

Super Gerenciador Musical

Sumário

Sumário	1
1- Pré-configurações e execução da aplicação	2
1.1 - Pré-configurações para o frontend	2
1.2 - Passos para a execução	2
2 - Detalhando a Jornada do Usuário	3
2.1 - Inicializando a aplicação	3
2.2 - Buscando e Adicionando Músicas	4
2.3 - Buscando Álbum, Playlists Públicas, Artistas e listando suas músicas	6
2.3.1 - Buscando Álbum e listando suas músicas	6
2.3.2 - Buscando Playlists Públicas e listando suas músicas	7
2.3.3 - Buscando Artista e listando suas músicas	8
2.4 - Playlists do usuário	9
2.4.1 - Listando Playlists do Usuário	9
2.4.2 - Criando Playlists	10
2.5 - Parâmetros das músicas em uma playlist	11
3 - Arquitetura e Padrão de Projeto	12
3.1 - Padrão de Projeto	12
3.2 - Arquitetura do sistema	12
4 - Organização do frontend	13
5 - Controllers	14
6 - Models e Services	15
6.1 - Autenticação	15
6.2 - Interface Serviço de modificação de músicas de uma playlist	15
Adicionador de músicas numa playlist	16
Removedor de músicas numa playlist	16
6.3 - Interface Serviço de busca	16
6.4 - Interface Serviço de procura única	18
6.5 - Serviços sem interface	18
6.6 - Classes secundárias	21

1- Pré-configurações e execução da aplicação

Nossa aplicação é dividida em frontend e backend, o nosso backend é implementado utilizando o framework Spring Boot, com a linguagem Java, já o nosso frontend é implementado utilizando a biblioteca React com a linguagem Javascript.

1.1 - Pré-configurações para o frontend

Para executar o frontend, é necessária a utilização do npm, a versão utilizada é a 8.5.5, além disso, também é necessário a utilização do node cuja versão utilizada foi a 16.15.0, recomendamos que ambos sejam instalados em suas versões mais recentes. Com estes passos feitos, dentro da pasta `supergerenciadormusical/front` rode o comando `npm install`.

1.2 - Passos para a execução

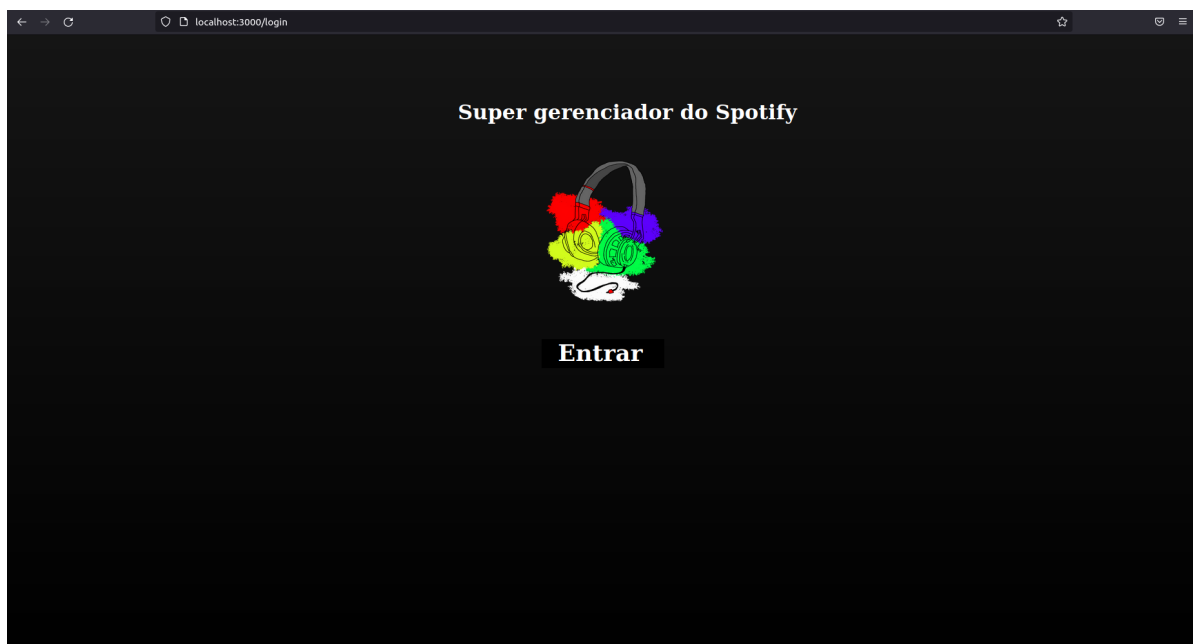
Execute o backend, executando o arquivo `SuperGerenciadorMusicalApplication.java` como Spring Boot App, feito isso, rode o seguinte comando `npm run dev` na pasta `supergerenciadormusical/front` para executar o frontend.

Feito isso, abra o navegador e siga para "<http://localhost:3000/login>". Nessa aba ocorre a autenticação, terminada a autenticação, por fim, abra uma nova aba e vá para <http://localhost:3000/>, onde todos os serviços da aplicação podem ser realizados.

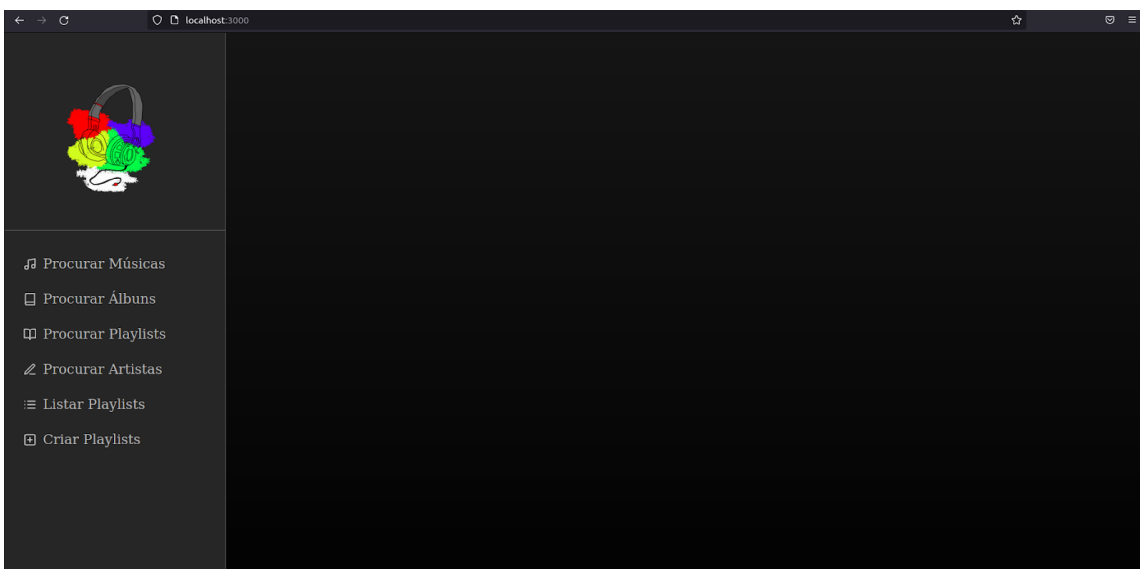
2 - Detalhando a Jornada do Usuário

2.1 - Inicializando a aplicação

Para rodar a aplicação, é necessário seguir até “<http://localhost:3000/login>”, assim como explicitado anteriormente. É importante ressaltar que o projeto do back já deve ter sido executado e estar funcionando.

