Data Song: uma aplicação para análise de dados de músicas integrada a API do Spotify

João Victor L. Amorim, André C. da Silva

Grupo de Pesquisa Mobilidade e Novas Tecnologias de Interação
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)
Campus Hortolândia – SP – Brasil

lopes.amorim@aluno.ifsp.edu.br, andre.constantino@ifsp.edu.br

Abstract. *This project consists of developing a web system that will build graphs and present them on a dashboard for the user. Justified by the lack of a simple method to analyze trends in the music industry using some user interface, the project seeks to provide insights for more accurate analyses on the subject of music and to transform this data into valuable information that will support more accurate studies or research. An incremental approach was used to develop the software, considering three graphs and a dashboard. As a result, a system was developed capable of obtaining music data from Spotify and using it to build graphs in an interface for the user to view.*

Resumo. *Este projeto consiste no desenvolvimento de um sistema web que constrói gráficos e os apresenta em um dashboard para o usuário. Justificado pela falta de um método simples para analisar as tendências no ramo da música utilizando alguma interface de usuário, o projeto busca disponibilizar insights para a realização de análises mais precisas sobre o tema música e para transformar esses dados em informações de valor que servirão de suporte para estudos ou pesquisas mais precisas. Foi utilizado uma abordagem incremental para o desenvolvimento do projeto, considerando três gráficos e um dashboard. Como resultado, foi desenvolvido um sistema capaz de obter os dados de músicas do Spotify e utilizá-los para a construção de gráficos em uma interface para o usuário visualizar.*

1. Introdução

A música é uma forma de arte e entretenimento que está presente no cotidiano da maioria das pessoas, podendo abranger aspectos culturais e históricos de um lugar, questões emocionais e formas de pensar ou de se expressar, e até mesmo como forma de conexão social e interação entre as pessoas. Essa forma de mídia é consumida todos os dias e, de acordo com o Relatório Global de Música da IFPI (2019), "o Brasil é o país responsável por 15,4% da receita gerada com música no continente."

Atualmente, a tecnologia está muito presente no compartilhamento de músicas, e está aparecendo na forma de aplicativos, como Spotify, Deezer, Youtube Music, entre outros. A relação entre os aplicativos citados acima é a utilização de algoritmos, que podem ser usados, por exemplo, no desenvolvimento de recomendações baseadas nas músicas escutadas por seus usuários, na criação de *playlists* relacionadas a gêneros musicais específicos, dentre outras.

Apesar de cada um desses *softwares* possuírem suas tecnologias e algoritmos para montagem de *playlists* e recomendações individuais para cada usuário, não há como obter uma aparente análise de como são feitas as recomendações a eles. Além disso, por mais que seja possível visualizar em números a popularidade de certos artistas e músicas nas aplicações

existentes, não há um método simples para analisar as tendências no ramo da música utilizando alguma interface que possa filtrar o que deve ser apresentado em relação às músicas. Para pessoas que desejam estudar, analisar dados ou descobrir mais artistas de um determinado gênero, por exemplo, se torna uma tarefa árdua, já que é preciso se limitar a recomendações feitas por esses aplicativos.

Como exemplo de trabalho atual relacionado ao uso de sistemas para explorar os resultados da música, pode ser citado o trabalho de Rea *et al*. (2021) que apresenta uma contribuição para a área de análise musical destacando algoritmos de aprendizagem de máquina na compreensão e previsão da popularidade musical, apresentando uma ligação direta com este trabalho.

Neste trabalho, será documentado o desenvolvimento de um sistema web que irá construir gráficos que disponibilizarão *insights* para a realização de análises mais precisas sobre o tema música e para a transformação desses dados em informações de valor que podem servir de suporte para estudos ou pesquisas mais precisas. Para o sucesso do desenvolvimento, será necessário o uso do conceito de integração de sistemas, pois, nos dias de hoje, um sistema que não possui integração externa vive isolado, prejudicando o próprio sistema (Silveira *et al*., 2012). Será feita então a utilização da API da plataforma Spotify para fazer requisições de músicas, artistas, álbuns e outros dados a fim de desenvolver informações sobre o tema música.

Na fundamentação teórica (Seção 2), serão citados os conceitos estudados e fundamentos utilizados durante o desenvolvimento deste trabalho. Os trabalhos correlatos estão disponíveis na Seção 3. Na Seção 4 é brevemente apresentada as principais ações tomadas durante o desenvolvimento do projeto, como a criação dos gráficos para o usuário. A Seção 5 ficará responsável por apresentar as etapas seguidas e decisões tomadas durante o planejamento do sistema e a construção da interface e da lógica de negócio. Na conclusão do trabalho (Seção 6), será resgatado o objetivo do projeto e serão descritos os resultados obtidos ao final dele.

2. Fundamentação Teórica

2.1. Sistemas web

Um sistema web é um *software* que está hospedado na *Internet* (Cunha, 2022). Os sistemas web podem trazer várias vantagens para uma empresa e seus usuários, como:

- Integração: Os sistemas web possuem facilidade para fazer integração com outros sistemas;
- Flexibilidade de acesso: Como esse sistema fica armazenado em servidores e não em máquinas locais, ele pode ser acessado a partir de qualquer aparelho que tenha um navegador;
- Alta personalização: É possível criar um sistema web que se adeque a qualquer tipo de objetivo que uma empresa possua.

2.2. Integração entre sistemas

É a conexão entre sistemas para que eles trabalhem em conjunto e de forma automatizada (SYDLE, 2024). É a unificação de *softwares* diferentes, como *softwares* para gerir pagamentos, recuperar dados de um produto específico, processos etc. Isso ajuda na

comunicação entre sistemas, deixando mais prático e eficiente, podendo reduzir gastos financeiros e aumentar a produtividade de uma empresa.

A integração entre sistemas pode ser usada:

1. Em casos de demanda por transformação digital;
2. Quando quiser aumentar a produtividade;
3. Quando quiser aumentar a consistência de dados entre sistemas;
4. Em necessidade de consumo de dados externos.

