MY MUSIC API

LUCAS F TELLAROLI

1. Introduction

My Music API is a SpringBoot/Java application that consumes Deezer open API and lets you create and edit your music playlists with your desired tracks. The application is containerized and run on any OS.

It has security control by JWT authentication with bearer token that need to be set as authorization header. The users registered on the system need to send a request to receive the token, which unlock other requests.

This documentation is designed to:

- Application developers
- Integration developers
- Testers
- Potential Clients
- Enthusiasts

2. Syntax Notations

This chapter is about the syntax used in the application endpoints.

1. Search patterns:

1. Search tracks from deezer library:

This search request is used to get the music IDs, which can be used when adding tracks to a playlist.

<scheme>://<url>/<path>?q=<product>

scheme = used protocol.

url = external API server and host.

path = API enpoint.

product = keyword for searching; music or album titles or artist name such as: "beat it", "michael jackson", "the dark side of the moon", and "50 cent".

2. Search users by ID:

This search request is used to find and get users info by inputting user ID.

scheme>://<server>:<port>/<path>/<object ID>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = API enpoint.

object ID = the user registered ID on application’s database.

3. Search playlist by user:

This search request is used to find and get all user playlists by inputting user ID.

<scheme>://<server>:<port>/<path>/ <playlistId>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = API enpoint.

userId = the user registered ID on application’s database.

4. Search playlist by ID:

This search request is used to find and get a playlists by inputing its ID.

<scheme>://<server>:<port>/<path>/<playlistId>/<product>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = API enpoint.

playlistId = the playlist registered ID on application’s database.

product = “tracks”.

2. Create Patterns:

1. Create new user:

This create request is used to create a new user.

<scheme>://<server>:<port>/<path>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = "users".

2. Create new playlist:

This create request is used to create new playlist.

<scheme>://<server>:<port>/<path>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = "playlists".

3. Update Patterns:

1. Update existing playlist:

This update request is used to edit playlists info.

<scheme>://<server>:<port>/<path>/<playlistId>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = "playlists".

playlistId = the playlist ID application's database.

4. Delete Patterns:

1. Deleting existing user:

This delete request is used to erase a user from repository.

<scheme>://<server>:<port>/<path>/<userId>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = "users".

userId = the user ID on application's database.

2. Deleting existing playlist:

This delete request is used to erase a user from repository.

<scheme>://<server>:<port>/<path>/<playlistId>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = "playlists".

playlistId = the playlist ID on application's database.

3. Deleting track from playlist:

This delete request is used to remove tracks from playlists.

<scheme>://<server>:<port>/<path>/<playlistId>/<product>/<trackId>

scheme = used protocol.

server = running application server.

port = port used to listening the application.

path = "playlists ".

playlistId = the playlist ID on application's database.

product = "tracks"

trackId = the track ID on application's database.

5. Response codes

Request Call	**Success status code**	**Fail status code**
GET	200	400, 401 or 404
POST	200 or 201	400, 401, 403 or 404
PUT	200	400, 401 or 404
DELETE	200	400, 401 or 404

3. Resources

1. User:

This object represents the clients of the system and their accounts. It is used to register and authenticate clients data and allow them to use the system.

Parameters type	**Parameters name**	**Description**
id	Long	User identifier
playlists	List<Playlist>	A list of Playlist objects
username	String	User chosen nickname
email	String	User's email address

password	String	User's chosen password
profiles	List<Profiles>	Represents user permissions in the system

1. User operations:

1. Create new user.

1. By sending a POST request to the endpoint "http://localhost:8081/users", clients can create a new user, inserting some information through the request body, in JSON format. The success status code for this operation is 201, created.

2. Example:

The request body must have the follow key/value variables:

```
{
    "name": "Pedro",
    "email": "qa@test.com",
    "password": "12345678"
}
```

The expected success response body for this example is the new user data:

```
{
    "id": 3,
    "name": "Pedro",
    "email": "qa@test.com"
}
```

2. Get user by ID.

1. By sending a GET request to the endpoint `http://localhost:8081/users/<userId>`, clients can search a user by inserting the user ID in the endpoint. This request does not need a request body. The success status code for this operation is 200, ok.