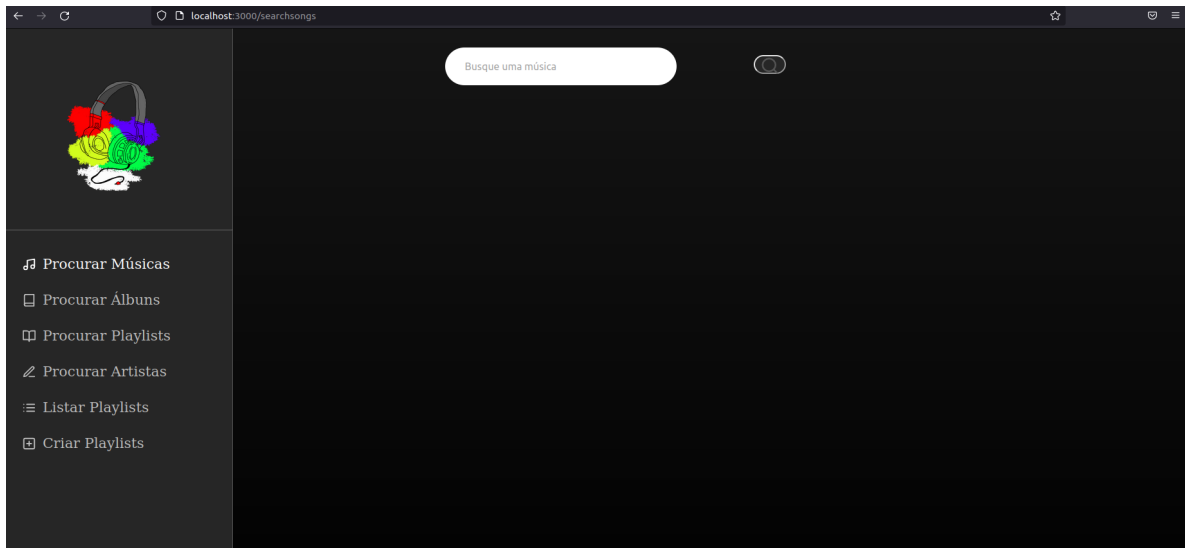


Em seguida, ao clicar em entrar, o usuário é redirecionado à tela do Spotify, e quando ele aceita, é direcionado a uma tela preta. Assim, ele deve abrir uma nova aba e seguir para “<http://localhost:3000/>”, a qual é a home da nossa aplicação.

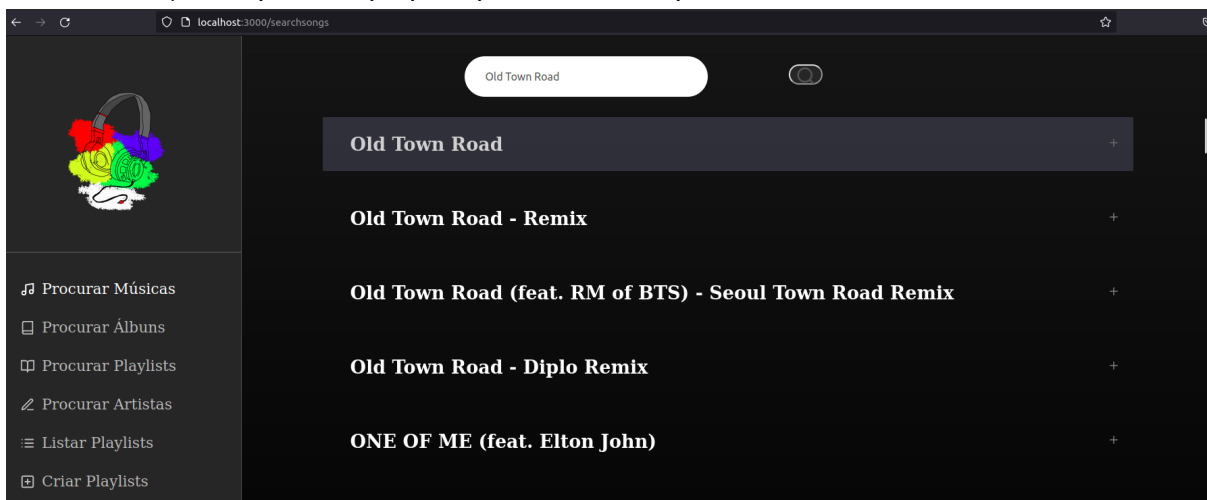


2.2 - Buscando e Adicionando Músicas

Clique em Procurar Músicas.

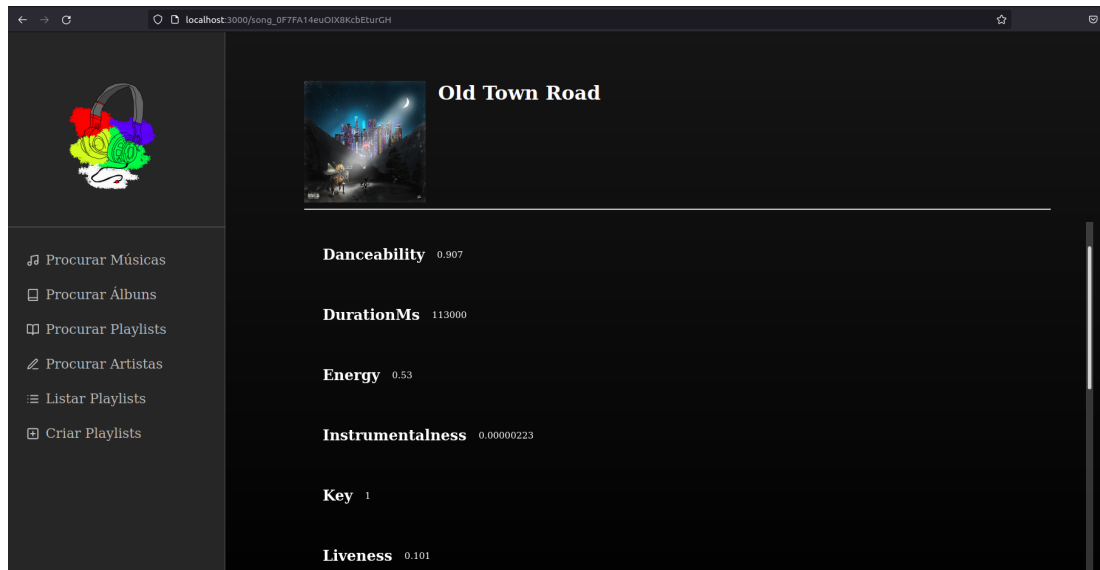


Digite o título da música que deseja procurar (você também pode informar seu artista, álbum, e etc.), e clique na lupa para procurar. Exemplo:

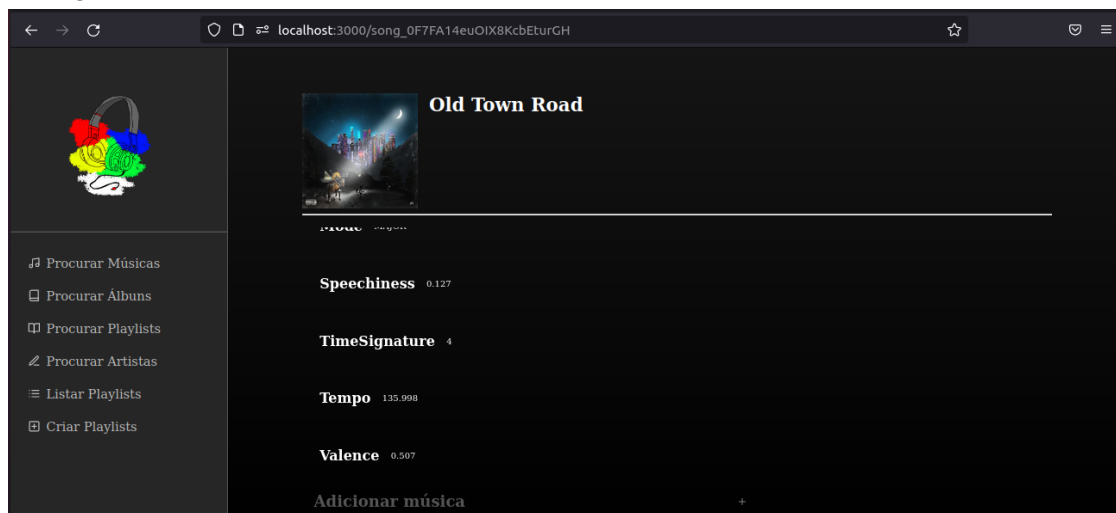


Para cada música, o usuário tem duas opções: clicar no nome e receber os parâmetros da música, ou clicar no ícone na lateral direita e eventualmente selecionar uma playlist para adicionar a música. No final da lista de parâmetros também há uma opção de adicionar a música em uma playlist.

