UNIVERSITÀ DI TRENTO

Dipartimento di Ingegneria
e Scienza dell'Informazione

Ingegneria del software

Documento di sviluppo
Versione: 0.01

Web Music Player

Gruppo T27

Anno accademico 2022/2023

Indice

1 Scopo del documento

Il presente documento riporta lo sviluppo di una parte del progetto Web Music Player. Dalla definizione delle API, alla loro implementazione, al deployment del progetto, questo documento descriverà le varie fasi che hanno portato alla creazione del software. Queste fasi sono:

- la specifica delle risorse
- l'implementazione e documentazione delle API
- il testing delle API
- l'implementazione del frontend
- il deployment dell'applicazione

Segue una breve descrizione dell'oggetto dello sviluppo.

Il sito web realizzato permette la **registrazione** e **accesso** alla piattaforma da parte degli utenti. Gli utenti standard possono **ricercare** canzoni tramite il loro titolo e aggiungere i risultati alla loro lista dei **preferiti**. Da qui le canzoni possono essere rimosse. Gli utenti creator possono inoltre **caricare** nuovi brani sulla piattaforma. Quando ricercano un brano, se questo risulta essere stato caricato da loro, possono decidere di **modificarlo** o di **eliminarlo** dalla piattaforma. Entrambe le tipologie di utenti possono far rimuovere il proprio account dalla piattaforma, cancellando tutti i dati a loro associati.

2 UserFlow

In questa sezione del documento si riportano gli "User Flows" per il ruolo dell'utente. L'immagine che segue descrive lo user flow relativo alle varie funzioni trattate in questa fase di sviluppo. Sarà anche presente una breve legenda che descrive i simboli utilizzati.

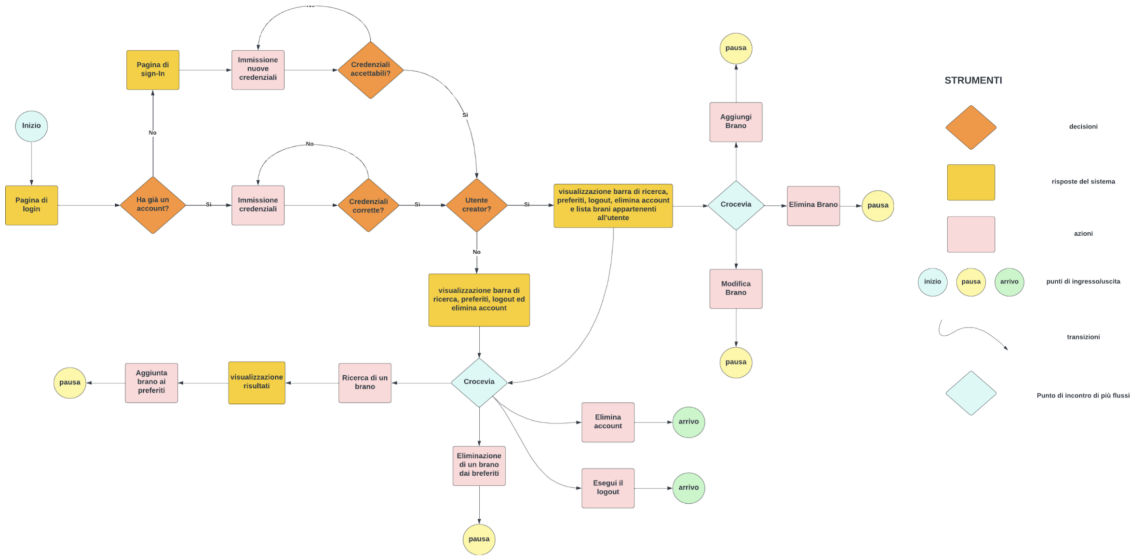

Figura 1: Struttura del backend del progetto

3 Implementazione e documentazione delle API

Questa sezione comprende lo sviluppo e la documentazione del backend del progetto, ovvero delle API che abbiamo descritto nelle sezioni precedenti del documento.

3.1 Struttura del backend

La struttura del backend è riportata nella figura 2. All'interno della cartella `/src` sono presenti i file che implementano le API e i modelli dei dati che faranno da ponte tra il backend e il database sul quale le informazioni verranno salvate. La cartella `/test` contiene i file per il testing; verrà approfondito nella prossima sezione.

Il file `index.ts` è il file principale del backend, in quanto si occupa di istanziare l'oggetto **app**, di effettuare la connessione al database e di avviare il server. Il file `app.ts` si consiste nell'applicazione stessa, alla quale verranno aggiunte le rotte per le API e per la documentazione. Il file `scripts.ts` contiene funzioni utilizzate in più punti del codice. I restanti file sono di configurazione.

