



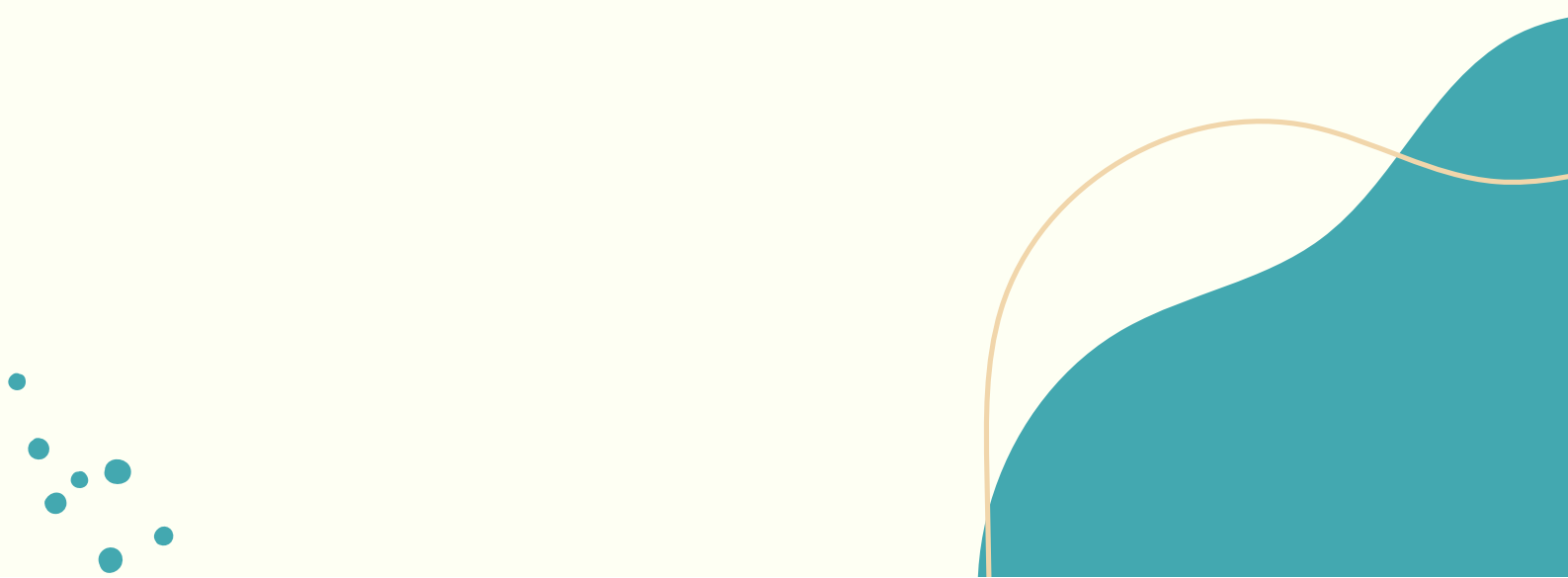
# **BASES DE DADOS**

## **Installation manual**

Ana Carolina dos Santos Morais N°2021222056  
anamorais@student.dei.uc.pt

Fernanda Margarida Rodrigues Fernandes N°2021216620  
mfernandes@student.dei.uc.pt

João Gonalo Reis Lopes N°2012170913 uc2012170913@student.uc.pt





## Introdução

Para este projeto é necessário obter algumas ferramentas. Assim, para facilitar a compreensão do uso dessas mesmas ferramentas, elaborámos este documento que vai permitir uma melhor compreensão.

## Linguagens de Programação necessárias

- Python
- SQL e pgSQL

## Sistema de Gerenciamento da base de dados

- PostgreSQL

## Bibliotecas utilizadas em Python

- flask
- psycopg2
- hashlib
- datetime
- calendar
- time
- random

## Outras tecnologias requeridas

- Onda
  - Postman
- 

## Instalação de ferramentas

Antes de começar a elaborar o projeto é necessário verificar se temos todas as bibliotecas e, se não tivermos é preciso instalar. Logo, encontra-se abaixo um guia com comandos para verificar o que é preciso.

- `pip install flask`
- `pip install psycopg2`

Ao realizar este comando, caso apareça os detalhes de cada pacote significa que o mesmo se encontra instalado. Caso o pacote não esteja, irá aparecer uma mensagem a indicar, que o mesmo não se encontra.

Relativamente ao *SQL* é necessário instalar, se possível, a última versão da *PostgreSQL* ou atualizar.

Para utilizar o *Postman* também é necessário instalar, pode ser feito a partir de uma página *web*.

## Configurações da Base de Dados

Para o acesso à base de dados, através da *psql* ou *pgadmin4* é necessário configurar o acesso, com *username* e *password* à escolha, a *porta* e o *localhost* já estão pré-definidos, não é preciso modificar, caso seja necessário. Após esta configuração é possível criar a nossa base de dados para este projeto.

## Mais informação

- <https://www.python.org/>
- <https://www.postgresql.org/>
- <https://www.postman.com/>

# BASES DE DADOS

## User manual

Descreve as solicitações da coleção do *Postman* enviadas para testar a aplicação.