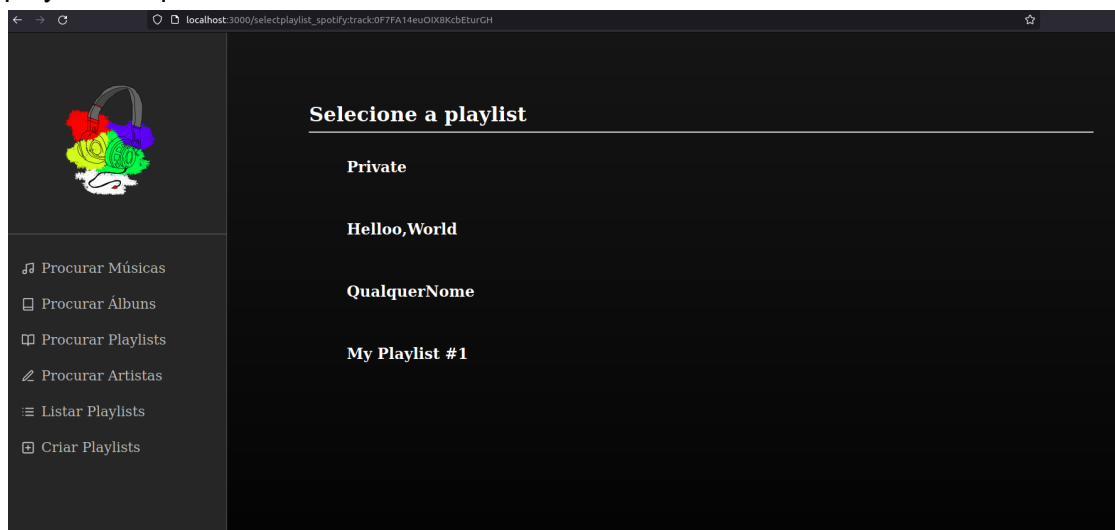
Por exemplo, clicando no nome da música:



Navegando para o fim da lista de parâmetros:



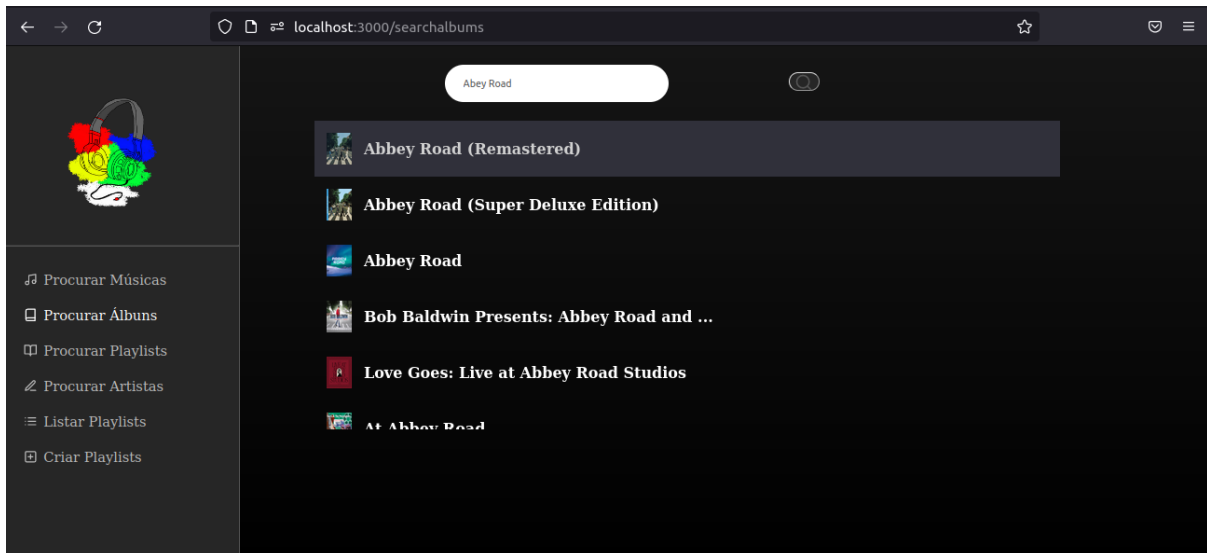
Tanto ao clicar em “Adicionar música”, quanto ao clicar no ícone à direita em uma música buscada, levarão o usuário a tela onde é mostrado suas playlists, para ele selecionar a playlist em que a música será adicionada.



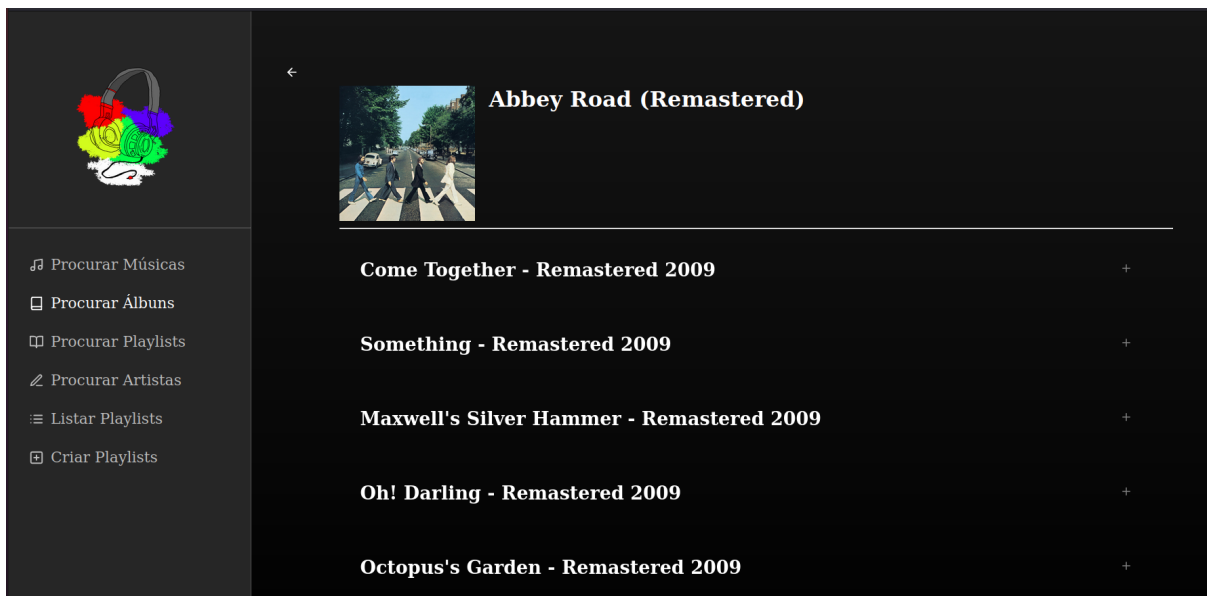
2.3 - Buscando Álbum, Playlists Públicas, Artistas e listando suas músicas

2.3.1 - Buscando Álbum e listando suas músicas

O procedimento de busca é análogo, clique em “Procurar Álbuns”, pesquise um título de álbum e clique na lupa para procurar, será listado os resultados da pesquisa, e selecione o álbum desejado clicando em seu nome.