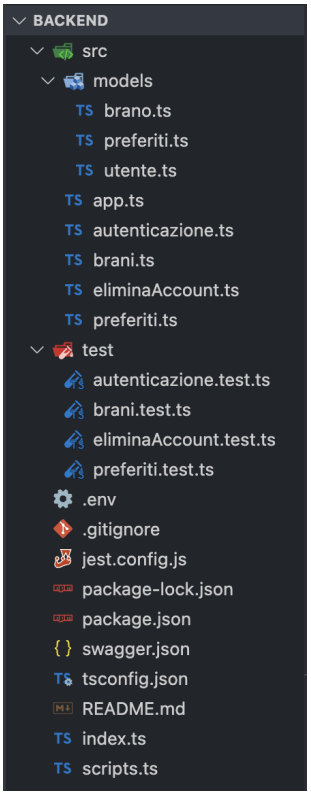

Figura 2: Struttura del backend del progetto

3.2 Dipendenze del progetto

Il progetto dipende da diverse librerie NodeJS per il suo funzionamento. Queste sono:

- `cors` per permettere ad un server esterno di accedere alle risorse presenti nel server di backend
- `dotenv` per le variabili d'ambiente
- `express` come framework per la creazione del server
- `jsonwebtoken` per la creazione e validazioen degli utenti tramite token
- `mongoose` come ponte tra il backend e il database MongoDB
- `multer` come middleware
- `swagger-ui-express` per la documentazione
- `typescript` come aiuto allo sviluppo

3.3 Modellazione dati nel database

Abbiamo utilizzato come base di dati MongoDB, un database non relazionale orientato ai documenti. Abbiamo creato tre *Collections*, gruppi di documenti di tipo diverso, una per tipo di dato che necessità di essere salvato. Questi sono il tipo di dato **Utente**, **Brano** e **Preferiti**.

Ciascuno corrisponde ad un file presente nella cartella `/src/models` del progetto, all'interno dei quali sono stati definiti i corrispettivi *Schema*.

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
Brano	18	1.97KB	113B	36KB	2	72KB	36KB
Preferiti	47	3.31KB	73B	36KB	1	36KB	36KB
Utente	8	913B	115B	36KB	1	36KB	36KB

Figura 3: Collezioni presenti nel database

```
1  _id: ObjectId('63ae416baa85ec273e32fcf9')          ObjectId
2  email: "silvanus.bordignon@studenti.unitn.it"      String
3  password: "PasswordValida&"                        String
4  tipoAccount: "standard"                            String
5  __v: 0                                             Int32
```

Figura 4: Modello tipo di dato Utente

```
1  _id: ObjectId('63af7d91f09d4c0b5980cd75')  ObjectId
2  nome: "Il tempo vola"                      String
3  artista: 63ae4cefc675dce04d60dfd0          ObjectId
4  durata: 120                                Int32
5  › tags: Array                              Array
6  __v: 0                                     Int32
```

Figura 5: Modello tipo di dato Brano

3.4 Specifica delle risorse

Questa sezione comprende due fasi distinte, entrambe antecedenti e necessarie allo sviluppo delle API.

Figura 6: Modello tipo di dato Preferiti

3.4.1 Estrazione delle risorse

In questo paragrafo vengono descritte le risorse estratte dal Class Diagram. Per ogni sottoparagrafo avrò una risorsa principale e tutte le API che questa risorsa può andare ad utilizzare.

«resource» Utente

La risorsa Utente ha come attributi:

- email
- password
- tipoAccount, che può essere Standard oppure Creator
- idUtente
- tokenUtente

«resource» Brano

Questa risorsa identifica un brano ed ha come attributi:

- idBrano
- nomeBrano
- nomeArtista
- durata
- tags

Le API contrassegnate da doppio asterisco (**) interagiscono sia con la risorsa **Utente** che con **Brano**.

API

«resource» signUp: Permette al nuovo utente di registrarsi. Viene svolta una POST e i parametri in ingresso sono email, password e tipoAccount. E' un API del backEnd.

«resource» accesso: Permette all'utente di accedere al servizio. Viene svolta una GET per il prelievo dell'utente dal database e i parametri in ingresso sono l'email e la password dell'utente. E' un API del BackEnd.

«resource» eliminaAccount: Permette ad un utente di eliminare il proprio account e di conseguenza tutti i suoi dati dal servizio. Viene svolta una DELETE e il parametro in ingresso è l'idUtente. E' un API del BackEnd.

«resource» ottieniPreferiti: Permette all'utente di visualizzare la playlist dei preferiti. Viene svolta una GET e il parametro in ingresso è l'idUtente. E' un API del FrontEnd.

«resource» modificaPreferiti: Permette all'utente di eliminare oppure aggiungere un brano dalla playlist dei preferiti. Viene svolta una PUT e i parametri in ingresso sono: idUtente, idBrano, azione (parametro di tipo String che identifica l'aggiunta oppure la rimozione). E' un API del BackEnd.

«resource» carica Brano**: Permette ad un utente di tipo Creator di caricare un brano. Viene svolta una POST e i parametri in ingresso sono: nomeBrano, idUtente, durata e tags (lista dei tag). E' un API del BackEnd.

«resource» modificaBrano**: Permette ad un utente di tipo Creator di modificare il nome oppure i tags di un suo brano. Viene svolta una PUT e i parametri in ingresso sono idBrano, nomeBrano, idUtente e tags. E' un API del BackEnd

«resource» eliminaBrano**: Permette ad un utente di tipo Creator di eliminare un suo brano. Viene svolta una DELETE e i parametri in ingresso sono idBrano e idUtente. E' un API del BackEnd.