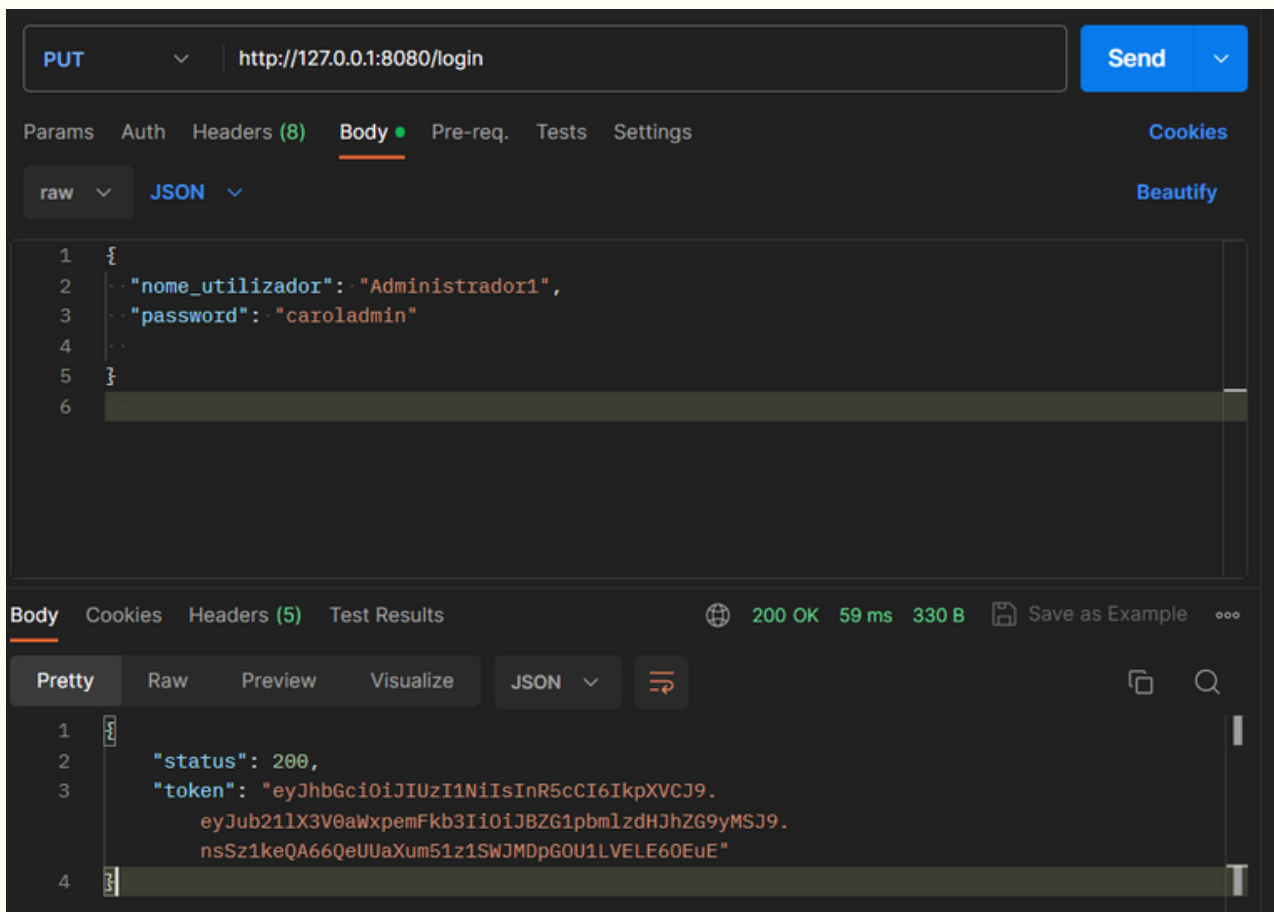
**Nota:** Antes de começar é importante correr o seguinte projeto em *python*  
⇒ *ProjetoFinal.py* e, a seguir abrir o Postman.

# User Authentication

**Descrição:** É pedido que cada utilizador faça 'login' na aplicação e, para isso é necessário fornecer o seu "nome\_utilizador" e a sua "password", caso seja bem-sucedido é devolvido um 'token' que será necessário para outras funcionalidades.

**URL:** /login

**Método:** PUT



# User Registration

**Descrição:** É nesta etapa que um utilizador se pode registar. Todos os utilizadores se podem registar como consumidores, mas apenas os artistas podem ser criados por um administrador. Neste caso, permitimos que os administradores se possam registar através desta função, dado que para a segurança das 'password' na base de dados é realizado uma encriptação na altura do armazenamento.

**URL:** /register

**Método:** POST

**Registo:** Artista / Consumidor

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8080/register`. The request body is a JSON object with the following fields: `nome_utilizador`, `nome`, `morada`, `email`, `contacto`, `password`, `nome_artistico`, `nome_gravadora`, and `token`. The response status is 200 OK, and the response body is a JSON object with `message`, `result`, and `status` fields.

```
POST http://127.0.0.1:8080/register

{
  "nome_utilizador": "Dillazz",
  "nome": "Andre Silva",
  "morada": "Guarda",
  "email": "mcdillaz@pt.com",
  "contacto": 987893267,
  "password": "dillazSilva",
  "nome_artistico": "Dillaz",
  "nome_gravadora": "Gravadora F",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lX3V0aWxpcmFkb3IiOiJBZG1pbmlzdHJhZG9yMSJ9.nsSz1keQA66QeUUaXum51z1SWJMDpG0U1LVELE60EuE"
}

{
  "message": "Registration completed",
  "result": "Dillazz",
  "status": 200
}
```

