UNIVERSITÀ DI TRENTO

Dipartimento di Ingegneria
e Scienza dell'Informazione

Ingegneria del software

Documento di architettura
Versione: 0.01

# Web Music Player

*Gruppo T27*

Anno accademico 2022/2023

# Indice

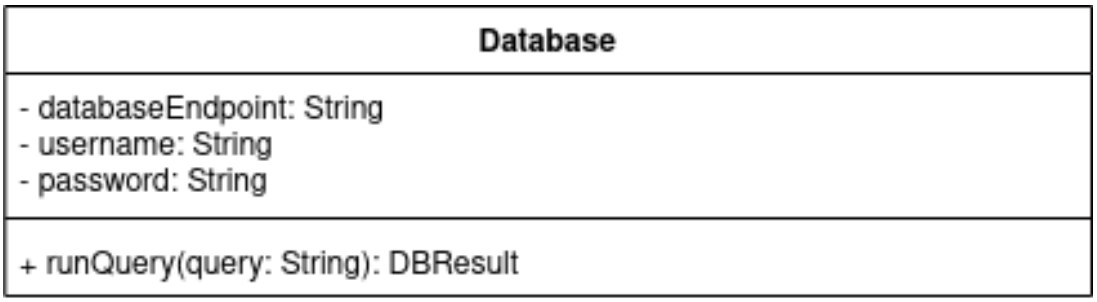
# 1 Scopo del documento

Il presente documento riporta l'analisi dell'architettura del progetto Web Music Player, sotto forma di classi e linguaggio OCL. Lo scopo di questo documento è quello di:

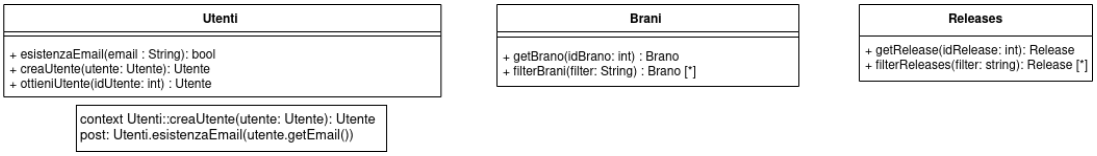
- elencare le classi utilizzate
- approfondirle mediante il linguaggio OCL
- presentare il diagramma delle classi

# 2 Elenco delle classi

1. **Database**




La classe dedita a comunicare col database, mandandogli le query e ritornando il risultato ricevuto.




```
context Utenti::creaUtente(utente: Utente): Utente
post: Utenti.esistenzaEmail(utente.getEmail())
```

2. **Utenti**

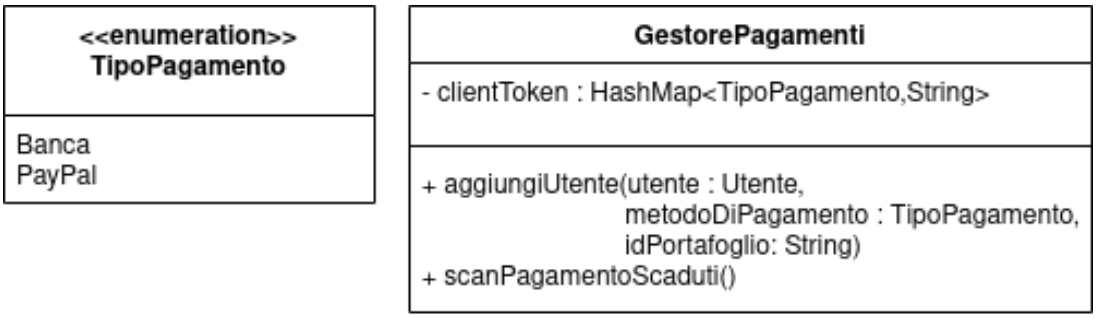
   La classe Utenti implementa un'interfaccia utile per la gestione degli utenti nel database. L'OCL specifica che, dopo la chiamata a `creaUtente`, la mail usata nella creazione deve essere nel database, indipendentemente dal fatto che l'utente sia stato inserito o che l'indirizzo mail fosse già presente.

3. **Brani**

   La classe Brani implementa un'interfaccia utile per la gestione dei singoli brani nel database. Oltre ad ottenere un brano dal suo ID, è possibile ottenere una lista di brani filtrando quelli nel database.