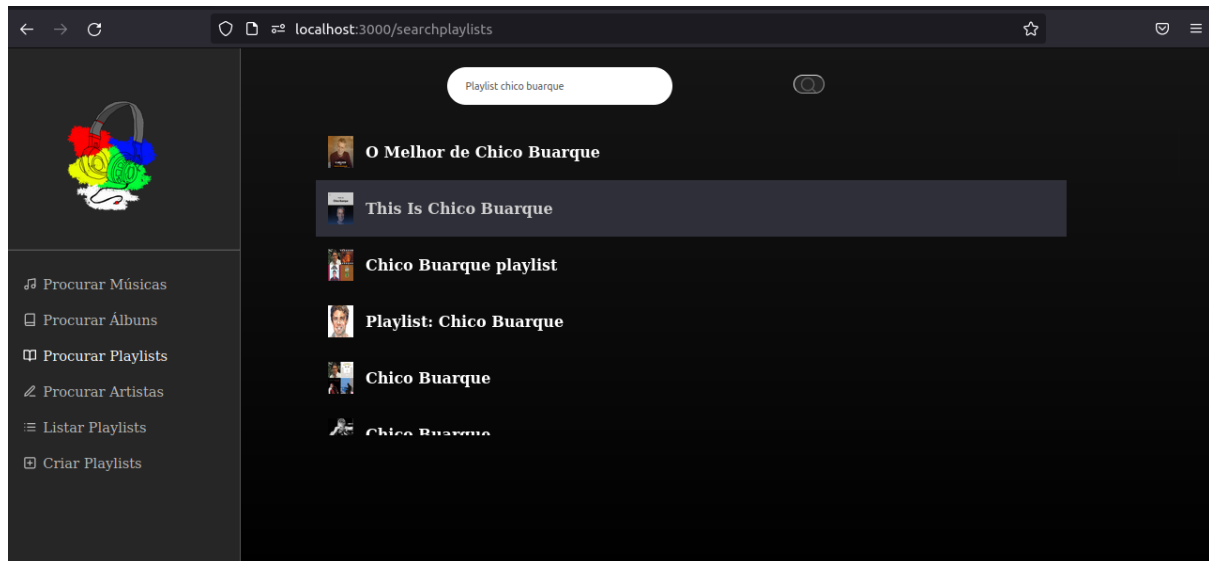
Após isso, suas músicas serão listadas:



Você pode adicionar essas músicas em uma de suas playlists, fazendo o procedimento já explicado em 2.2.

2.3.2 - Buscando Playlists Públicas e listando suas músicas

O procedimento de busca é análogo, clique em “Procurar Playlists”, pesquise um título de playlist pública e clique na lupa para procurar, será listado os resultados da pesquisa, e selecione a playlist desejada clicando em seu nome



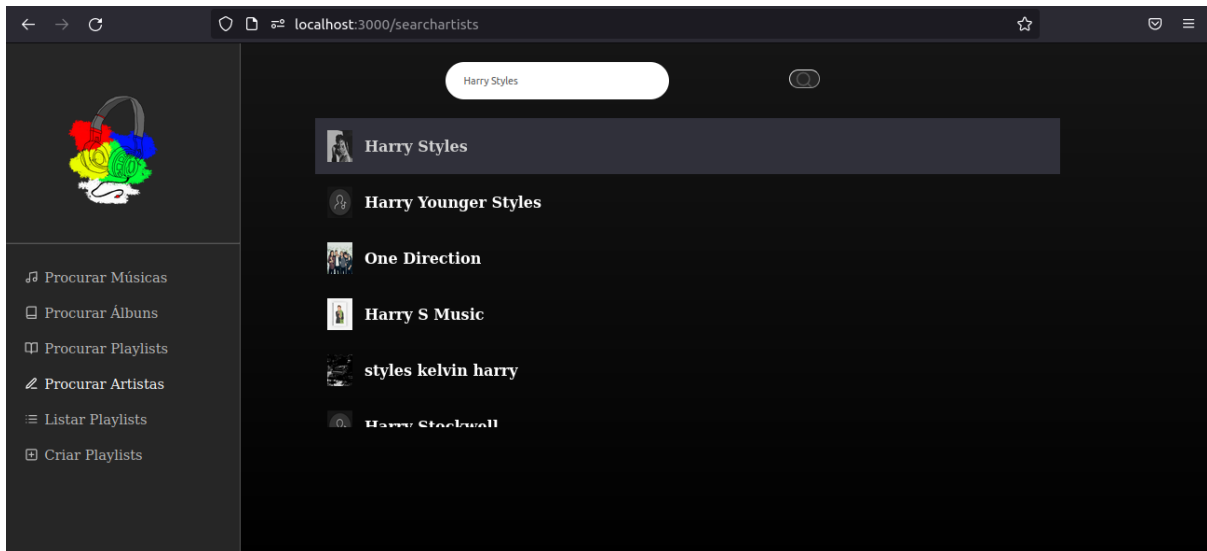
Após isso, suas músicas serão listadas



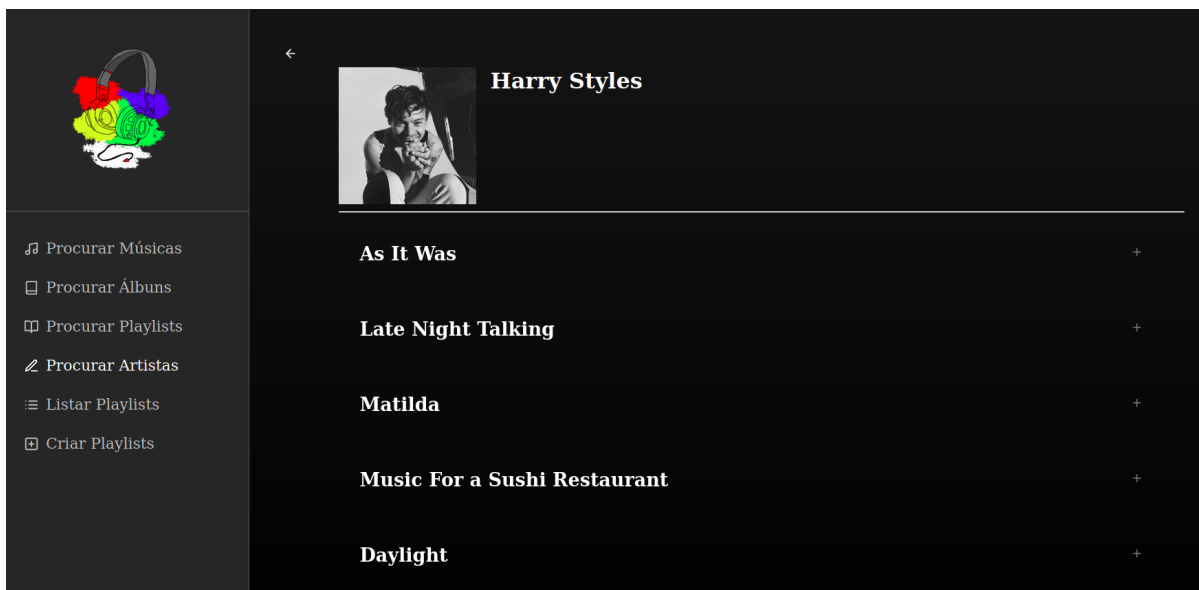
Você pode adicionar essas músicas em uma de suas playlists, fazendo o procedimento já explicado em 2.2.

2.3.3 - Buscando Artista e listando suas músicas

O procedimento de busca é análogo, clique em “Procurar Artista”, pesquise um nome de artista e clique na lupa para procurar, será listado os resultados da pesquisa, e selecione o artista desejado clicando em seu nome



