

# Database Query GUI

Lorenzo Maria Alberto Paoria

26 febbraio 2025

## Sommario

Il progetto "Piattaforma di streaming musicale per sistemi distribuiti" nasce dall'idea di creare una soluzione tecnologica moderna per l'ascolto e la gestione di contenuti musicali in un ambiente distribuito. L'obiettivo principale è sviluppare una piattaforma che consenta agli utenti di accedere a un vasto catalogo musicale in modo rapido, affidabile e scalabile, sfruttando i vantaggi offerti dai sistemi distribuiti.

## 1 L'idea

L'idea iniziale parte dall'osservazione di come le piattaforme di streaming musicale abbiano rivoluzionato il modo in cui consumiamo musica. Tuttavia, queste soluzioni spesso presentano complessità tecniche legate alla gestione dei dati, alla sincronizzazione degli utenti e alla scalabilità del sistema per supportare un numero crescente di utenti contemporanei.

La piattaforma proposta si propone di affrontare queste sfide attraverso un'architettura distribuita. Quindi utilizzare una progettazione che sfrutti database distribuiti e microservizi per garantire un alto livello di disponibilità e resilienza. Oltre alla gestione scalabile degli utenti: Consentire il supporto per milioni di utenti contemporanei senza compromettere le prestazioni.

Il progetto si basa su tecnologie moderne di programmazione e database distribuiti, utilizzando il linguaggio Java per lo sviluppo del backend e le librerie necessarie per la comunicazione tra i vari componenti del sistema oltre all'utilizzo di phpMyAdmin come gestore del database.



Figura 1: Logo del programma creato tramite IA

## 2 Tecnologie utilizzate

### 2.1 Java

Linguaggio di programmazione versatile e orientato agli oggetti, scelto per la sua affidabilità e ampia compatibilità. È usato per implementare la logica di business e la gestione dei processi di rete nel progetto. Nel progetto copre il ruolo per l'implementazione del backend, con particolare attenzione alla gestione della comunicazione tra nodi distribuiti e alla manipolazione dei dati. Oltre alla GUI dell'applicativo creata utilizzando la libreria Jframe. Inoltre una api utilizzata è JDBC (Java Database Connectivity), API standard fornita da Java per consentire alle applicazioni Java di interagire con database relazionali. È un ponte che permette al codice Java di inviare query SQL al database, recuperare i risultati, e gestire i dati.

### 2.2 phpMyAdmin

phpMyAdmin è uno strumento open-source basato sul web per la gestione di database MySQL. Fornisce un'interfaccia grafica (GUI) che semplifica l'interazione con i database, rendendo accessibili operazioni che altrimenti richiederebbero la scrittura di comandi SQL.

### 2.3 Maven

Strumento di gestione delle build e delle dipendenze per progetti Java. Automatizza il processo di compilazione, test e packaging del software. Consente di includere librerie e framework necessari al progetto semplicemente aggiungendo le loro coordinate al file pom.xml. Le dipendenze vengono automaticamente scaricate da repository remoti come Maven Central.

## 3 Database

Il database della piattaforma di streaming musicale è progettato per gestire vari aspetti di un sistema di streaming, tra cui gli utenti, i contenuti musicali (come album e canzoni), le playlist, le preferenze musicali, e i pagamenti degli utenti.

Al centro del sistema ci sono le tabelle che gestiscono i contenuti musicali, come la tabella `Contenuto`, che memorizza le canzoni o altri tipi di contenuti, e la tabella `Album`, che collega le canzoni agli album di un artista. Gli artisti sono memorizzati in una tabella separata, `Artista`, che tiene traccia di ogni artista tramite il suo nome.

Il database prevede anche la gestione dei tipi di utenti (come 'free' o 'premium') nella tabella `TipoUtente`, e le informazioni degli utenti stessi nella tabella `Utente`, che include dettagli come nome, cognome, email, e anche la password. Gli utenti possono anche avere preferenze per i generi musicali, salvate nella tabella `PreferenzaGenere`, e possono creare playlist per organizzare i contenuti a loro piacimento, questo è permesso solo agli utenti di tipo premium. Le playlist degli utenti sono gestite nella tabella `Playlist`, mentre i contenuti aggiunti alle playlist sono tracciati nella tabella `ContenutiPlaylist`.

Inoltre, il database tiene traccia delle riproduzioni dei contenuti tramite la tabella `RiproduzioneContenuto`, che registra ogni volta che un utente ascolta un contenuto. C'è anche una tabella che gestisce le relazioni fra i contenuti e i generi musicali, chiamata `AppartieneGenere`, permettendo di associare vari contenuti ai generi musicali corrispondenti.