2. Example:

For the endpoint: "http://localhost:8081/users/3", the expected success response body for this example is the user data:

```
{
    "id": 3,
    "name": "Pedro",
    "email": "qa@test.com"
}
```

3. Login.

1. After created, users can log into system by sending a POST request to the endpoint “http://localhost:8081/login”. Clients can retrieve the authentication token by inserting email and password through the request body, in JSON format. The expected success status code for this operation is 200, ok.

2. Example:

The request body must have the follow key/value variables:

```
{
    "email": "qa@test.com",
    "password": "12345678"
}
```

The expected success response body for this example is the token and its type:

```
{
    "token": "eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJNeSBtdXNpYyBhcGkiLCJzdWIiOiIzIiwiaWF0IjoxNjQ0MjYzMjA3LCJleHAiOjE2NDQyNjUwMDd9.tJSvzuQhgd4Jpx1GZTtY-xiSUMyfnS5e47qsBpsrjDA",
    "type": "Bearer"
}
```

3. Delete user by ID.

1. By sending a DELETE request to the endpoint “http://localhost:8081/users/<userId>”, clients can remove a user from the database repository by inserting the user ID in the endpoint. This request does not need a request body. The expected success status code for this operation is 200, ok, and the response body for this request is:

```
“User <userId> deleted.”
```

2. Playlist

This object represents the playlists created by the clients of the system. It is used to register and manage the client’s desired tracks and can be edited by them.

Parameters name	**Parameters type**	**Description**
id	Long	Playlist identifier
title	String	Playlist’s chosen title
description	String	Playlist’s description, info
user	User	The owner User of the Playlist
trackIds	List<String>	A list of the tracks identifiers added to this playlist

2. Playlist operations:

1. Create new playlist.

1. By sending a POST request to the endpoint “http://localhost:8081/playlists”, clients can create a new playlist, inserting some information through the request body, in JSON format. The expected success status code for this operation is 201, created.

2. Example:

The request body must have the follow key/value variables:

```
{
    "title": "title",
    "description": "my first playlist",
    "userId" : "1"
}
```

The expected success response body for this example is a dto of the playlist created:

```
{

    "id": 8,
    "title": "title",
    "description": "my first playlist",
    "userId": 5,
    "tracks": []
}
```

2. Get user playlists.

1. By sending a GET request to the endpoint “http://localhost:8081/playlists/user/<userId>”, clients can get all the playlists of a user inserting their ID in the endpoint. This request does not need a request body. The expected success status code for this operation is 200, ok.

2. Example:

For the endpoint: “http://localhost:8081/playlists/user/8”, the expected success response body for this example is the playlist data:

```
[
    {
        "id": 8,
        "title": "title",
        "description": "my first playlist",
        "userId": 5,
        "tracks": []
    },
    {
        "id": 9,
        "title": "title 2",
        "description": "my second playlist",
```

```
        "userId": 5,
        "tracks": []
    }
]
```

3. Edit playlist title and description.

1. By sending a PUT request to the endpoint “http://localhost:8081/playlists/<playlistId>”, clients can update a existing playlist, inserting some information through the request body, in JSON format. The expected success status code for this operation is 200, ok.

2. Example:

The request body must have the follow key/value variables:

```
{
    "title": "another title",
    "description": "my playlist",
    "userId" : "5"
}
```

The expected success response body for this example is the playlist updated data:

```
{
    "id": 8,
    "title": "another title",
    "description": "my playlist",
    "userId": 5,
    "tracks": []
}
```

4. Delete playlist.

1. By sending a DELETE request to the endpoint “http://localhost:8081/playlists/<playlistId>”, clients can delete an existing playlist from the database repository, inserting the playlist ID in the endpoint. This request does not need a request body. The expected success status code for this operation is 200, ok, and the response body for this request is:

```
“Playlist <playlistId> deleted.”
```