Tipos de integração de sistemas:

1. Troca de dados eletrônicos;
2. Banco de dados;
3. API's.

Este trabalho irá abordar a integração por APIs, definidas na próxima Subseção.

2.3. API

Podemos definir uma API como uma forma de comunicação com o *software* que procura abstrair totalmente os detalhes da implementação subjacente (Alexandre, 2021). Uma API (*Application Programming Interface*) é um mecanismo que irá permitir que um sistema se comunique com um ou mais sistemas diferentes, escondendo a complexidade por trás, utilizando protocolos para a criação de *softwares*.

Fazemos uma requisição à API usando métodos como *GET*, *POST*, *PUT* e *DELETE* e ela nos traz a resposta desejada. As respostas da API vêm com um código de *status* que nos disponibiliza alguma mensagem adicional, como o de solicitação bem-sucedida (código 200) e não encontrado (código 404).

Além do código de *status*, recebemos os próprios dados que fizemos requisição, podendo ser retornados em JSON ou semelhantes.

2.4. JSON

O JSON (*JavaScript Object Notation*) é um formato de arquivo que é organizado de maneira que seu acesso seja facilitado e legível. Em um arquivo JSON, os dados são organizados no padrão "chave/valor" como é demonstrado na Figura 1. Apesar de levar em seu nome JavaScript, esse formato de arquivo é reconhecido por inúmeras linguagens de programação. Por conta disso, o JSON é muito utilizado em serviços web e aplicativos para leitura e compartilhamento de dados.

Figura 1. Exemplo de retorno de uma API em formato JSON. Fonte: (Sa, 2023)

2.5. Conceitos necessários para o uso da API do Spotify

2.5.1. *Access token*

O *access token* é uma *string* que contém as credenciais e permissões que podem ser usadas para acessar um determinado recurso ou dados do usuário. Na Figura 2 há um exemplo de uso do *Access Token*. O *access token* é passado no cabeçalho da requisição (linhas 5 a 7). Os dados retornados pela requisição estão sendo armazenados em formato JSON (linha 10).

```
1  async function getProfile(accessToken) {
2    let accessToken = localStorage.getItem('access_token');
3
4    const response = await fetch('https://api.spotify.com/v1/me', {
5      headers: {
6        Authorization: 'Bearer ' + accessToken
7      }
8    });
9
10   const data = await response.json();
11 }
```

Figura 2. Exemplo de um *access token* sendo usado para obter o perfil de um usuário.

2.5.2. Autorização

Refere-se ao processo de permitir o acesso a dados e ferramentas. O Spotify usa para autorização o *framework* OAuth2.0. Para realizar a autorização, o usuário final concede o acesso de seus dados protegidos (lista de reprodução, informações pessoais etc.) ao aplicativo, o aplicativo requisita o acesso dos dados do usuário, e o servidor hospeda os recursos protegidos e fornece a autenticação e autorização via OAuth 2.0.

2.6. Arquitetura MVC

De acordo com Fowler (2002), o padrão MVC (*Model-View-Controller*) foi criado para separar a lógica de negócios da apresentação, tornando a manutenção do código mais eficiente e modular. O *Model* representa a lógica de negócio a ser utilizada, dependendo do objetivo do projeto e do resultado esperado. O *View* é a parte responsável pela interface e pela exibição das informações e interação com o usuário. E o *Controller* faz a mediação entre o *Model* e a *View*, lidando com as requisições e dados de entrada do usuário, permitindo a interação entre a parte lógica e a parte visual.

2.7. Visualização de informações

A Visualização de Informações é a área responsável por apresentar os dados de forma visual de modo que eles possam ser compreendidos facilmente. Para representar os dados de forma visual, os dados são transformados em imagens mentais ou reais para que os seres humanos consigam visualizá-los (Nascimento; Ferreira, 2011). Na Figura 3 é apresentado um modelo de referência para visualização de informações, em que são apresentadas as etapas da visualização de informação, sendo elas:

- Transformação de dados: Os dados brutos são estruturados em uma visualização lógica, como tabelas;
- Mapeamento visual: A visualização lógica é transformada em uma estrutura visual, como gráficos;

- Transformação visual: A estrutura visual pode ser alterada pelo usuário, adicionando ou removendo dados, como o uso de filtros em gráficos para alterar seu resultado final.

Existem métodos de visualização de informações muito comuns, como a utilização de gráficos de barra, gráficos de linha, gráficos de pizza, histogramas, entre outros. Todas essas formas de representação de informação auxiliam na compreensão dos dados a serem analisados, muitas vezes colaborando para a descoberta de informações difíceis de serem observadas sem uma representação visual.

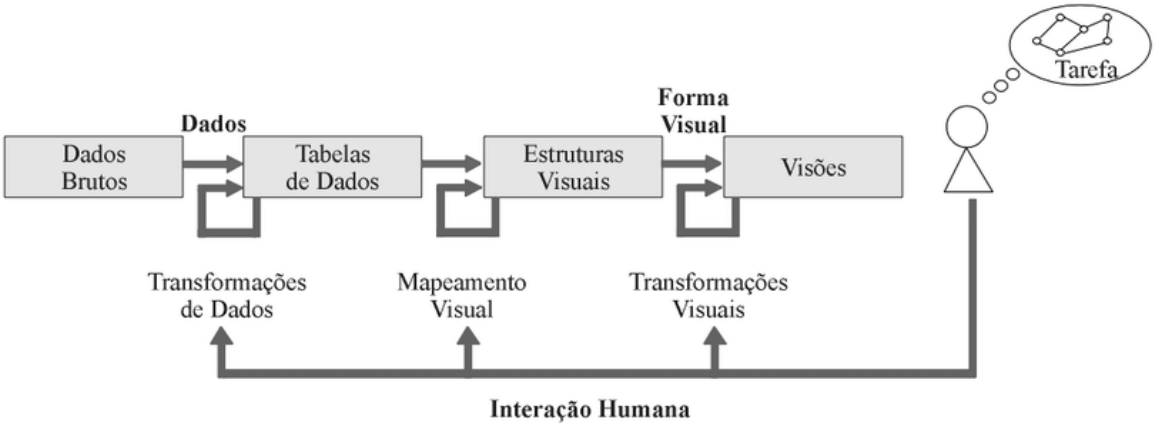


Figura 3. Modelo de referência para visualização de informações. Fonte: (Card *et al*., 1999).