Nel complesso, questo database è progettato per supportare tutte le funzionalità principali di una piattaforma di streaming musicale, offrendo un'architettura relazionale che consente di organizzare, tracciare e gestire contenuti musicali, utenti, playlist, preferenze, abbonamenti e pagamenti in modo efficiente.

## 4 Aspect-Oriented Programming (AOP)

L'Aspect-Oriented Programming (AOP) è utilizzato per gestire le operazioni di logging in modo trasversale, senza modificare il codice principale delle classi. È stato scelto per separare logiche di debug (come il logging) dalla logica di base del programma.

## 5 Schema progetto Maven

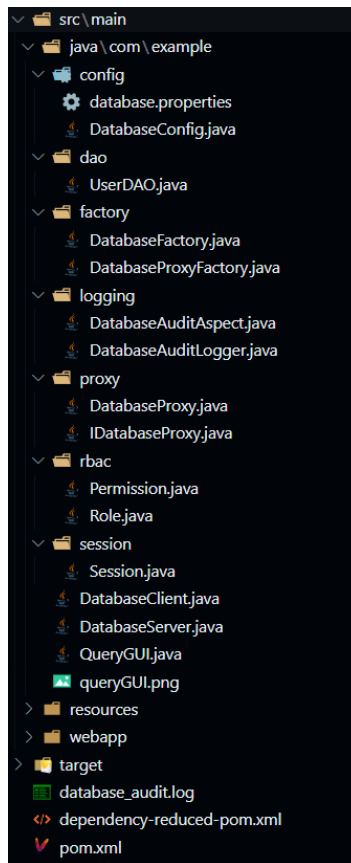


Figura 2: Schema di maven

Una lista delle classi presente è:

1. DatabaseServer: La classe principale che gestisce il server, le sessioni, i ruoli e le connessioni dei client.
2. QueryGUI: L'interfaccia grafica per l'utente che interagisce con il server tramite un proxy.
3. Session: Rappresenta una sessione utente con un ID, un utente e i ruoli attivi.
4. Permission: Rappresenta un permesso con un'operazione e un oggetto.
5. Role: Rappresenta un ruolo con una lista di permessi.
6. DatabaseConfig: Gestisce la configurazione del database.
7. database.properties: un file contenente indirizzo ip, porta, url per jdbc e username e password per entrare nel database.
8. UserDAO: Gestisce l'accesso ai dati degli utenti, inclusa l'autenticazione e l'esecuzione delle query.
9. DatabaseFactory: Fornisce una connessione al database.
10. DatabaseProxyFactory: Fornisce un'istanza dell'interfaccia al proxy per il database.
11. DatabaseAuditAspect: Gestisce il logging delle operazioni di autenticazione e query tramite AspectJ.

12. DatabaseAuditLogger: Gestisce il logging delle operazioni di audit.
13. DatabaseProxy: Gestisce la connessione al server e l'invio di richieste.
14. IDatabaseProxy: Fornisce un'interfaccia per l'uso di DatabaseServer e DatabaseProxy.

## 5.1 Dipendenze di Maven

Per il corretto funzionamento di Maven con il programma sono state usate delle dipendenze come:

- MySQL connector, per la connessione al database MySQL
- Flatlaf, per la costruzione della gui
- HikariCP, per la pool di connessioni effettuabili
- AspectJ Runtime
- AspectJ Weaver

## 6 Design Pattern utilizzati

### 6.1 Role-Based Access Control (RBAC) Pattern

Il Role-Based Access Control (RBAC) viene implementato tramite le classi Role, Permission e Session. Questo pattern è utilizzato per gestire il controllo degli accessi basato sui ruoli degli utenti. È stato scelto per garantire una gestione flessibile dei permessi, separare l'autenticazione dall'autorizzazione e permettere un controllo degli accessi gerarchico.

Nel flusso del programma, il RBAC interviene dopo l'autenticazione, creando una sessione con i ruoli appropriati (ad esempio, "free", "premium" o "admin"). Prima di eseguire ogni query, il sistema verifica i permessi associati ai ruoli attivi nella sessione. Se l'utente non ha i permessi necessari per eseguire una determinata operazione, l'accesso viene negato.

### 6.2 Proxy Pattern

Il Proxy Pattern viene implementato tramite la classe DatabaseProxy e la sua interfaccia IDatabaseProxy. Questo pattern funge da intermediario tra il client e il server del database, gestendo la comunicazione di rete e proteggendo l'accesso al database. È stato scelto per centralizzare la gestione delle richieste, mantenere lo stato della sessione e semplificare l'interazione tra il client e il server.

Nel flusso del programma, il IDatabaseProxy riceve le richieste dal client (ad esempio, autenticazione o esecuzione di query), le inoltra al server e restituisce i risultati al client. Inoltre, il proxy gestisce la connessione al server e verifica che la sessione sia valida prima di eseguire qualsiasi operazione.