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8080/register`. The request body is a JSON object with the following fields: `nome_utilizador`, `nome`, `morada`, `email`, `contacto`, `password`, and `tipo`. The response status is 200 OK, and the response body is a JSON object with `message`, `result`, and `status` fields.

```
POST http://127.0.0.1:8080/register

{
  "nome_utilizador": "Consumidor3",
  "nome": "Amália Rodrigues",
  "morada": "Coimbra",
  "email": "Amalia2000@pt.com",
  "contacto": 987891211,
  "password": "consumidor3",
  "tipo": "consumidor"
}

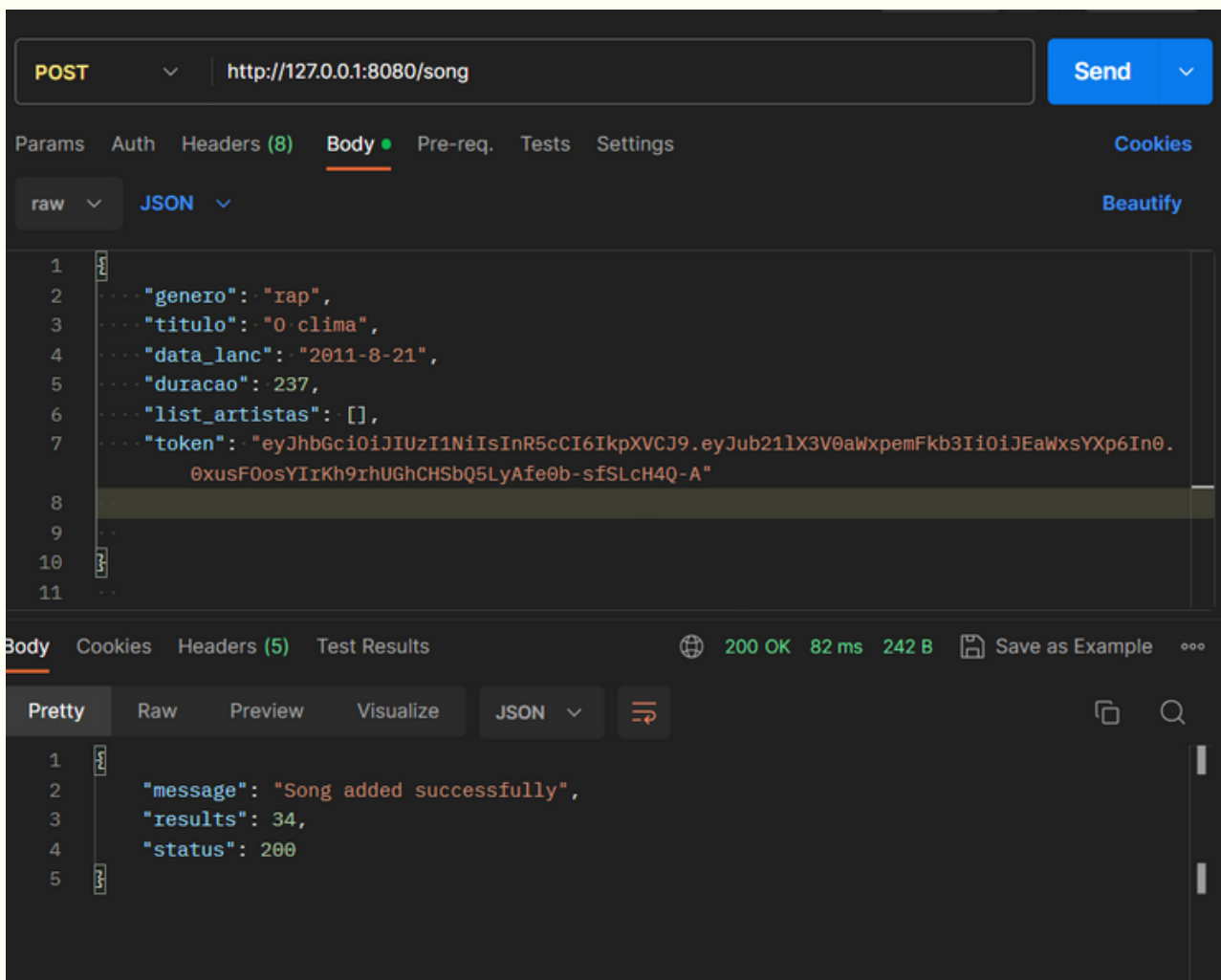
{
  "message": "Registration completed",
  "result": "Consumidor3",
  "status": 200
}
```

# Add song

**Descrição:** Os artistas podem publicar as suas músicas na plataforma. Eles podem fornecer informações adicionais sobre os outros artistas que também participam nas suas músicas. Não esquecendo que a duração é em segundos (s).

**URL:** /song

**Método:** POST

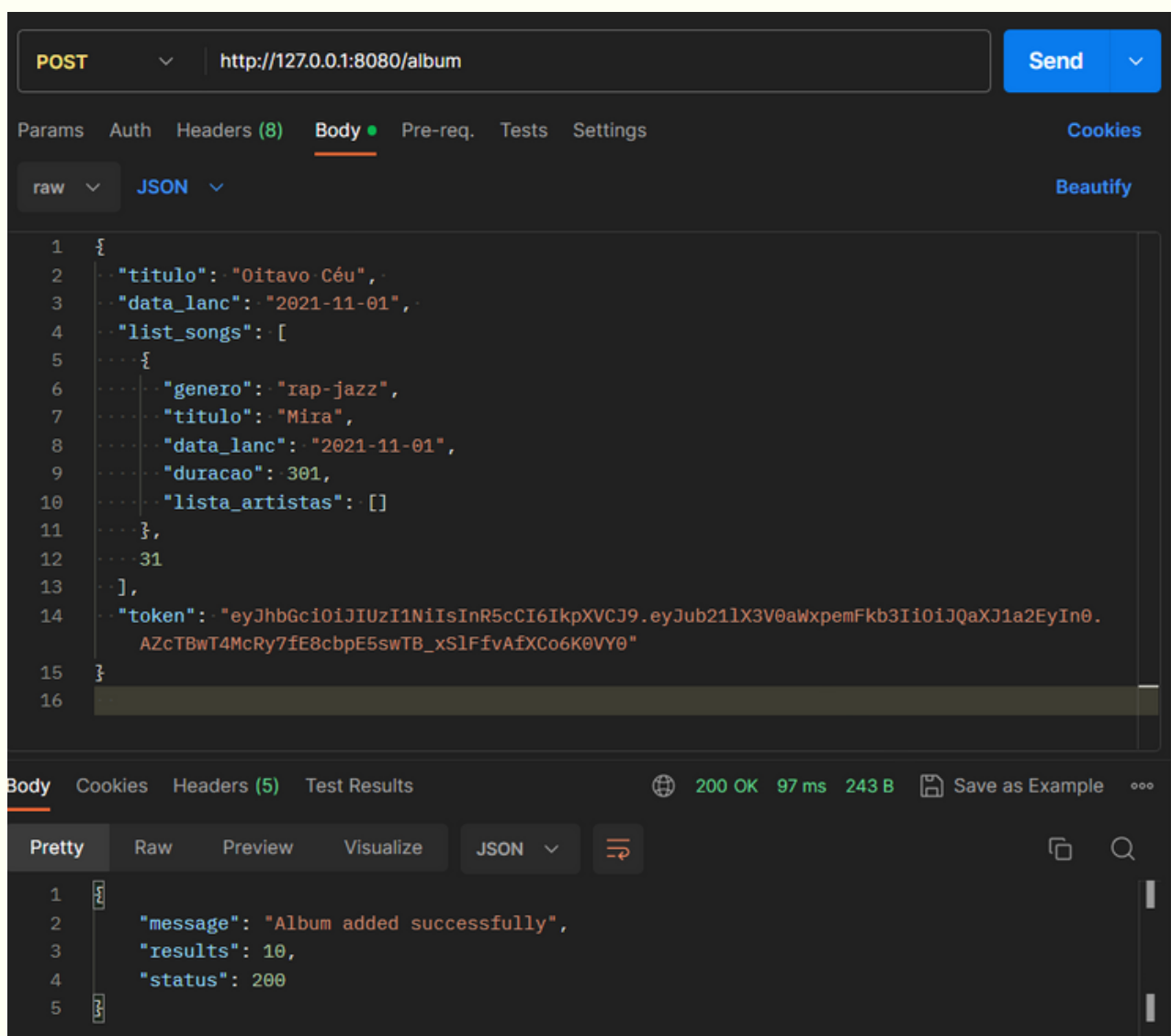


# Add album

**Descrição:** Os artistas podem publicar os seus álbuns na plataforma. É possível fornecer detalhes de novas músicas, bem como detalhes de músicas existentes já na plataforma. É aqui que podemos associar um álbum a uma música já existente, dado que quando se insere uma música não a associamos a um álbum.

