

Relazione progetto

Il progetto si propone di sviluppare un'applicazione web per la gestione e la condivisione di playlist musicali tra utenti.

L'applicazione permette di:

- Registrare un utente creando profili personalizzati.
- Gestire le playlist.
- Condividere le playlist create da una comunità.

L'applicazione sarà sviluppata usando come framework bootstrap per renderla totalmente responsiva.

Analisi dei requisiti

Il progetto è composto da tre principali macro-scenari:

- Gestione degli utenti.
- Gestione delle playlist.
- Gestione delle condivisioni.


Ogni utente una volta registrato può: creare e modificare il proprio profilo, organizzare playlist personalizzate attingendo ai dati forniti tramite le API REST di Spotify e condividere queste playlist con altri utenti. Le playlist possono essere cercate e visualizzate attraverso criteri di ricerca specifici come tag e titolo.

Profilo utente FRONT-END spiegazione interfaccie

Registrazione:

Basterà inserire i relativi dati nel form di registrazione.

[About](#) [Login](#)



Nome

Cognome

Data di nascita

Paese

Paese ▾

Email

Password (min 6)

Register

Show Custom

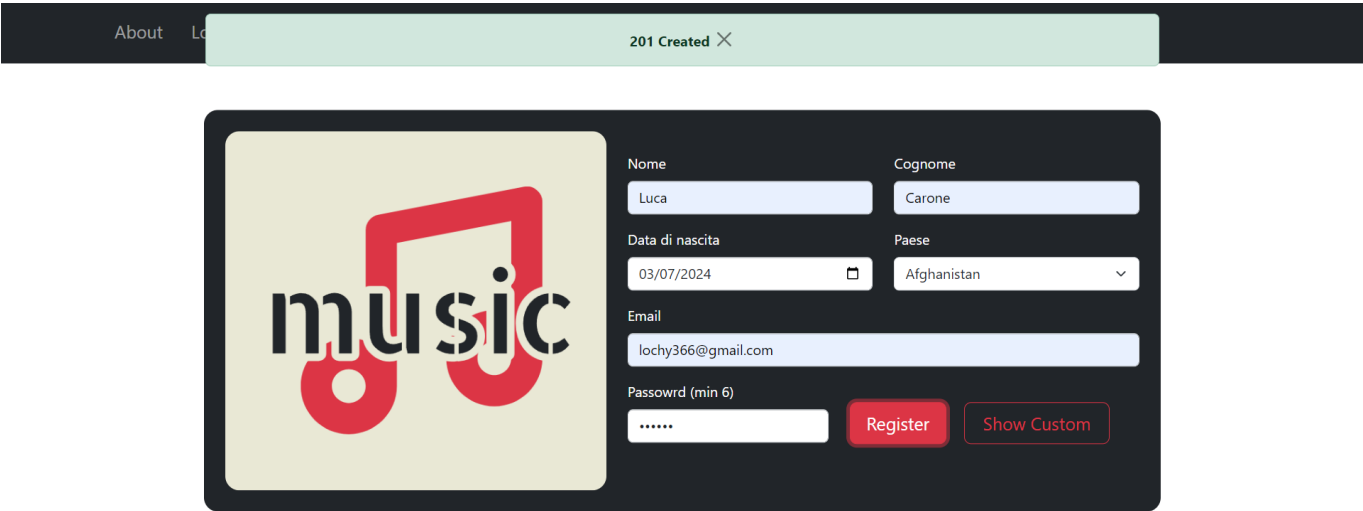
Cliccando show custom si apriranno le info aggiuntive.



La ricerca degli artisti viene effettuata usando le api di spotify, una volta cercato un artista basterà aggiungerlo o rimuoverlo.




Una volta finita la registrazione basterà premere reigstra, successivamente apparirà un banner per capire se la registrazione è avvenuta con successo o meno.



Login:

Per effettuare un login basterà inserire i dati nel form e premere login.

[Register](#) [About](#)



Email

abc@mail.com

Password

Log in

[Forgot password?](#)


Don't have an account?

Create new

Forgot password

Se la mail esiste invia una richiesta di reset password al server, essa viene stampata nella console.