### 6.3 Factory Pattern

Il Factory Pattern viene implementato tramite le classi DatabaseFactory e DatabaseProxyFactory. Questo pattern è utilizzato per centralizzare la creazione di oggetti complessi, come le connessioni al database e le istanze del proxy.

DatabaseFactory: Si occupa della creazione e gestione delle connessioni al database. Utilizza le configurazioni fornite da DatabaseConfig per stabilire la connessione. È stato scelto per evitare la duplicazione del codice di connessione e per fornire un punto unico di configurazione.

DatabaseProxyFactory: Si occupa della creazione e gestione di un'istanza di DatabaseProxy, configurata con i parametri forniti da DatabaseConfig. Questo approccio garantisce che ci sia una sola istanza del proxy, evitando problemi di consistenza e sprechi di risorse.

## 6.4 Authenticator Single Sign-On Pattern

L'Authenticator Single Sign-On Pattern viene implementato tramite la classe Session. Questo pattern gestisce le sessioni degli utenti, memorizzando dati come l'ID utente, il timestamp di accesso e i ruoli attivi. Il metodo `isExpired()` verifica se la sessione è ancora valida confrontando il tempo attuale con il tempo di scadenza (impostato a 5 minuti).

Nel flusso del programma, una nuova sessione viene creata dal DatabaseServer quando un utente si autentica con successo. Prima di eseguire qualsiasi operazione, il sistema verifica se la sessione è scaduta chiamando `isExpired()`. Se la sessione è scaduta, l'utente deve riautenticarsi; altrimenti, l'operazione viene eseguita.

## 6.5 Data Access Object (DAO) Pattern

Il Data Access Object (DAO) Pattern viene implementato tramite la classe UserDAO. Questo pattern fornisce uno strato di astrazione tra l'applicazione e il database, separando la logica di accesso ai dati dalla logica di business. È stato scelto per incapsulare le query SQL, semplificare la gestione delle operazioni sul database e garantire un'interfaccia coerente per l'accesso ai dati.

Nel flusso del programma, il UserDAO gestisce l'autenticazione degli utenti, esegue le query sul database e formatta i risultati prima di restituirli. Ad esempio, durante l'autenticazione, il UserDAO verifica le credenziali dell'utente e restituisce il tipo di utente ("free" o "premium"). Durante l'esecuzione di una query, il UserDAO esegue la query sul database e formatta i risultati in una stringa leggibile.

## 7 Funzionalità di sicurezza

Le funzionalità di sicurezza includono un meccanismo di timeout della sessione, il controllo di accesso basato sui ruoli (RBAC), il logging di audit per autenticazioni e query, e un controllo granulare dei permessi.

### 7.1 Timer di sessione

Il sistema implementa un timeout di sessione automatico per proteggere gli utenti da accessi non autorizzati. Ogni sessione ha un limite di tempo di 5 minuti di inattività, il tempo viene aggiornato ad ogni interazione dell'utente e se una sessione scade, l'utente deve riautenticarsi. Inoltre questo protegge da sessioni rimaste aperte e abbandonate.

Listing 1: Meccanismo session timeout

```
public class Session {
    private static final int SESSION_TIMEOUT_MINUTES = 5;
    private LocalDateTime lastAccessTime;

    public boolean isExpired() {
        return LocalDateTime.now().minusMinutes(SESSION_TIMEOUT_MINUTES)
            .isAfter(lastAccessTime);
    }

    private void updateLastAccessTime() {
        lastAccessTime = LocalDateTime.now();
    }
}
```

### 7.2 Controllo Accessi Basato sui Ruoli (RBAC)

Il sistema utilizza un sofisticato sistema RBAC con tre livelli di accesso:

- Utente Free: può solo eseguire query SELECT

- Utente Premium: può eseguire SELECT, INSERT e UPDATE
  - Admin: ha accesso completo (SELECT, INSERT, UPDATE, DELETE, CREATE, DROP)
- Ogni operazione viene eseguita solo dopo la verifica dei permessi assegnati al ruolo dell'utente.