3. Tracks

Parameters name	**Parameters type**	**Description**
id	String	Track identifier
title	String	Track title
duration	String	Track duration, in seconds
artist	ArtistDto	The artist who released this track

album	AlbumDto	The name of the album which this track belongs

2. Tracks operations:

1. Search tracks in the external API (Deezer).

1. By sending a GET request to the endpoint “https://deezerdevs-deezer.p.rapidapi.com/search?q=<keyword>”, clients can look for music, albums, and artists in the deezer repository, inserting the keyword in the endpoint. This request does not need a request body. The expected success status code for this operation is 200, ok, and the response body for this example is a list of tracks that have that keyword, by relevance order. Clients will need the track’s ID value from the response in order to add that music to a playlist.

2. Example:

The follow response is received when <keyword> = peaches:


```
{
    "data": [
        {
            "id": 1637412872,
            "readable": true,
            "title": "Peaches",
            "title_short": "Peaches",
            "title_version": "",
            "link": "https://www.deezer.com/track/1637412872",
            "duration": 198,
            "rank": 279052,
            "explicit_lyrics": true,
            "explicit_content_lyrics": 1,
            "explicit_content_cover": 0,
            "preview": "https://cdns-preview-3.dzcdn.net/stream/c-
39ac9995ca433ba9ccdf3721b87152f1-8.mp3",
            "md5_image": "626c80c646832b8240a126d9597accbb",
            "artist": {
                "id": 288166,
                "name": "Justin Bieber",
                "link": "https://www.deezer.com/artist/288166",
                "picture": "https://api.deezer.com/artist/288166/image",
                "picture_small": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/56x56-
000000-80-0-0.jpg",
                "picture_medium": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/250x250-
000000-80-0-0.jpg",
```

```
                "picture_big": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/500x500-
000000-80-0-0.jpg",
                "picture_xl": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/1000x1000
-000000-80-0-0.jpg",
                "tracklist": "https://api.deezer.com/artist/288166/top?li
mit=50",
                "type": "artist"
            },
            "album": {
                "id": 291212762,
                "title": "AFTER Ski Pt.1",
                "cover": "https://api.deezer.com/album/291212762/image",
                "cover_small": "https://e-cdns-
images.dzcdn.net/images/cover/626c80c646832b8240a126d9597accbb/56x56-
000000-80-0-0.jpg",
                "cover_medium": "https://e-cdns-
images.dzcdn.net/images/cover/626c80c646832b8240a126d9597accbb/250x250-
000000-80-0-0.jpg",
                "cover_big": "https://e-cdns-
images.dzcdn.net/images/cover/626c80c646832b8240a126d9597accbb/500x500-
000000-80-0-0.jpg",
                "cover_xl": "https://e-cdns-
images.dzcdn.net/images/cover/626c80c646832b8240a126d9597accbb/1000x1000-
000000-80-0-0.jpg",
                "md5_image": "626c80c646832b8240a126d9597accbb",
                "tracklist": "https://api.deezer.com/album/291212762/trac
ks",
                "type": "album"
            },
            "type": "track"
        },
        {
            "id": 1280165222,
            "readable": true,
            "title": "Peaches",
            "title_short": "Peaches",
            "title_version": "",
            "link": "https://www.deezer.com/track/1280165222",
            "duration": 198,
            "rank": 992645,
            "explicit_lyrics": true,
            "explicit_content_lyrics": 1,
            "explicit_content_cover": 1,
            "preview": "https://cdns-preview-1.dzcdn.net/stream/c-
179bee1bfc65440d7df06d2246209699-4.mp3",
            "md5_image": "87468622c8e7ac9dce7b541be136aa4c",
```

```
            "artist": {
                "id": 288166,
                "name": "Justin Bieber",
                "link": "https://www.deezer.com/artist/288166",
                "picture": "https://api.deezer.com/artist/288166/image",
                "picture_small": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/56x56-
000000-80-0-0.jpg",
                "picture_medium": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/250x250-
000000-80-0-0.jpg",
                "picture_big": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/500x500-
000000-80-0-0.jpg",
                "picture_xl": "https://e-cdns-
images.dzcdn.net/images/artist/22dd86b628a03d8dad3c7dfb33320a91/1000x1000
-000000-80-0-0.jpg",
                "tracklist": "https://api.deezer.com/artist/288166/top?li
mit=50",
                "type": "artist"
            },
            "album": {
                "id": 215962322,
                "title": "Justice",
                "cover": "https://api.deezer.com/album/215962322/image",
                "cover_small": "https://e-cdns-
images.dzcdn.net/images/cover/87468622c8e7ac9dce7b541be136aa4c/56x56-
000000-80-0-0.jpg",
                "cover_medium": "https://e-cdns-
images.dzcdn.net/images/cover/87468622c8e7ac9dce7b541be136aa4c/250x250-
000000-80-0-0.jpg",
                "cover_big": "https://e-cdns-
images.dzcdn.net/images/cover/87468622c8e7ac9dce7b541be136aa4c/500x500-
000000-80-0-0.jpg",
                "cover_xl": "https://e-cdns-
images.dzcdn.net/images/cover/87468622c8e7ac9dce7b541be136aa4c/1000x1000-
000000-80-0-0.jpg",
                "md5_image": "87468622c8e7ac9dce7b541be136aa4c",
                "tracklist": "https://api.deezer.com/album/215962322/trac
ks",
                "type": "album"
            },
            "type": "track"
        },
```


