

INGEGNERIA DEL SOFTWARE

SOUNDMATCH

SVILUPPO APPLICAZIONE

Alessandro Fontana, Davide Pedrotti, Nicolas Torriglia

Università di Trento

Indice

Scopo del documento	4
1 User Flows	4
2 Application Implementation and Documentation	5
2.1 Project Structure	5
2.2 Project Dependencies	6
2.3 Project Data or DB	6
2.4 Project APIs	9
2.4.1 Resources Extraction from the Class Diagram	9
2.4.2 Resources Models	11
2.5 Sviluppo API	15
2.5.1 Registrazione	15
2.5.2 Login	16
2.5.3 Invia un nuovo messaggio in una Community	17
2.5.4 Visualizza i messaggi di una Community	17
2.5.5 Visualizza i messaggi che un utente ha inviato in una Community	18
2.5.6 Elimina un messaggio specifico	18
2.5.7 Elimina tutti i miei messaggi inviati in una Community	19
2.5.8 Invia un nuovo messaggio ad un Artista	20
2.5.9 Visualizza messaggi di una Collaborazione	21
2.5.10 Segna messaggi come già letti	21
2.5.11 Visualizza messaggi non letti	22
2.5.12 Elimina messaggio	22
2.5.13 Invia richiesta di Collaborazione	22
2.5.14 Accetta richiesta di Collaborazione	24

2.5.15 Rifiuta richiesta di Collaborazione	24
2.5.16 Visualizza Collaborazioni	25
2.5.17 Pubblica Brano	25
2.5.18 Visualizza Brani	26
2.5.19 Cerca Brano	26
2.5.20 Modifica Brano	27
2.5.21 Elimina Brano	28
2.5.22 Elimina i miei Brani	28
3 API documentation	29
3.1 Gestione degli errori	30
3.2 Models	31
4 FrontEnd Implementation	31
5 GitHub Repository and Deployment Info	36
6 Testing	36

Scopo del documento

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione SoundMatch. In particolare, presenta tutti gli artefatti necessari per realizzare alcuni servizi disponibili all'utente, come la registrazione, il login e l'utilizzo sia della chat tra artisti che di quella globale. Partendo dalla descrizione degli user flow dell'applicazione, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per il funzionamento dell'applicazione. Per ogni API realizzata, oltre ad una descrizione delle funzionalità fornite, il documento presenta la sua documentazione e i test effettuati. Infine una sezione e' dedicata alle informazioni del Git Repository e il deployment dell'applicazione stessa.

1 User Flows

In questa sezione di documento si riportano gli "User Flows" per il ruolo sia di utente autenticato che di artista. Nella features accessibili solo dall'utente artista, lo schema dello user flow presenta dei controlli, in modo che esso sia corretto analizzando entrambi i ruoli disponibili del progetto SoundMatch. L'immagine che segue descrive le API trattate in questa fase di sviluppo. È presente inoltre una legenda che descrive i simboli utilizzati.

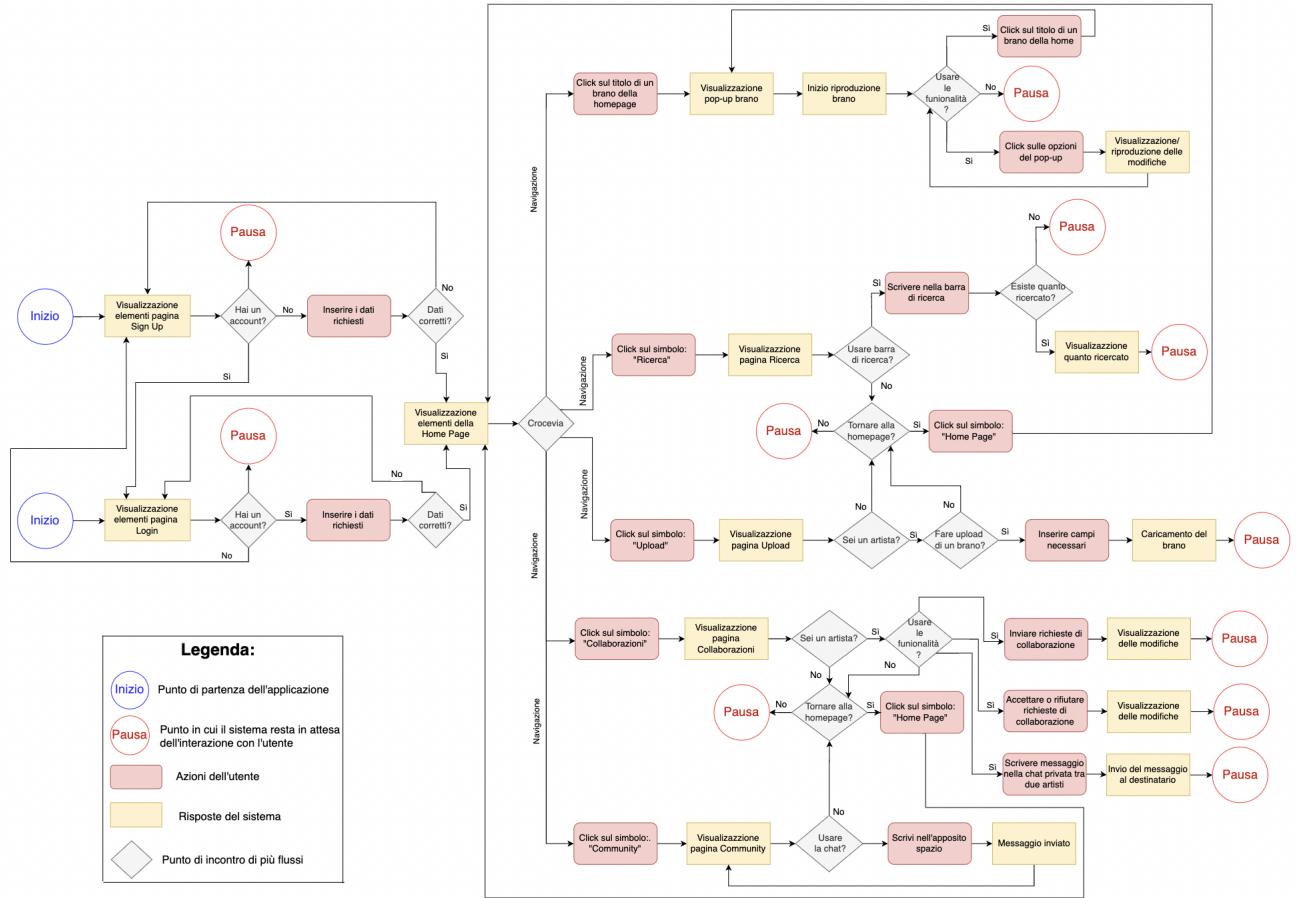


Figura 1: User Flows del progetto SoundMatch

2 Application Implementation and Documentation

2.1 Project Structure

La struttura del progetto, come da figura, è composta di una cartella "models", contenente i modelli utilizzati per il database (MongoDB), di una cartella "controllers", contenente tutti i metodi utilizzati dalle api e di una cartella "routes" per specificare le routes dei metodi usati dalle api. Infine, per il frontend, abbiamo creato una cartella "public" la quale contiene tutte le pagine html.

2.2 Project Dependencies

I moduli Node che abbiamo utilizzato e che sono presenti nel file Package.json sono i seguenti:

- Dotenv
- Express
- Jsonwebtoken
- Mongoose
- Http
- Socket.io
- Body-parser

2.3 Project Data or DB

Per poter gestire tutti i dati abbiamo definito 5 strutture dati come illustrato in figura 2.