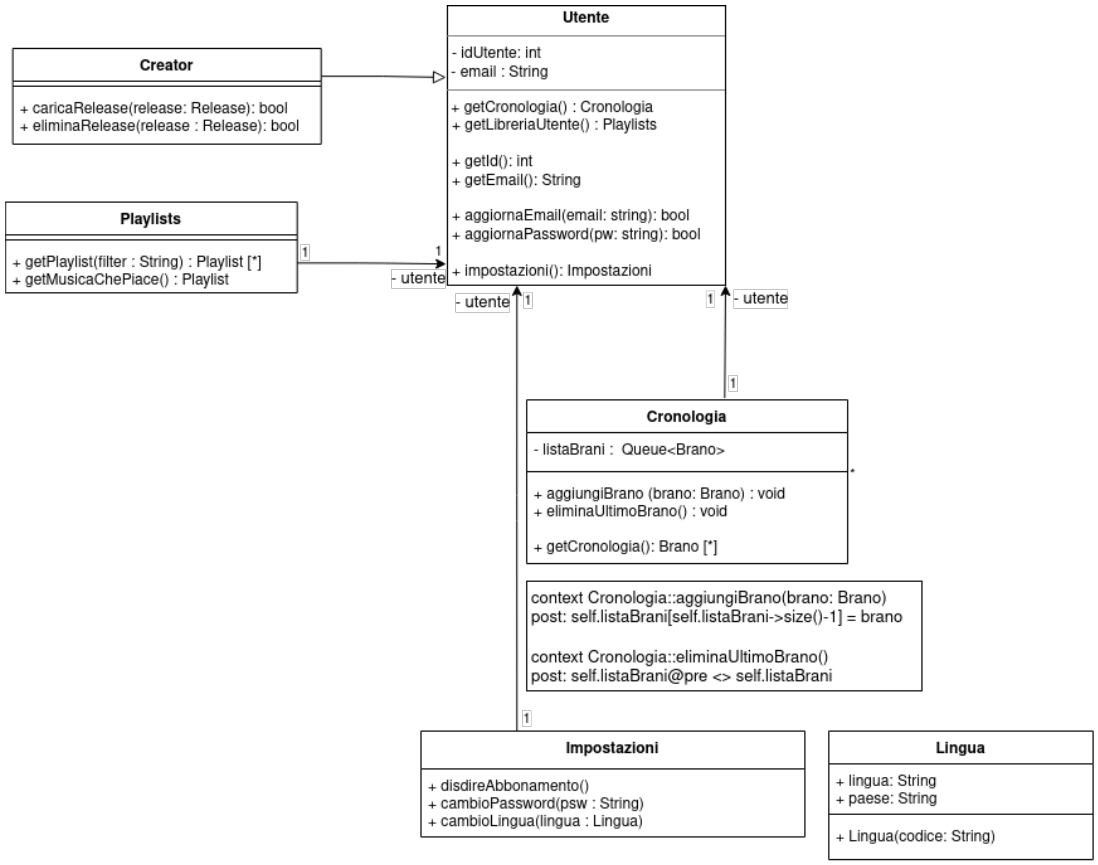
4. **Releases**

   La classe Releases implementa un'interfaccia utile per la gestione delle Releases (Albums, Singoli, EPs, etc...) nel database. Anche qui è possibile ottenere una Release attraverso il suo ID e filtrare tra tutte quelle disponibili.




5. **GestorePagamenti**

La classe è un singleton dedito all'aggiungere abbonamenti al database.
`aggiungiUtente` viene chiamato alla fine del flow di autorizzazione gestito dal metodo di pagamento stesso (es. Autenticazione e Autorizzazione attraverso l'endpoint OAuth di PayPal), dove `idPortafoglio` è un token dato dalla piattaforma per il pagamento utile per richiedere il pagamento ogni volta che l'abbonamento scade per il rinnovo automatico.
Una volta ogni giorno viene lanciato `scanPagamentoScaduti`, che scorre il database per provare a rinnovare gli abbonamenti scaduti.
`clientToken` è una HashMap che mappa le piattaforme ai presunti `clientId` richiesti per identificare l'applicazione, ad esempio in uno dei flow di autorizzazione OAuth.
`TipoPagamento` è l'enumerazione che elenca le piattaforme di pagamento supportate.