… And so it goes with a list of tracks that have this <keyword> in common.

2. Add tracks to existing playlist.

1. By sending a POST request to the endpoint “http://localhost:8081/playlists/<playlistId>/tracks”, clients can add tracks to an existing playlist, inserting the playlist ID in the endpoint and the deezer’s track ID through the request body, in JSON format. The expected success status code for this operation is 200, ok.

2. Example:

The request body must have the follow key/value variables:

```
{
    "id": "1280165222"
}
```

The expected success response body for this example is a Track object:

```
{
    "id": "1280165222",
    "title": "Peaches",
    "duration": "198",
    "artist": {
        "name": "Justin Bieber"
    },
    "album": {
        "title": "Justice"
    }
}
```

3. Get playlist tracks.

1. By sending a GET request to the endpoint “http://localhost:8081/playlists/<playlistId>/tracks”, clients can get all the tracks added to a playlist, inserting the playlist ID in the endpoint. This request does not need a request body. The expected success status code for this operation is 200, ok.

2. Example:

For the endpoint: “http://localhost:8081/playlists/8/tracks”, the expected success response body for this example is the playlist object with a list of the playlist’s tracks:

```
{
    "id": 8,
    "title": "another title",
    "description": "my playlist",
    "userId": 5,
    "tracks": [
        {
            "id": "1280165222",
            "title": "Peaches",
            "duration": "198",
            "artist": {
                "name": "Justin Bieber"
```

```
            },
            "album": {
                "title": "Justice"
            }
        },
        {
            "id": "4763165",
            "title": "Beat It",
            "duration": "257",
            "artist": {
                "name": "Michael Jackson"
            },
            "album": {
                "title": "Michael Jackson's This Is It"
            }
        }
    ]
}
```


4. Delete tracks from playlist.

1. By sending a DELETE request to the endpoint "http://localhost:8081/playlists/<playlistId>/tracks/<trackId>", clients can remove a track from the playlist by inserting the playlist ID and the rack ID in the endpoint. The expected success status code for this operation is 200, ok, and the response body for this request is:

```
"Track 4763165 deleted from playlist 8."
```

4. Error Representation

The follow table represent the expected errors status codes for this application:

Error status code	**Description**
400	Bad request
401	Unauthorized
403	Forbidden
404	Bad request
405	Method not allowed
500	Internal server error

5. Version Updates

This documentation refers to the base version of this API (1.0.1).

MY MUSIC API

LUCAS F TELLAROLI

,

INATEL DEVELOPERS PROGRAM – IDP

FEBRUARY, 2022