3. Trabalhos correlatos

O trabalho de Rea *et al.* (2021) busca classificar músicas pela popularidade, propondo o uso de técnicas de processamento de linguagem natural, junto aos recursos sobre músicas que são disponibilizados pelo Spotify. Foi utilizado também um algoritmo de aprendizagem de máquina chamado árvore de decisão. Uma das motivações citadas para esse trabalho foi a necessidade crescente de artistas usarem plataformas de *streaming* como o Spotify para serem reconhecidos pelos seus ouvintes e para obterem sua renda na área da música. O projeto é importante para que os artistas consigam compreender o que pode tornar uma música mais popular entre os usuários e, em suas próximas obras, consigam alcançar ainda mais o seu público.

O trabalho de Silva (2017) teve como objetivo principal desenvolver um serviço *web*, que faz a integração de dois sistemas para diminuir o tempo de cadastro de clientes e de lançamento de pedidos de um sistema que utiliza o Magento, que é uma plataforma para criação de lojas virtuais. A criação do *web service* deve diminuir a utilização de recursos, diminuir os custos e aumentar a eficiência do sistema.

O trabalho de Rea *et al*. (2021) foca na análise da popularidade musical com processamento de linguagem natural, enquanto Silva (2017) aborda a otimização de processos em *e-commerce*. Este trabalho busca combinar a análise musical de Rea *et al*. com a interface *web* de da Silva para transformar dados do Spotify em gráficos interativos.

4. Metodologia

Empregamos uma abordagem incremental para o desenvolvimento do *software*, considerando três gráficos no total (cada gráfico será considerado um incremento), com as seguintes atividades:

- Estudo sobre integração de sistemas;
- Seleção das tecnologias de desenvolvimento;
  - Spring: *Framework* para construção do *backend* do trabalho;
  - Angular: *Framework frontend* usado na construção da interface do usuário;
  - Postman: Ferramenta utilizada para realizar testes de *endpoints* do *backend**;***
  - IntelliJ: IDE utilizada para programação do *backend* em Spring;
  - Visual Studio Code: IDE utilizada para programação do *frontend* em Angular.
- Estudo das APIs de *softwares* de execução de música e análise dos dados retornados por elas;
- Definição de três gráficos possíveis para analisar os dados retornados pela API;
- Implementação do primeiro gráfico;
- Implementação do segundo gráfico;
- Implementação do terceiro gráfico;
- Construção da tela de *dashboard* para exibir os gráficos.

5. Desenvolvimento

5.1. Estudo das APIs de aplicativos de música

Foram estudadas as APIs existentes de músicas, para selecionar a que fosse consumida pelo sistema proposto. Durante a pesquisa, cinco plataformas de *streaming* de música se destacaram:

- Spotify: Na página do Spotify para desenvolvedores, é dito que a plataforma possui 100 milhões de músicas e 5 milhões de *podcasts*. Além disso, em um relatório do quarto trimestre de 2023 do Spotify, existem informações relevantes para a escolha desta plataforma para este projeto, como os 602 milhões de usuários ativos por mês e o crescimento de 23% ao ano;
- Apple Music: Esta plataforma não divulga seu produto através do número de usuários, e acaba focando nos recursos que ela disponibiliza, como *lossless audio*, *Dolby Atmos* e a integração com dispositivos Apple;
- Deezer: De acordo com a *Deezer NewsRoom*, a Deezer possui 10,5 milhões de assinantes e mais 1,1 milhão de assinantes em 2023 graças ao fortalecimento de mercado na França e expansão da presença global;
- SoundCloud: De acordo com "*SoundCloud Revenue And Usage Statistics* (2024)", o SoundCloud possuía, até 2023, 140 milhões de usuários registrados e 375 milhões de faixas em sua plataforma;
- Youtube Music: Não é dito a quantidade de usuários total, mas no *blog* oficial do Youtube, é dito que, em Janeiro de 2024, o Youtube Music e *Premium* alcançaram 100 milhões de assinantes (considerando as contas em tempo de teste).

A Tabela 1 expõe os dados analisados de cada uma das plataformas citadas. Baseando-se nos dados obtidos através da pesquisa feita, a plataforma do Spotify é a que pode trazer mais benefícios para esse trabalho. É possível notar que a plataforma possui a maior quantidade de usuários ativos. Pelo Spotify possuir uma maior quantidade de usuários, é

possível que o projeto seja mais utilizado se for feito baseando-se na API do Spotify, já que é necessário que o usuário do sistema possua uma conta na plataforma de *streaming*. A API do Spotify não exige assinatura para consumi-la, sendo outra vantagem.

Tabela 1. Principais plataformas de *streaming* de música e suas principais características.

Plataforma	***Spotify***	***Apple Music***	***Youtube Music***
Usuários (estimado em 2023)	+ 600 milhões	-	100 milhões
Tipos de assinatura	Grátis e *Premium*	Somente pago	Grátis e *Premium*
Recursos	*Playlists* personalizadas, *Podcasts*	*Lossless Audio*, *Dolby Atmos*, Integração com dispositivos *Apple*	Integração com *Youtube*, Vídeos
Conteúdo (estimado em 2023)	+ 100 milhões de músicas, 5 milhões de *podcasts*	+ 100 milhões de músicas	Conteúdo do *Youtube*

5.1.1. A API do Spotify

A API do Spotify (Spotify, 2023) possui uma documentação rica em detalhes, além de tutoriais ensinando a usar seus recursos e explicações de alguns conceitos utilizados pela API, como: autorização, chamada de API, *token* de acesso, etc. Considerando esses e outros benefícios, foi decidido que a API do Spotify seria utilizada neste trabalho para receber informações relacionadas a músicas, artistas, álbuns e usuários, já que o Spotify possui uma grande quantidade de usuários ativos, fazendo com que mais pessoas possam fazer uso das ferramentas desse projeto.