**URL:** /album

**Método:** POST





# Search song

**Descrição:** Devolve todas as músicas que contenham a palavra-chave fornecida pelo utilizador. Não tem argumentos.

**URL:** /song/{keyword}

**Método:** GET

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8080/song/mi`. The response is a JSON object with a status of 200. The JSON body is as follows:

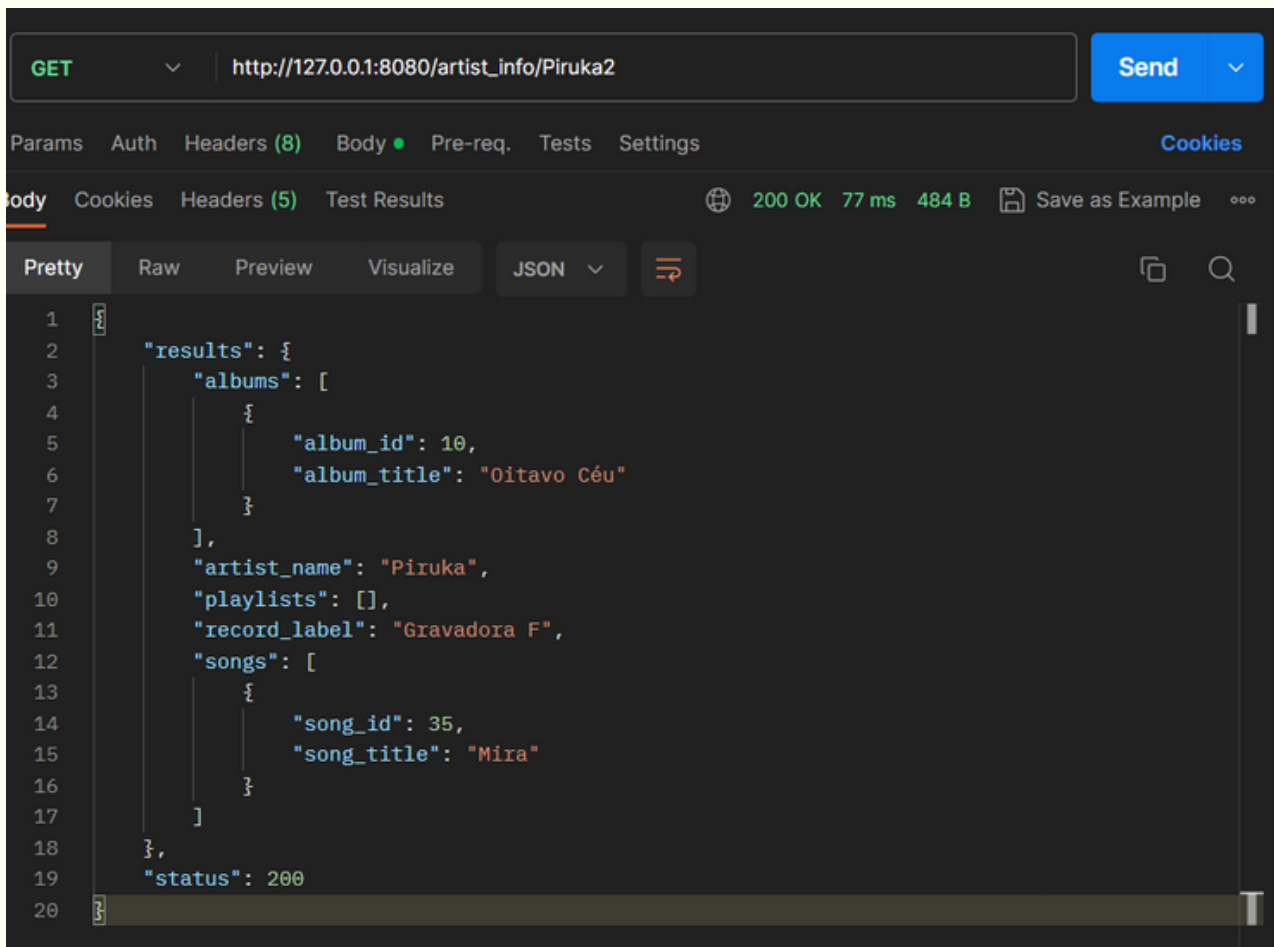
```
1  {
2    "results": [
3      {
4        "albums": [
5          9,
6          9
7        ],
8        "artists": [
9          "Dillaz",
10         "Piruka"
11       ],
12       "title": "Cemitério"
13     }
14   ],
15   "status": 200
16 }
```

# Detail artist

**Descrição:** Lista todas as informações relevantes sobre um artista. Todas as músicas, álbuns e listas de reprodução públicas. Não tem argumentos.

**URL:** /artist\_info/{artista\_username}

**Método:** GET



# Subscribe to Premium

**Descrição:** O consumidor fornece o número do(s) cartão(es) pré-pago(s) e o período de assinatura.

**URL:** /subscription

**Método:** POST

The screenshot displays a REST client interface with a POST request to `http://127.0.0.1:8080/subscription`. The request body is in JSON format, containing subscription details. The response is also in JSON format, indicating a successful subscription completion.

**Request:**

```
POST http://127.0.0.1:8080/subscription
```

**Body (JSON):**

```
{  "tipo_assinatura": "month",  "lista_cartoes": [2411977402723617, 4880301526414283, 9692779131183195],  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lX3V0aWxpeWpFkb3IiOiJ4cHRvIn0.YsdzYXWWpm3TplHtqymC9QfC5e6e41qGryVmEkIKUHM"}
```