### 6. Utente e Creator

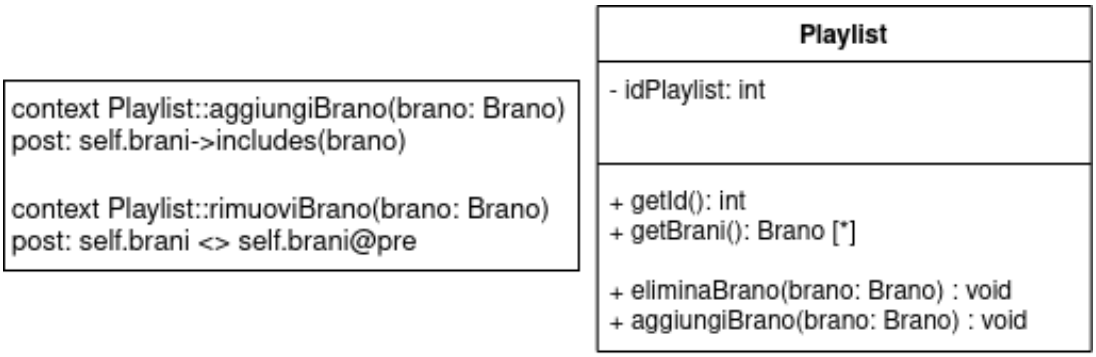
Le classi rappresentano gli oggetti nel database, le funzioni `aggiornaEmail` e `aggiornaPassword` modificano anche l'oggetto nel database, ma possono fallire (ad esempio se l'indirizzo mail non è unico o se la password non risponde alle policy).
Alcuni dei metodi ritornano altre classi utili per fare query al database unicamente riguardo un utente.
Il Creator, per l'applicazione, differisce solo nelle operazioni che può fare, dedite alla gestione delle sue Releases.

### 7. Playlists, Cronologia e Impostazioni

Queste classi possono essere create solo da oggetti Utente e sono utili per fare queries al database filtrando automaticamente per utente.
`Impostazioni` ha metodi per cambiare alcune preferenze dell'utente.
`Playlists` permette di ottenere le playlist create dall'utente e la sua playlist 'Musica Che Mi Piace'. `Playlists` fa le query in maniera 'pigra', ovvero le queries sono mandate solo quando necessario invece di domandare direttamente tutte le playlists al database.
`Cronologia` permette di ottenere la recente cronologia dell'utente. L'OCL specifica che, una volta chiamato `aggiungiBrano`, il brano passato come parametro deve risultare come l'ultimo aggiunto, e che `eliminaUltimoBrano` porti a una lista diversa.

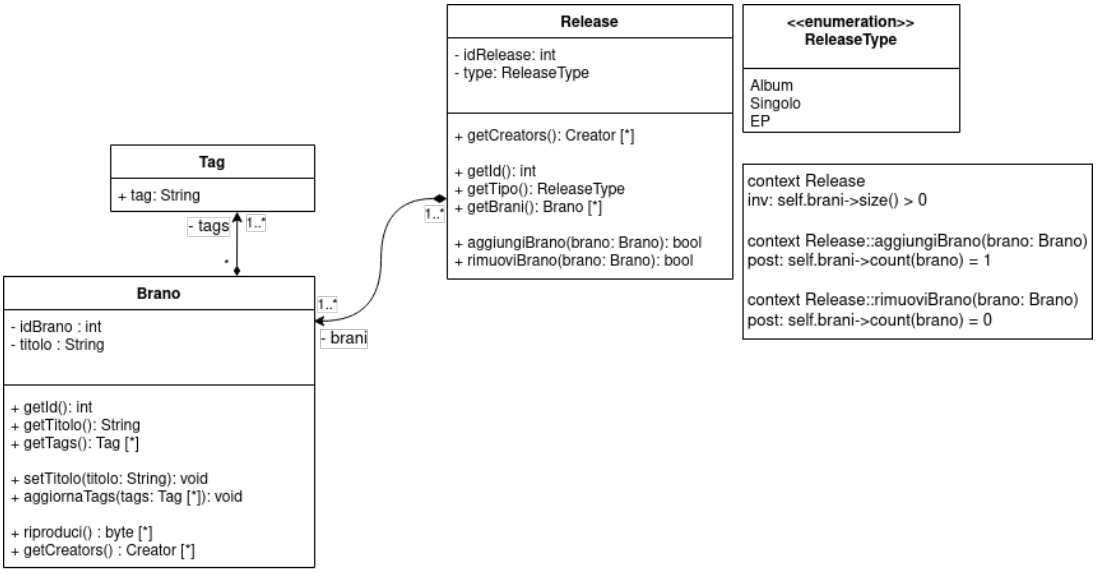
### 8. Lingua

`Lingua` è una classe utile per confermare che la stringa passata al costruttore sia valido ISO 639.