O Spotify utiliza o OAuth 2.0, permitindo que um aplicativo se autentique em outro. Para isso, é necessário que o usuário permita acesso sem pedir o *login* e senha a ele. Na Figura 4 há um exemplo do funcionamento do OAuth 2.0. O sistema solicita uma autorização ao usuário. Caso a autorização seja concedida, o sistema solicita um *token* de acesso para realizar requisições de recursos que são protegidos na API.

Para fazermos uso dos recursos disponibilizados pelo Spotify, foi necessário criar uma forma de autorizar as requisições feitas para sua API, e para isso é preciso informar o *token* de acesso no cabeçalho das requisições HTTP. Para conseguir o código de acesso, o usuário deve fazer *login* com sua conta do Spotify (Figura 5a) e autorizar o sistema a usar algumas informações que serão solicitadas (Figura 5b), como exemplo conceder acesso aos artistas e músicas que o usuário escuta para poder obter os *top* artistas ou *top* músicas do usuário.

Após o usuário autorizar o uso de seus dados do Spotify, um código de autorização será gerado e retornado pela API do Spotify para o sistema que o requisitou (Figura 6). Em seguida, o código de autorização será inserido no cabeçalho de uma nova requisição HTTP, dessa vez para obter o *token* de acesso em si (Figura 7) que será usado nas requisições da API. O *token* de acesso tem a duração de 3.600 segundos (uma hora).

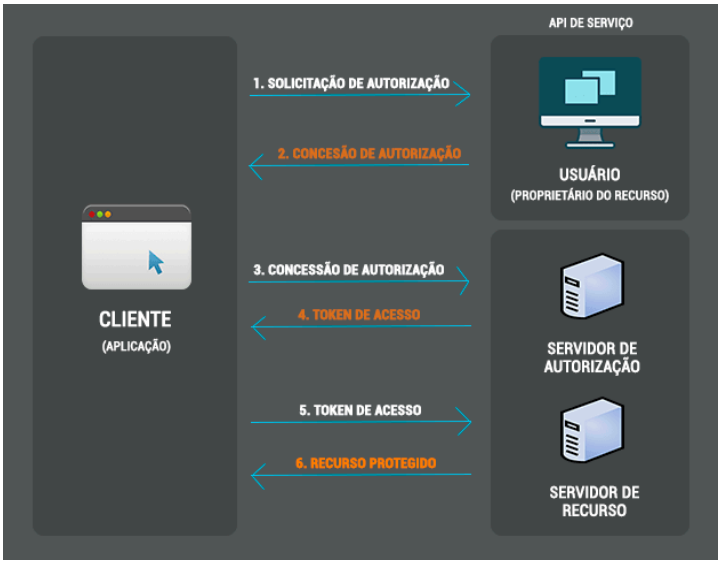
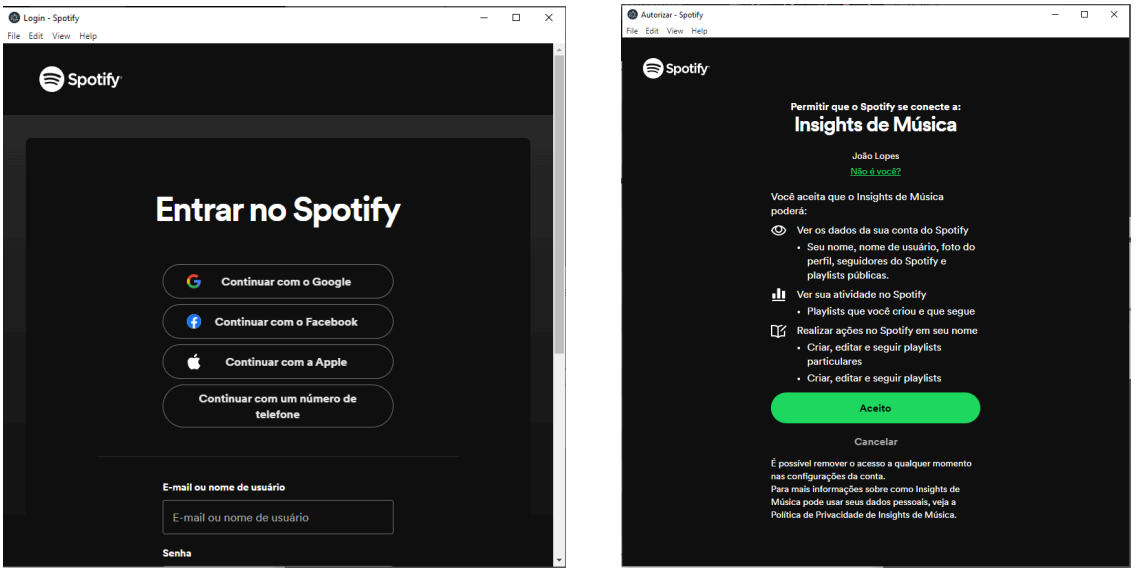


Figura 4. Funcionamento do OAuth 2.0 empregado para acesso a API do Spotify. Fonte: (Guedes, 2019).



(a) (b)