«resource» ottieniBrano: Questa API permette di ottenere una risorsa di tipo Brano a partire dal suo id. Viene svolta una GET e il parametro in ingresso è l'idBrano. E' un API del BackEnd.

«resource» ricerca: Permette all'utente di cercare un brano all'interno del database. Viene svolta una GET e il parametro in ingresso è il nomeBrano. E' un API del FrontEnd.

3.4.2 Modello delle risorse

Seguono i modelli delle risorse, una descrizione più accurata di ciascuna API, che comprende URI e metodo HTTP, descrizione della richiesta e delle varie risposte che quell'endpoint può restituire.

Autenticazione

Registrazione

Questa API, all'indirizzo `/api/auth/registrazione`, ha un metodo `POST` e viene utilizzata per memorizzare un nuovo utente nel database.
La request body è formata dalla risorsa Utente che ha come attributi i campi *email*(String), *password*(String) e *tipoAccount*(tipoAccount).
Può avere 3 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 201 e *message* = "Created" allora l'operazione

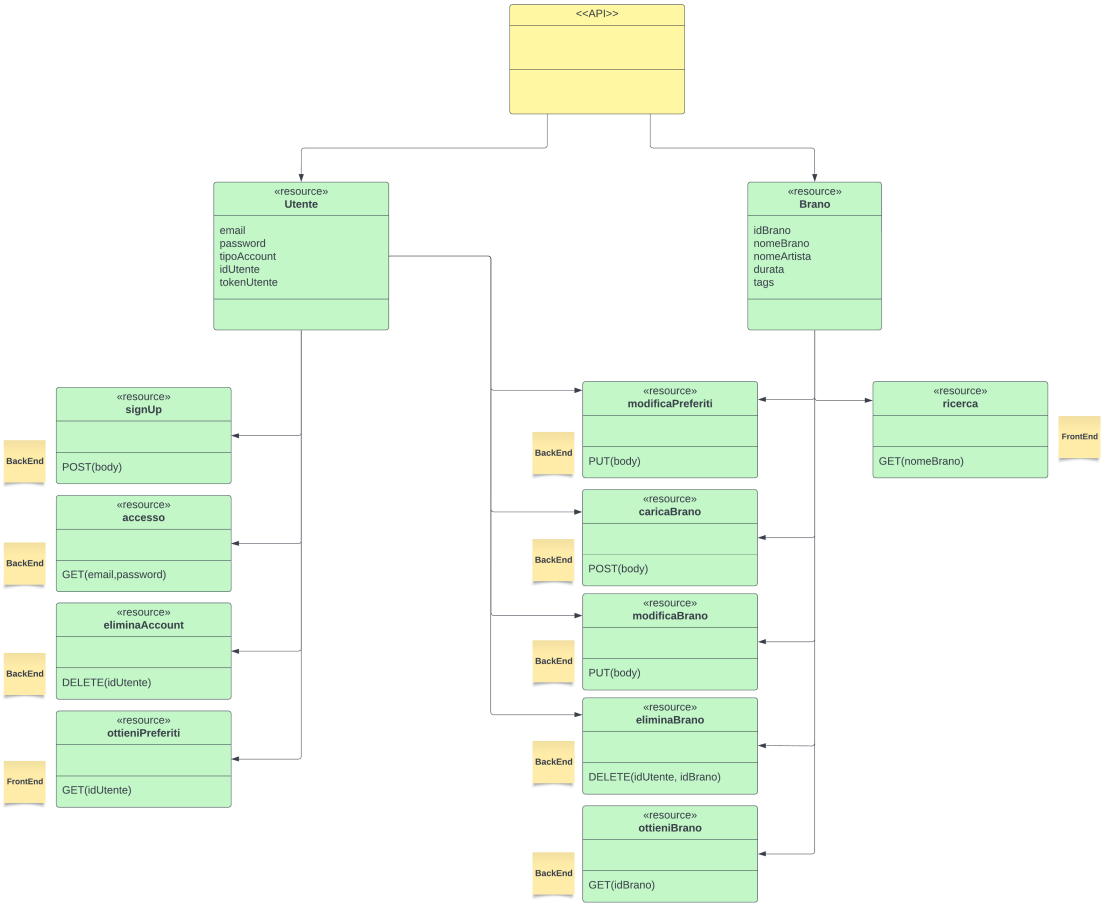

Figura 7: Diagramma di estrazione delle risorse

è riuscita e viene restituito l'*idUtente*(String), il *tokenUtente*(String), l'*email*(String) e il *tipoAccount*(String).
Nel caso in cui *code* = 409 e *message* = "Conflict" allora l'operazione non è riuscita in quanto l'email inserita è già presente nel database.
Se invece *code* = 400 e *message* = "Bad request" allora l'operazione non è riuscita in quanto l'email o la password inseriti non sono corretti o non conformi con le specifiche del sito.