**Response:**

```
200 OK 87 ms 239 B
```

**Body (JSON):**

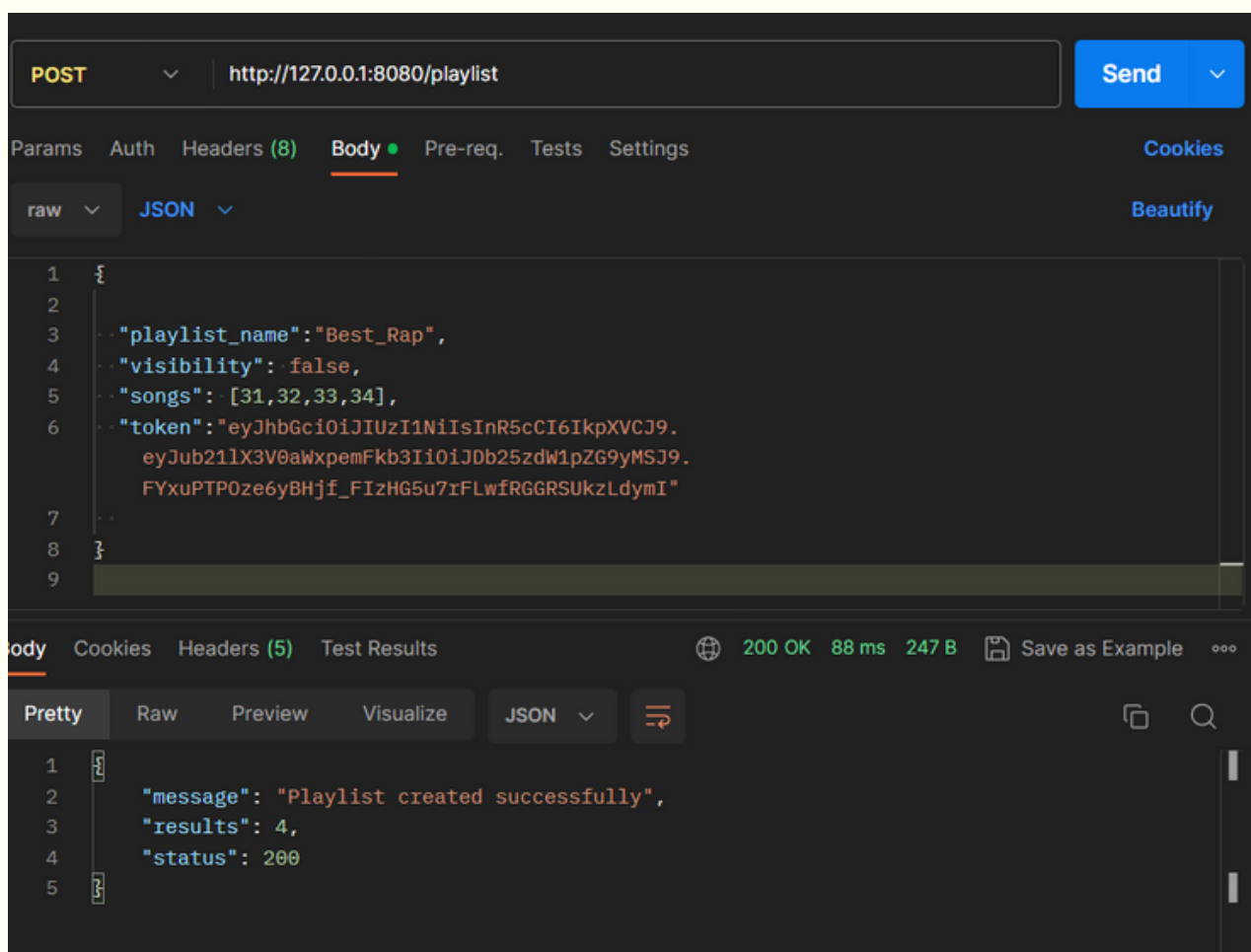
```
{  "message": "SUBSCRIPTION COMPLETED",  "result": 3,  "status": 200}
```

# Create Playlist

**Descrição:** Esta função só está disponível a consumidores com uma assinatura válida.

**URL:** /playlist

**Método:** POST

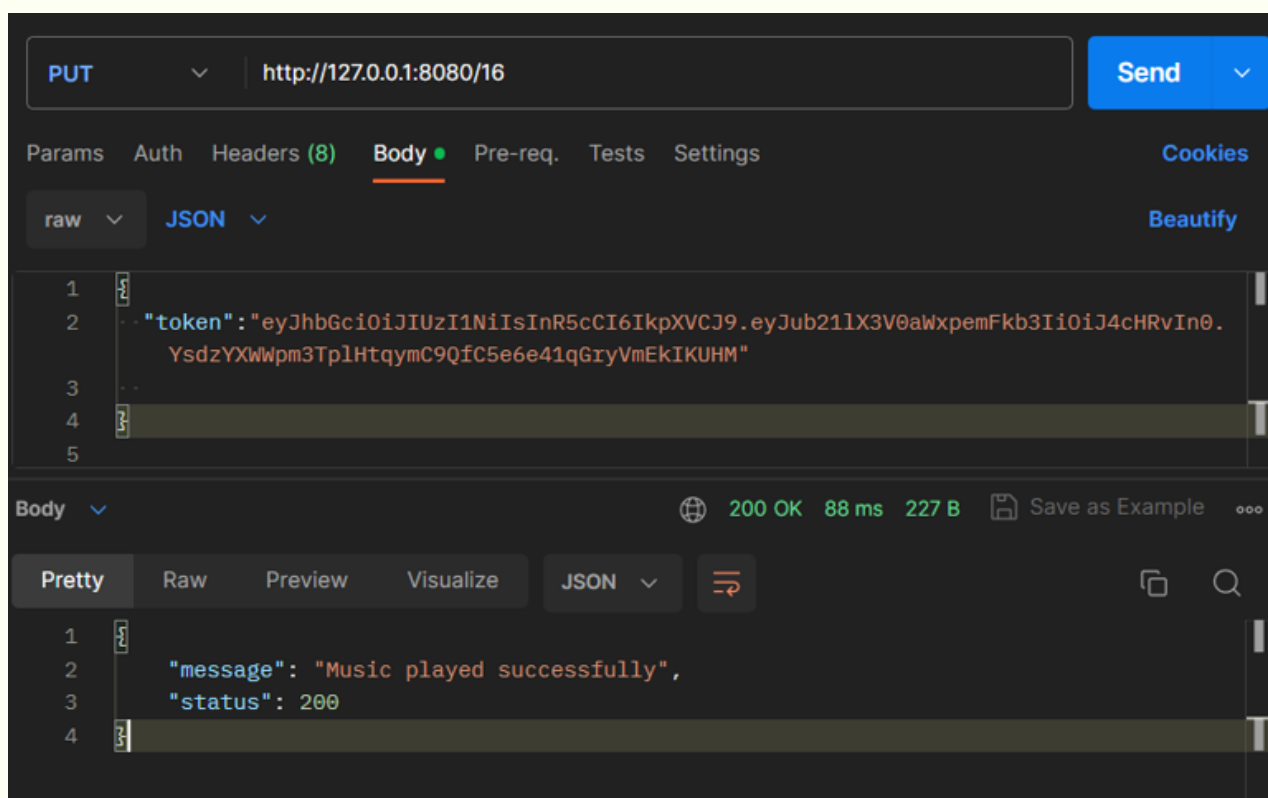


# Play song

**Descrição:** Ao "tocar uma música", basicamente assumimos que foi ouvida através do histórico e atualizamos o top 10 do consumidor, mediante um ‘trigger’.