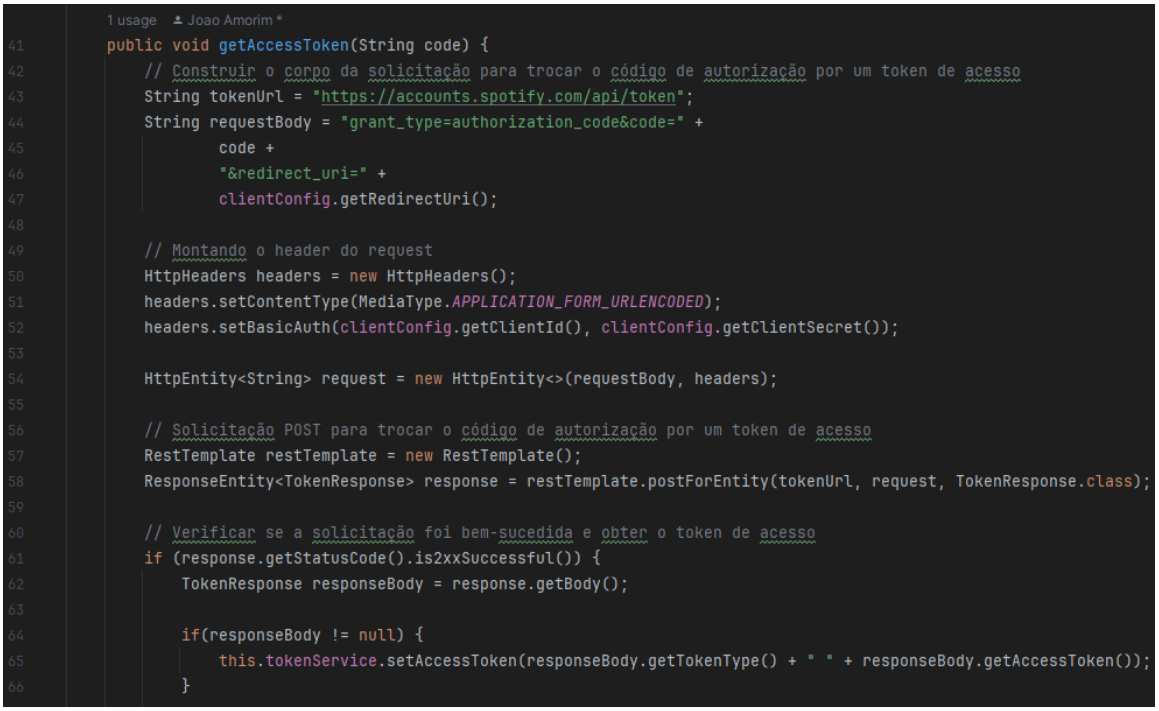
Figura 5. Telas (a) de *login* do Spotify que aparecerá caso o usuário não esteja logado ainda em sua conta e (b) de autorização que aparece ao usuário para autorizar que o sistema utilize os dados da conta de Spotify do usuário.

O Spring Boot é uma ferramenta que auxilia e agiliza o desenvolvimento de aplicativos web e microsserviços com o Spring *Framework*. Ele facilita a criação de aplicativos autônomos e de nível de produção. Para ajudar o desenvolvedor, as configurações mínimas para iniciar um novo projeto com o *framework* são simples de serem executadas, além de poder fornecer algumas dependências iniciais opinativas para simplificar a configuração de compilação. Neste projeto, foi utilizado o Spring Boot para o desenvolvimento do código para autenticar usuários, autorizar o uso dos dados dos usuários, obter dados da API do Spotify, entre outros.

```
1 usage   Joao Amorim *
public String getAuthCode() {
    String spotifyAuthorizeUrl = "https://accounts.spotify.com/authorize";
    UriComponentsBuilder builder = UriComponentsBuilder.fromHttpUrl(spotifyAuthorizeUrl)
            .queryParam( name: "client_id", clientConfig.getClientId())
            .queryParam( name: "client_secret", clientConfig.getClientSecret())
            .queryParam( name: "response_type", ...values: "code")
            .queryParam( name: "redirect_uri", clientConfig.getRedirectUri())
            .queryParam( name: "scope", ...values: "playlist-read-private " +
                    "playlist-read-collaborative " +
                    "user-read-private " +
                    "user-read-email " +
                    "user-top-read"
            );

    return builder.toUriString();
}
```

Figura 6. Trecho de código criado para obter o código de autorização.



```
1 usage   Joao Amorim *
public void getAccessToken(String code) {
    // Construir o corpo da solicitação para trocar o código de autorização por um token de acesso
    String tokenUrl = "https://accounts.spotify.com/api/token";
    String requestBody = "grant_type=authorization_code&code=" +
            code +
            "&redirect_uri=" +
            clientConfig.getRedirectUri();

    // Montando o header do request
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
    headers.setBasicAuth(clientConfig.getClientId(), clientConfig.getClientSecret());

    HttpEntity<String> request = new HttpEntity<>(requestBody, headers);

    // Solicitação POST para trocar o código de autorização por um token de acesso
    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<TokenResponse> response = restTemplate.postForEntity(tokenUrl, request, TokenResponse.class);

    // Verificar se a solicitação foi bem-sucedida e obter o token de acesso
    if (response.getStatusCode().is2xxSuccessful()) {
        TokenResponse responseBody = response.getBody();

        if(responseBody != null) {
            this.tokenService.setAccessToken(responseBody.getTokenType() + " " + responseBody.getAccessToken());
        }
```

Figura 7. Trecho de código criado para obter o *token* de acesso.

5.2. Uso da plataforma Postman para teste de requisições

A plataforma Postman foi usada para fazer testes de requisições para a API do Spotify. Ela simplifica os passos para fazer as requisições a outras API 's. Para o caso deste trabalho, o Postman também ajudou nos testes de autorização e uso do *token* de acesso.

É necessária uma requisição ao Spotify informando as credenciais do cliente, nesse caso serão informados o *Client ID* e o *Client Secret* (são credenciais do aplicativo que serão usadas para autorização da API e obtenção do *token* de acesso). Para exemplificar, será utilizado o Postman, que é uma ferramenta que permite fazer requisições de forma prática e receber os resultados retornados.

No Postman é possível realizar a autorização necessária para receber o *Access Token* (*token* de acesso) retornado do Spotify. Para isso, deve ser informado alguns dados, como o

Client ID e o *Client Secret*. A Figura 8 demonstra um exemplo de configuração de um novo *token* e na Figura 9 é possível visualizar o retorno de um *token* de acesso.