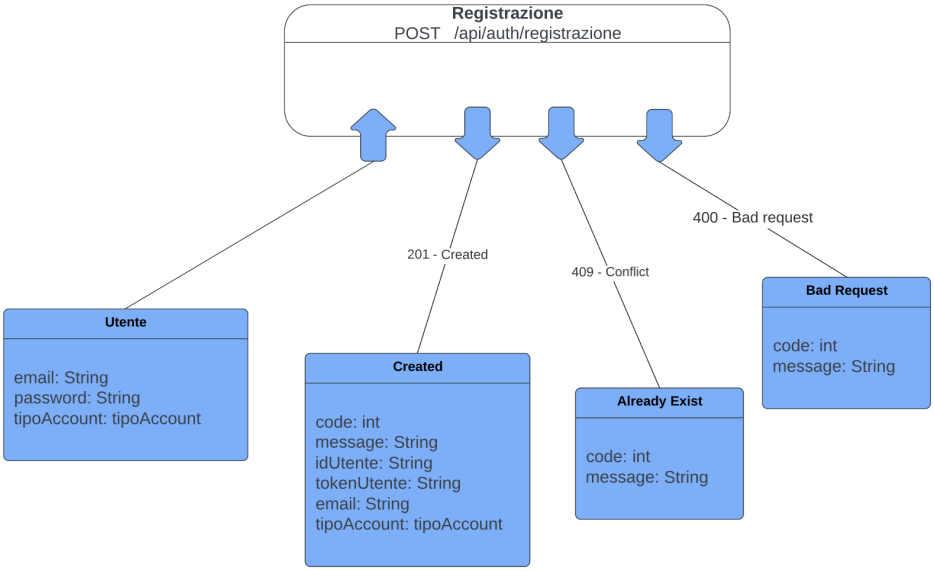

Figura 8: Modello della risorsa Registrazione

Accesso

Questa API, all'indirizzo `/api/auth/accesso`, ha un metodo `GET` e viene utilizzata per fare il login di un utente. La request body è la risorsa Utente, la quale ha come attributi il campo *email* (String) e *password* (String).
Può avere 3 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 200 e *message* = "OK" allora l'operazione è riuscita e nel corpo della risorsa viene restituito *l'idUtente*(String), il *tokenUtente*(String), *l'email*(String) e il *tipoAccount*(String).
Nel caso in cui *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto l'email inserita non è presente nel database.
Se invece *code* = 403 e *message* = "Forbidden" allora l'operazione non è riuscita in quanto la password inserita non è corretta.

Ricerca

Ricerca

Questa API, permette di ricercare all'interno del database un determinato brano e restituisce uno o più brani a seconda del testo digitato dall'utente.
All'indirizzo `/api/ricerca`, ha un metodo `GET` e prende in ingresso una stringa di testo

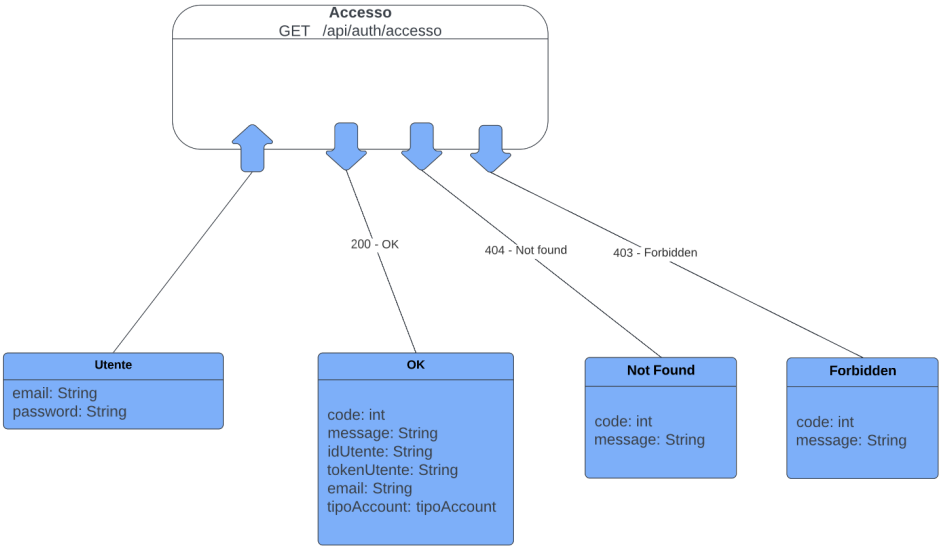

Figura 9: Modello della risorsa Accesso

nomeBrano. La response body è formata da *code*(int) e *message*(String) che saranno rispettivamente uguali a 200 e "OK" e dal campo *brani*(Brano[*]) formato da una lista di oggetti di tipo Brano.