Após isso, suas músicas serão listadas

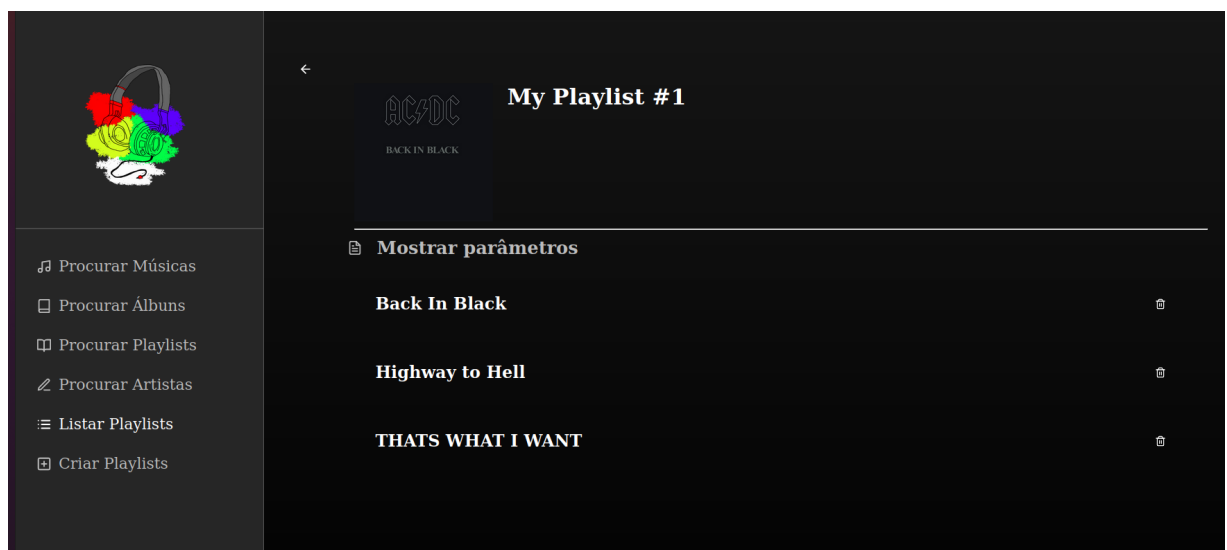
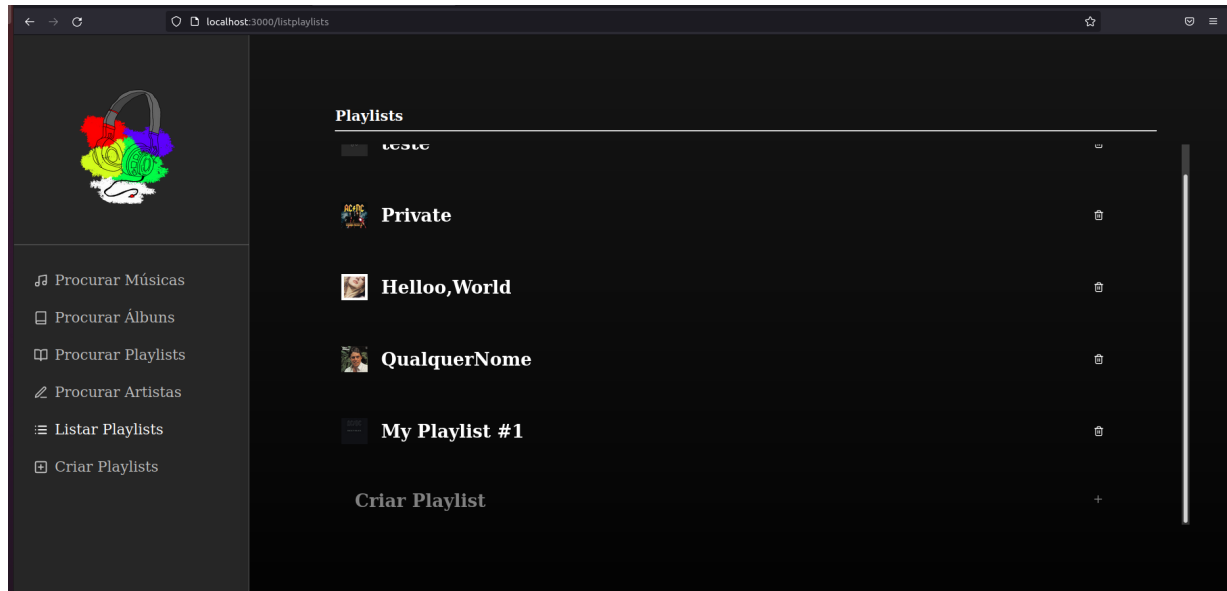


Você pode adicionar essas músicas em uma de suas playlists, fazendo o procedimento já explicado em 2.2.

2.4 - Playlists do usuário

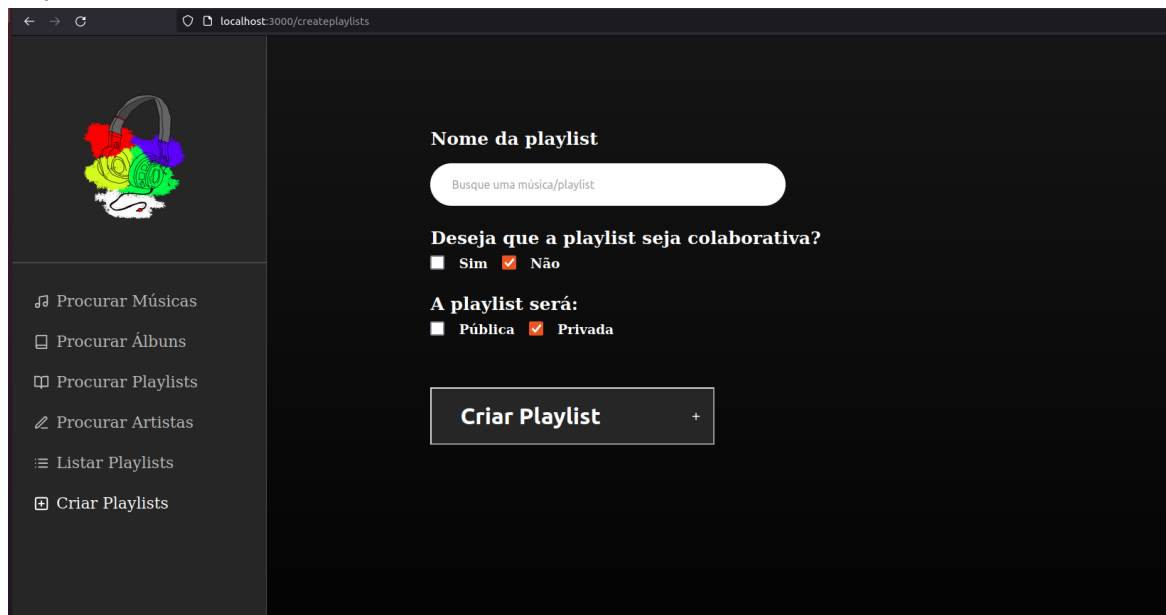
2.4.1 - Listando Playlists do Usuário

Ao clicar em “Listar Playlists”, é fornecido uma listagem com as playlists criadas pelo usuário; ele pode selecionar uma playlist para visualizar (clcando no nome) ou para remover (clcando no ícone na lateral direita); o usuário pode também clicar na opção de criar uma playlist e será direcionado para a tela de criação de playlists.