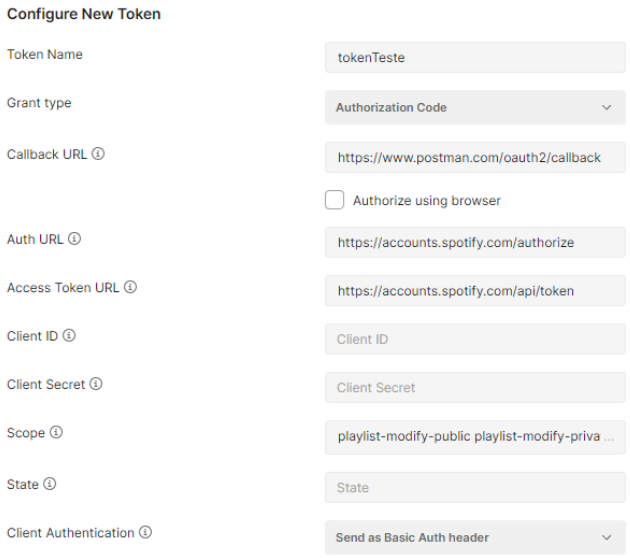


Figura 8. Configuração de um novo *token* na ferramenta Postman.

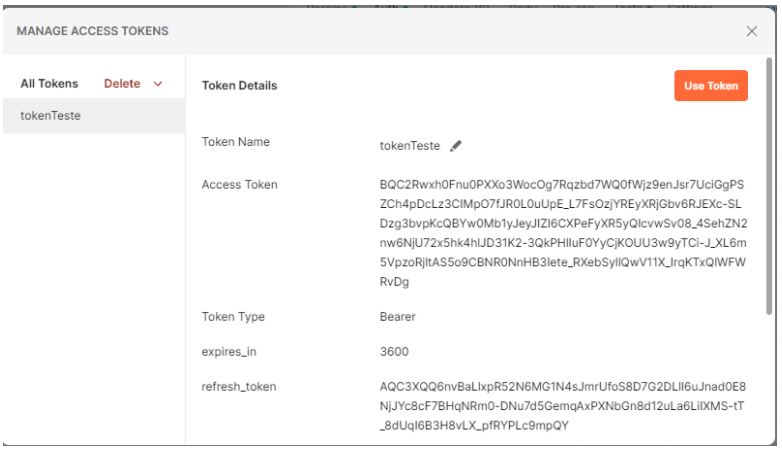


Figura 9. Retorno da API com informações do *token* de acesso.

5.3. Definição de gráficos para análise de dados

Para que a construção dos gráficos seja possível é necessário analisar os dados que são possíveis de recebermos da API do Spotify. Para isso, foi feita uma pesquisa na documentação da API do Spotify (Spotify, 2023) para analisar os dados disponibilizados e utilizar aqueles que podem ser considerados valiosos para utilização em gráficos:

- **Acusticidade:** É uma medida de 0.0 a 1.0 para saber se uma música é acústica. Quanto mais próximo o valor retornado for de 1.0, mais confiável será que essa música é acústica;
- **Dançabilidade:** É feita uma combinação dos dados da música para analisar o nível de dançabilidade da música, ou seja, será avaliado o quanto essa música é "adequada" para dançar;

- **Energia:** Representa a intensidade da música. Por exemplo: músicas no estilo Metal provavelmente terão um nível mais elevado de energia quando comparado com músicas clássicas;
- **Valência:** Esse recurso representa uma medida que vai nos dizer a positividade de uma música. É um valor que estará entre 0.0 e 1.0, em que quanto mais próximo de 1.0, mais positiva é a música (alegre, eufórica), e quanto mais próximo de 0.0, mas negativa é a música (triste, deprimente);
- **Popularidade:** As músicas possuem um dado que nos dizem sua popularidade, e seu nível de popularidade é medido de 0 a 100. O nível de popularidade é calculado por um algoritmo que analisa, principalmente, a quantidade total de reproduções que uma música tem e o quão recente são essas reproduções. As músicas que possuem mais reproduções recentemente tendem a possuir um nível de popularidade maior do que as que eram mais reproduzidas antigamente. Os artistas e os álbuns também possuem um nível de popularidade, e ele é derivado matematicamente da popularidade de suas músicas;
- ***Top* músicas do usuário:** Após a autorização do usuário para uso de alguns dados de seu perfil do Spotify, é possível obter suas *top* músicas baseado em um cálculo de afinidade (a documentação da API não entra em detalhes, mas provavelmente a quantidade de reproduções de uma certa música entra nesse cálculo). Alguns dos dados retornados são: as informações do álbum que a música pertence; os artistas que estão participando da música; a existência ou não de letras explícitas; sua popularidade. É possível passar no *endpoint* o limite de músicas retornadas. Também podemos definir o tempo que será considerado para retorno das *top* músicas. Nesse caso, temos três opções disponíveis: curto prazo (últimas 4 semanas); médio prazo (últimos 6 meses); longo prazo (último ano). Os prazos não são exatos, apenas aproximados;
- ***Top* artistas do usuário:** Funciona da mesma forma que o *top* músicas do usuário, mas dessa vez retornando seus *top* artistas. No caso do *top* artistas, podemos obter imagens de vários tamanhos do artista, seus gêneros musicais, sua popularidade, entre outros.

A partir desses dados, é possível plotar alguns gráficos que trazem informações significativas e que podem ajudar a tirar algumas conclusões sobre a relação da música com as pessoas e seus comportamentos.

5.3.1. Atualizações sobre dados disponibilizados pelo Spotify

Em novembro de 2024, o Spotify realizou um *post* com o título "*Introducing some changes to our Web API"* (Spotify, 2024) em sua comunidade e introduziu algumas mudanças em sua WebAPI. A partir dessa atualização, não será mais possível obter alguns recursos da API, como: recomendações, recursos de áudio, pré-visualização de músicas, entre outros. Essa mudança afeta diretamente um dos gráficos que seriam construídos no *dashboard* do projeto. Ainda assim, continuará sendo possível construir os demais gráficos.

5.4. Desenvolvimento da interface do sistema web

Desenvolvido pelo Google, o Angular é um *framework* de código aberto que se baseia na linguagem TypeScript, que proporciona tipagem estática e segurança, e usa a abordagem de orientação a componentes para criar interfaces. Este *framework* é muito utilizado para a criação de aplicativos web, pois além de transformar os aplicativos dinâmicos e interativos, ele fornece suporte a gerenciamento de estado, roteamento, validação de formulários, entre

outros benefícios. O Angular também promove uma manutenção facilitada graças a sua estrutura modularizada. Baseados nos benefícios que este *framework* oferece, foi decidido que este *framework* seria utilizado para a construção da interface *front-end* do sistema.

