Università degli studi – Insubria

Corso di informatica

Manuale Tecnico

Laboratorio interdisciplinare B

L'applicazione è sviluppata in Java ed è organizzata in base a tre moduli:

Common:

Si tratta del modulo che implementa gli oggetti remoti utilizzati in RMI e la **ServerInterface**, da cui provengono i moduli ereditati dal server.
Conterrà le seguenti classi:

- Canzone: classe usata per gestire le canzoni, che implementa gli **attributi**:
  - **titolo**: titolo della canzone
  - **autore**: autore della canzone
  - **anno**: anno di pubblicazione della canzone
- Emozione: classe usata per gestire le emozioni, che implementa gli **attributi**:
  - **categoria**: categoria dell'emozione provata
  - **punteggio**: punteggio assegnato all'emozione
  - **note**: eventuali note dell'utente
  - **canzone**: titolo della canzone
  - **autore**: autore della canzone
  - **utente**: utente che ha inserito l'emozione
- Playlist: classe usata per gestire le playlist, che implementa gli **attributi**:
  - **nomePlaylist**: nome della playlist
  - **listaCanzoni**: lista di canzoni della playlist
  - **nomeUtente**: nome dell'utente che possiede la playlist
- Utente: classe usata per gestire gli utenti, che implementa gli **attributi**:
  - **username**: nome utente dell'utente
  - **nome**: nome dell'utente
  - **cognome**: cognome dell'utente
  - **indirizzo**: indirizzo dell'utente
  - **email**: email dell'utente
  - **password**: password dell'utente
- ServerInterface: classe usata per rendere implementabili i **metodi**:
  - **Registrazione**: metodo per effettuare la registrazione
  - **Login**: metodo per effettuare il login
  - **InserisciEmozione**: metodo per inserire un'emozione
  - **CreaPlaylist**: metodo per creare una playlist
  - **InserisciCanzone**: metodo per inserire una canzone
  - **EliminaCanzone**: metodo per eliminare una canzone
  - **CercaBranoT**: metodo per cercare un brano per titolo
  - **CercaBranoA**: metodo per cercare un brano per autore
  - **CercaBranoY**: metodo per cercare un brano per anno
  - **VisualizzaPlaylist**: metodo per visualizzare le playlist
  - **VisualizzaCanzoni**: metodo per visualizzare le canzoni
  - **CercaEmozioni**: metodo per cercare le emozioni registrate

ClientES:

Si tratta del modulo che implementa l'interfaccia utente ed effettua le operazioni RMI per richiamare i metodi del server.
Conterrà le seguenti classi:

- Client: classe usata per richiamare i metodi del server con RMI, in base alle operazioni scelte sull'interfaccia che implementa i metodi:
    - **Connessione**: per eseguire la connessione con il registro RMI
    - **Registrazione**: che richiama il metodo server "Registrazione"
    - **Login**: che richiama il metodo server "Login"
    - **InserisciEmozione**: che richiama il metodo server "InserisciEmozione"
    - **CreaPlaylist**: che richiama il metodo server "CreaPlaylist"
    - **InserisciCanzone**: che richiama il metodo server "InserisciCanzone"
    - **EliminaCanzone**: che richiama il metodo server "EliminaCanzone"
    - **CercaBranoT**: che richiama il metodo server "CercaBranoT"
    - **CercaBranoA**: che richiama il metodo server "CercaBranoA"
    - **CercaBranoY**: che richiama il metodo server "CercaBranoY"
    - **VisualizzaPlaylist**: che richiama il metodo server "VisualizzaPlaylist"
    - **VisualizzaCanzoni**: che richiama il metodo server "VisualizzaCanzoni"
    - **CercaEmozioni**: che richiama il metodo server "CercaEmozioni"
    - **main**: crea un'istanza della classe e la fa partire
- InterfacciaApplication: classe usata per inizializzare il lato client del programma, che avrà un metodo **main** per inizializzare l'interfaccia.
- InterfacciaInizialeController: classe usata per controllare e implementare le funzioni dell'interfaccia grafica, attraverso i seguenti metodi:
    - **initialize**: inizializza l'interfaccia e richiama i metodi client della ricerca
    - **completaRegistrazioneButton**: richiama il metodo client corrispondente
    - **completaLoginButton**: richiama il metodo client corrispondente
    - **inserisciEmozioniCButton**: richiama il metodo client corrispondente
    - **creaPlaylistCButton**: richiama il metodo client corrispondente
    - **aggiungiCanzoneCButton**: richiama il metodo client corrispondente
    - **eliminaCanzoneCButton**: richiama il metodo client corrispondente
    - **visualizzaEmozioniCButton**: richiama il metodo client corrispondente
    - **logout**: effettua il logout
    - **visualizzaPlaylistButton**: richiama il metodo client corrispondente
    - **visualizzacanzoniButton**: richiama il metodo client corrispondente
    - **i metodi rimanenti** sono utilizzati per funzioni di interfaccia