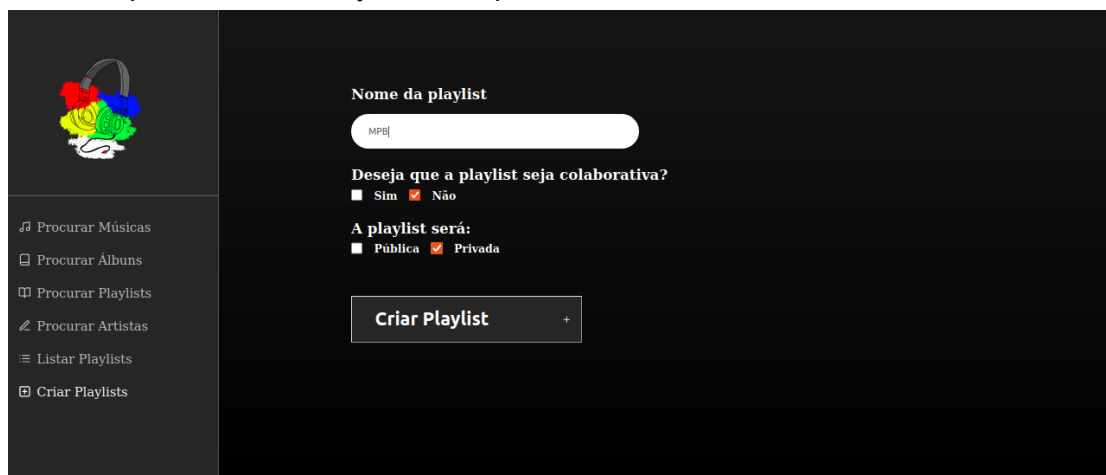


2.4.2 - Criando Playlists

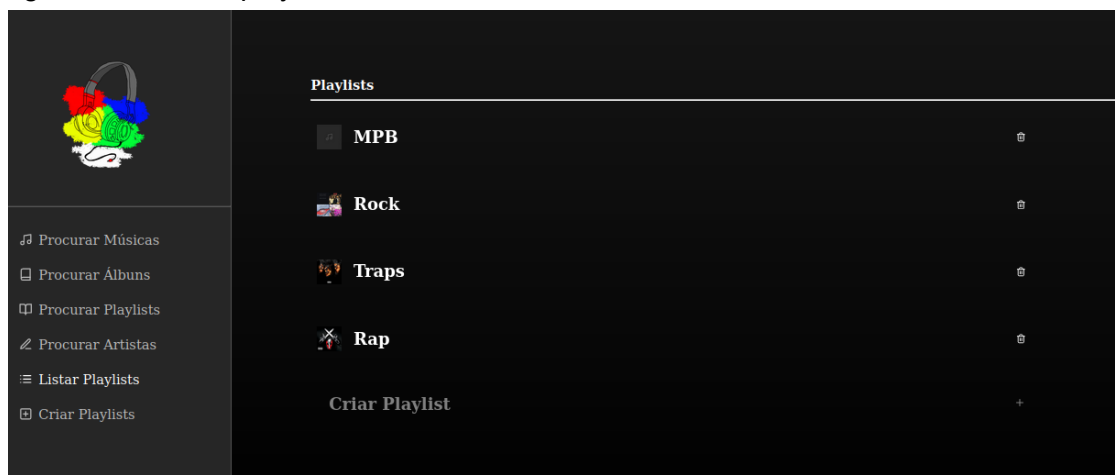
A criação de uma playlist ocorre com a passagem de três parâmetros: uma string para o nome e dois booleanos que indicam se ela será colaborativa ou não, pública ou privada. A playlist criada não pode ser pública e colaborativa simultaneamente.



Por exemplo, criando a Playlist MPB, privada e não colaborativa:

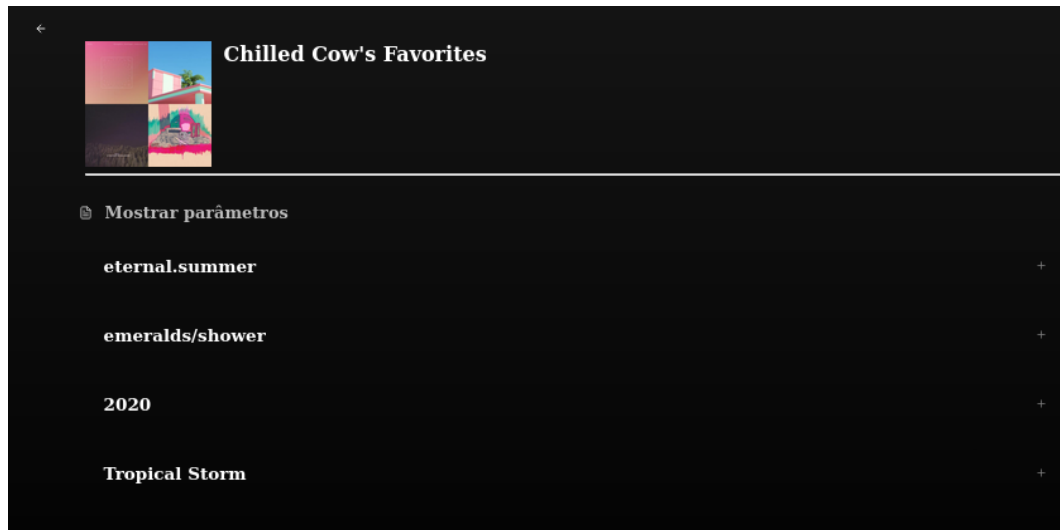


Agora, essa nova playlist foi criada:



2.5 - Parâmetros das músicas em uma playlist

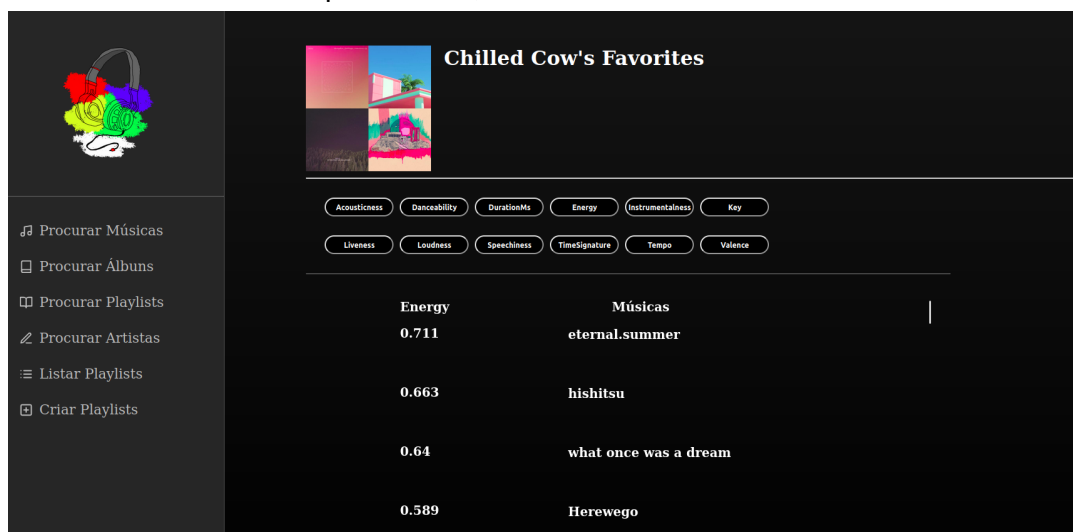
Para uma playlist pública ou privada, há a opção “Mostrar parâmetros”:



Ao clicar nela, mostram-se os parâmetros ao quais podemos ordenar as músicas dessa playlist:



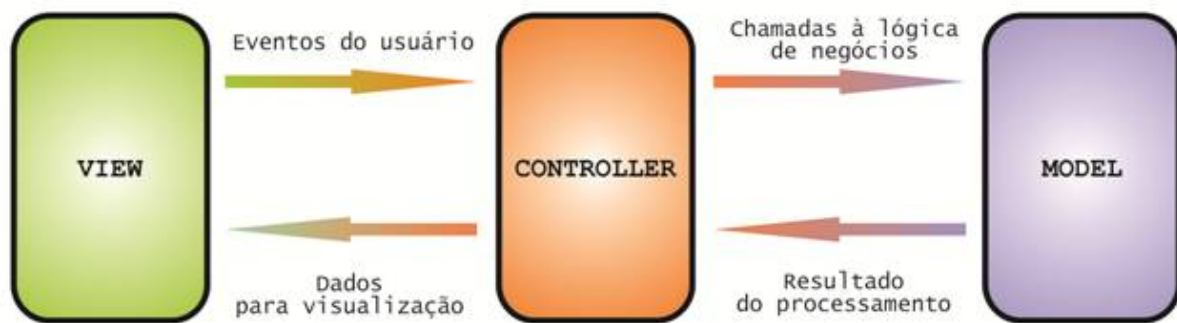
Clicando em um desses parâmetros, as músicas serão ordenadas:



3 - Arquitetura e Padrão de Projeto

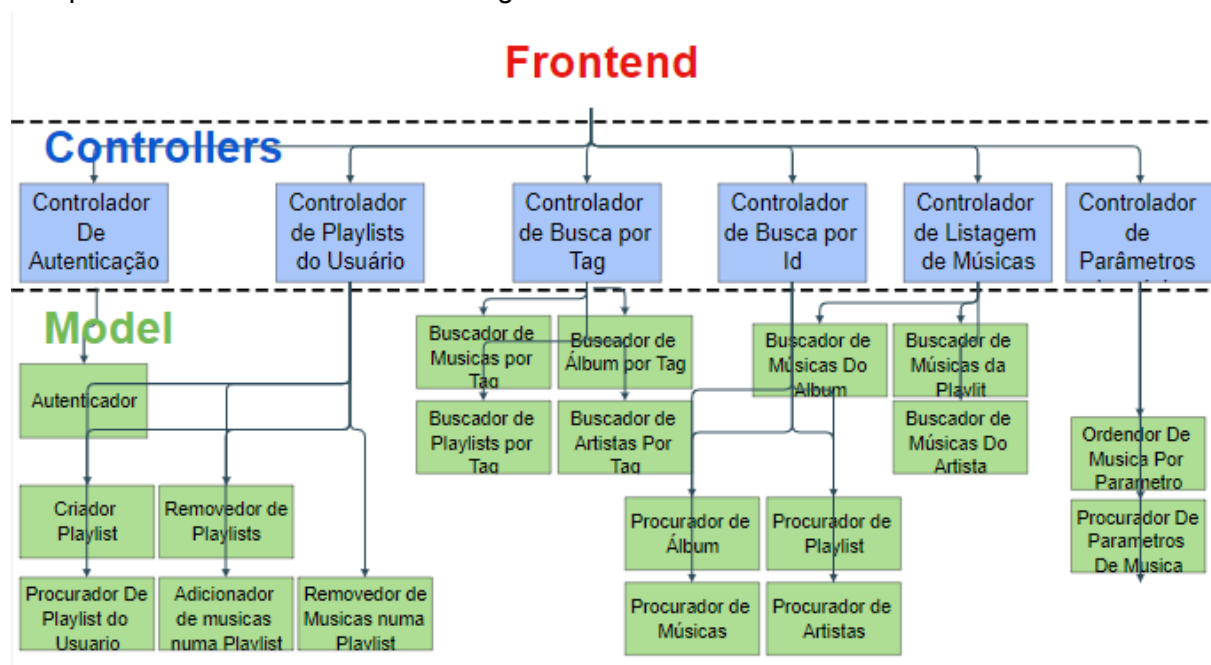
3.1 - Padrão de Projeto

Para este sistema seguimos o padrão MVC, no qual possui camada de frontend (interface com o usuário), controllers (realizam a intermediação do frontend com o backend) e o models onde se encontra a lógica de negócios do sistema e os services que realizam a comunicação com os dados da API do spotify.



3.2 - Arquitetura do sistema

A arquitetura do nosso sistema é a seguinte:



4 - Organização do frontend

O frontend se encontra na pasta **supergerenciadormusical/front/** sendo dividido em:

App:

A pasta App apresenta uma tela que passa suas propriedades para cada child; representa a barra lateral da aplicação, na qual está o logo e o link para as principais telas, e o background escuro no qual cada informação é trazida em cada tela.

Components:

A pasta components apresenta componentes que são reutilizados em múltiplas telas:

- GoBack: um elemento com um ícone de uma seta que permite o retorno para a tela anterior na navegação;
- ElementHeader: representa a estrutura de uma apresentação de um elemento qualquer, baseando-se especialmente em imagem e nome;
- Parameter e ParameterButton: fornecem o nome dos parâmetros que uma música apresenta (e o valor para o caso do primeiro componente);
- SearchData: componente que implementa a busca de um elemento baseando-se em um critério de busca;
- A pasta table apresenta componentes que são utilizados para formar a tabela de listagem das músicas de uma playlist de acordo com os parâmetros.

Images:

Apresenta a Logo utilizada e imagens substitutivas de imagens de playlists e artistas, utilizadas quando necessário.

Pages:

A pasta pages apresenta as páginas existentes no front.

Styles:

Apresenta algumas características de estilização; em especial, as cores utilizadas.

5 - Controllers

Os controllers, como já foi dito anteriormente, são responsáveis pela integração do frontend com o backend, essa integração ocorre por meio de requisições http enviadas pelo frontend (origins = "http://localhost:3000/") e recebidas pelo backend http://localhost:8080/. Cada Rest Controller, fica situado em uma rota, por exemplo, o ControladorDePlaylistsDoUsuário recebe as requisições enviadas na rota: http://localhost:8080/playlists.

Controlador De Autenticação

Responsável pela intermediação do frontend com o Autenticador do model.

Rota: "/autenticacao"

Do ponto de vista do front-end, a autenticação consiste em duas etapas:

1- requisição do tipo GET na rota "/autenticacao/iniciar"

2- redirecionar o usuário para a uri retornada pela requisição

Todo o resto, que refere-se à parte burocrática da conexão com a API do Spotify, está implementado pelo Autenticador.

Controlador De Playlists Do Usuário

Responsável pela intermediação do frontend com os serviços de criação de playlist, listagem playlist, remoção de playlist, adição de músicas numa playlist e remoção de músicas numa playlist.

Rota: "/playlists"

Controlador De Busca Por Tag

Responsável pela intermediação do frontend com os serviços de busca de músicas, de álbuns, playlists públicas e de artistas.

Rota: "/buscar-por-tag"

Controlador De Busca Por Id

Responsável pela intermediação do frontend com os serviços de obtenção de músicas, de álbuns, playlists públicas e de artistas pelo seu ID.

Controlador De Listagem De Músicas

Responsável pela intermediação do frontend com os serviços de listagem de músicas em um álbum, playlist e artista.

Rota: "/listar-musicas"

Controlador De Parâmetros De Música

Responsável pela intermediação do frontend com os serviços de obter os parâmetros de músicas pelos seus ids, e de ordenar músicas por um determinado parâmetro.

Rota: "/parametros"

6 - Models e Services

Primeiramente, irei listar as classes abstratas e interfaces do pacote - junto com as respectivas classes que as implementam - elas têm como objetivo uniformizar as funções das classes concretas, de forma que outras partes do código possam utilizar o polimorfismo genérico para se referenciar a essas interfaces e classes abstratas ao invés de se referenciar as classes concretas, assim dando uma maior flexibilidade para o nosso código, mediante a isso muitas funções devolvem objetos da classe `AbstractModelObject`, mas antes desses objetos serem enviados ao front eles sofreram um cast para uma subclasse dessa classe.

Classe abstrata - Serviços do aplicativo

Essa classe abstrata abrange todas as classes que realizam algum tipo de serviço no nosso projeto.

Classe abstrata - Serviço Spotify

Herda de: Serviço do aplicativo

Essa classe abstrata abrange de todos os serviços do nosso gerenciador musical que possuem como atributo um objeto do tipo `spotifyApi` (basicamente, um gerador de urls de requisição à api, definido pela biblioteca citada anteriormente), ou seja, todos os serviços que realizam requisições diretamente no banco de dados do Spotify.

6.1 - Autenticação

Autenticador

Classe responsável pela autenticação e autorização do uso da API do Spotify do usuário, implementada com o padrão de projeto Singleton, ela permite que outras classes possam usufruir do token de acesso do usuário para que elas possam realizar requisições na API e também permite a atualização dos tokens, assim possibilitando que outros usuários usem a aplicação.

6.2 - Interface Serviço de modificação de músicas de uma playlist

Método: `public AbstractModelObject executaServiço(String playlistID, String uris [])`

Essa interface é responsável por modificar as músicas, referenciadas por seus uris, de uma playlist - especificada pelo `playlistID` - ou seja, remover ou acrescentar músicas nela, as funções que a implementam retornam o `AbstractModelObject` retornado pela api durante a requisição. As exceções sobre a privacidade da playlist (ser pública ou não) são

administradas pela API. Os serviços dessa interface são performados ao se realizar uma requisição do tipo POST na rota https://api.spotify.com/v1/playlists/{playlist_id}/tracks.

Adicionador de músicas numa playlist

Herda de: Serviço Spotify

Implementa: Serviço de modificação de músicas de uma playlist

Método: veja sua interface.

Essa classe implementa o ato de adicionar músicas de uma playlist por meio da sua interface, retorna os snapshot id da playlist modificada.

Removedor de músicas numa playlist

Herda de: Serviço Spotify

Implementa: Serviço de modificação de músicas de uma playlist

Método: veja sua interface.

Essa classe implementa o ato de remover músicas de uma playlist por meio da sua interface, retorna os snapshot id da playlist modificada.