1. La collezione "users" contiene tutte le informazioni riguardanti gli utenti che hanno creato un account SoundMatch
2. La collezione "songs" contiene i dati riguardanti le canzoni caricate
3. La collezione "collabs" gestisce le richieste di collaborazione tra artisti
4. La collezione "collab_chats" contiene informazioni circa i messaggi inviati tra artisti
5. "chats" è una collezione che viene utilizzata per gestire le chat globali

SoundMatch								CREATE COLLECTION
LOGICAL DATA SIZE: 2.07KB	STORAGE SIZE: 180KB	INDEX SIZE: 252KB	TOTAL COLLECTIONS: 5					
Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size	
chats	1	136B	136B	36KB	1	36KB	36KB	
collab_chats	4	460B	115B	36KB	1	36KB	36KB	
collabs	2	168B	84B	36KB	1	36KB	36KB	
songs	5	525B	105B	36KB	1	36KB	36KB	
users	6	833B	139B	36KB	3	108KB	36KB	

Figura 2: Collezioni MongoDB

Per rappresentare gli utenti abbiamo definito la seguente struttura:

```
_id: ObjectId('638db3ddc985e85dbafc7ccb')
nome_utente: "nTorre02"
email: "nicolas.torriglia02@gmail.com"
password_hash: "test"
is_artista: true
__v: 0
```

Figura 3: User Schema

Per rappresentare le canzoni abbiamo definito la seguente struttura:

```
_id: ObjectId('639837de669b2b44a8b4c652')
title: "titolo 2"
artist: ObjectId('638db3ddc985e85dbafc7ccb')
> collaborations: Array
> genres: Array
__v: 0
```

Figura 4: Songs Schema

Per rappresentare collaborazioni abbiamo definito la seguente struttura:

```
_id: ObjectId('63a1939b32148e71bcd4c42')
from: ObjectId('638db3ddc985e85dbafc7ccb')
to: ObjectId('638db62a75e2e0464268b306')
status: "SENT"
__v: 0
```

Figura 5: Collabs Schema

Per rappresentare i messaggi inviati tra artisti abbiamo definito la seguente struttura:

```
_id: ObjectId('639d6ala3afb5bc4a9da6e03')
from: ObjectId('638db3ddc985e85dbafc7ccb')
collab: ObjectId('63998c51f1b186a7a8b0c1ef')
text: "Ciao"
datetime: 1671260698330
status: "READ"
__v: 0
```

Figura 6: Collab_chats Schema

Per rappresentare la chat globale abbiamo definito la seguente struttura:

```
_id: ObjectId('63ala54f19ff38bcad22b790')
nome_utente: "nTorre022"
id_utente: "638db5935b7f7e86e6b3f19c"
message: "test"
community_id: "2"
__v: 0
```

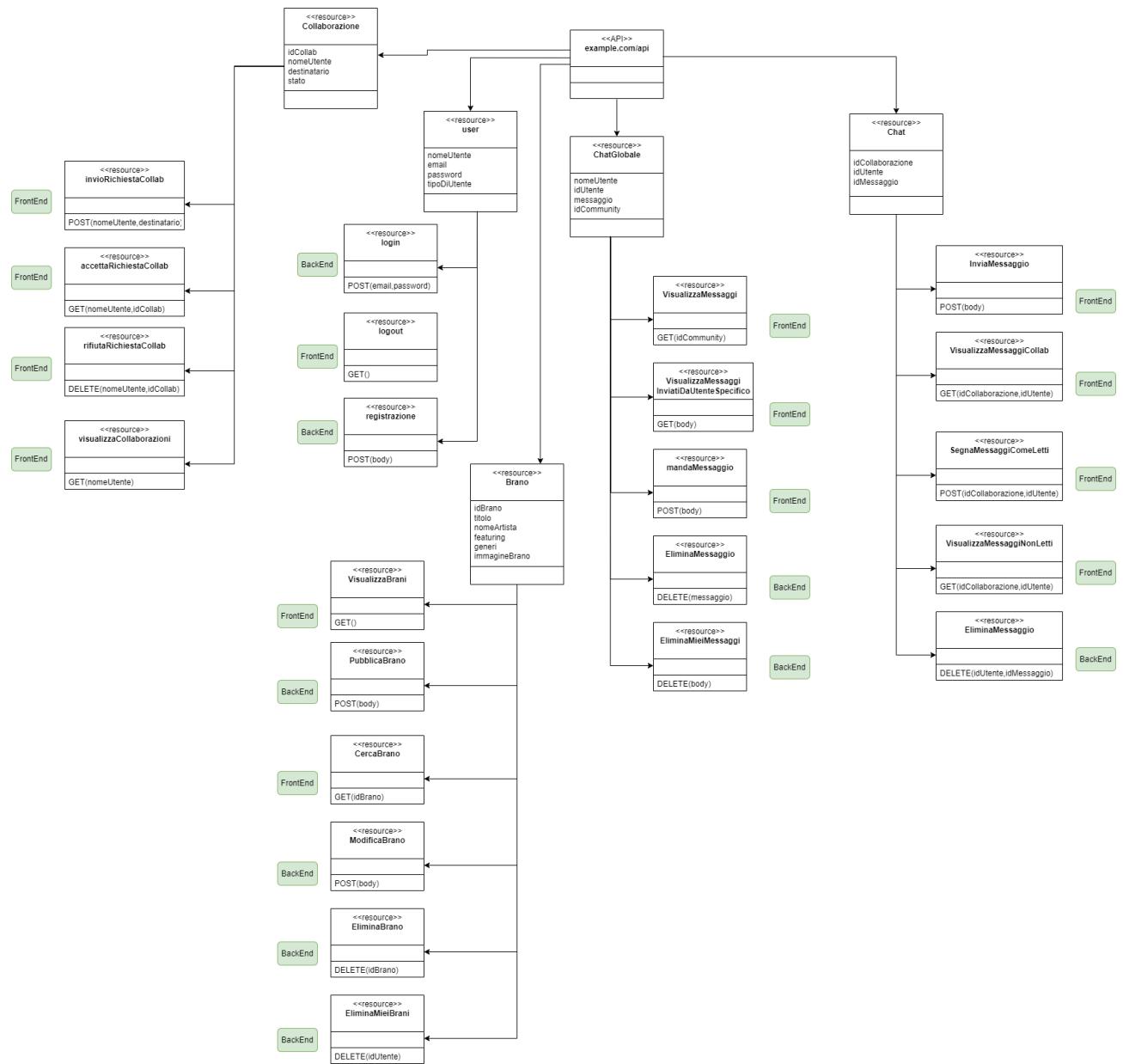
Figura 7: Chats Schema

2.4 Project APIs

2.4.1 Resources Extraction from the Class Diagram

Di seguito mostriamo il diagramma delle risorse estratte dal Class Diagram. Come da figura possiamo notare la presenza di 6 differenti API riguardanti:

- La gestione delle collaborazioni tra artisti
- Le operazioni di registrazione, login e logout
- La possibilità di caricare, modificare e visualizzare brani
- Una chat globale attraverso la quale gli utenti possono interagire tra di loro
- La presenza di una chat tra artisti che hanno avviato una collaborazione



2.4.2 Resources Models

Nei modelli delle risorse riguardanti le API implementate mostriamo le informazioni contenute nel body della richiesta e ciò che viene ritornato dalle API. Di seguito mostriamo i Resources Models.

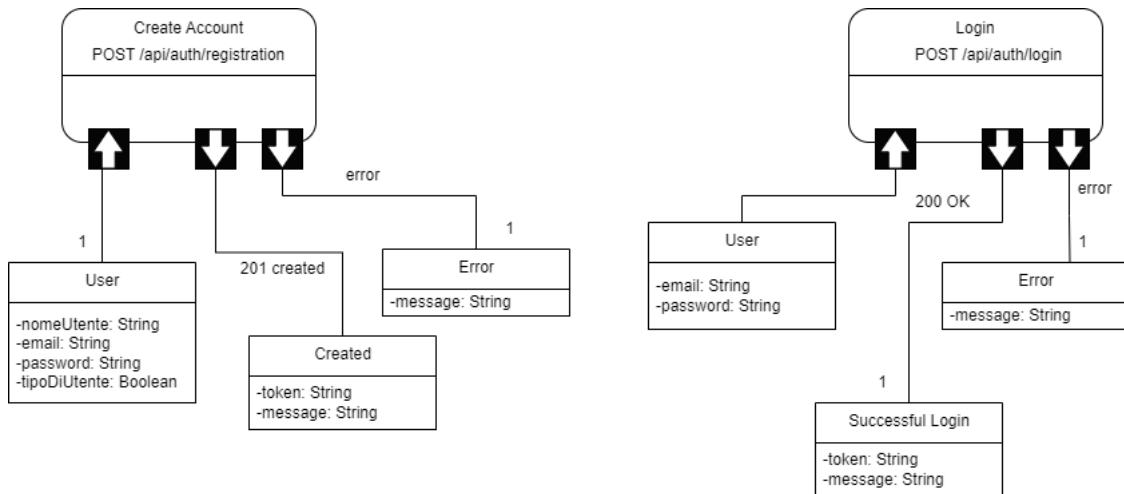
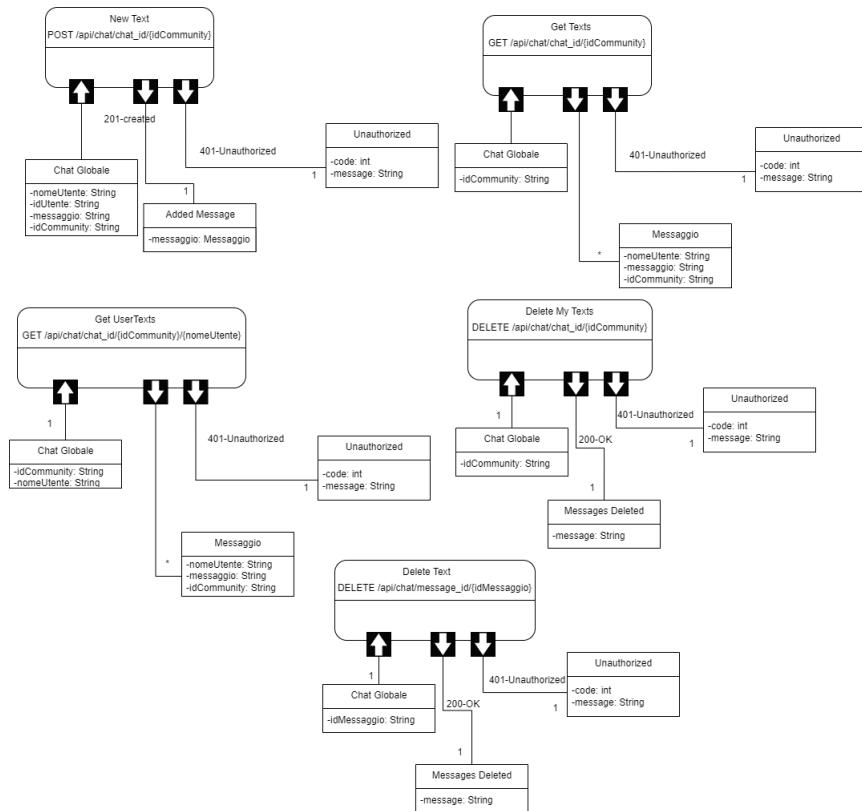