Listing 2: Meccanismo di controllo per le query basato sul ruolo

```
public class DatabaseServer {
    private boolean validateQueryPermissions(String query, Set<Role> roles) {
        String operation = extractOperation(query);
        String object = extractObject(query);

        return roles.stream()
            .anyMatch(role -> role.hasPermission(new Permission(operation, object)) ||
                role.hasPermission(new Permission(operation, "*")));
    }
}
```

### 7.3 Sistema di logging

Il sistema mantiene un registro dettagliato di tutte le attività per scopi di sicurezza e debugging:

- Registra ogni tentativo di autenticazione (riuscito o fallito)
- Traccia tutte le query eseguite con il loro risultato
- Memorizza informazioni contestuali come ID cliente e timestamp
- Permette di ricostruire la sequenza degli eventi in caso di problemi

Listing 3: Meccanismo di audit logging

```
public class DatabaseAuditLogger {
    public void logAuthentication(String clientId, String sessionId, String email,
        String tipoUtente, boolean success) {
        logger.info(String.format("[%s] - Authentication attempt -- Client: %s,
        ----- Session: %s, - User: %s, - Tipo - Utente: %s, - Success: %s", LocalDateTime.now(),
            clientId, sessionId, email, tipoUtente, success));
    }

    public void logQuery(String sessionId, String query, boolean success) {
        logger.info(String.format("[%s] - Query execution -- Session: %s,
        ----- Query: %s, - Success: %s", LocalDateTime.now(), sessionId, query, success));
    }
}
```

### 7.4 Prevenzione SQL injection

Prevenzione SQL Injection e Controllo Permessi Granulare Il sistema implementa diverse misure per prevenire attacchi SQL:

- Uso di PreparedStatement per ogni query al database
- Validazione delle query prima dell'esecuzione
- Controllo granulare dei permessi a livello di operazione e oggetto

Come funziona nel dettaglio:

1. L'utente invia una query
2. Il sistema estrae l'operazione (SELECT, INSERT, ecc.)
3. Verifica i permessi dell'utente per quell'operazione
4. Prepara la query usando PreparedStatement
5. Esegue la query solo se tutti i controlli sono passati

Listing 4: Prevenzione SQL injection

```
public class UserDao {
    public String authenticate(String email, String password) throws SQLException {
        // Uso di PreparedStatement per prevenire SQL injection
        String query = "SELECT-u.tipo FROM Utente-u WHERE u.email =-?-AND-u.passw =-?";
        try (PreparedStatement stmt = connection.prepareStatement(query)) {
            stmt.setString(1, email);
            stmt.setString(2, password);
            ResultSet rs = stmt.executeQuery();
            //ecc...
        }
    }
}
```

## 7.5 Lista di Aspect Pointcut

Questa lista di aspetti implementa un sistema di logging che:

Traccia chi si autentica al database (con clientId, sessionId, email e ruolo), registra quali query vengono eseguite e se hanno successo. Lo fa senza modificare il codice delle classi originali grazie all'AOP. Il tutto è progettato per funzionare in un ambiente multi-thread, mantenendo separati i contesti di esecuzione.

- readObjectPointcut(): Intercetta le chiamate a ObjectInputStream.readObject() all'interno di ClientHandler
- authenticationPointcut(): Intercetta il metodo handleAuthentication()
- queryPointcut(): Intercetta il metodo handleQuery()

### 7.5.1 Aspetti per l'autenticazione

**beforeAuthentication()** si attiva prima dell'esecuzione di handleAuthentication(), crea un nuovo AuthContext e memorizza il clientId e salva il contesto nel ThreadLocal.

**afterReadAuthData()** si attiva dopo la lettura di oggetti durante l'autenticazione, cattura i dati letti (email dell'utente) e li aggiunge al contesto.

**afterAuthentication()** si attiva dopo che l'autenticazione è completata, recupera informazioni sul ruolo dell'utente e il successo dell'operazione, registra l'evento di autenticazione tramite auditLogger e rimuove il contesto dal ThreadLocal.

### 7.5.2 Aspetti per le query

**beforeQuery()** si attiva prima dell'esecuzione di handleQuery(), crea un nuovo QueryContext e salva il contesto nel ThreadLocal.

**afterReadQueryData()** si attiva dopo la lettura di oggetti durante l'elaborazione di una query, cattura prima il sessionId e poi la query stessa e aggiunge questi dati al contesto.

**afterQuery()** si attiva dopo che la query è stata eseguita, determina se la query ha avuto successo controllando il risultato, registra l'evento di query tramite auditLogger e rimuove il contesto dal ThreadLocal.