URL: /<int:ismn>

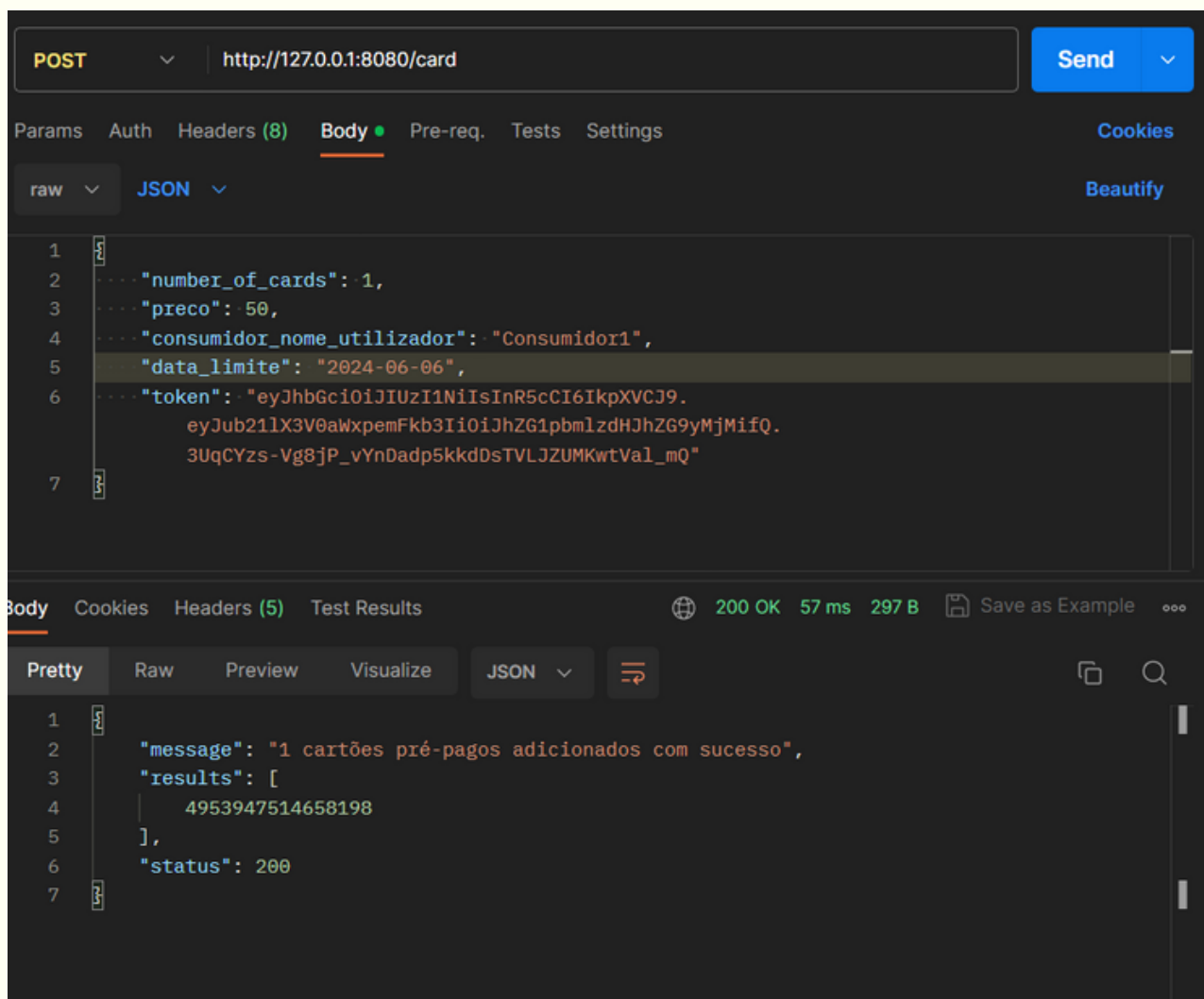
Método: PUT



## Generate pre-paid cards

**Descrição:** Apenas os administradores podem gerar cartões pré-pagos, mas é necessário fornecer a que consumidor, vai atribuir os cartões e, a data de validade.

URL: /card  
Método: POST



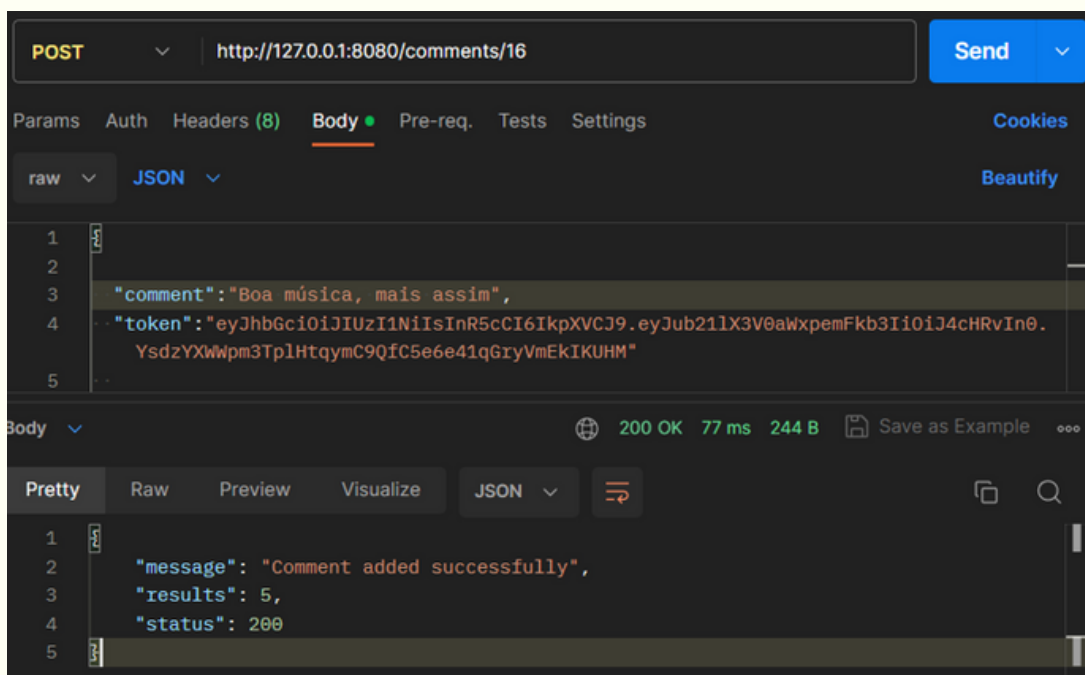
# Leave comment/feedback

**Descrição:** Apenas os consumidores podem deixar comentários e/ou responder a comentários a músicas. Para isso, é necessário fornecer o 'ismn' da música.