Para a construção dos gráficos no *front-end*, foi utilizado o Chart.js, uma biblioteca que possibilita a criação de gráficos interativos e responsivos em aplicações web. O Chart.js possui uma variedade de tipos de gráficos para serem usados, entre eles estão: gráfico de linha, gráfico de barra, gráfico de pizza, e gráfico de dispersão.

A quantidade de opções disponibilizadas pela biblioteca agiliza o desenvolvimento e traz várias alternativas para melhorias ou adesão de gráficos no futuro.

Antes da implementação do primeiro gráfico, foi necessário fazer a integração com o *backend* para obter os dados necessários para os gráficos. A integração foi feita através da criação de um serviço no *front-end* que possibilita a construção de requisições HTTP para realizar chamadas à API do projeto. A resposta da API é obtida em formato JSON e então deve ser trabalhada de forma que a biblioteca Chart.js consiga construir os gráficos com os dados recebidos.

Para implementar os gráficos no navegador, foi necessário a criação de uma variável para armazenar os nomes dos gráficos e uma variável para armazenar os dados que cada gráfico apresenta (Figura 10). São passados o nome e os dados dados do gráfico, e adiciona em um *array* de gráficos para ser utilizado.

```
110   private addToCharts(chartName: string, chartData: any) {
111     this.charts.push({name:chartName, data:chartData});
112     this.chartList.push(chartName);
113   }
```

Figura 10. Método para adicionar um gráfico no *dashboard*.

Também foi preciso criar um método para criação de gráficos (Figura 11). Esse método utiliza a biblioteca Chart.js, passando o nome do gráfico e os dados a serem exibidos, então o gráfico é criado e pode passar a ser exibido no navegador após ser invocado pelo HTML da página. O método instancia um novo objeto do tipo “Chart” (linha 90), passando o nome do gráfico e os dados a serem apresentados (linha 90 e 95).

```
89   createChart(chartName: string, chartData:any) {
90     this.chart = new Chart(chartName, {
91       type: 'bar',
92       data: {
93         datasets: [{
94           label: '# Media',
95           data: chartData,
96           borderWidth: 1
97         },
98       ]
99       },
100      options: {
101        scales: {
102          y: {
103            beginAtZero: true
104          }
105        }
106      }
107    });
108  }
```

Figura 11. Método para criar um gráfico no *dashboard*.

A página HTML carrega, dinamicamente, o *array* de gráficos e adiciona um elemento *canvas* para cada gráfico (Figura 12).

```
9           <section class="chart-main">
10              <h2>{{ chartList[0] }}</h2>
11              <div class="chart">
12                  <canvas [id]="chartList[0]"></canvas>
13              </div>
14          </section>
```

Figura 12. Trecho em HTML que utiliza o *array* de gráficos para apresentar no navegador a partir do elemento "canvas".

Para que o usuário possa trocar entre os gráficos a serem apresentados, foi necessário a criação de um método para atualizar os dados do gráfico que já está sendo exibido (Figura 13). O método recebe apenas o nome do gráfico e, a partir do nome do gráfico, é possível buscar dentro do *array* de gráficos os dados a serem apresentados. Na Figura 14 é exibido o trecho em HTML para apresentar os botões para troca de gráficos. Os botões possuem os nomes dos gráficos para que seja possível identificar qual gráfico será apresentado no navegador. Quando o botão é pressionado, o método para atualizar o gráfico é chamado, passando o nome do gráfico para o método.

```
115 protected updateChartData(chartName: string) {
116   this.chart.data.datasets[0].data = this.charts[this.charts.findIndex(chart => chart.name === chartName)].data;
117   this.chart.update();
118
119   const currentIndex = this.chartList.findIndex(currentChart => chartName === currentChart);
120   const currentChart = this.chartList[currentIndex];
121
122   this.chartList[currentIndex] = this.chartList[0];
123   this.chartList[0] = currentChart;
124 }
```

Figura 13. Método para atualização dos dados do gráfico.

```
15          <section class="chart-list">
16              @for (chart of chartList; track chart) {
17                  <button (click)="updateChartData(chart)" class="btn--primary-main">{{chart}}</button>
18              }
19          </section>
```

Figura 14. Código para alternância dos gráficos pelo usuário .

5.5. Incremento 1: implementação do primeiro gráfico

Na Figura 15, na linha 58, é chamado o método para obter o *top* músicas de um usuário com sua popularidade. Os dados são salvos em um *array*. Em seguida, da linha 61 até a linha 64, os dados são mapeados para serem separados em chave/valor, em que o nome da música é a chave, e a popularidade o valor. Após isso, os dados são armazenados no *array* de gráficos e o gráfico é criado.

```
58      this.track$ = this.graphicService.getTopTracksByPopularity();
59      this.track$.subscribe(all => {
60        this.allTracks = all;
61        const trackMap = this.allTracks.reduce((acc, track) => {
62          acc[track.name] = track.popularity; //define name como chave e popularity como valor
63          return acc;
64        }, {} as { [key: string]: number});
65
66        this.addToCharts("trackPopularity", trackMap);
67        this.createChart(this.charts[0].name, this.charts[0].data);
68      });
```

Figura 15. Tratamento de dados realizado para a exibição das *top* músicas de um usuário.

No método usado para buscar os dados necessários para o primeiro gráfico, passamos o *endpoint* no cliente HTTP para que o *front-end* consiga realizar uma requisição e obter a resposta da API. O *endpoint* basicamente é um URL que irá definir qual requisição será feita e o que irá ser retornado pela API.

No código da Figura 16, o método do serviço de gráficos para obter do *backend* os dados necessários para criar o gráfico de *top* músicas do usuário com a popularidade das músicas. No exemplo abaixo, foi usado o seguinte *endpoint* para recuperar a popularidade das *top* músicas de um usuário: "*/graphic/track/popularity/top*". Na requisição ao *backend*, é necessário passar o "*jwtToken*" que foi armazenado no "*localStorage*". Esse *token* é necessário para autenticar as requisições feitas ao *backend* e retornar as músicas do usuário correto.

```
26    getTopTracksByPopularity() {
27      return this.httpClient.get<Track[]>(this.url + '/graphic/track/popularity/top', {
28        headers: {
29          'Authorization': 'Bearer ' + localStorage.getItem('jwtToken')
30        }
31      });
32    }
```

Figura 16. Código para obtenção dos dados sobre popularidade das *top* músicas de um usuário.