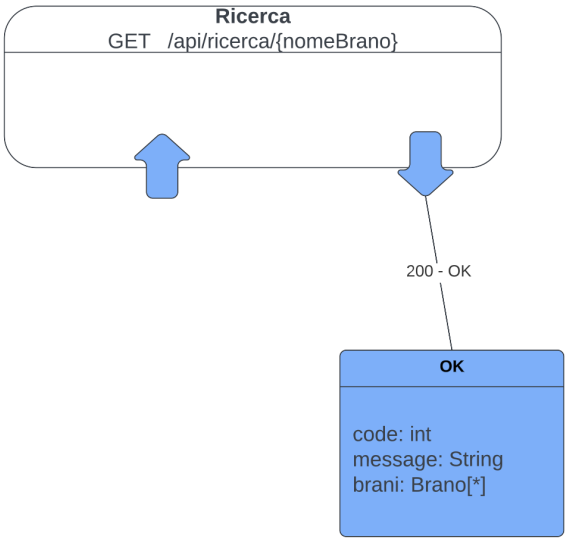

Figura 10: Modello della risorsa Ricerca

Eliminazione account

Elimina Account

Questa API all'uri `/api/eliminaAccount` permette all'utente la cancellazione del suo account e di conseguenza di tutti i suoi dati all'interno del sito web. Viene utilizzato il metodo `DELETE`.

La request body è formata da un solo campo: *idUtente* (String). Può avere 3 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 204 e *message* = "No content " allora l'azione è stata eseguita e non devono essere fornite ulteriori informazioni.
Nel caso in cui *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto l'*idUtente* inserito non è presente nel database.
Se invece *code* = 400 e *message* = "Bad Request" allora l'operazione non è riuscita in quanto l'*idUtente* passato non è valido.

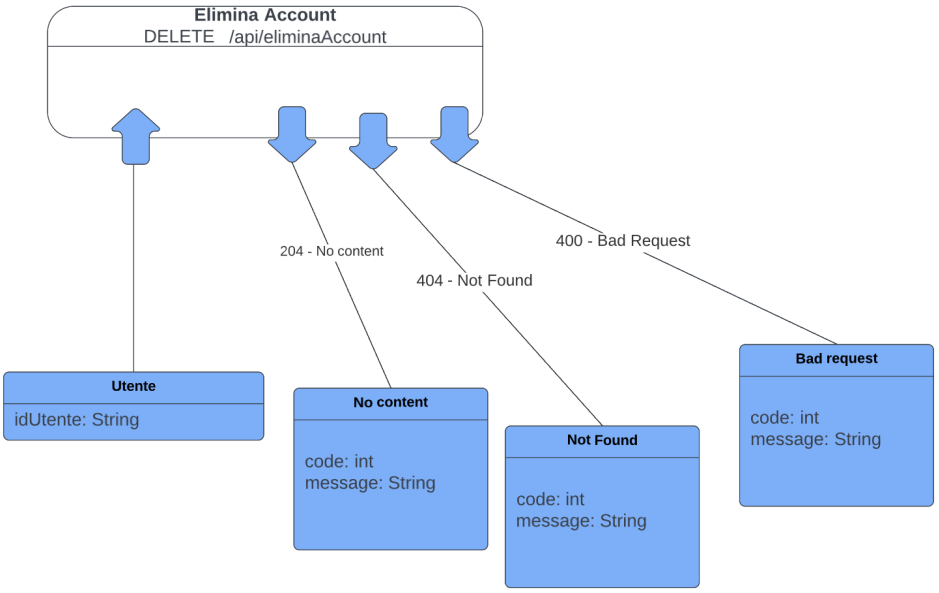

Figura 11: Modello della risorsa Eliminazione account

Operazioni creator

Carica Brano

Questa API consente ad un utente Creator di caricare un nuovo brano. Viene fatta una `POST` all'indirizzo: `/api/brano`.
La request body (Brano e Utente) è formata da *nomeBrano*(String), *idUtente*(String), *durata*(int) e *tags*(Tag[*]).
Può avere 4 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 201 e *message* = "Created " allora l'azione è stata eseguita e il brano è stato caricato correttamente.
Nel caso in *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto l'*idUtente* non corrisponde ad un utente registrato.
Nel caso in cui *code* = 409 e *message* = "Conflict" allora l'operazione non è riuscita in quanto l'utente Creator ha già caricato un brano con quel nome.
Se invece *code* = 400 e *message* = "Bad Request" allora l'operazione non è riuscita in quanto almeno uno dei parametri passati non è valido oppure *idUtente* non corrisponde ad un utente Creator.