[Register](#) 200 OK ✕



Email

lochy54@outlook.it

Invia richiesta

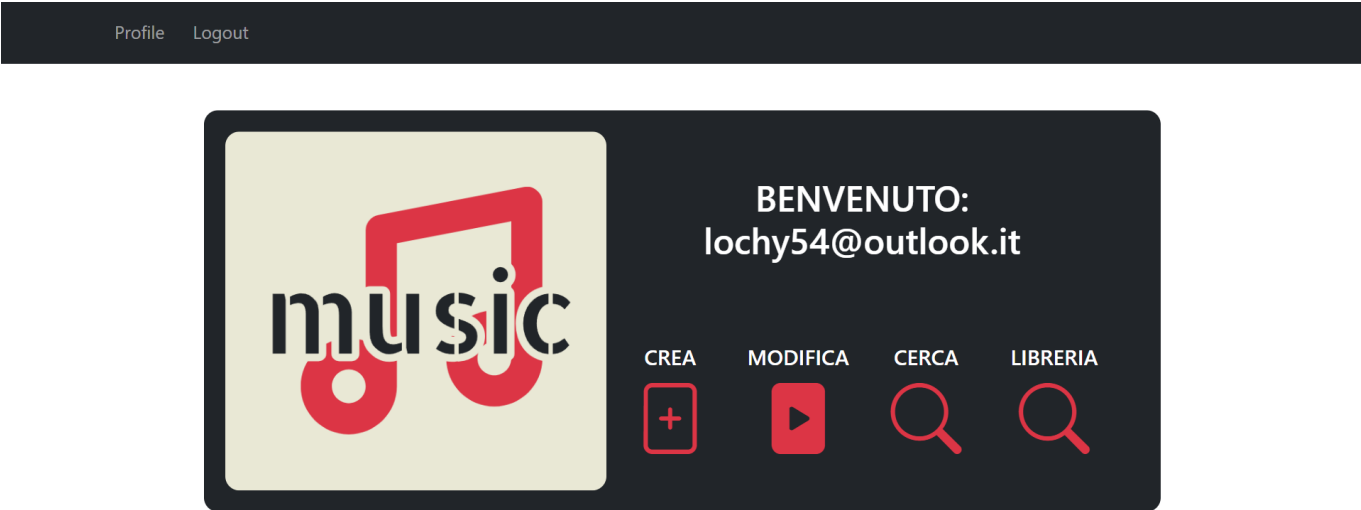
```
Forgot passwor: { email: 'lochy54@outlook.it' }
```

Homepage utente

Qui si potrà:

- Visualizzare il proprio profilo per:
 - eliminarlo
 - modificarlo.
- Effettuare logout.
- Creare una playlist (crea).
- Modificare una playlist (modifica)
 - eliminarla
 - modificarla.

- Cercare una playlist di un altro utente (eventualmente aggiungendo la playlist al proprio profilo).
- Visualizzare la libreria delle playlist personali + aggiunte.



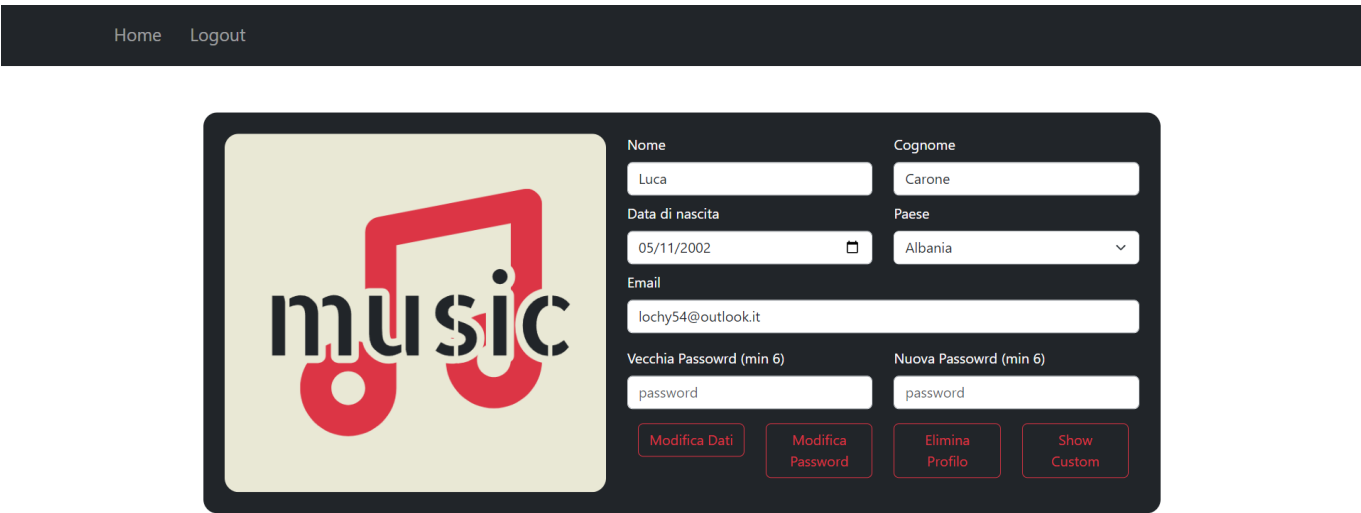
Profile

Tramite questa interfaccia è possibile modificare tutti i dati del profilo. Una volta cambiato un dato basterà premere "modifica dati" per inviare la richiesta al server.