Figura 8: Autenticazione

**Figura 9: Chat Globale**

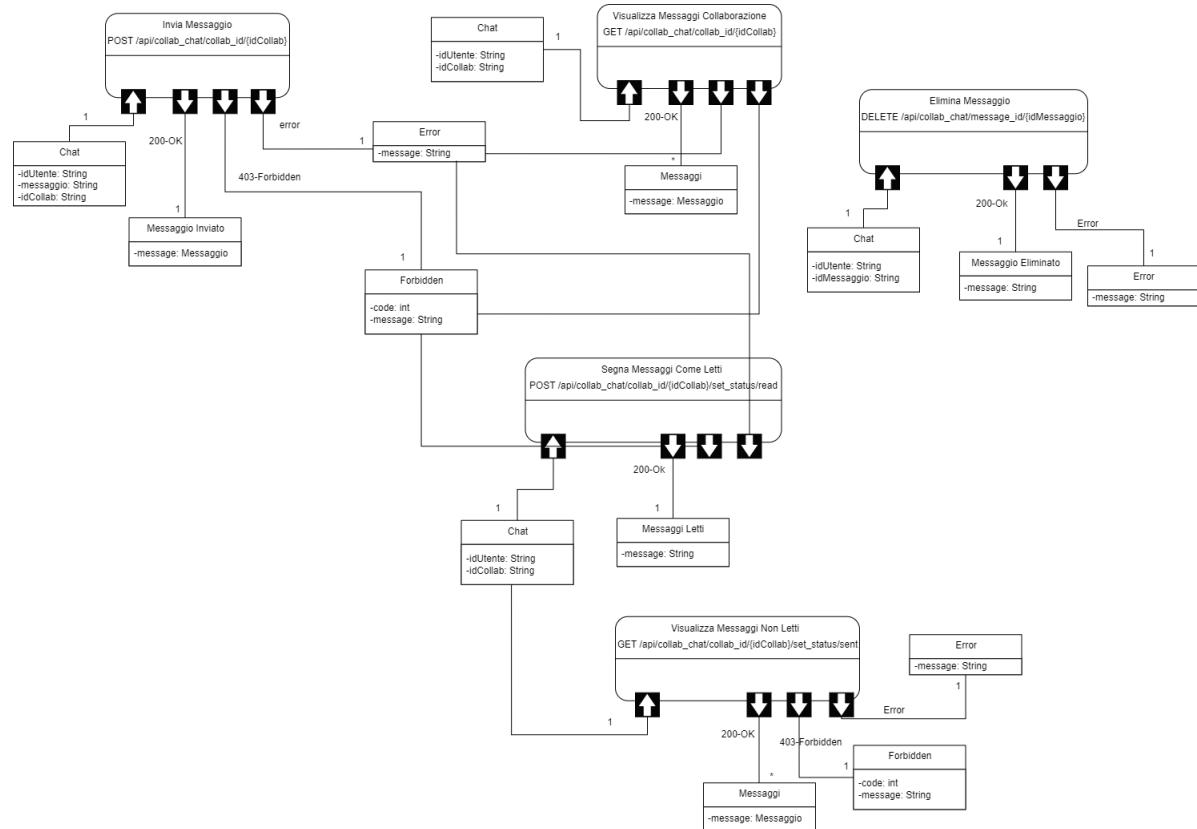


Figura 10: Chat tra due Artisti

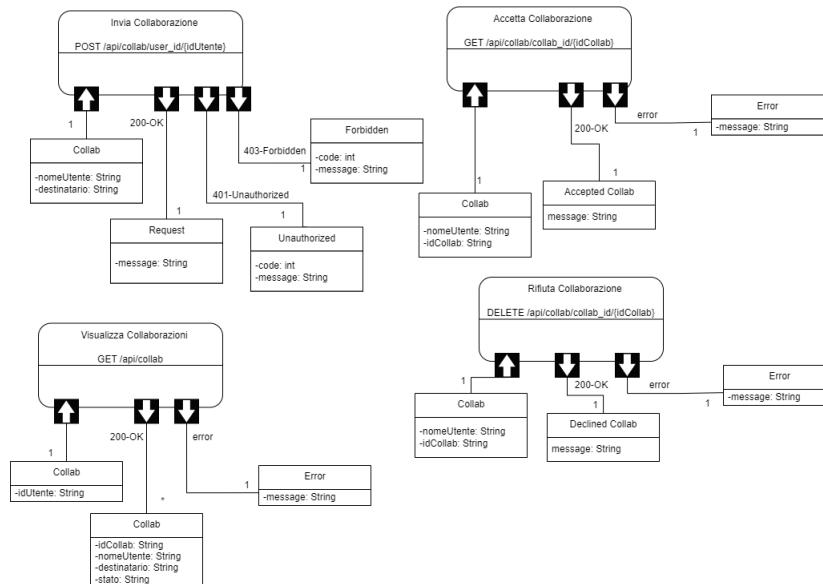
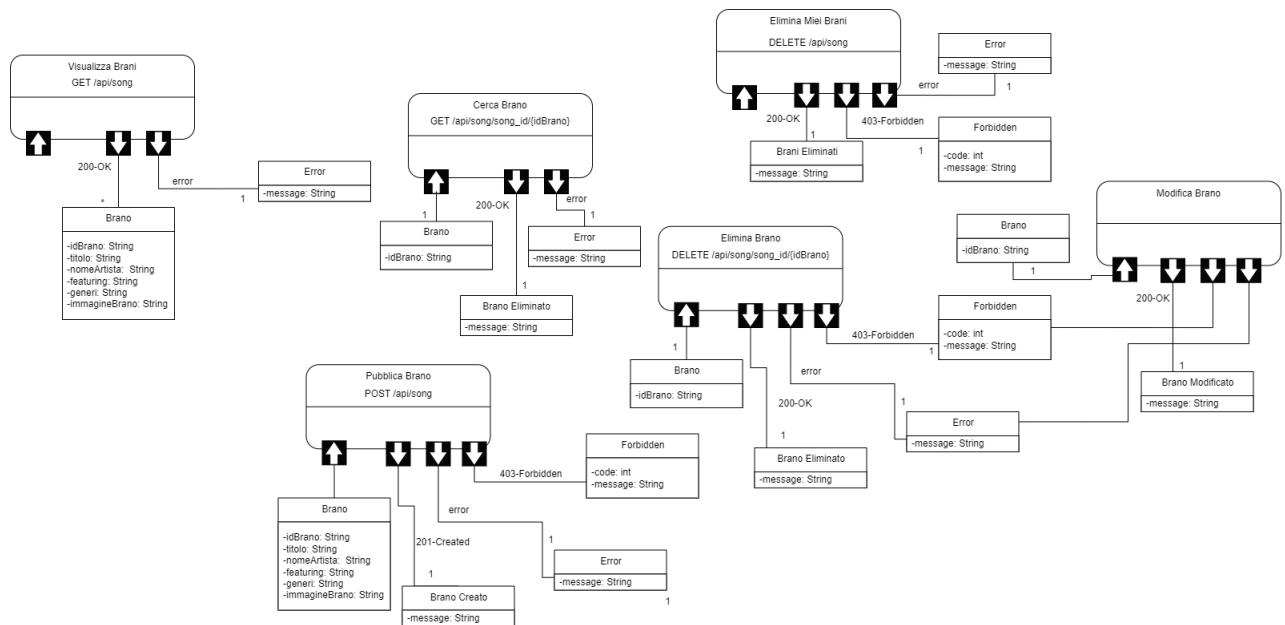