## 8 Diagramma UML delle classi

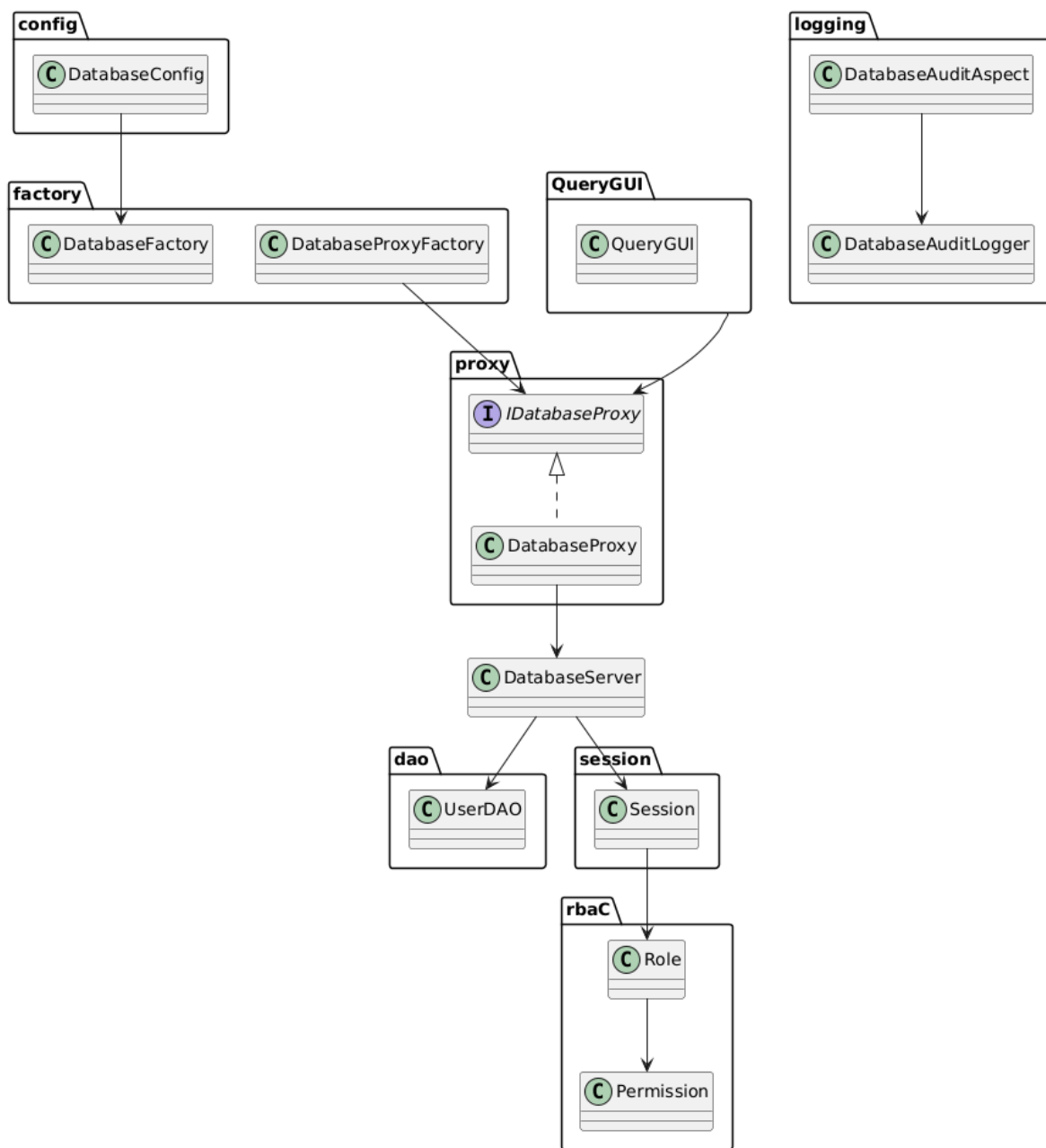


Figura 3: Diagramma UML delle classi



## 9 Diagramma di sequenza del programma

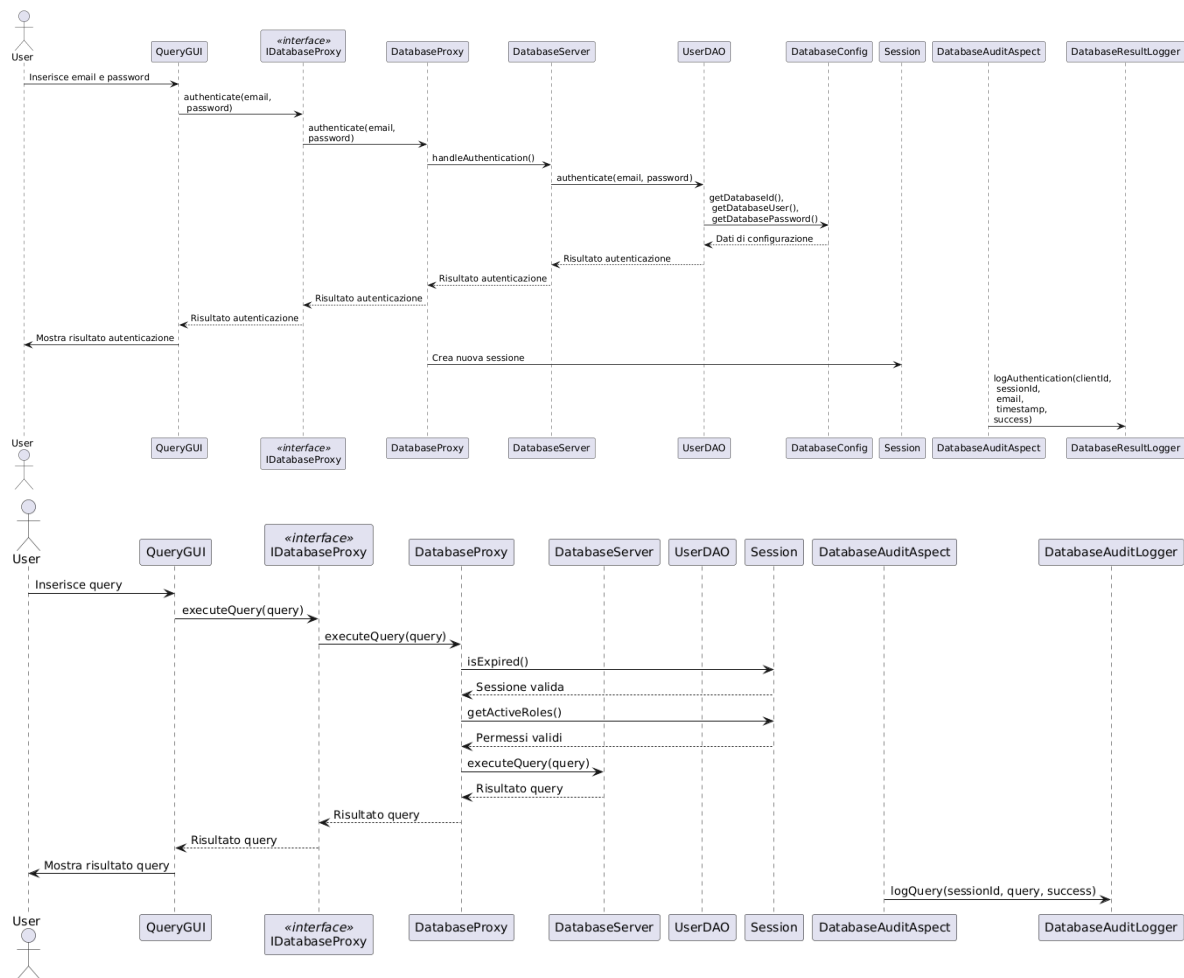


Figura 4: Diagramma di sequenza del programma

## 10 Foto esecuzione del programma

Il primo passaggio è l'avvio del server che si deve collegare al database.

Una volta che il server è in ascolto, i client possono essere avviati:

Quando il client si avvia e dopo aver fatto l'accesso, possono essere fatte tutte le query al database che si vogliono, purché siano permesse. Sotto, un esempio di query accettata e rifiutata:

```

→mvnProject git:(main) mvn -Pserver exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:mvnProject >-----
[INFO] Building mvnProject 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ mvnProject ---
gen 27, 2025 1:46:00 PM com.example.DatabaseServer main
INFO: Starting Database Server...
gen 27, 2025 1:46:00 PM com.example.DatabaseServer start
INFO: Server listening on port 12345