Per cambiare la password sarà necessario inserire anche la vecchia password per motivi di sicurezza, successivamente si dovrà premere su "modifica password".

Premendo show custom si potranno visualizzare anche le info aggiuntive sul profilo.

Per eliminare il profilo basterà premere su elimina profilo.



Premendo il tasto home si tornerà alla schermata principale.

Una volta inviata la richiesta apparirà un banner con lo stato.

Logout

Premendo logout si uscirà dal profilo tornando alla pagina di login.


Crea

In questa pagina sarà possibile creare una playlist inserendo:

- Titolo.
- Descrizione.
- Tag.
- Pubblica o privata.

Una volta inseriti i campi obbligatori si potrà creare la playlist vuota o aggiungere ad essa delle canzoni.

Profile Home Logout



Titolo

Nome

Descrizione

Descrizione

Tag (separati da virgola no spazi)

Tag1,Tag2,Tag3

☐ Pubblica

Aggiungi Canzoni

Save

Una volta cercata una canzone si potrà aggiungerla o rimuoverla a piacimento, una volta aggiunta una canzone il timer del "totale" aumenterà del tempo necessario.

La ricerca delle canzoni viene effettuata attraverso le api di spotify.

Durata totale: 2:53

Cerca

Nome: AOK

Album: AOKDate: 2021-05-06

Artista: Tai VerdesDurata: 2:53

ADD

Nome: Foi Intenso - Ao Vivo

Canzoni Inserite

Nome: AOK

Album: AOKDate: 2021-05-06

Artista: Tai VerdesDurata: 2:53

REM

Una volta inviata la richiesta apparirà un banner con lo stato.

Premendo il tasto home si tornerà alla schermata principale.

Modifica


In questa interfaccia si potrà selezionare una playlist da noi creata per:

- Modificarla.
- Eliminarla.

Premendo "MOD" si entrerà nella schermata di modifiche, la quale è analoga a quella di creazione.

Per l'eliminazione basterà premere il tasto "DEL" ed apparirà un banner per la conferma dello stato.