Figura 11: Collaborazione tra due Artisti

**Figura 12:** Gestione dei Brani

2.5 Sviluppo API

2.5.1 Registrazione

Questa API viene utilizzata ogni volta che un utente si vuole registrare al sito. Viene creato un nuovo oggetto di tipo "user" con un ID autogenerato, un nome utente, un'indirizzo email, l'hash della password e la tipologia di utente (artista o ascoltatore). Tutti questi parametri vengono passati tramite il body della richiesta. Ritorna un messaggio di errore stampato su console se ci sono stati problemi durante la creazione dell'account, altrimenti genera un token di sessione e ritorna un messaggio di avvenuto inserimento.

```
// create an account
const createAccount = (req, res, next) => {

  let email = req.body.email;
  let password = req.body.password;
  let nome_utente = req.body.nome_utente;
  let is_artista = req.body.is_artista;

  // controllo che non ci sia già questo nome utente in DB, altrimenti do errore

  // salvo utente
  const user = new User({ nome_utente: nome_utente,
    email: email,
    password_hash: password,
    is_artista: is_artista});

  user.save(function (err, item) {
    if (err){
      console.log(err)
      return res.json({success: false, message: "Error, user not saved"});
    }
    console.log(item.id)
    let token = token.createToken(item.email, item.id, item.is_artista);
    console.log(token)
    return res.json({success: true,
      token: token,
      message: "Success, user saved"});
  });
};
```

Figura 13: Registrazione

2.5.2 Login

Questa API viene utilizzata per eseguire il login. L'utente deve inserire email e password, queste due informazioni verranno utilizzate per verificare l'esistenza di un account con tale email e password. Se l'email non è collegata ad un account presente nel Database o se la password non corrisponde verrà ritornato un messaggio di errore. Se invece le informazioni inserite sono corrette verrà generato, e successivamente ritornato, un token di sessione assieme ad un messaggio di generazione di tale token.

```
// login, verifico credenziali e restituisco token
const login = function (req, res, next) {
  let email = req.body.email
  let password = req.body.password

  // verifico che ci siano email e password nella richiesta
  if(!email || !password)
    return res.json({success: false, message: "Error, give all the information"})

  User.findOne({ email: email }, function(err, user){
    if(err) return res.json({success: false, message: "Error"});
    // TODO: password hash
    if (user.password_hash != password){
      // password non corretta
      return res.json({success: false, message: "Wrong password"})
    }
    console.log(user);
    let stringtoken = token.createToken(user.email, user.id, user.is_artista);
    return res.json({success: true, message: "Token generated", token: stringtoken})
  });
}
```

2.5.3 Invia un nuovo messaggio in una Community

Tramite questa api gli utenti possono inviare messaggi nella chat di una Community. Tramite il token di sessione e il body della richiesta (contenente il messaggio) viene creato un nuovo oggetto contenente il messaggio, l'id della Community, il nome utente e l'id dell'utente.

```
// creating new message
const newText = async function (req,res,next) {
  try{
    // getting all the variables from token, body and parameters
    var utente = await User.findById(req.loggedUser.id).exec();
    var nome_utente = utente.nome_utente
    var message = req.body.message;
    var community_id = req.params.chat_id;
    /* var savedMessage = await message.save() */
    if(utente.nome_utente != null) {
      Chat.insertMany([
        {nome_utente: utente.nome_utente, id_utente: req.loggedUser.id, message: message, community_id: community_id.toString()}
      ])
      console.log('saved');
      return res.send(nome_utente);
    }
    else {
      console.log("error nome_utente null")
    }
  }
  catch (error){
    console.log('error',error);
    return res.sendStatus(500);
  }
  finally{}
}
```

2.5.4 Visualizza i messaggi di una Community

Grazie a questa api gli utenti possono visualizzare i messaggi di una Community. Nella richiesta è presente l'id della Community, grazie al quale possiamo eseguire una ricerca nel database per trovare, e successivamente ritornare, tutti i messaggi legati ad essa. Nel caso venga trovato un errore, il programma ritorna un messaggio di errore con codice 500.

```
// gets all messages from a community
const getTexts = async function(req,res,next) {
  var community_id = req.params.chat_id;
  // only logged-in users can look at the community chat
  if(req.loggedUser.id != null) {
    Chat.find({community_id: community_id.toString()},(err, messages)=> {
      if(err)
        return res.send(messages);
    })
  }
}
```

2.5.5 Visualizza i messaggiche un utente ha inviato in una Community

Questa api viene utilizzata per poter visualizzare tutti i messaggi che un particolare utente ha inviato in una community. Il procedimento è identico alla ricerca dei messaggi di una Community eccetto per il fatto che la ricerca include anche il nome dell'utente interessato. Nel caso venga trovato un errore, il programma ritorna un messaggio di errore con codice 500.

```
// gets all messages from a user
const getUserTexts = async function(req,res,next) {
  var community_id = req.params.chat_id;
  if(req.loggedUser.id != null) {
    Chat.find({nome_utente: req.params.nome_utente, community_id: community_id.toString()},(err, messages)=> {
      //res.redirect()
      return res.send(messages);
    })
  }
}
```

2.5.6 Elimina un messaggio specifico

Ogni utente ha la possibilità di eliminare un messaggio specifico che ha inviato. Questa api utilizza l'id del messaggio e l'id dell'utente (ottenuto tramite token) per eliminare il messaggio. Nella ricerca del messaggio includiamo anche l'id dell'utente per evitare che qualcuno possa eliminare messaggi inviati da altri. Nel caso venga trovato un errore, il programma ritorna un messaggio di errore.

```
// delete single text
const deleteText = async function (req,res,next) {
  try {
    // since we want to delete a specific message we need it's id (the same user might send the same message multiple times)
    // we don't need the community since the message_id is unique
    var message_id = req.params.message_id
    //console.log(id);
    // finding the message and making sure that the user that wants to delete it is the same user that sent it
    await Chat.findOneAndDelete({_id: message_id, id_utente: req.loggedUser.id},(err, messages)=> {
      return res.send("eliminato");
    })
    return res.sendStatus(200);
  }
  catch(error) {
    console.log(error)
  }
  finally{}
}
```

2.5.7 Elimina tutti i miei messaggi inviati in una Community

Grazie all'id dell'utente contenuto nel token e all'id della community contenuto nell'url, ogni utente ha la possibilità di eliminare tutti i propri messaggi inviati in una community specifica. Nel caso venga trovato un errore, il programma ritorna un messaggio di errore con codice 500.

```
const deleteMyTexts = async function (req,res,next) {
    // we can obtain the community id from the parameters
    var community_id = req.params.chat_id;

    try {
        // using deleteMany method on user and community
        await Chat.deleteMany({id_utente: req.loggedUser.id, community_id: community_id.toString()}).then(function(){
        }).catch(function(error){
            console.log(error);
        })
        return res.sendStatus(200);
    }
    catch(error) {
        console.log(error)
        return res.sendStatus(500);
    }
    finally{}
}
```