```

Figura 5: Avvio del server

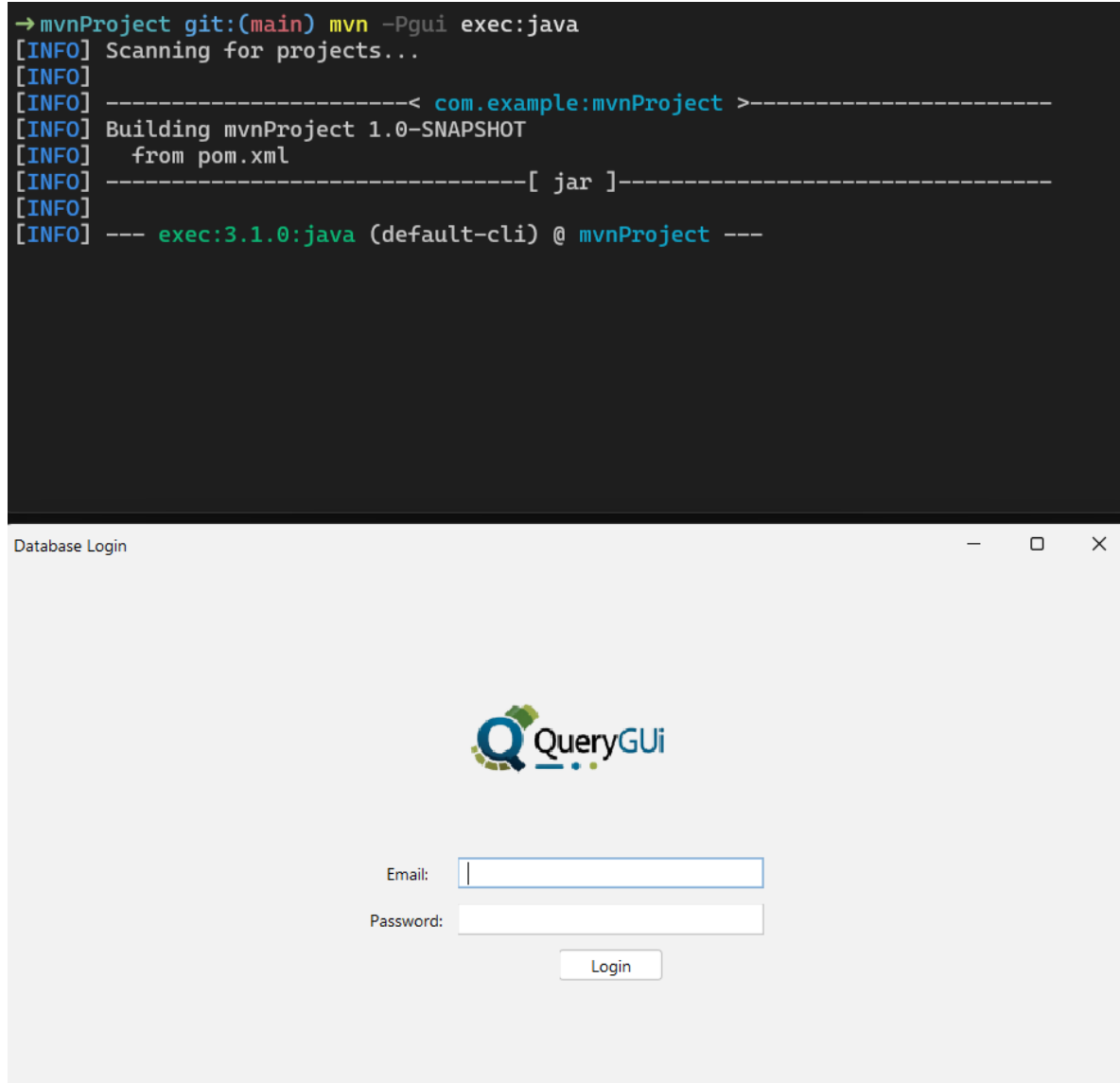
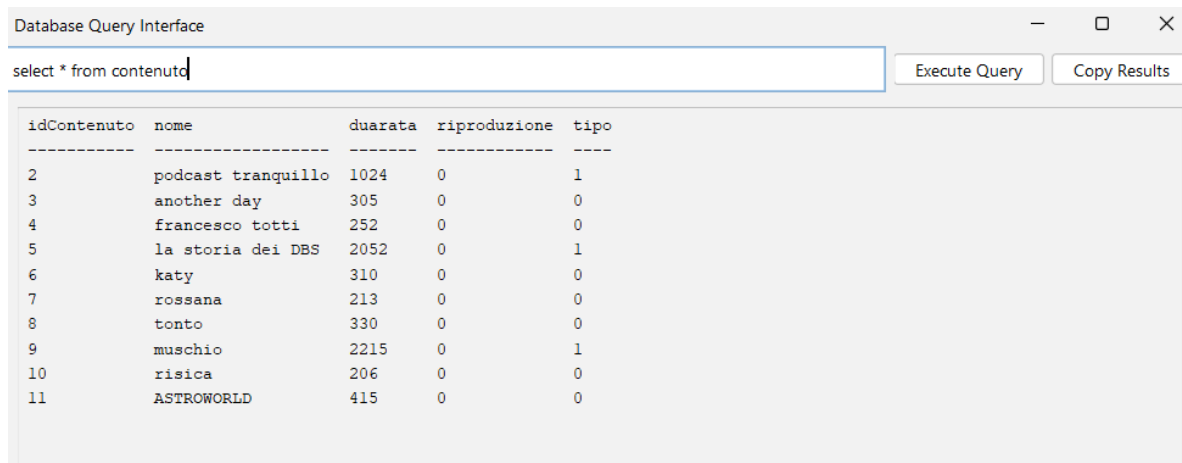