Profile Home Logout



Cerca

Luca1

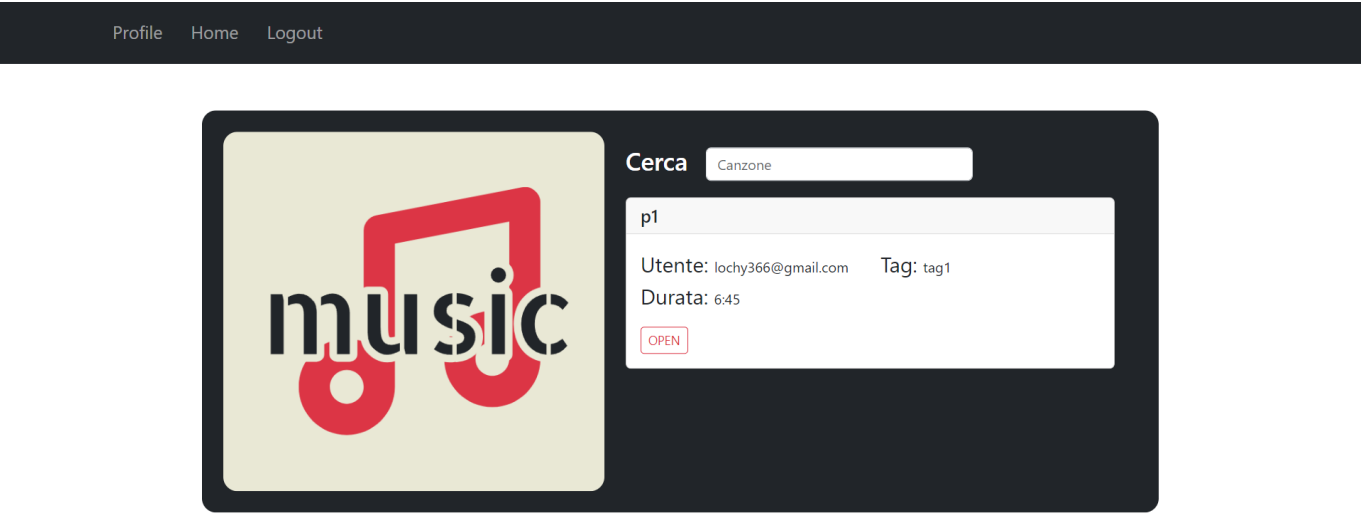
Utente: lochy54@outlook.itTag: tag

Durata: 6:45

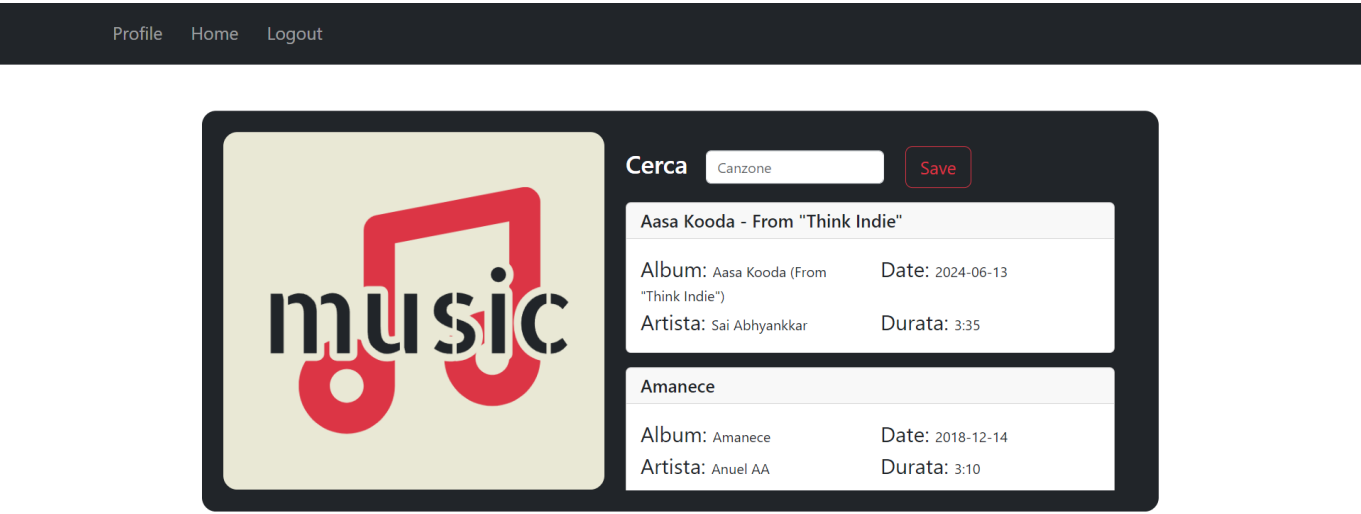
MODDEL

Cerca

In questa pagina si potranno cercare playlist di altri utenti per aggiungerle alla nostra libreria. La ricerca potrà essere effettuata per tag o nome playlist.



Premendo il tasto open si andrà ad aprire la playlist per controllarne il contenuto.

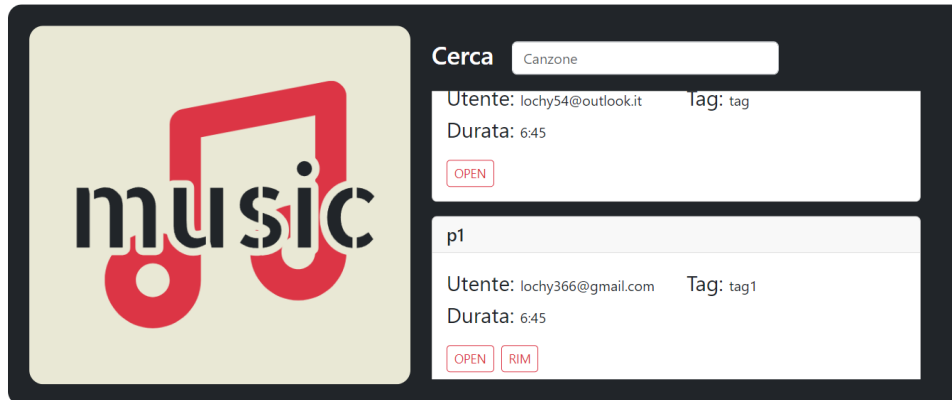


Qui possiamo decidere se salvare la playlist nel nostro profilo o meno. E' possibile ricercare canzoni all'interno della playlist tramite la barra di ricerca. Una volta salvata una playlist si tornerà alla schermata di ricerca playlist.

Libreria

In questa schermata è possibile visualizzare tutte le playlist della libreria.

- Posso rimuovere delle playlist aggiunte da altri account
- Visualizzare le playlist (aggiunte+create)

[Profile](#) [Home](#) [Logout](#)

Per rimuovere una playlist aggiunta basterà premere "RIM". Una volta inviata la richiesta sarà visualizzato un banner con lo stato di essa.

Il tasto "OPEN" permette di aprire una playlist ,visualizzarne il contenuto e cercare delle canzoni.