2.5.8 Invia un nuovo messaggio ad un Artista

Questa api viene utilizzata per inviare un messaggio ad un Artista col quale è stata avviata una collaborazione. Per fare ciò è necessario conoscere l'id della collaborazione, l'id dell'utente che vuole inviare il messaggio e il messaggio (ottenuto tramite body). Successivamente viene creato un oggetto contenente l'id del mittente, l'id della collaborazione, il corpo del messaggio, l'ora alla quale è stato inviato il messaggio e lo stato del messaggio (impostato a: "SENT"). Se l'utente non è un artista o se prova ad inviare un messaggio ad un Artista con il quale non ha avviato una collaborazione, verrà inviato un messaggio di errore.

```
const sendMessage = (req, res, next) =>{
  let user_id = req.loggedUser.id;

  // controllo ci sia il corpo del messaggio
  let text = req.body.text;
  if (text == null)
    return res.json({success: false, message: "Testo vuoto"});

  // verifico che nella collab io sia presente e lo stato sia ACCEPTED
  var collab_id = req.params.collab_id;

  Collab.findById(collab_id, function(err, collab){
    if (err) return res.json({success: false, message: "Errore nel trovare la collaborazione"});
    if (collab.from != user_id && collab.to != user_id){
      // utente manda messaggio ad una collaborazione non sua
      return res.json({success: false, message: "Non puoi inviare messaggi in questa collaborazione"});
    }
    if (collab.status != "ACCEPTED"){
      // utente manda messaggio a chat ancora da accettare
      return res.json({success: false, message: "Stato della collaborazione non ACCEPTED"});
    }

    // manda messaggio
    CollabChat.create({from: user_id,
                      collab: collab_id,
                      text: req.body.text,
                      datetime: Date.now(),
                      status: "SENT"}, function(err, message){
      if (err) return res.json({success: false, message: "Messaggio non inviato"});
      return res.json({success: true, message: "Messaggio inviato", message: message});
    });
  });
}
```

2.5.9 Visualizza messaggi di una Collaborazione

Questa api serve per poter visualizzare i messaggi che sono stati scambiati tra due artisti che hanno avviato una collaborazione. L'api utilizza l'id della collaborazione ottenuto tramite url e l'id dell'utente ottenuto tramite token per cercare nel database tutti i messaggi appartenenti a quella determinata collaborazione. Se l'id dell'utente coincide con quello di uno dei due artisti, vengono ritornati tutti i messaggi. In caso contrario viene ritornato un messaggio di errore.

```
const getMessages = (req, res, next) => {
  var collab_id = req.params.collab_id;
  var user_id = req.loggedUser.id;

  // prendo la collab e vedo se è dell'utente
  Collab.findById(collab_id, function(err, collab){
    if (err) return res.json({success: false, message: "Qualcosa è andato storto"});
    if (collab.from != user_id && collab.to != user_id)
      return res.json({success: false, message: "Non puoi ottenere i messaggi di questa chat"});

    // restituisco chat
    CollabChat.find({collab: collab_id}, function(error, chat){
      return res.json({success: true, messages: chat});
    });
  });
}
```

2.5.10 Segna messaggi come già letti

Grazie all'id dell'utente (ottenuto sempre tramite token) e l'id della collaborazione (ottenuto tramite url) questa api segna tutti i messaggi presenti nella collaborazione come "già letti" ritornando un messaggio di successo. Se ci sono stati errori ritorna un messaggio di errore.

```
const setRead = (req, res, next) => {
  // setta tutti i messaggi che hanno come destinatario me e chat id quella passata come letti
  let collab_id = req.params.collab_id;
  let user_id = req.loggedUser.user_id;

  CollabChat.updateMany({collab: collab_id, to: user_id}, {status: "READ"}, function(err){
    if (err) return res.json({success: false, message: "Non è stato possibile impostare come letti i tuoi messaggi"});
    res.json({success: true, message: "Messaggi impostati come letti"});
  });
}
```

2.5.11 Visualizza messaggi non letti

Questa api viene utilizzata per permettere all'artista di visualizzare i messaggi non letti di una conversazione avvenuta con un altro artista. Per fare ciò utilizza l'id dell'utente e l'id della collaborazione, cercando nel database i messaggi che non sono stati letti (ovvero con stato: "SENT"). Se l'utente non ha i permessi per poter accedere alla conversazione viene ritornato un messaggio di errore.

```
const getUnreadMessages = (req, res, next) => {
  var collab_id = req.params.collab_id;
  var user_id = req.loggedUser.id;

  // prendo la collab e vedo se è dell'utente
  collab.findById(collab_id, function(err, collab){
    if (err) return res.json({success: false, message: "Qualcosa è andato storto"});
    if (collab.from != user_id && collab.to != user_id)
      return res.json({success: false, message: "Non puoi ottenere i messaggi di questa chat"});

    // restituisco chat
    CollabChat.find({collab: collab_id, status: "SENT"}, function(error, chat){
      return res.json({success: true, messages: chat});
    });
  });
}
```

2.5.12 Elimina messaggio

Questa api prende in input l'id dell'utente e l'id del messaggio che si vuole eliminare e, successivamente, lo rimuove dal database ritornando un messaggio di successo se non ci sono stati problemi. In caso contrario viene inviato un messaggio di errore.

```
const deleteMessage = (req, res, next) => {
  // posso cancellare un messaggio se sono stato io a inviarlo
  let user_id = req.loggedUser.id;
  let message_id = req.message_id;

  CollabChat.findOneAndDelete({from: user_id, id: message_id}, function(err){
    if (err) return res.json({success: false, message: "Errore nell'eliminare il messaggio"});
    return res.json({success: true, message: "Messaggio eliminato"});
  });
}
```

2.5.13 Invia richiesta di Collaborazione

Questa api viene utilizzata ogni volta che un artista vuole inviare una richiesta di collaborazione ad un altro artista. Per poter inviare la richiesta di collaborazione bisogna controllare che i

due utenti interessati siano entrambi artisti e che un utente non stia inviando una richiesta a se stesso. Successivamente controlliamo che la collaborazione non sia già esistente. Nel caso in cui uno di questi requisiti non sia soddisfatto viene inviato un messaggio di errore, altrimenti viene creata la richiesta di collaborazione assieme ad un messaggio di successo.

```
const sendCollab = (req, res, next) => {
  // TODO: verifica requisiti?
  let destination_user_id = req.params.user_id;
  let sender_user_id = req.loggedUser.id;

  // verifico che siano differenti (no collab con se stessi)
  if(destination_user_id == sender_user_id){
    return res.json({success: false, message: "Collaborazione con se stessi non disponibile"});
  }

  // verifico che entrambi gli account siano di artisti
  if (!req.loggedUser.is_artist){
    return res.json({success: false, message: "Devi essere un artista per avviare una collaborazione"});
  }

  let isArtist = true;
  User.findOne({id: destination_user_id}, function(err, item){
    if (err) isArtist = false; return;
    if(!item.is_artist) isArtist = false;
  });

  // destinatario non è un'artista
  if(!isArtist) return res.json({success: false, message: "Non puoi collaborare con un utente non artista"});

  // vedo se la collaborazione è già esistente
  Collab.find({from: sender_user_id, to: destination_user_id}, function(err, items){
    if (err) return res.json({success: false, message: "Error, qualcosa è andato storto"});
    if (items.length == 0){
      // collaborazione non esistente in precedenza
      Collab.create({from: sender_user_id, to: destination_user_id, status: "SENT"}, function(err, item){
        if (err) return res.json({success: false, message: "Error creating a collab request"});
        console.log(item);
        return res.json({success: true, message: "Collab request created", accept: {path: "/collab/collab_id/" + item.id, method: "GET"}, decline: {path: "/collab/collab_id/" + item.id, method: "DELETE"}});
      })
    } else {
      // collaborazione già esistente, restituisco errore e stato vecchia collab
      return res.json({success: false, message: "Collaborazione già esistente", status: items[0].status});
    }
  });
}
```

2.5.14 Accetta richiesta di Collaborazione

Questa api viene chiamata ogni volta che un artista decide di accettare una richiesta di collaborazione. Per poter funzionare necessita l'id dell'utente e l'id della collaborazione; successivamente controlla che la collaborazione non sia già stata accettata, se non ci sono errori lo stato della collaborazione viene impostato ad: "ACCEPTED", ritornando un messaggio di successo.

```
const acceptCollab = (req, res, next) => [
  let user_id = req.loggedUser.id;
  let collab_id = req.params.collab_id;

  // controllo che la collab sia in stato pendente (controllo non necessario, ma utile)
  Collab.findOne({id: collab_id}, function(err, item){
    if (err) return res.json({success: true, message: "Errore nell'accettare la richiesta di collaborazione"});
    if (item.status == "ACCEPTED") {return res.json({success: false, message: "Collab già accettata"})}
  });

  // imposto la collab con stato attivo
  // la imposto cercando con user_id e collab_id, per verificare che sia effettivamente rivolta a me
  Collab.findOneAndUpdate({to: user_id, id: collab_id}, {status: "ACCEPTED"}, function(err){
    if (err) res.json({success: false, message: "Impossibile accettare la richiesta di collaborazione"});
    res.json({success: true, message: "Collaborazione accettata"});
  })
]
```