**URL:** /comments/<int:ismn>

**Resposta:** URL: /comments/<int:ismn>/<int:id\_comentario>

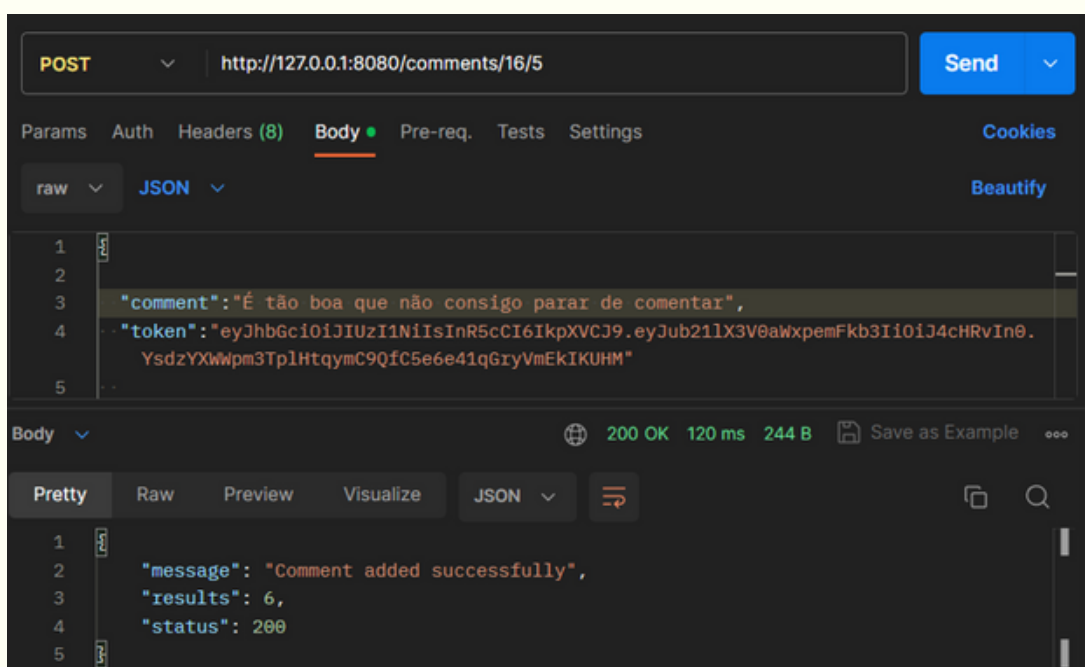
**Método:** POST



The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8080/comments/16`. The request body is a JSON object with a comment and a token. The response is a JSON object with a success message, 5 results, and a 200 status.

```
POST http://127.0.0.1:8080/comments/16
{
  "comment": "Boa música, mais assim",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lX3V0aWxpemFkb3IiOiJ4cHRvIn0.YsdzYXWwpm3TplHtqymC9QfC5e6e41qGryVmEkIKUHM"
}

200 OK 77 ms 244 B
{
  "message": "Comment added successfully",
  "results": 5,
  "status": 200
}
```



The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:8080/comments/16/5`. The request body is a JSON object with a comment and a token. The response is a JSON object with a success message, 6 results, and a 200 status.

```
POST http://127.0.0.1:8080/comments/16/5
{
  "comment": "É tão boa que não consigo parar de comentar",
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21lX3V0aWxpemFkb3IiOiJ4cHRvIn0.YsdzYXWwpm3TplHtqymC9QfC5e6e41qGryVmEkIKUHM"
}

200 OK 120 ms 244 B
{
  "message": "Comment added successfully",
  "results": 6,
  "status": 200
}
```

# Generate a monthly report

**Descrição:** Tem apenas em atenção ao histórico e ao género das músicas e, devolve a última vez e o número de vezes que um dado género foi ouvido. Não tem argumentos.

**URL:** /report/<ano-mes>

**Método:** GET

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:8080/report/2023-06`. The response is a JSON object with a status of 200 OK, 57 ms response time, and 461 B body size. The JSON body contains an array of results and a status field.

```
1  {
2    "results": [
3      {
4        "genre": "pop-rock",
5        "month": "06-2023",
6        "playbacks": 9
7      },
8      {
9        "genre": "rap-pop",
10       "month": "06-2023",
11       "playbacks": 1
12     },
13     {
14       "genre": "rap",
15       "month": "06-2023",
16       "playbacks": 34
17     }
18   ],
19   "status": 200
20 }
```