ServerES:

Si tratta del modulo che implementa le funzioni per la connessione al database, implementa i metodi utilizzati nel programma e offre una connessione RMI.
Conterrà le seguenti classi:

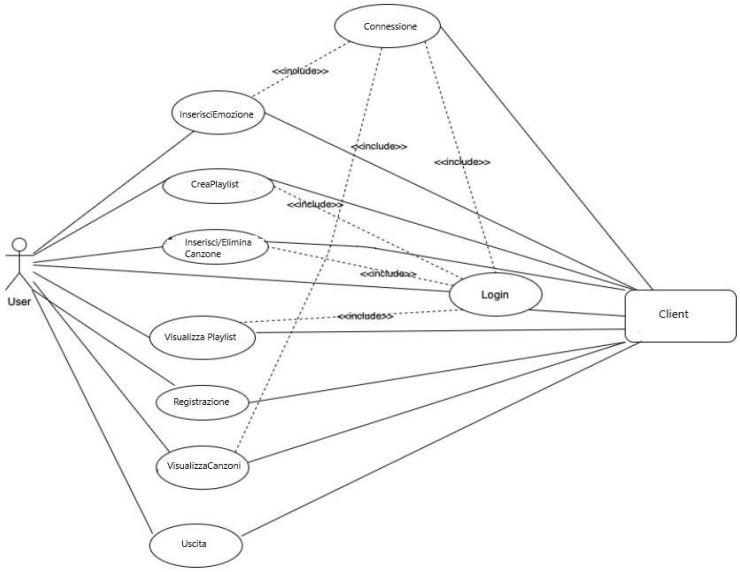
- DataBaseHandler: classe usata per connettersi al database Postgre, ed implementa i seguenti metodi:
    - **getInstance**: cattura l'istanza attuale della connessione
        - **Complessità: O(1)**
    - **connectDB**: si connette al database Postgre, inserendo anche username e password di Posgre e richiamando il metodo fillTableEmozione
        - **Complessità: O(1)**
    - **insert**: metodo per eseguire query di inserimento
        - **Complessità: O(1)**
    - **select**: metodo per eseguire query di selezione
        - **Complessità: O(1)**
    - **disconnect**: permette di disconnettersi dal database Postgre
        - **Complessità: O(1)**
    - **fillTableEmozione**: metodo per inserire le emozioni base nel database, in base al valore della connessione che gli è stata passata
        - **Complessità: O(1)**
- ServerImpl: classe usata per implementare i metodi della ServerInterface, che eseguiranno le operazioni utili alla modifica del database Postgre
    - **run**: crea un'istanza del server, crea la connessione al registro RMI e al database, ritornando informazioni sull'esito di queste operazioni
        - **Complessità: O(1)**
    - **Registrazione**: dato in input un oggetto utente, lo inserisce nel database, verificando, allo stesso tempo, se questo è già presente nel database
        - **Complessità: lineare(O(n))**
    - **Login**: dati in input un username ed una password, verificherà se ci sono utenti nel database con i dati corrispondenti e ritornerà il risultato di questa operazione
        - **Complessità: lineare(O(n))**
    - **InserisciEmozione**: dato in input un oggetto emozione, cerca la canzone corrispondente, se la trova, inserisce l'emozione nel database, verificando, allo stesso tempo, se questa è già presente nel database(per la stessa canzone, la stessa emozione non può essere registrata più di una volta)
        - **Complessità: lineare(O(n))**
    - **CreaPlaylist**: dato in input il nome della playlist, la canzone da inserire e l'utente che l'ha create, crea una playlist, verificando, allo stesso tempo, se questa è già presente nel database
        - **Complessità: lineare(O(n))**
    - **InserisciCanzone**: dato in input la canzone da inserire e la playlist in cui inserirla, cerca la playlist corrispondente nel database e se esiste, inserisce la canzone(se esiste nel database), verificando, allo stesso tempo, se questa è già presente nella playlist caricata nel database
        - **Complessità: lineare(O(n))**