Figura 6: Avvio della GUI del client



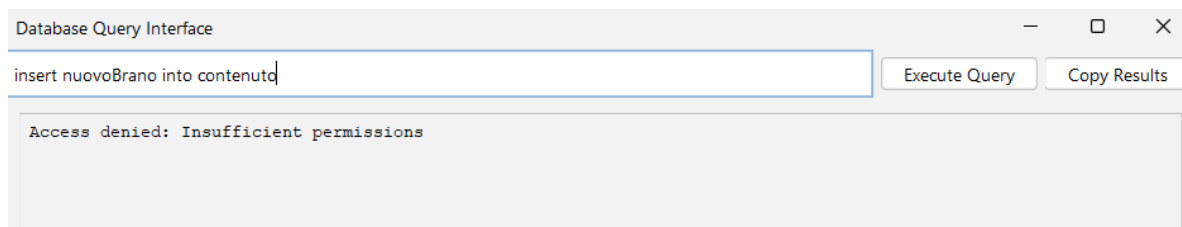
Database Query Interface

select \* from contenuto

Execute Query Copy Results

idContenuto	nome	duarata	riproduzione	tipo
2	podcast tranquillo	1024	0	1
3	another day	305	0	0
4	francesco totti	252	0	0
5	la storia dei DBS	2052	0	1
6	katy	310	0	0
7	rossana	213	0	0
8	tonto	330	0	0
9	muschio	2215	0	1
10	risica	206	0	0
11	ASTROWORLD	415	0	0

Figura 7: Esecuzione query - Successo



Database Query Interface

insert nuovoBrano into contenuto

Execute Query Copy Results

Access denied: Insufficient permissions

Figura 8: Esecuzione query - Fallimento

## 11 Test

Per testare meglio i tipi di risultati ottenuti e la funzionalità del programma, possiamo avviare il test creato che simula le stesse query con due account differenti: uno di tipo free e l'altro di tipo premium.

```
→mvnProject git:(main) mvn -Pclient exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.example:mvnProject >-----
[INFO] Building mvnProject 1.0-SNAPSHOT
[INFO]    from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ mvnProject ---

--- Testing User: margheritaursino@gmail.com (Role: free) ---
Authentication Result: Authentication successful:b9e7733d-9978-4620-a4eb-d2b41863b988

Query Test:
SQL: SELECT * FROM Contenuto LIMIT 5
Expected Allowed: true
User Role: free
Query Result: idContenuto      nome      duarata riproduzione      tipo
2      podcast tranquillo      1024      0      1
3      another day      305      0      0
4      francesco totti 252      0      0
5      la storia dei DBS      2052      0      1
6      katy      310      0      0

Permission Check: PASS

Query Test:
SQL: UPDATE Contenuto SET nome = 'Test Title' WHERE idContenuto = 1
Expected Allowed: true
User Role: free
Query Result: Access denied: Insufficient permissions
Permission Check: PASS

Query Test:
SQL: INSERT INTO Contenuto (nome, duarata, riproduzione, tipo) VALUES ('New Song', 180, 0, 1)
Expected Allowed: true
User Role: free
Query Result: Access denied: Insufficient permissions
Permission Check: PASS

Query Test:
SQL: DELETE FROM Contenuto WHERE idContenuto = 1
Expected Allowed: false
User Role: free
Query Result: Access denied: Insufficient permissions
Permission Check: PASS
```

Figura 9: Client Free - Test delle query

Come vediamo, per l'utente di tipo free l'unica query accettata è stata la SELECT.

```

--- Testing User: annapistorio@gmail.com (Role: premium) ---
Authentication Result: Authentication successful:100febf7-8791-4260-af31-a7d6e8754605

Query Test:
SQL: SELECT * FROM Contenuto LIMIT 5
Expected Allowed: true
User Role: premium
Query Result:


|   | idContenuto        | nome | duarata | riproduzione | tipo |
|---|--------------------|------|---------|--------------|------|
| 2 | podcast tranquillo | 1024 | 0       | 1            |      |
| 3 | another day        | 305  | 0       | 0            |      |
| 4 | francesco totti    | 252  | 0       | 0            |      |
| 5 | la storia dei DBS  | 2052 | 0       | 1            |      |
| 6 | katy               | 310  | 0       | 0            |      |


Permission Check: PASS

Query Test:
SQL: UPDATE Contenuto SET nome = 'Test Title' WHERE idContenuto = 1
Expected Allowed: true
User Role: premium
Query Result: 0 rows affected
Permission Check: PASS

Query Test:
SQL: INSERT INTO Contenuto (nome, duarata, riproduzione, tipo) VALUES ('New Song', 180, 0, 1)
Expected Allowed: true
User Role: premium
Query Result: 1 rows affected
Permission Check: PASS

Query Test:
SQL: DELETE FROM Contenuto WHERE idContenuto = 1
Expected Allowed: false
User Role: premium
Query Result: Access denied: Insufficient permissions
Permission Check: PASS

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.539 s
[INFO] Finished at: 2025-01-27T13:50:51+01:00
[INFO] -----

```

Figura 10: Client Premium - Test delle query

Mentre per l'utente di tipo premium sono state accettate le query `SELECT`, `UPDATE` e `INSERT`, mentre la `DELETE` è stata rifiutata in quanto è un comando riservato agli admin.