La ricerca potrà essere effettuata per tag o nome playlist

Server BACK-END

Server

Il back-end creato per gestire le risposte alle chiamate http del front-end utilizza express per creare un server sulla porta 3000.

```
const app = express(); // inizializzazione
const port = 3000; // port
// Middleware
app.use(express.json());
app.use(mongoSanitize());
app.use((req, res, next) => {
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.setHeader("Access-Control-Allow-Methods", "POST, GET, PUT, DELETE");
  res.setHeader("Access-Control-Allow-Headers", "Content-Type");
  next();
})
```

All' interno di una funzione asincrona ci sarà l'avvio del server.

```
//setup generi ask ogni 5 min (potrebbero cambiare)
(async () => {
  -----
  // Start server
  app.listen(port, () => {
```



```
    console.log(`Server is running on port ${port}`);  
  });  
  })();
```

Funzioni aggiuntive server

Introduciamo 3 funzioni aggiuntive prima di analizzare le varie routes:

chektoken:

Il server utilizza dei token di accesso generati ogni volta che un nuovo client effettua un login. Questi token saranno salvati dentro l'array dei token assieme ad un timestamp del tempo d'inserimento e la mail dell'utente. Il token sarà successivamente inviato al client, come risposta, per richieste future.

```
var tokenlis = [];
```

Ogni volta che un client fa una richiesta al server deve includere il token. Se il token è contenuto nel server e il timestamp associato è minore di 10 minuti la richiesta sarà garantita e il timestamp sarà aggiornato al tempo della nuova richiesta; in caso contrario la richiesta sarà rifiutata e il token sarà scartato.

```
//ce un token attivo , se c'è aggiorniamo l'orario  
function chektoken(value) {  
  const currentTime = new Date();  
  const tenMinutesAgo = new Date(currentTime.getTime() - (5 * 100 * 1000));  
  
  for (let index = 0; index < tokenlis.length; index++) {  
    if (tokenlis[index].token === value) {  
      if (tokenlis[index].time <= tenMinutesAgo) {  
        tokenlis.splice(index, 1);  
        return false;  
      } else {  
        tokenlis[index].time = currentTime;  
        console.log("Token aggiornato");  
        return true;  
      }  
    }  
  }  
}  
  
// Se il token non è stato trovato  
return false;  
}
```

findtoken:

Per effettuare alcune ricerche nel db è necessario usare la mail dell'utente; dato che nelle varie richieste dell'utente non viene mai inclusa la mail ma soltanto il token. Esiste una funzione che dato un token attivo restituisce una mail.

```
//dato un token trova l'email
function findtoken(token){
  for (let index = 0; index < tokenlis.length; index++) {
    if (tokenlis[index].token === token) {
      return tokenlis[index].user;
    }
  }
}
```

getGeneri:

Donde evitare di richiedere i generi a spotyfi ogni volta che un nuovo client effettua una modifica del profilo o una registrazione, ho introdotto questa funzione asincrona che ogni 5 minuti richiede i generi e li salva nell'array dei generi del server.

```
//setup generi ask ogni 5 min (potrebbero cambiare)
(async () => {
  generi = await getgenere();
  setInterval(async () => {
    generi = await getgenere();
  }, 5 * 60 * 1000);
  ---
})();
```

Zod

Questo pacchetto viene usato per controllare che i dati mandati dal client, nelle varie richieste, siano corretti e rispettino gli standard assegnati, prima di inserirli nel db.

```
import { z } from 'zod';
const userDataSchema = z.object({
  nome: z.string().min(2),
  cognome: z.string().min(2),
  data: z.date(),
  paese: z.string(),
  email: z.string().email(),
  artisti: z.array(z.string().min(2)).default([]),
});
---
try {
  userData.data = new Date(userData.data);
  userDataSchema.parse(userData);
  ----
```

Spotyfi api

Questa parte di codice introduce una funzione che restituisce un client spotyfi che ci permette di fare chiamate sfruttando le api

```

import request from "request";
import SpotifyWebApi from "spotify-web-api-node";
async function gettoken() {
  const client_id = '80c861fd6b084de3bddd82e305be6fcc';
  const client_secret = '0d45b84ca78e436eaf57b2a7d8961944';
  const authOptions = {
    url: 'https://accounts.spotify.com/api/token',
    headers: {
      'Authorization': 'Basic ' + (new Buffer.from(client_id + ':' +
client_secret).toString('base64'))
    },
    form: {
      grant_type: 'client_credentials'
    },
    json: true
  };
  return new Promise((resolve, reject) => {
    request.post(authOptions, function(error, response, body) {
      if (!error && response.statusCode === 200) {
        resolve(body.access_token);
      } else {
        reject(error);
      }
    });
  });
}
async function getapi() {
  var access_token = await gettoken();
  var spotifyApi = new SpotifyWebApi();
  spotifyApi.setAccessToken(access_token);
  return spotifyApi;
}
export {getapi}

```

Utilizzando il client id e il client seacret (presi dalla pagina ufficiale di spotyfi) genero un token per effettuare richieste al server di spotyfi. Questo token sarà poi usato per inizializzare un client che sarà poi usato per le varie richieste http.