- **EliminaCanzone**: dato in input la canzone da inserire e la playlist da cui eliminarla, cerca la playlist corrispondente nel database e se esiste, elimina la canzone(se esiste nel database) da quest'ultima, verificando, allo stesso tempo, se questa è presente nella playlist caricata nel database
    - **Complessità: lineare(O(n))**
- **CercaBranoT**: dato in input il titolo della canzone da cercare, cerca la/le canzone/i presenti nel database
    - **Complessità: lineare(O(n))**
- **CercaBranoA**: dato in input l'autore della canzone da cercare, cerca la/le canzone/i presenti nel database
    - **Complessità: lineare(O(n))**
- **CercaBranoY**: dato in input l'anno di pubblicazione della canzone da cercare, cerca la/le canzone/i presenti nel database
    - **Complessità: lineare(O(n))**
- **VisualizzaPlaylist**: dato in input il nome dell'utente, ritorna una lista di tutte le playlist che ha registrato nel database
    - **Complessità: lineare(O(n))**
- **VisualizzaCanzoni**: ritorna una lista di tutte le canzoni presenti nel database
    - **Complessità: lineare(O(n))**
- **CercaEmozioni**: data in input una canzone da cercare, visualizza tutte le emozioni registrate per quella canzone(se esiste nel database), se presenti le ritorna come una lista
    - **Complessità: lineare(O(n))**

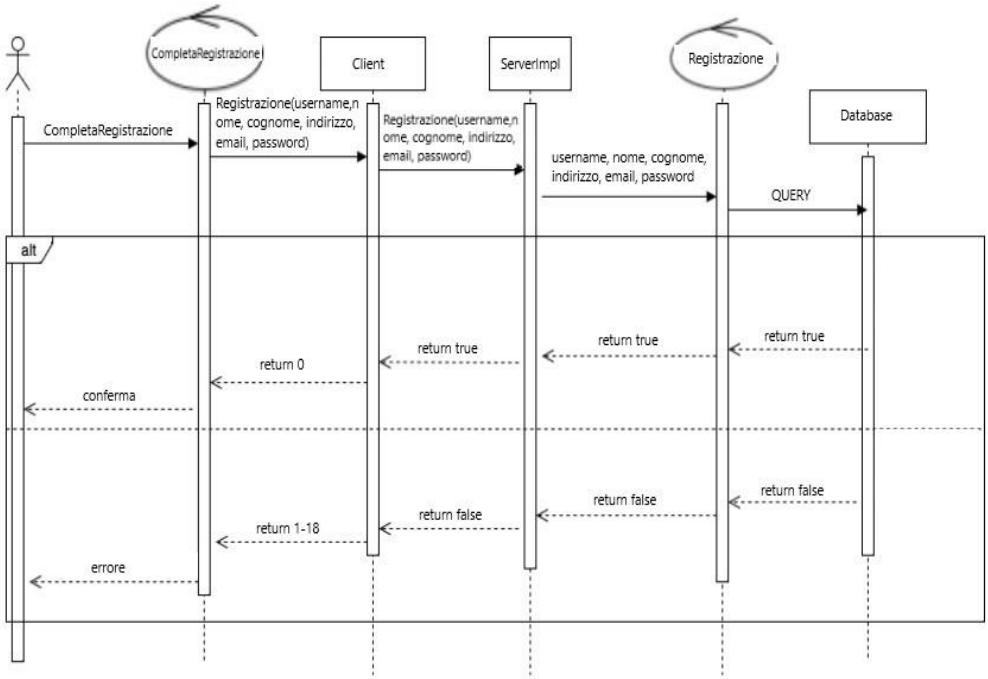
- ServerMain: classe che inizializza il server, mediante un metodo **main**