Elimina Brano

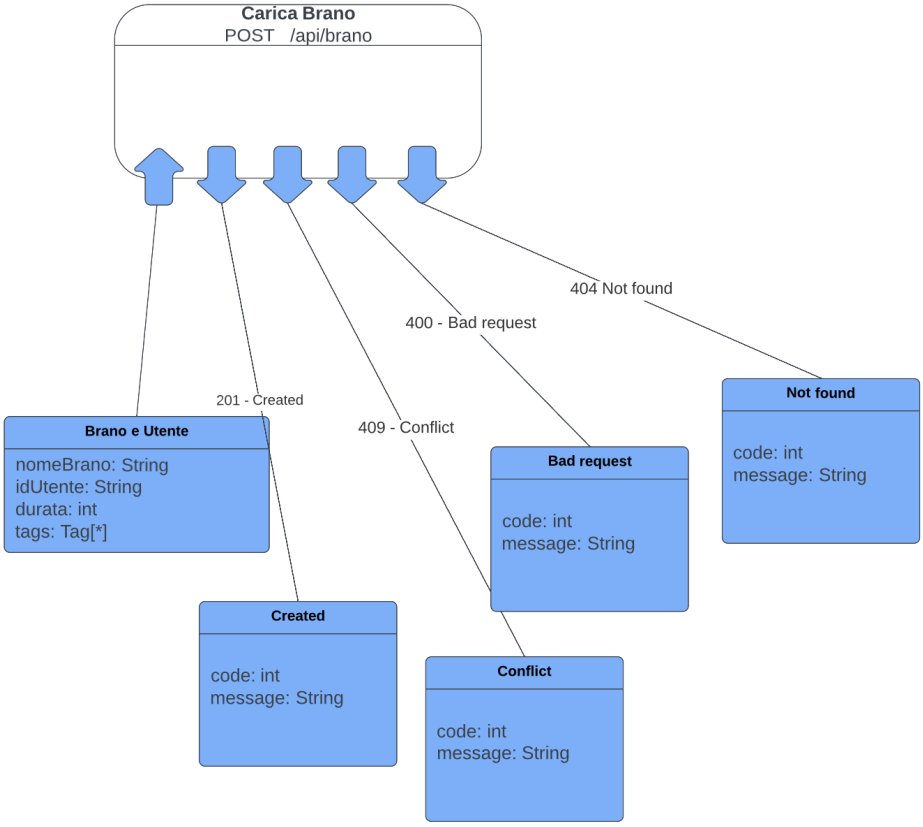

Figura 12: Modello della risorsa Carica brano

Questa API consente ad un utente Creator di eliminare un brano. Viene fatta una `DELETE` all'uri: `/api/brano`. La request body (Brano e Utente) è formata da *idBrano*(String) e *idUtente*(String).
Può avere 3 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 204 e *message* = "No Content " allora l'azione è stata eseguita e non devono essere fornite ulteriori informazioni.
Nel caso in cui *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto idBrano o idUtente passati non sono presenti nel database.
Se invece *code* = 400 e *message* = "Bad Request" allora l'operazione non è riuscita in quanto almeno uno dei parametri passati non è valido.

Modifica Brano

Questa API consente ad un utente Creator di modificare un suo brano andando a cambiare il nome del brano oppure i tags. Viene fatta una `PATCH` all'indirizzo `/api/brano`.
La request body (Brano e Utente) è formata da *idBrano*(String) *idUtente*(String), *nomeBrano*(String), e *tags*(Tag[*]).
Può avere 4 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 200 e *message* = "OK" allora l'azione è stata eseguita e il brano è stato modificato correttamente.
Nel caso in cui *code* = 409 e *message* = "Conflict" allora l'operazione non è riuscita in quanto è possibile che il nuovo nome dato al brano sia già il nome di un altro brano del Creator.

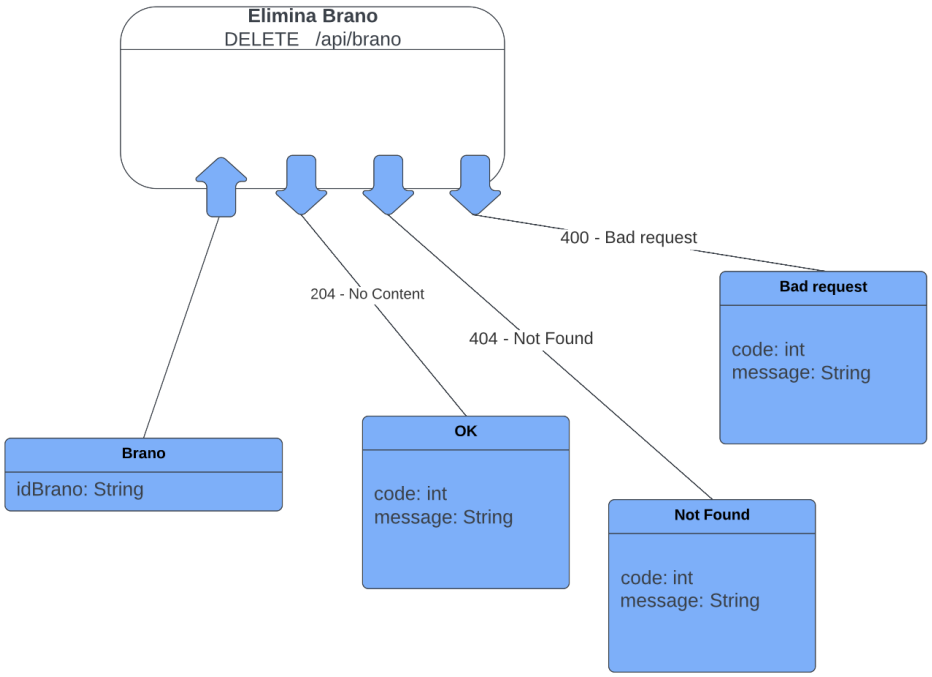

Figura 13: Modello della risorsa Elimina Brano

Se *code* = 400 e *message* = "Bad Request" allora l'operazione non è riuscita in quanto almeno uno dei parametri passati non è valido.
Nel caso in cui *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto idBrano o idUtente (oppure entrambi) non sono presenti nel database.