Mongodb

Connect

Questa parte di codice implementa una funzione che ritorna un mongoClient usato per fare chiamate al db.

```

import { MongoClient } from 'mongodb';
async function connectToCluster() {
  let mongoClient;

  try {

```

```
    mongoClient = new MongoClient("mongodb://localhost:27017");
    await mongoClient.connect();
    return mongoClient;
  } catch (error) {
    throw new Error ('problemi di connessione');
  }
}
export {connectToCluster};
```

Una volta connesso al db ritorna il client sul quale fare le operazioni.

Sanitize

Utilizzo mongoSanitize per pulire tutte le query che effettuo verso il db (dove evitare una no-sql-Injection).

```
import mongoSanitize from 'express-mongo-sanitize';
app.use(mongoSanitize());
```

MongoDB

Collezioni

Nel mio db ho 2 collezioni:

Utenti

```
{
  "_id": ObjectId('65fd8c4fe25fadd091652acb'),
  "nome": "Luca",
  "cognome": "Carone",
  "data": 2002-11-05T00:00:00.000+00:00,
  "paese": "Albania",
  "email": "lochy54@outlook.it",
  "password": "123456",
  "generi": Array (1),
  "artisti": Array (1)
}
```



Un utente è una persona che si è registrata al sito

- id: identificativo univoco del profilo
- nome: nome del registrato
- cognome: cognome del registrato
- data: data di nascita del registrato
- paese: paese del registrato
- email: email del registrato
- password: password del profilo
- generi: generi che piacciono al registrato
- artisti: artisti che piacciono al registrato

negli utenti ho 2 indici unici:

Name and Definition	Type	Size	Usage	Properties
> _id_	REGULAR ⓘ	36.9 KB	3 (since Tue Jul 16 2024)	UNIQUE ⓘ
> email_1	REGULAR ⓘ	36.9 KB	0 (since Tue Jul 16 2024)	UNIQUE ⓘ

Oltre all'id ho impostato come indice unco anche l'email (ogni email può essere registrata una sola volta dato che identifica un utente nel db)

Playlist

```
_id: ObjectId('669528e66211955e962bde34')
nome: "Luca1"
tag: Array (1)
descrizione: "desc"
canzoni: Array (2)
public: true
durata: 405514
email: Array (1)
```

Una playlist è una collezione di canzoni

- id: identificativo univoco della playlist
- nome: nome della playlist
- tag: tag della playlist
- descrizione: breve descrizione testuale della playlist
- canzoni: array di id delle canzoni aggiunte alla playlist
- public: bool che identifica se la playlist è pubblica o privata
- durata: durata della playlist in ms
- email: lista di email che hanno la playlist in libreria
 - posizione 0 : creatore della playlist
 - posizione n: coloro che hanno aggiunto la playlist in libreria

Dato che i collegamenti profilo-playlist sono salvati in un record della playlist stessa, se il creatore di una playlist la elimina scomparirà a tutti.

nelle playlist ho 2 indici unici:

Name and Definition	Type	Size	Usage	Properties
> _id_	REGULAR ⓘ	36.9 KB	4 (since Tue Jul 16 2024)	UNIQUE ⓘ
> email_1_nome_1	REGULAR ⓘ	36.9 KB	0 (since Tue Jul 16 2024)	UNIQUE ⓘ COMPOUND ⓘ

Oltre all'id ho impostato come indice unico anche la coppia email-nomePlaylist (gli utenti non possono creare playlist con lo stesso nome di playlist già create dal loro profilo (attive)).

Swagger

E attivo lo swagger delle varie route del server.

```
import swaggerUi from "swagger-ui-express";
import swaggerDocument from "../swagger-output.json" with { type: "json" };
```

```
---  
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerDocument ));
```

Si può accedere ad esso sul localhost alla porta 3000 /api-docs.