2.5.15 Rifiuta richiesta di Collaborazione

Questa api funziona allo stesso modo della api per l'accettazione della richiesta di collaborazione, con l'unica differenza che in caso di successo lo stato della collaborazione viene impostato a: "REJECTED".

```
const declineCollab = (req, res, next) => {
  let user_id = req.loggedUser.id;
  let collab_id = req.params.collab_id;

  // controllo che la collab sia in stato pendente (controllo non necessario, ma utile)
  Collab.findOne({id: collab_id}, function(err, item){
    console.log(item.status);
    if (err) return res.json({success: true, message: "Errore nell'accettare la richiesta di collaborazione"});
    if (item.status == "REJECTED") {return res.json({success: false, message: "Collab già rifiutata"})}
  });

  // imposto la collab con stato rifiutato
  // la imposto cercando con user_id e collab_id, per verificare che sia effettivamente rivolta a me
  Collab.findOneAndUpdate({to: user_id, id: collab_id}, {status: "REJECTED"}, function(err, item){
    console.log(item);
    if (err) res.json({success: false, message: "Impossibile rifiutare la richiesta di collaborazione"});
    res.json({success: true, message: "Collaborazione rifiutata"});
  })
}
```

2.5.16 Visualizza Collaborazioni

Tramite l'id dell'utente per cercare tutte le collaborazioni collegate a quell'utente. Se c'è un errore viene inviato un messaggio di errore, altrimenti vengono ritornate le collaborazioni.

```
const getAllCollabs = (req, res, next) => {
  let user_id = req.loggedUser.id;

  Collab.find({$or:[{from: user_id}, {to: user_id}]}, function(err, items){
    if (err) return res.json({success: false, message: "Errore nel cercare le tue richieste"})
    return res.json({success: true, collabs: items})
  });
}
```

2.5.17 Pubblica Brano

Questa api viene utilizzata ogni volta che un artista vuole pubblicare un brano. Prende il titolo del brano, contenuto nel body della richiesta, e il nome dell'artista contenuto nel token per creare un nuovo oggetto, salvandolo nel database. Anche in questa api controlliamo che l'utente sia un artista e ritorniamo messaggi di errore nel caso vengano rilevati errori. In caso contrario ritorniamo un messaggio di successo

```
const newSong = (req, res, next) => {
  console.log(req.loggedUser);
  // controllo sia artista, altrimenti restituisco error
  // TODO: cambiare da isArtista a is_artista
  if (!req.loggedUser.is_artista){
    return res.status(403).json({success:false, message:'Not Authorized'});
  }

  let song = req.body.song;
  const songToSave = new Song({title: song.title,
    artist: req.loggedUser.id})

  songToSave.save(function (err, item) {
    if (err){
      console.log(err)
      return res.json({success: false, message: "Error, song not saved"});
    }
    return res.json({success: true,
      message: "Success, song saved"});
  });
};
```

2.5.18 Visualizza Brani

Questa api viene utilizzata tutte le volte che un utente si trova in una pagina contenente brani musicali. Ritorna tutti i brani contenuti nel database.

```
// print all songs
const getSongs = async function (req, res, next) {
  res.json({success: true, songs: await Song.find()})
};
```

2.5.19 Cerca Brano

Quando un utente vuole cercare un brano specifico viene chiamata questa api. Tramite l'id della canzone controlla che essa sia presente nel database e, in caso di successo, ritorna la canzone. Se invece sono stati rilevati errori viene ritornato un messaggio di errore.

```
// get one song by id
const getSong = (req, res, next) => {
  let song_id = req.params.song_id;
  Song.findOne({id: song_id}, function(err, song){
    if (err) return res.json({success: false, message: "Song not found, is the id correct?"})

    return res.json({success: true, song: song})
  })
};
```

2.5.20 Modifica Brano

Questa api viene chiamata ogni volta che un artista vuole modificare le informazioni presenti in una canzone che ha caricato. L'api estraе l'id della canzone dal body della richiesta e l'id dell'utente dal token. Controlla che l'utente sia un artista (nel caso non lo sia ritorna un messaggio di errore), successivamente cerca una canzone contenente i due id estratti. Se l'id dell'utente coincide con l'id dell'autore del brano e non vengono rilevati errori, viene ritornato un messaggio di avvenuta modifica. In tutti gli altri casi viene inviato un messaggio di errore.

```
// modify one song by id
const modifySong = (req, res, next) => {
  if (!req.loggedUser.is_artista){
    return res.status(403).json({success:false, message:'Not Authorized'});
  }

  // prendo id della canzone dal corpo della richiesta
  let song = req.body.song;
  let song_id = song.id;

  // id dell'artista presente nel token
  let token_user_id = req.loggedUser.id;
  // confronto con l'id dell'artista della canzone
  Song.findOne({id: song_id}, function(err, song){
    if (err) return res.json({success: false, message: "Song not found, is the id correct?"});
    if (token_user_id != song.artist){
      // qualcuno sta cercando di modificare la canzone di un altro
      return res.json({success: false, message: "Not Authorized"});
    }

    // qui tutto regolare, posso modificare la canzone
    Song.findByIdAndUpdate(song_id, song, function(err){
      if (err) return res.json({success: false, message: "Uncatched error"});
      return res.json({success: true, message: "Song modified"});
    });
  });
};
```

2.5.21 Elimina Brano

Questa api è molto simile alla api che viene utilizzata per modificare un brano, con l'unica differenza che, in caso di successo, il brano viene eliminato dal database.

```
const deleteSong = (req, res, next) => {
  // solo un artista può eliminare una sua canzone
  if (!req.loggedUser.is_artista){
    return res.status(403).json({success:false, message:'Not Authorized'});
  }

  // prendo id della canzone dal corpo della richiesta
  let song_id = req.params.song_id;
  let token_user_id = req.loggedUser.id;

  Song.findOne({id: song_id}, function(err, song){
    if (err) return res.json({success: false, message: "Song not found, is the id correct?"});
    if (token_user_id != song.artist){
      // qualcuno sta cercando di modificare la canzone di un altro
      return res.json({success: false, message: "Not Authorized"});
    }

    // qui tutto regolare, posso modificare la canzone
    Song.findByIdAndRemove(song_id, song, function(err){
      if (err) return res.json({success: false, message: "Uncatched error"});
      return res.json({success: true, message: "Song deleted"});
    });
  });
};
```

2.5.22 Elimina i miei Brani

Questa api viene utilizzata tutte le volte che un artista vuole eliminare tutti i brani che ha pubblicato. Essa controlla che l'utente sia un artista (ricavando l'id dal token). Successivamente, se non vengono rilevati errori, elimina tutti i brani che hanno come autore quell'utente inviando un messaggio di avvenuta eliminazione di tutti i brani.

```
// delete all songs by artist
const deleteSongs = (req, res, next) => {
  if (!req.loggedUser.is_artista){
    return res.status(403).json({success:false, message:'Not Authorized'});
  }
  // utente autenticato ed artista
  Song.deleteMany({artist: req.loggedUser.id}, function(err){
    if (err) return res.json({success: false, message: "Error deleting all songs"});
    return res.json({success: true, message: "All songs deleted"});
  });
};
```


3.1 Gestione degli errori

Nella documentazione la maggior parte degli errori è stata classificata come: "error 400", mostrando una lista dei potenziali errori che potrebbero comparire. Di seguito un'immagine che mostra la documentazione della API di login che mostra anche la lista degli errori.

The screenshot shows the API documentation for the `/api/auth/login` endpoint. The `POST` method is selected. In the **Parameters** section, there is a single required parameter named `Login` (User Login) located in the `(body)` section. The schema for this parameter is defined as:

```
{  
  "email": "string",  
  "password": "string"  
}
```

The **Responses** section contains two entries:

- Code 200:** Description Token Generated. Example value: { "token": "string" }
- Code 400:** Description Example Value | Model. Example value: {
 "error": "string",
 "Possible error": "string",
 "Enum": "string"
}
 Possible error values: [Error, give all the information, Wrong password, Error]

A `Try it out` button is located in the top right corner of the interface.

3.2 Models

Di seguito mostriamo la lista dei modelli utilizzati per documentare le nostre API



4 FrontEnd Implementation

Il frontend consiste in delle interfacce grafiche interattive tramite le quali è possibile eseguire diverse funzioni grazie alle API descritte nelle sezioni precedenti. Il front-end è stato realizzato con i linguaggi HTML, CSS e JavaScript. Le due schermate iniziali sono quella di SignUp e Login che richiamano le corrispettive API viste precedentemente. Nella schermata di SignUp è possibile creare un account inserendo i dati correttamente e scegliendo tra il ruolo artista o ascoltatore. Dalla pagina SignUp è presente un link che punta alla pagina di Login, nel caso si avesse già un account; la stessa funzione è presente nella pagina Login. Con la creazione di un account si crea un token, il quale sarà necessario per le API seguenti.