Diagrammi UML

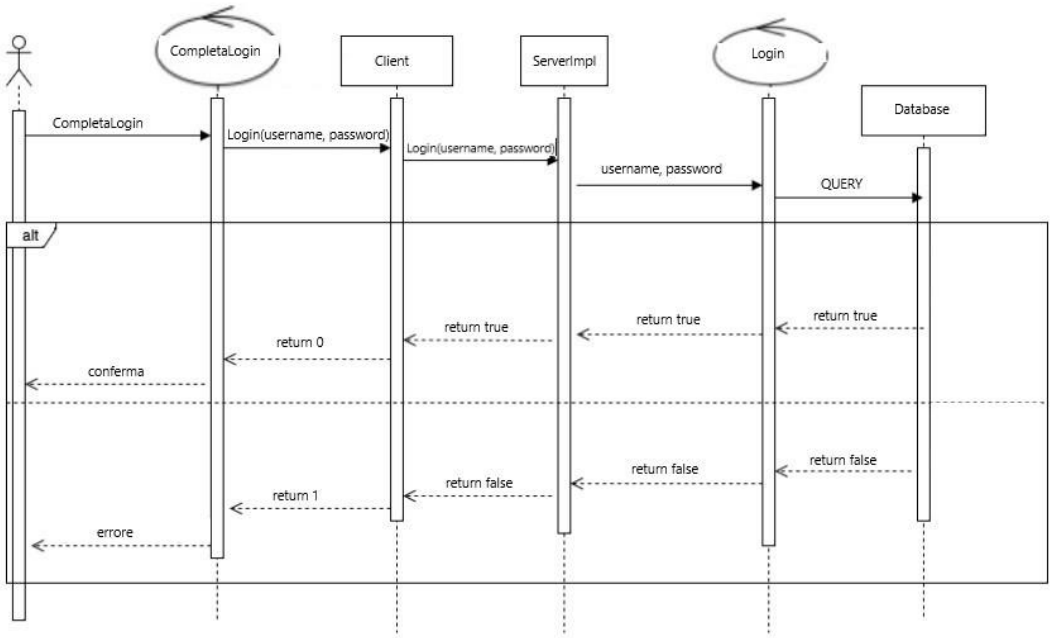
1. UseCase:



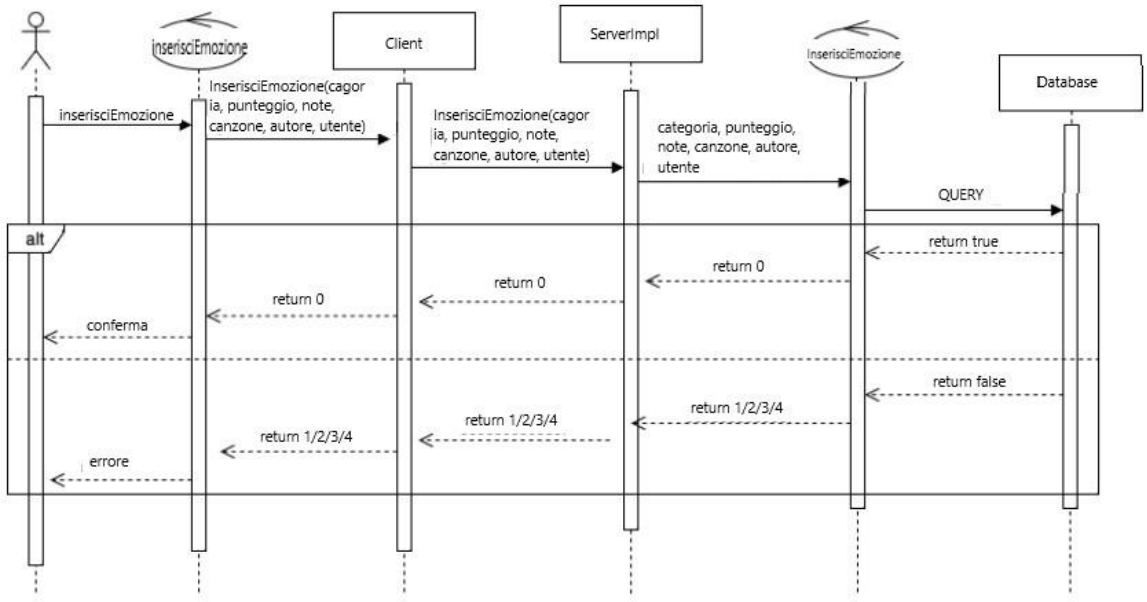
2. Sequence Registrazione:



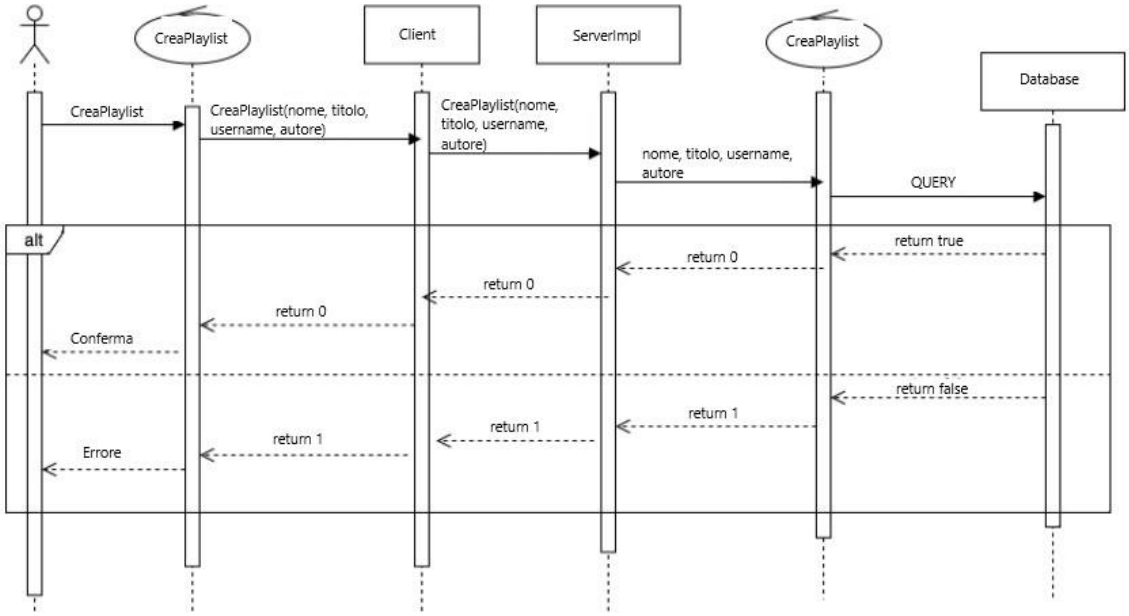
3. Sequence Login:



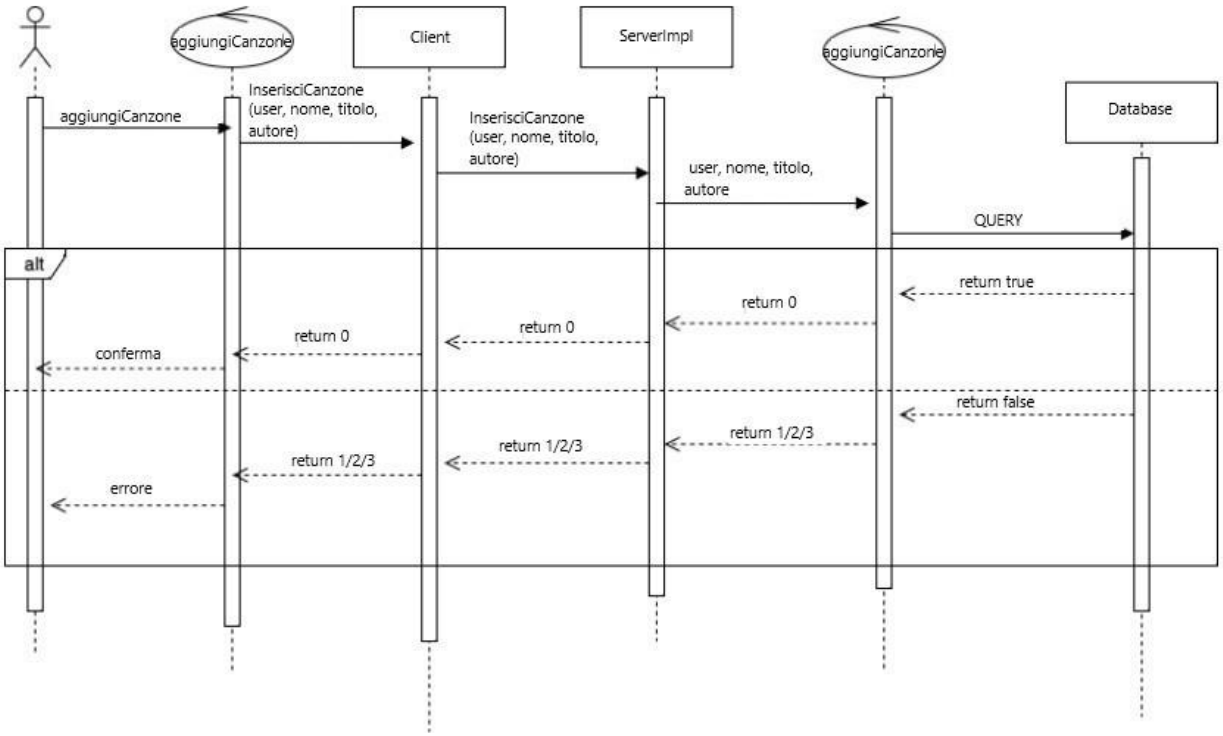
4. Sequence InserisciEmozione



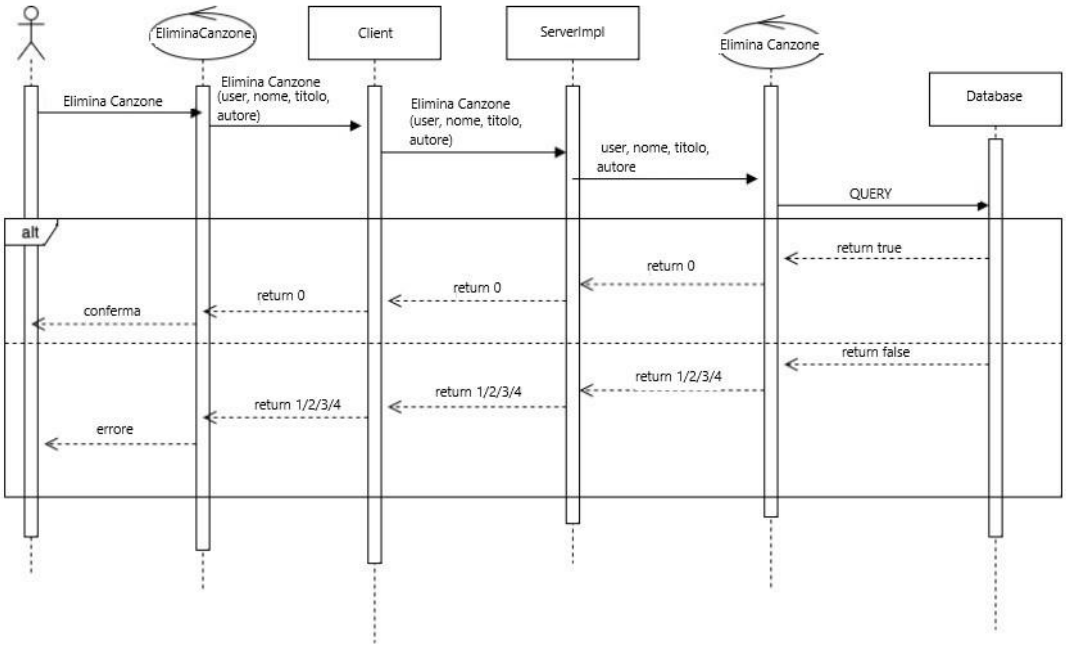
5. Sequence CreaPlaylist



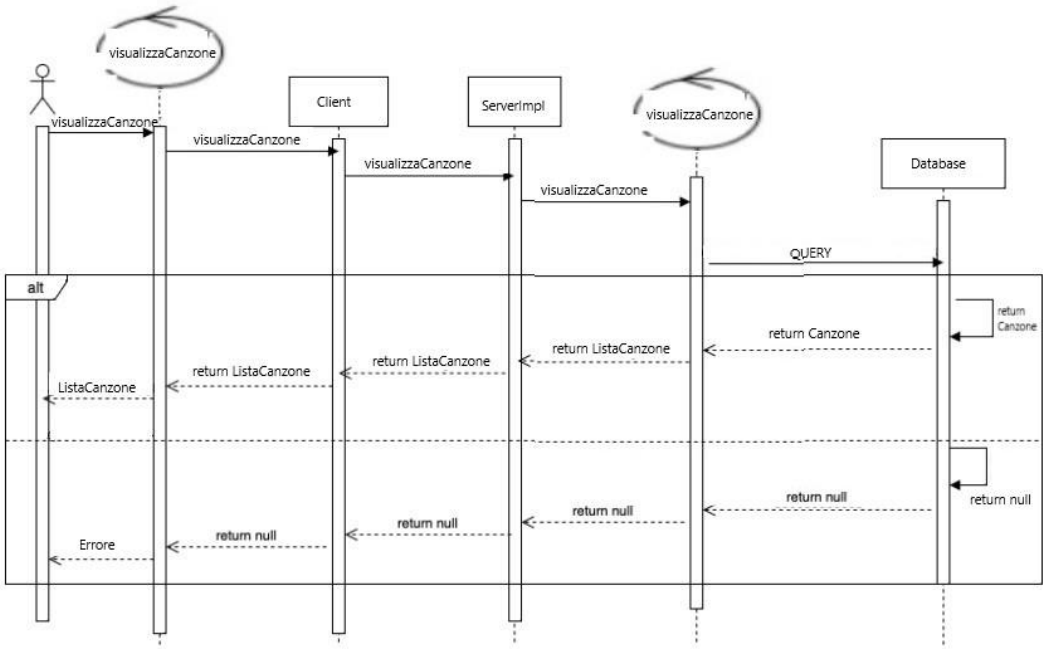