Routes:

Genere

Questa route serve per richiedere la lista dei generi utilizzando le api di spotify. Ritorna 200 e la lista di generi.

```
app.get('/genere', (req, res) => {  
  console.log("generi richiesti");  
  res.status(200).json(generi);  
});
```

Elimina

Questa route serve per eliminare un profilo. In caso l'eliminazione venga eseguita con successo ritorna 200, nel caso si sia verificato un errore in fase di risposta o eliminazione manda uno status 500.

```
app.delete('/elimina', async (req, res) => {  
  console.log("Received elimination request with message:", req.body);  
  if(chektoken(req.body.token)){  
    let v = await elimina(findtoken(req.body.token))  
    res.status(v.code).json(v);  
  }else{  
    res.status(500).json({ res:false , code:500});  
  }  
});
```

modplaylist1

Dato un token (attivo) di un profilo restituisce le playlist create e ancora attive di quel profilo (200), nel caso si sia verificato un errore in fase di risposta o ricerca manda uno status 500.

```
app.post('/modplaylist1', async (req, res) => {  
  console.log("Received mod request with message:", req.body);  
  if(chektoken(req.body.token)){  
    let v = await modplaylist1(findtoken(req.body.token))  
    res.status(v.code).json(v);  
  }else{  
    res.status(500).json({ res:false , code:500});  
  }  
});
```

modplaylist2

Dato un token (attivo) di un profilo e un nome di una playlist creata da quel profilo e ancora attiva trova e ritorna le canzoni della playlist cercata (200), nel caso si sia verificato un errore in fase di ricerca o risposta manda uno status 500.

```
app.post('/modplaylist2', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await modplaylist2(findtoken(req.body.token) ,req.body.playlist)
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

modplaylist3

Dato un token (attivo) di un profilo trova e ritorna le playlist create e non da quel profilo ancora attive (200), nel caso si sia verificato un errore in fase di ricerca o risposta manda uno status 500.

```
app.post('/modplaylist3', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await modplaylist3(findtoken(req.body.token))
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

modplaylist4

Dato un token (attivo) di un profilo e un nome di una playlist creata o non da quel profilo e ancora attiva. Trova e ritorna le canzoni della playlist cercata (200), nel caso si sia verificato un errore in fase di ricerca o risposta manda uno status 500.

```
app.post('/modplaylist4', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await modplaylist4(findtoken(req.body.token) ,req.body.playlist)
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

eliminaPlaylist

Dato un token (attivo) di un profilo e un nome di una playlist elimina la playlist di quel profilo (200), nel caso si sia verificato un errore in fase di eliminazione o risposta manda uno status 500. Tutti gli altri utenti che si sono salvati la playlist non la vedranno più.

```
app.delete('/eliminaPlaylist', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await delPlaylist(findtoken(req.body.token),req.body.nome)
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500 });
  }
});
```

togliPlaylist

Dato un token (attivo) di un profilo e un nome di una playlist toglie la playlist di quel profilo (200), nel caso si sia verificato un errore in fase di eliminazione o risposta manda uno status 500. La playlist rimarrà attiva negli altri profili in cui è stata salvata.

```
app.delete('/togliPlaylist', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await remPlaylist(findtoken(req.body.token),req.body.nome)
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

modplaylist5

Dato un token (attivo) di un profilo trova e ritorna le playlist non create e non aggiunte dall'utente (200), nel caso si sia verificato un errore in fase di ricerca o risposta manda uno status 500.

```
app.post('/modplaylist5', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await modplaylist5(findtoken(req.body.token))
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```


modplaylist6

Dato un token (attivo) di un profilo e un nome di una playlist non creata da quel profilo e ancora attiva trova e ritorna le canzoni della playlist cercata (200), nel caso si sia verificato un errore in fase di ricerca o risposta manda uno status 500.

```
app.post('/modplaylist6', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await modplaylist2(req.body.emailpass, req.body.playlist)
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

register

Registra un nuovo utente nella piattaforma (200), nel caso ci siano problemi in fase di inserimento manda uno status 500, nel caso ci siano problemi di controllo dei dati inseriti manda uno status 400.

```
app.put('/register', async (req, res) => {
  console.log("Received registration request with message:", req.body);
  let v = await register(req.body, generi)
  console.log(v);
  res.status(v.code).json(v);
});
```

login

Effettua il login di un utente nella piattaforma (200), nel caso ci siano problemi in fase di connessione manda uno status 500, nel caso ci siano problemi di controllo dei dati inseriti manda uno status 400. Inoltre crea il token di accesso, lo salva nel server assieme al timestamp e lo manda all'utente (per chiamate future).

```
app.post('/login', async (req, res) => {
  console.log("Received login request with message:", req.body);
  let v = await login(req.body)
  console.log(v);
  res.status(v.code).json(v);
  if(v.res !== false){
    if(!chektoken(v.res)){
      tokenlis.push({token: v.res , time: new Date() , user: req.body.email})
      console.log(tokenlis);
    }
  }
});
```