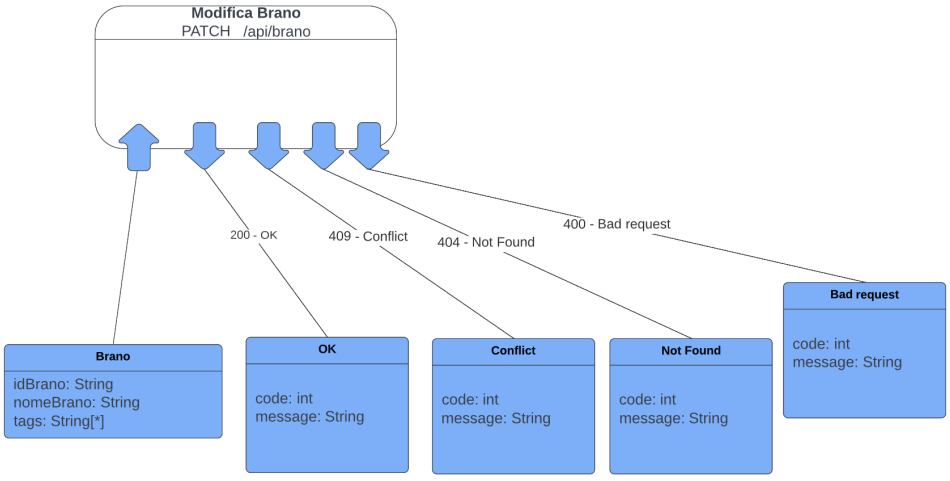

Figura 14: Modello della risorsa Modifica brano

Preferiti e ottenimento brano

Ottieni Brano

Questa API consente di ottenere un oggetto di tipo Brano a partire da un idBrano. Viene utilizzato il metodo `GET` all'indirizzo `/api/brano` e la request body è formata dall'*idBrano*(String).
Può avere 3 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 200 e *message* = "OK" allora l'azione è stata eseguita correttamente e nel corpo della response troviamo anche l'attributo *brano*(Brano).
Nel caso in cui *code* = 400 e *message* = "Bad Request" allora l'operazione non è riuscita in quanto l'idBrano passato alla funzione non è valido.
Nel caso in cui *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto l'idBrano passato all'API non è presente nel database.

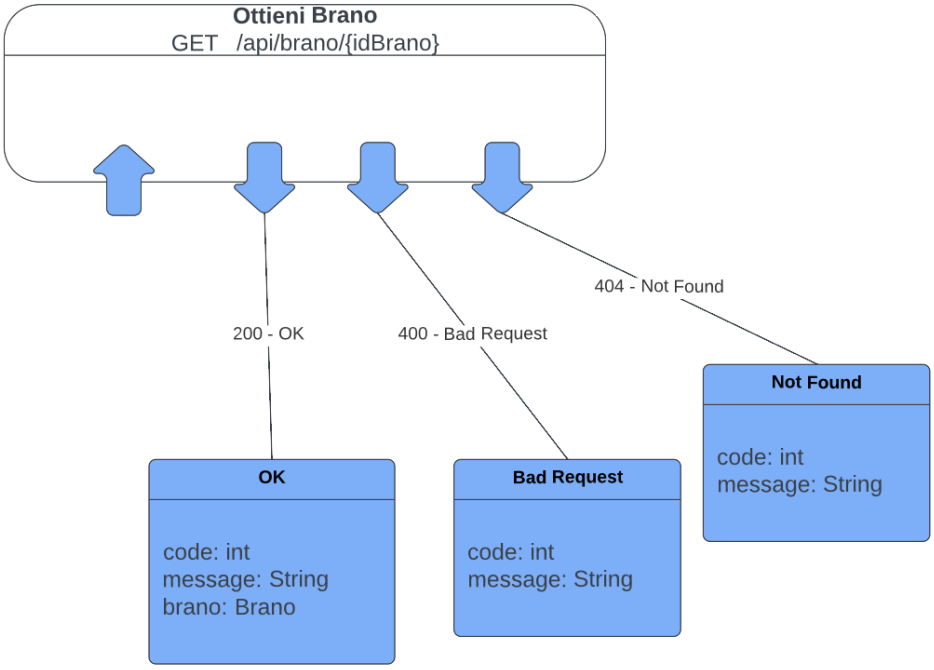

Figura 15: Modello della risorsa Ottieni Brano

Ottieni Preferiti

Questa API permette di ottenere e visualizzare la lista dei preferiti. Viene utilizzato il metodo `GET` all'indirizzo `/api/preferiti`.
La request body è formata da un solo parametro: *idUtente*.
Può avere 3 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 200 e *message* = "OK" allora l'azione è stata eseguita correttamente e nel corpo della response troviamo anche l'oggetto *idBrani* (String[*]), ovvero una lista degli id dei brani appartenenti alla playlist Preferiti di quel particolare utente.
Nel caso in cui *code* = 400 e *message* = "Bad Request" allora l'operazione non è riuscita in quanto l'idUtente passato alla funzione non è valido.
Nel caso in cui *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto l'idUtente passato all'API non è presente nel database.