6. Sequence InserisciCanzone



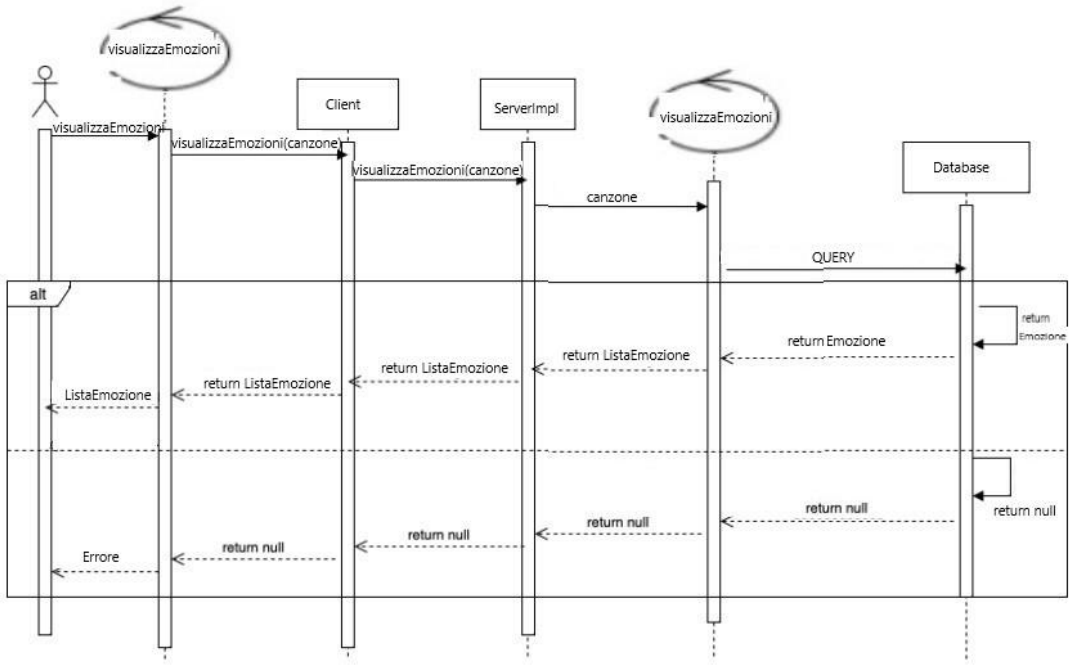
7. Sequence EliminaCanzone



8. Sequence VisualizzaCanzone



9. Sequence VisualizzaEmozione



10. ER