6.3 - Interface Serviço de busca

Método: public AbstractModelObject executaServiço(String tagDeProcura, int offset)

Essa interface realiza a busca de algum tipo de estrutura de dados da API, como álbuns, músicas e playlists, de acordo com certos parâmetros(a tagDeProcura e o offset) e retorna um Paging<> de objetos que satisfazem a tagDeProcura dada. A tagDeProcura é uma string dada pelo usuário que servirá como base para o algoritmo de busca de dados da API. O offset deve ser incrementado de 50 em 50 a cada vez que o usuário faz uma requisição consecutiva usando a mesma tagDeProcura, para assim obter novos resultados, seu valor máximo é 1000. Nessa interface realizamos uma requisição do tipo GET na rota <https://api.spotify.com/v1/search>.

Buscador de álbuns

Herda de: Serviço Spotify

Implementa: Serviço de busca

Método: veja sua interface

Essa classe implementa o ato de buscar álbuns por meio de sua interface, nesse caso a tagDeProcura é o nome do álbum buscado.

Buscador de artistas

Herda de: Serviço Spotify

Implementa: Serviço de busca

Método: veja sua interface

Essa classe implementa o ato de buscar artistas por meio de sua interface, nesse caso a tagDeProcura é o nome do artista buscado.

Buscador de músicas da playlist

Herda de: Serviço Spotify

Implementa: Serviço de busca

Método: veja sua interface

Essa classe implementa o ato de buscar as músicas de uma playlist por meio de sua interface, nesse caso a tagDeProcura é o nome da playlist.

Buscador de músicas do álbum

Herda de: Serviço Spotify

Implementa: Serviço de busca

Método: veja sua interface

Essa classe implementa o ato de buscar as músicas de um álbum por meio de sua interface, nesse caso a tagDeProcura é o nome do álbum.

Buscador de músicas por tag

Herda de: Serviço Spotify

Implementa: Serviço de busca

Método: veja sua interface

Essa classe implementa o ato de buscar as músicas do Spotify por meio de sua interface, nesse caso a tagDeProcura é o nome da música buscada.

Buscador de playlists públicas

Herda de: Serviço Spotify

Implementa: Serviço de busca

Método: veja sua interface

Essa classe implementa o ato de buscar as playlists públicas por meio de sua interface, nesse caso a tagDeProcura é o nome da playlist buscada.

6.4 - Interface Serviço de procura única

Método: public AbstractModelObject executaServiço(String id)

A partir do id de um objeto do banco de dados, ele retorna o objeto em si, podendo ser um artista, um álbum, uma playlist, etc. Para isso, realiza-se uma requisição do tipo GET, a rota pode variar dependendo do objeto a ser obtido.

Procurador De Álbum

Herda de: Serviço Spotify

Implementa: Serviço de procura única

Método: veja sua interface

Implementa a sua interface na busca de um álbum, realizando uma requisição do tipo GET na rota <https://api.spotify.com/v1/albums/id>.

Procurador De Artista

Herda de: Serviço Spotify

Implementa: Serviço de procura única

Método: veja sua interface

Implementa a sua interface na busca de um artista, realizando uma requisição do tipo GET na rota <https://api.spotify.com/v1/artists/id>.

Procurador De Playlist

Herda de: Serviço Spotify

Implementa: Serviço de procura única

Método: veja sua interface

Implementa a sua interface na busca de uma playlist, realizando uma requisição do tipo GET na rota https://api.spotify.com/v1/playlists/playlist_id.

6.5 - Serviços sem interface

As próximas classes concretas não implementam nenhuma interface até o momento, pois devido a grande diversidade de requisições para a API que podemos fazer, as seguintes requisições não tem um padrão muito parecido de argumentos e retorno da API, dessa forma elas não implementam nenhuma interface por enquanto, mas com o avanço do projeto isso pode mudar.

Criador de playlist

Herda de: Serviço Spotify

Método: public AbstractModelObject executaServiço(String userID, String nome_da_playlist, boolean serColaborativa, boolean serPública, String descrição)

A partir do userID, cria uma playlist vazia desse usuário, com nome dado por nome_da_playlist e suas características(ser colaborativa, ser pública e sua descrição) são dadas pelos outros argumentos da função, dados pelo usuário. Essa classe realiza uma requisição do tipo POST na rota https://api.spotify.com/v1/users/{user_id}/playlists e retorna a playlist criada.

Procurador de músicas

Herda de: Serviço Spotify

Método: public Track[] executaServiço(String [] ids)

Essa classe obtêm as músicas referenciadas pelos seus ids no array ids. Para isso, realiza-se uma requisição do tipo GET na rota <https://api.spotify.com/v1/tracks>.

Procurador de top músicas do artista

Herda de: Serviço Spotify

Método: public Track[] executaServiço(String id)

Essa classe obtêm as músicas mais tocadas de um artista - referenciado pelo seu id. Para isso, realiza-se uma requisição do tipo GET na rota <https://api.spotify.com/v1/artists/id/top-tracks>.

Procurador de parâmetros de músicas

Herda de: Serviço Spotify

Método: public AudioFeatures[] executaServiço(String [] ids_das_músicas)

Essa classe obtêm os parâmetros de cada uma das músicas referenciadas pelos seus ids no array ids_das_músicas. Para isso, realiza-se uma requisição do tipo GET na rota <https://api.spotify.com/v1/audio-features>.

Procurador de playlists do usuário atual

Herda de: Serviço Spotify

Método: public AbstractModelObject executaServiço(int offset)

Essa classe retorna as playlists de um usuário, contudo a cada busca será devolvido no máximo 50 playlists. Para permitir o acesso a mais playlists, usamos o argumento offset, na

qual o offset deve ser incrementado de 50 em 50 a cada vez que o usuário faz uma requisição consecutiva desse tipo, para assim obter novos resultados, seu valor máximo é 1000. Esse serviço é executado através de uma requisição do tipo GET na rota <https://api.spotify.com/v1/me/playlists>.

Removedor de playlists

Herda de: Serviço Spotify

Método: public int executaServiço(String userID, String playlistID)

A partir do id do usuário, ele remove a playlist desse usuário que contenha o determinado playlistID, a função dessa classe retorna 1 caso haja sucesso na remoção e 0 caso contrário. Essa classe realiza uma requisição do tipo DELETE na rota https://api.spotify.com/v1/playlists/playlist_id/followers.

Usuário Atual

Herda de: Serviço Spotify

Método: public User executaServiço()

Essa classe tem como objetivo obter os dados do usuário atualmente logado para isso ela gera uma requisição do tipo GET na rota <https://api.spotify.com/v1/me> e obtém como resultado dessa requisição esses dados e os retorna por meio do objeto User.

Ordenador de músicas por parâmetro

Herda de: Serviço do aplicativo

Método: public <T extends Comparable<T>> String[] ordenaMúsicas(T[] parâmetro, String[] ids)

Dada dois vetores, um de parâmetros- obtido pelo AudioFeatures da música - e outro de ids, onde um mesmo índice dos dois vetores se referencia a mesma música, ordeno o vetor parâmetro de ordem decrescente e o vetor de ids é ordenado juntamente com ele, para que um mesmo índice dos dois vetores se remeta a mesma música, por fim devolvo o vetor de ids ordenado.

Método: private <T extends Comparable<T>> void quickSort(T[] parâmetro, int ini, int fim, String [] ids)

Ordeno o vetor de Comparable , e consequentemente o vetor de ids também, utilizando o algoritmo do quicksort.

Método: private <T extends Comparable<T>> int separaSedgewick(T[] parâmetro, int ini, int fim, String[] ids)

Utilizo o método proposto por Sedgewick para escolher o próximo pivô do QuickSort.

6.6 - Classes secundárias

As próximas classes são classes auxiliares, que não realizam requisições na API e nem prestam serviços diretamente ao usuário, mas são importantes para a realização de tarefas intermediárias e tratativas parciais dos dados para outros serviços.

Gerador de json

Método: `public JSONArray urisParaJsonArray(String [] array)`

Transforma um array de uris de músicas em um jsonArray e o devolve, essa classe é utilizada para remover músicas de uma playlist.

Trocador de elementos

Método: `public <T extends Comparable<T>> void trocar(T[] vetor, int i, int j)`

Troca os elementos `T[i]` e `T[j]` de lugar no vetor dado, sendo este um vetor de objetos que implementam a interface `Comparable<T>`.