9. **Playlist**

`Playlist` è l'oggetto che rappresenta una playlist. L'attributo `brani` è dato dalla relazione con `Brani`.
L'OCL specifica che, una volta aggiunto un brano, questo dev'essere presente nella playlist, e che una volta rimosso un elemento, la lista dev'essere diversa.

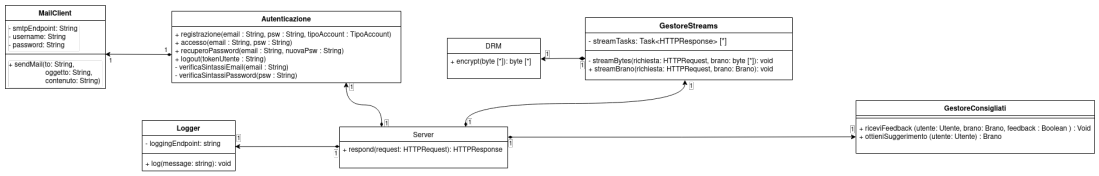



10. **Release**

Questo oggetto rappresenta una Release, l'OCL specifica che la lista di brani non dev'essere mai vuota e che ogni brano può apparire una sola volta.
`getCreators` permette di ottenere la lista di Creators proprietari della Release. `ReleaseType` è un'enumerazione che lista i tipi di Release.

11. **Brano**

`Brano` rappresenta un Brano nel database, `getTags` permette di ottenere tutti i suoi tags, `getCreators` permette di ottenere tutti i suoi Creators, e `riproduci` ritorna l'array di bytes che compone il file.




12. **Server**

    Questo oggetto si premura di rispondere alle richieste HTTP che arrivano al server, smistandole in base a directory richiesta e il loro contenuto.

13. **Logger**

    `Logger` manda i messaggi di log al server adibito a raccoglierli.

14. **GestoreConsigliati**

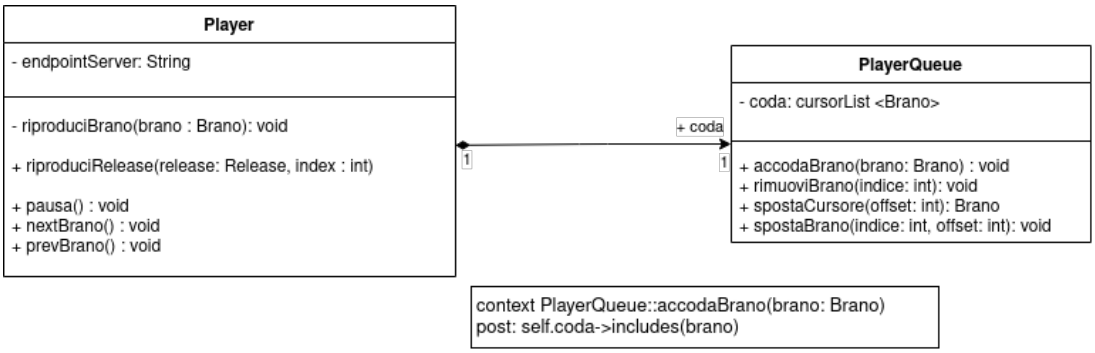
    Su richiesta del client GestoreConsigliati ritorna una canzone da consigliare all'utente e registra i feedback dati ai vari suggerimenti.

15. **GestoreStreams e DRM**

    Ogni volta che una canzone è richiesta, `GestoreStreams` lancia un Task per streammare asincronamente al client il file. Il file passa prima per il DRM in modo da venire cifrato in trasporto.

16. **Autenticazione e MailClient**

    Autenticazione si premura di gestire registrazione, login, logout e recupero password, e manda mails attraverso il MailClient in caso di registrazione o recupero.




17. **Player**

    Il Player è nel client e manda le richieste al server per ottenere informazioni e streammare audio.

18. **PlayerQueue**

    La Coda del client gli permette di tracciare le tracce richieste e di navigare tra quelle già riprodotte e da riprodurre

# 3 Diagramma delle classi con OCL