Modifica Preferiti

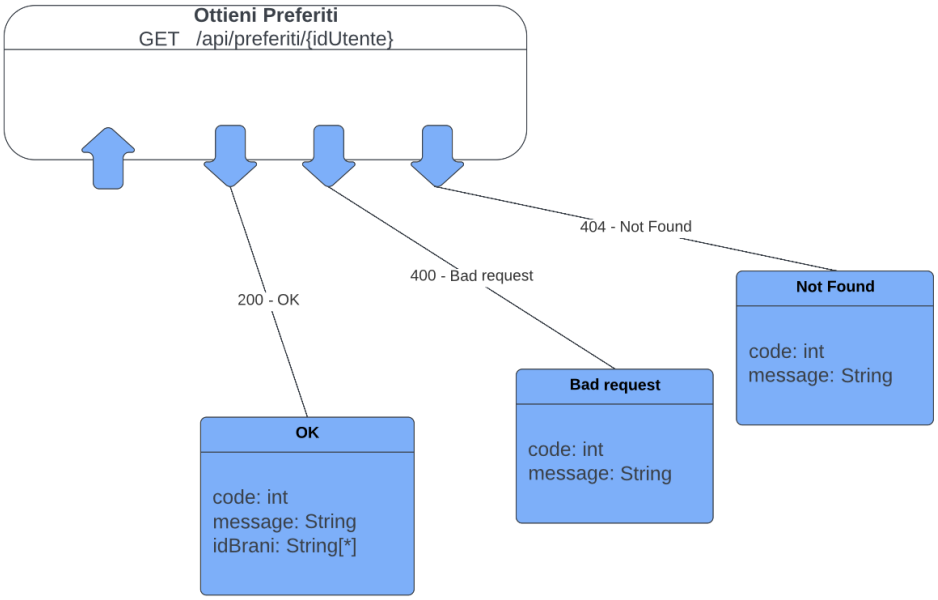

Figura 16: Modello della risorsa Ottieni preferiti

Questa API permette all'utente di modificare la propria playlist di preferiti. Grazie ad essa l'utente può aggiungere o eliminare un brano dai preferiti. Viene utilizzato il metodo `PATCH` all'indirizzo `/api/preferiti/modifica`.
La request body è formata da 3 parametri: *idUtente*(String), *idBrano*(String), e *azione*(String); quest'ultimo attributo può assumere solo i valori "aggiungi" ed "elimina" in relazione ad un determinato brano.
Può avere 4 tipologie di response body a seconda della riuscita dell'operazione.
Se la response body è formata da *code* = 200 e *message* = "OK" allora l'azione è stata eseguita correttamente e nel response body ottengo l'oggetto idBrani(String[*]).
Nel caso in cui *code* = 409 e *message* = "Conflict" allora l'operazione non è riuscita.
Se *code* = 400 e *message* = "Bad Request" allora l'operazione non è riuscita in quanto almeno uno dei parametri passati non è valido.
Nel caso in cui *code* = 404 e *message* = "Not Found" allora l'operazione non è riuscita in quanto idBrano o idUtente (oppure entrambi) non sono presenti nel database.

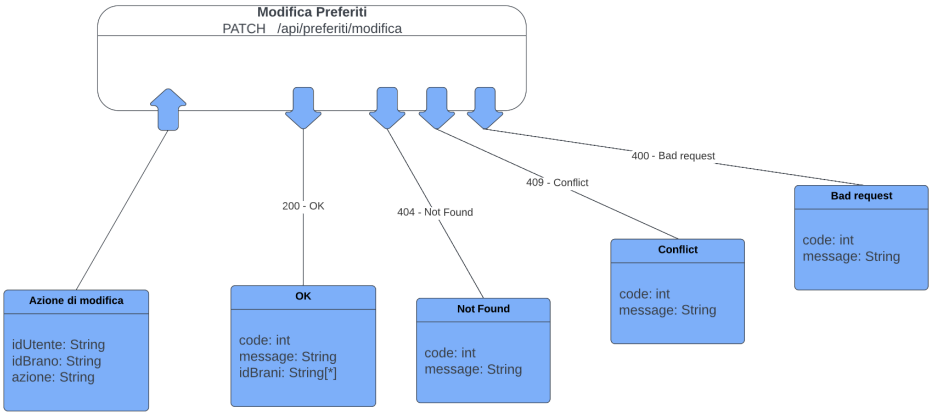

Figura 17: Modello della risorsa Modifica preferiti

4 Sviluppo delle API

5 Documentazione delle API

6 Implementazione del frontend

7 GitHub repository e deployment

8 Testing delle API