logout

Effettua il logout di un utente dalla piattaforma (200) rimuovendo il token dalla lista dei token presenti.

```
app.post('/logout', (req, res) => {
  console.log("Received logout request with message:", req.body);
  for (let index = 0; index < tokenlis.length; index++) {
    if (tokenlis[index].token === req.body.token) {
      tokenlis.splice(index, 1);
      console.log("rimuovo "+req.body.token);
    }
  }
  res.status(200).json({ res:true , code:200});
});
```

mod

Dato un token (attivo) di un profilo restituisce i dati di tale profilo (200), nel caso ci siano problemi in fase di connessione manda uno status 500.

```
app.post('/mod', async (req, res) => {
  console.log("Received mod request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await mod(findtoken(req.body.token))
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

modPass

Dato un token (attivo) di un profilo , la nuova password e la vecchia password imposta la password a quella nuova (200), nel caso ci siano problemi in fase di connessione manda uno status 500, nel caso ci siano problemi di controllo dei dati inseriti manda uno status 400.

```
app.put('/modPass', async (req, res) => {
  console.log("modifica richiesta: ", req.body);
  if(chektoken(req.body.token)){
    let v = await modPass(req.body, findtoken(req.body.token))
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

modData

Dato un token (attivo) di un profilo e un json di campi da modificare aggiorna i dati del profilo (200), nel caso ci siano problemi in fase di connessione manda uno status 500, nel caso ci siano problemi di controllo dei dati inseriti manda uno status 400.

```
app.put('/modData', async (req, res) => {
  var tokenre = req.body.token;
  delete req.body.token;
  console.log("modifica richiesta: ", req.body);
  if(chektoken(tokenre)){
    let v = await modData(req.body,findtoken(tokenre),generi)
    res.status(v.code).json(v);
    for (let index = 0; index < tokenlis.length; index++) {
      if (tokenlis[index].token === tokenre) {
        tokenlis[index].user= req.body.email;
        console.log(tokenlis)
      }
    }
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

ADDplaylist

Dato un token (attivo) di un profilo, una mail del creante di una playlist e il nome della playlist al quale mi voglio aggiungere, aggiungo al mio profilo la playlist (200), nel caso ci siano problemi in fase di connessione o aggiunta manda uno status 500.

```
app.put('/ADDplaylist', async (req, res) => {
  console.log("Received add request with message:", req.body);
  if(chektoken(req.body.token)){
    let v = await
    ADDplay(findtoken(req.body.token),req.body.emailpass,req.body.playlist)
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

cerca

Dato un token (attivo) di un profilo e un dato da cercare (artista, nome canzone, album) cerca le canzoni più simili a quel caso e le restituisce(200), nel caso ci siano problemi in fase di connessione o ricerca manda uno status 500.

```
app.post('/cerca', async (req, res) => {
  console.log("cercato", req.body.cercato);
  if(chektoken(req.body.token)){
```

```
    let v = await cercato(req.body.cercato)
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

artisti

Dato un artista da cercare, lo cerca e le restituisce(200), nel caso ci siano problemi in fase di connessione o ricerca manda uno status 500.

```
app.post('/artisti', async (req, res) => {
  console.log("cercato", req.body.cercato);
  let v = await artisti(req.body.cercato)
  res.status(v.code).json(v);
});
```

salva

Dato un token (attivo) di un profilo e un json di una playlist, la salva nel profilo(200). Nel caso ci siano problemi in fase di connessione o salvataggio manda uno status 500, nel caso ci siano problemi di controllo dei dati inseriti manda uno status 400.

```
app.put('/salva', async (req, res) => {
  console.log("salva playlist: ", req.body);
  if(chektoken(req.body.token)){
    let v = await salva(req.body, findtoken(req.body.token))
    res.status(v.code).json(v);
  }else{
    res.status(500).json({ res:false , code:500});
  }
});
```

salvaMod

Dato un token (attivo) di un profilo e un json di una playlist, salva le modifiche nel profilo(200). Nel caso ci siano problemi in fase di connessione o salvataggio manda uno status 500, nel caso ci siano problemi di controllo dei dati inseriti manda uno status 400.

```
app.put('/salvaMod', async (req, res) => {
  console.log("salva playlist: ", req.body);
  if(chektoken(req.body.token)){
    let v = await salvaMod(req.body)
    res.status(v.code).json(v);
  }else{
```

```
    res.status(500).json({ res:false , code:500});  
  }  
});
```

forgot

Riceve una richiesta di password forgot e la stampa in console.

```
app.post('/forgot', async (req, res) => {  
  console.log("Forgot passwor: ", req.body);  
  res.json(forgot(req.body.email));  
  
});
```