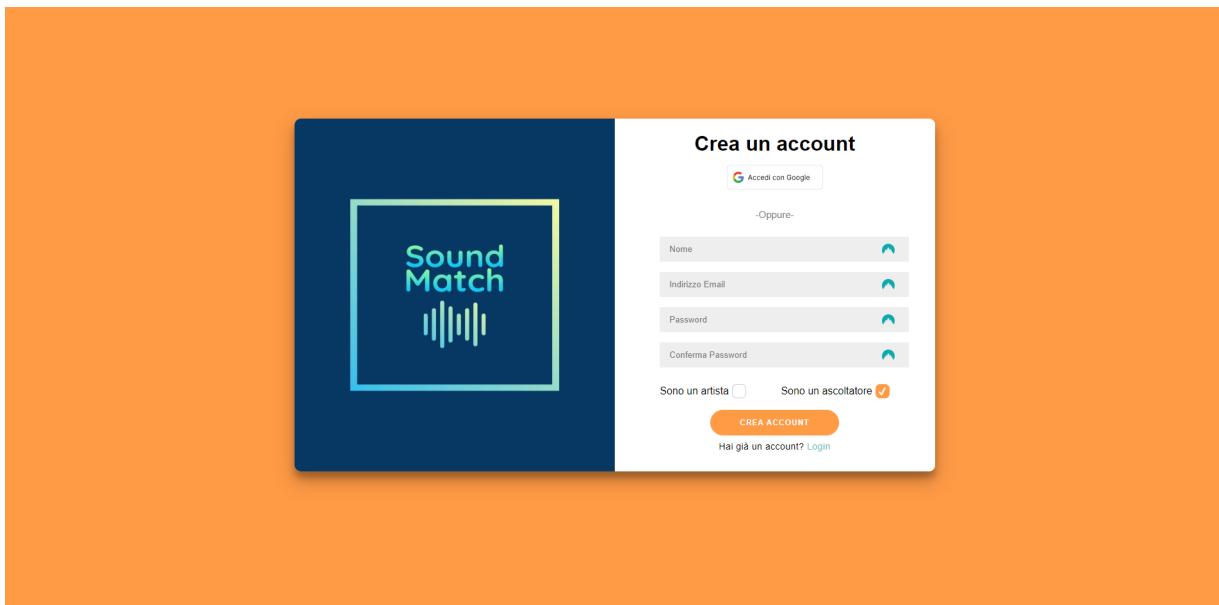


Figura 14: Pagina di SignUp

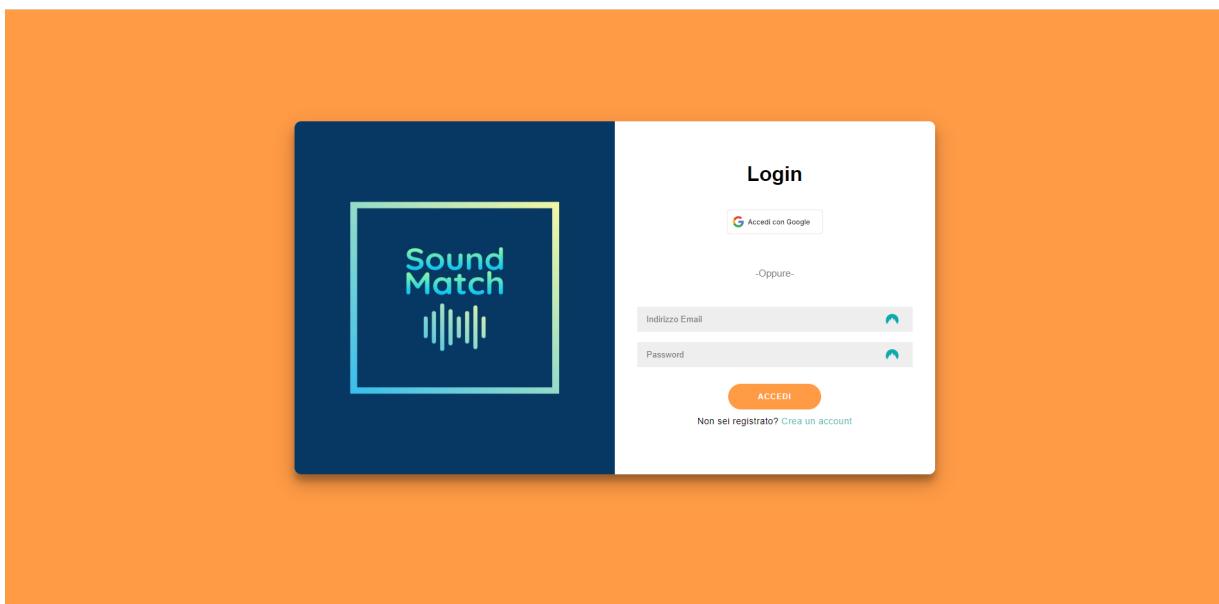


Figura 15: Pagina di Login

Se si crea un account artista, si avrà la possibilità di scegliere i generi musicali che caratterizzano la propria musica, attraverso la schermata Generi.