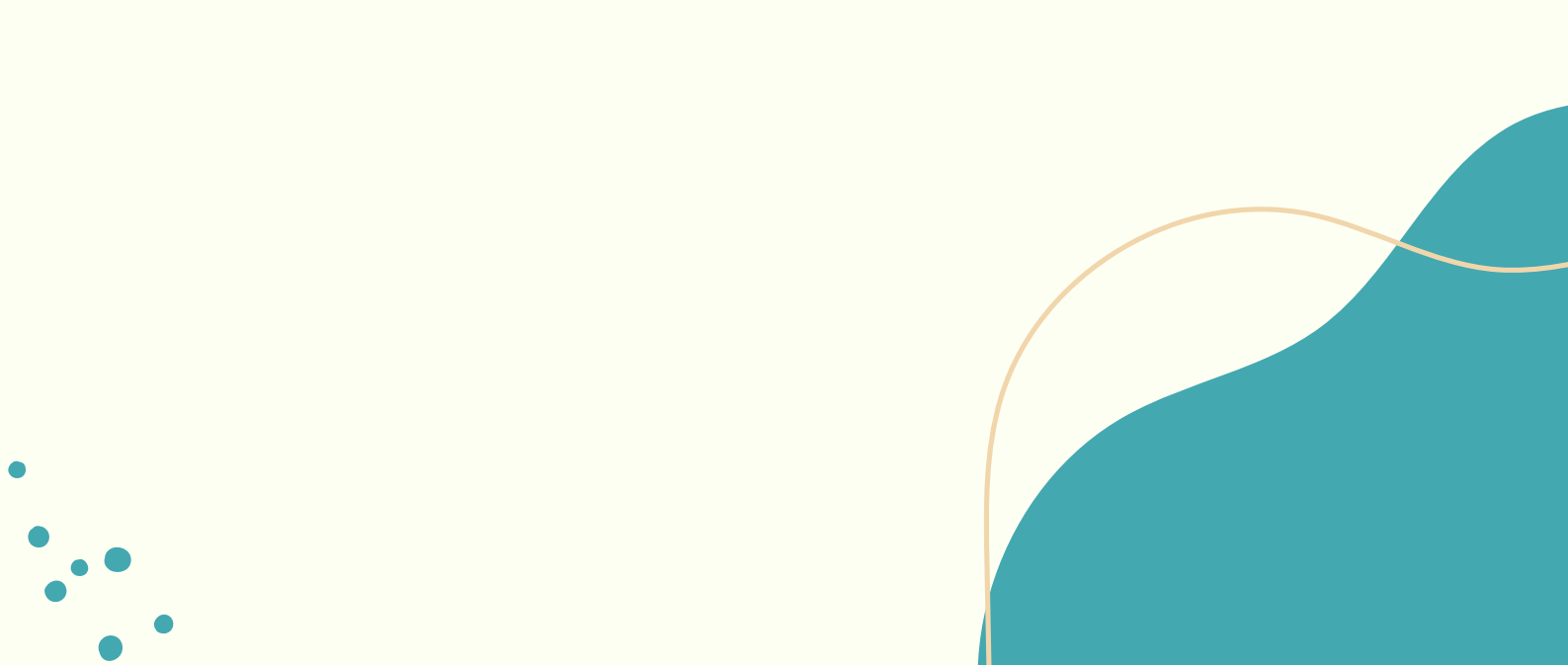


# BASES DE DADOS

## **Additional information**

Descreve algumas informações relevantes, tais como o número de horas de trabalho e o plano do grupo, os diagramas dos modelos, entre outras.

.



## Development plant

Dado que definimos um plano de trabalho na meta intermédia, limitámo-nos a segui-lo, assim, o plano foi este:

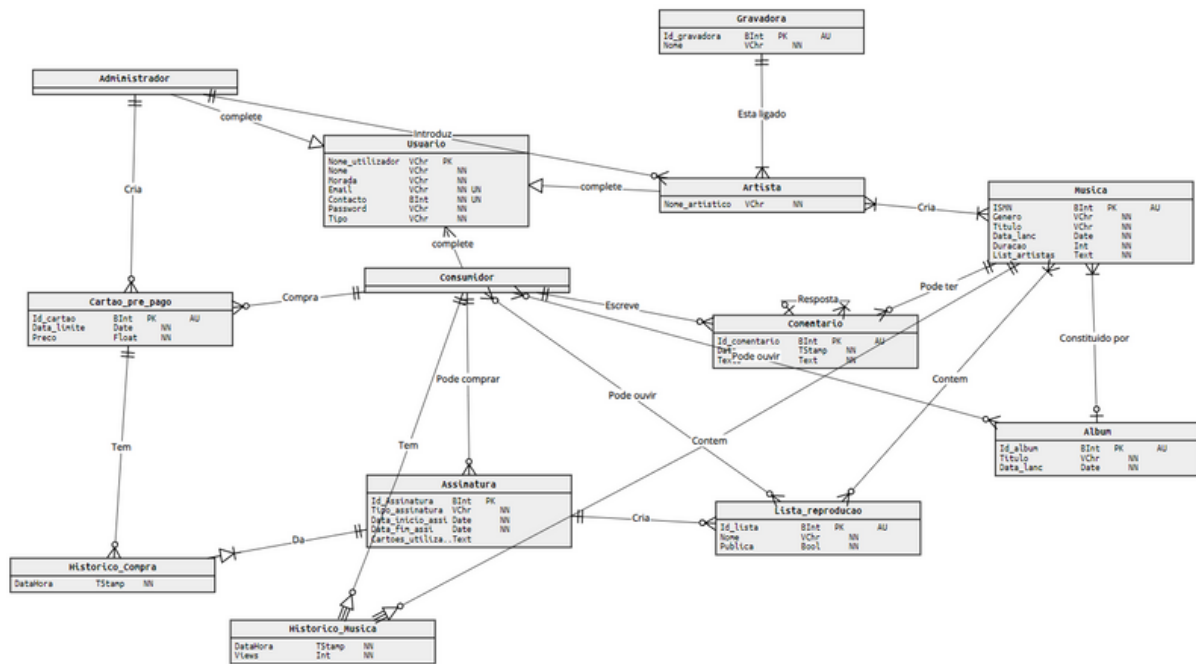
Ana	Fernanda	João
Login/Registo de utilizadores	Adicionar música/ álbum	Pesquisar música
Perfilar um artista	Subscrever um plano	Criar uma "Playlist"
Tocar uma música	Gerar cartões pré-pagos	Deixar comentário/resposta
Realização do relatório	Manual de Utilizador	Gerar relatório mensal

No total, foram despendidas cerca de 75h de trabalho. Semelhante à primeira meta, os vários problemas e objetivos a implementar, bem como os esquemas de organização do projeto foram ativamente discutidos por todos.

## Triggers

Juntamente com o código-fonte, foi entregue um código em linguagem SQL, que contém os vários ‘triggers’ que implementámos neste trabalho. Assim, foram desenvolvidos três ‘triggers’, um que sempre que uma música é inserida, ele é ativado e insere o artista e a música na tabela artista\_música, para que isso fosse feito, foi adicionado à tabela música, uma coluna (list\_artistas) que contém os artistas de uma música, para que o ‘trigger’ fosse facilmente implementado. Outro para inserir o utilizador na respetiva tabela, dado que como o consumidor e o administrador têm os mesmos argumentos, e a forma para distinguir é o tipo, devido à encriptação da palavra-passe, pois se não fosse a encriptação da palavra-passe, o administrador seria inserido diretamente na base de dados. Por fim, o terceiro é ativado sempre que é necessário atualizar o top\_musicas na tabela do consumidor, dado que sempre que se insere um histórico, essa atualização tem de ser feita.

# Diagrama Entidade-Relacionamento



# Diagrama Modelo-Relacional