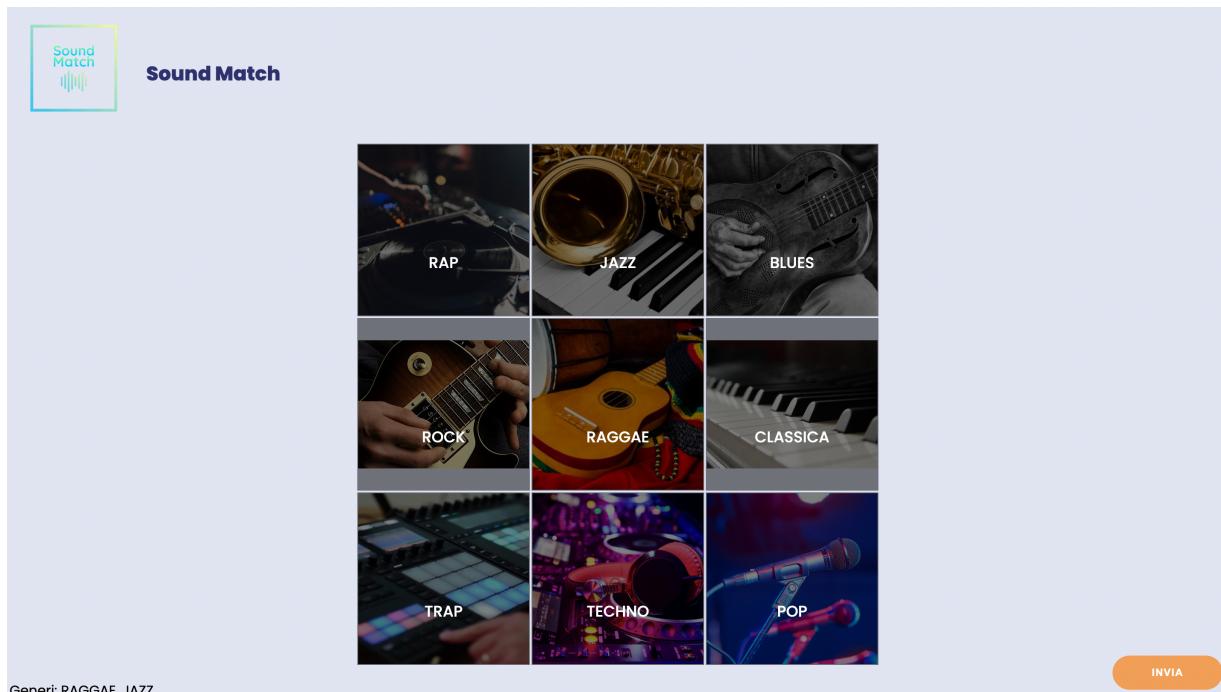


Figura 16: Pagina di scelta dei generi musicali

Figura 17: Pagina di Homepage

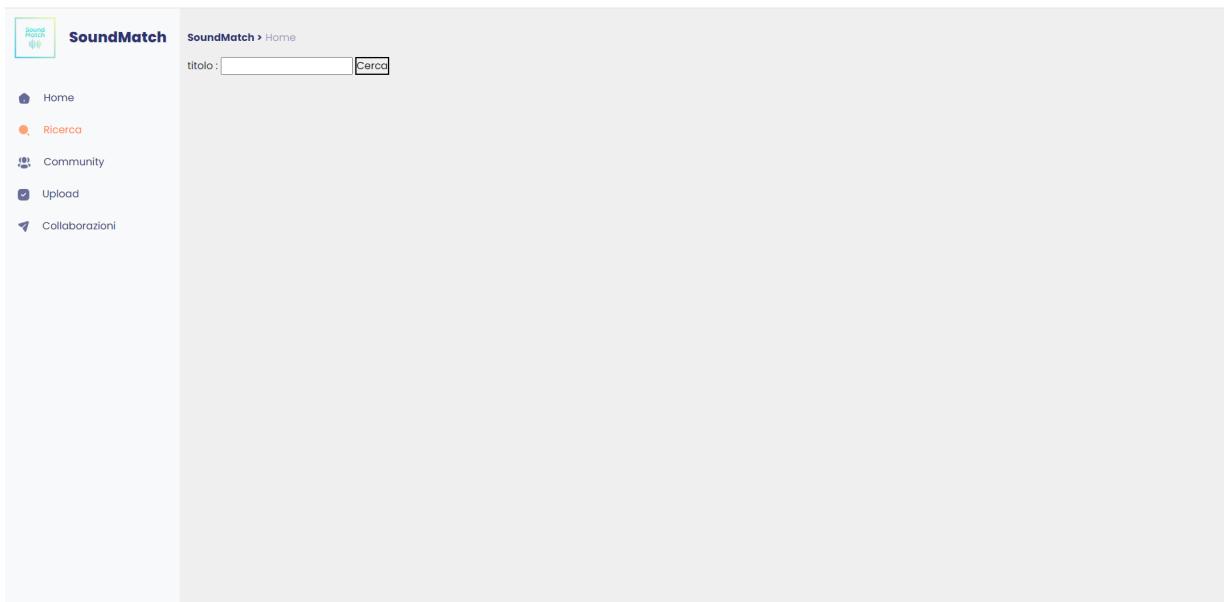
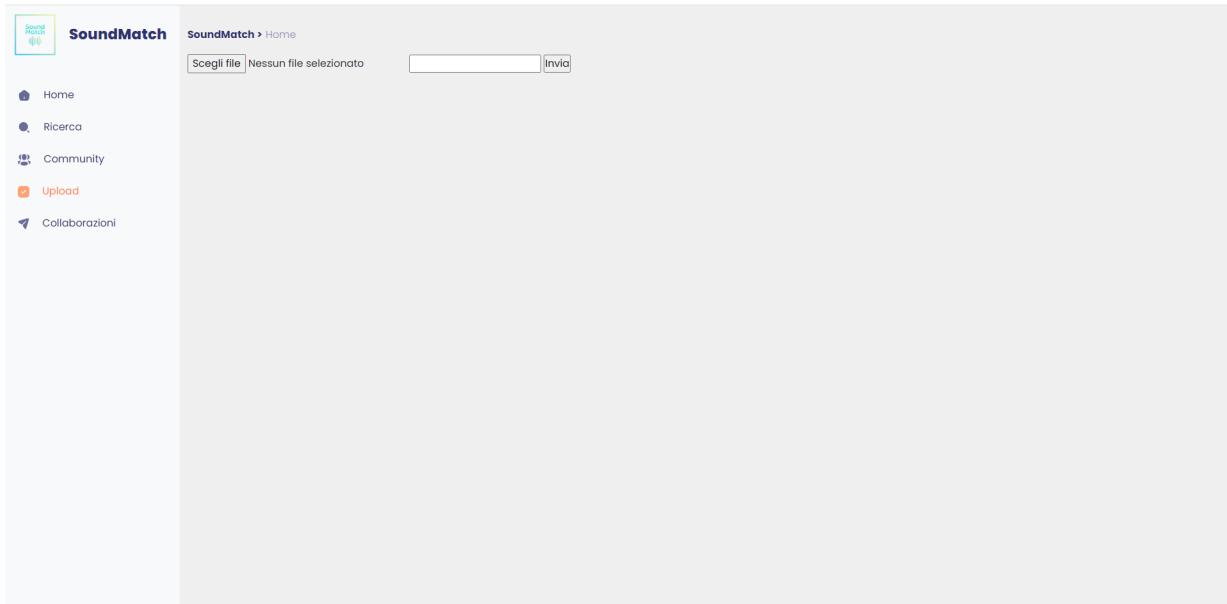


Figura 18: Pagina di Ricerca

A screenshot of the SoundMatch global chat interface. The title "Chat globale" is at the top. Below it is a text input field with the placeholder "Scrivi un messaggio". Underneath is a green "Invia" button. A red "Elimina i miei messaggi" button is also present. Further down is another text input field with the placeholder "Visualizza tutti i messaggi di un utente". Below this is a green "Cerca" button. At the bottom left are "Login" and "Fai il login" links. At the very bottom, there are two messages: "• Mario Rossi ciao a tutti D" and "• Mario Rossi Benvenuto nella chat globale D".

Figura 19: Pagina di Community

**Figura 20:** Pagina di Upload**Lista artisti**

- Gigi [Invia richiesta](#)
- Bebe K [Invia richiesta](#)
- Lil Billy [Invia richiesta](#)
- Mario Rossi [Invia richiesta](#)
- Peter Short [Invia richiesta](#)
- Apocalipsis [Invia richiesta](#)

Lista richieste collaborazioni ricevute**Lista richieste collaborazioni inviate**

- undefined [Elimina richiesta](#)

Lista collaborazioni avviate

- undefined [Chat](#)

Lista collaborazioni rifiutate**Figura 21:** Pagina di Collaborazioni

Completata la creazione dell'account, o effettuato completamente il login, si accede alla schermata homepage, nella quale è presente una lista scorrevole di brani, nei quali effettuando un click sul titolo, partita la musica aprendo una schermata di "pop-up" apposita. Sulla sinistra della pagina di homepage è presente una sidebar dove sono presenti i collegamenti

alle varie schermate che comprendono le funzionalità descritte dalle API.

La prima funzionalità partendo dall'alto della posizione della sidebar è la Ricerca. Essa permette di cercare brani tramite titolo, e se presenti, mostrare i titoli delle canzoni a video. Se si schiaccia su di essi, partirà la riproduzione della canzone scelta.

La seconda in ordine è la Community, la quale implementa una chat globale fra utenti che permette di mandare, visualizzare e eliminare i messaggi in chat. Inoltre è presente la funzionalità di ricercare i messaggi filtrandoli per il nome utente che si inserisce nell'apposito spazio. Un'altra è il caricamento di contenuti musicali, che è permessa solo ai profili artisti, accessibile attraverso la schermata Upload. In questa pagina è presente il tasto "Scegli file" il quale permette di navigare all'interno per proprio PC per selezionare il contenuto da caricare. Inoltre è presente una barra apposita per inserire il titolo della canzone di cui si vuole fare upload. L'ultima funzionalità riservata solo ai profili artisti è la schermata Collaborazioni la quale permette di inviare richieste di collaborazioni ad altri artisti presenti su SoundMatch, e gestire le proprie richieste ricevute. Se un'artista accetta una richiesta di collaborazione, verrà salvata nell'apposito spazio e verrà sbloccata la funzionalità chat tra di essi.

5 GitHub Repository and Deployment Info

La repository del sito web (frontend e backend) si trova su GitHub al seguente link: <https://github.com/ISuperBellissimi/SoundMatch> Le informazioni per eseguire il deploy sono contenute nel file README.md. Inoltre abbiamo hostato il sito web su un server linux, il link è: <http://38.54.13.226:3000/signup.html>

6 Testing

Per effettuare il testing delle API abbiamo utilizzato **Jest** e **Supertest**. Tutti i file di test sono contenuti nella cartella "testing", presente all'interno della cartella "controllers". In ogni file

sono presenti vari test che coprono tutte le API che appartengono alla stessa route.

Per eseguire i casi di test bisogna utilizzare il comando: **npm run testing <nomeFile>**.

Il testing va eseguito singolarmente per ogni file perché altrimenti jest segna errori (il problema dovrebbe essere dato dal fatto che jest esegue tutti i file di test in contemporanea, quindi la porta 3000 che viene utilizzata nel primo file di test risulta essere occupata per gli altri file).

Di seguito mostriamo il testing effettuato sulle API di autenticazione.

```
PASS controllers/testing/auth.test.js
  Test API auth
    ✓ should successfully create a user (641 ms)
    ✓ should successfully login (2 ms)
    ✓ should give error because of wrong password (1 ms)
    ✓ should give error because of missing username (3 ms)
    ✓ should give error because of missing password (3 ms)

  Test Suites: 1 passed, 1 total
  Tests:       5 passed, 5 total
  Snapshots:   0 total
  Time:        4.9 s, estimated 5 s
```