5.6. Incremento 2: implementação do segundo gráfico

O código exibido na linha 71 da Figura 17 chama o método do serviço de gráficos para obter o *top* artistas do usuário com sua popularidade. Esses dados são armazenados em um *array*. Na linha 74, os dados são mapeados para chave/valor, em que o nome do artista é a chave, e a popularidade do artista o valor. Após isso, os dados mapeados são adicionados ao *array* de gráficos a partir do método "*addToCharts()*".

```
71    this.artist$ = this.graphicService.getTopArtistsByPopularity();
72    this.artist$.subscribe(all => {
73      this.allArtists = all
74      const artistMap = this.allArtists.reduce((acc, artist) => {
75        acc[artist.name] = artist.popularity; //define name como chave e popularity como valor
76        return acc;
77      }, {} as { [key: string]: number});
78
79      this.addToCharts("artistPopularity", artistMap);
80    });
```

Figura 17. Geração dos dados para o gráfico sobre os *top* artistas de um usuário.

Na Figura 18 é exibido o método do serviço de gráficos para obter do *backend* os dados necessários para criar o gráfico de *top* artistas do usuário com a popularidade dos artistas. No exemplo abaixo, foi usado o seguinte *endpoint* para recuperar a popularidade dos *top* artistas de um usuário: "*/graphic/artist/popularity/top*". Na requisição ao *backend*, é necessário passar o "*jwtToken*" que foi armazenado no "*localStorage*". Esse *token* é necessário para autenticar as requisições feitas ao *backend* e retornar os artistas do usuário correto.

```
34    getTopArtistsByPopularity() {
35      return this.httpClient.get<Artist[]>(this.url + '/graphic/artist/popularity/top', {
36        headers: {
37          'Authorization': 'Bearer ' + localStorage.getItem('jwtToken')
38        }
39      });
40    }
```

Figura 18. Código para obtenção dos dados sobre popularidade das *top* músicas de um usuário.

5.7. Incremento 3: implementação do terceiro gráfico

O código da Figura 19 apresenta o método do serviço de gráficos para obter o *top* músicas do usuário com os recursos e características quantitativas das músicas. Esses dados são armazenados em um *array* específico de músicas com características quantitativas. No código da Figura 20, o *array* de músicas passa por um iterador para que sejam obtidas a média de cada uma das características quantitativas das músicas. Após a retirada da média, essas informações são armazenadas. Para, então, o gráfico ser adicionado na tela do usuário (Figura 21). Na linha 61, os dados do *array* de média das características das músicas são passados ao método "*addToCharts()*" para armazenar os dados e criar um gráfico.

```
42        this.trackFeatures$ = this.graphicService.getTopTracksByFeatures();
43        this.trackFeatures$.subscribe(all => this.allTrackFeatures = all);
```

Figura 19. Código para obter as *top* músicas do usuário e suas características.

```
45      const trackFeaturesAvg: Partial<TrackFeatures> = {};
46      this.allTrackFeatures.forEach(track => {
47        trackFeaturesAvg.energy? trackFeaturesAvg.energy = (trackFeaturesAvg.energy + track.energy)
48         / 2 : trackFeaturesAvg.energy = track.energy;
49        trackFeaturesAvg.valence? trackFeaturesAvg.valence = (trackFeaturesAvg.valence + track.valence)
50         / 2 : trackFeaturesAvg.valence = track.valence;
51        trackFeaturesAvg.danceability? trackFeaturesAvg.danceability = (trackFeaturesAvg.danceability + track.danceability)
52         / 2 : trackFeaturesAvg.danceability = track.danceability;
53        trackFeaturesAvg.instrumentalness? trackFeaturesAvg.instrumentalness = (trackFeaturesAvg.instrumentalness + track.instrumentalness)
54         / 2 : trackFeaturesAvg.instrumentalness = track.instrumentalness;
55        trackFeaturesAvg.acousticness? trackFeaturesAvg.acousticness = (trackFeaturesAvg.acousticness + track.acousticness)
56         / 2 : trackFeaturesAvg.acousticness = track.acousticness;
57        trackFeaturesAvg.speechiness? trackFeaturesAvg.speechiness = (trackFeaturesAvg.speechiness + track.speechiness)
58         / 2 : trackFeaturesAvg.speechiness = track.speechiness;
59      });
```

Figura 20. Geração dos dados para o gráfico sobre as *top* músicas de um usuário.

```
60
61        this.addToCharts("trackFeaturesAvg", trackFeaturesAvg);
62
```

Figura 21. Código de adição do gráfico de *top* músicas na tela do usuário.

No código exposto na Figura 22 está o método do serviço de gráficos para obter do *backend* os dados necessários para criar o gráfico de média de recursos das músicas do *top* músicas do usuário. No exemplo abaixo, foi usado o seguinte *endpoint* para recuperar a os recursos das músicas do *top* músicas de um usuário: "*/graphic/track/features/top*". Na requisição ao *backend*, é necessário passar o "*jwtToken*" que foi armazenado no "*localStorage*". Esse *token* é necessário para autenticar as requisições feitas ao *backend* e retornar os recursos das músicas do usuário correto.

```
getTopTracksByFeatures() {
  return this.httpClient.get<TrackFeatures[]>(this.url + '/graphic/track/features/top', {
    headers: {
      'Authorization': 'Bearer ' + localStorage.getItem('jwtToken')
    }
  });
}
```

Figura 22 Código para obtenção dos dados sobre características das *top* músicas de um usuário.

5.8. Incremento 4: construção do *dashboard*

Para a construção do *dashboard* foi utilizado a linguagem de marcação HTML para criar a estrutura do conteúdo da página web. O HTML é utilizado para criar elementos como título, imagens e *links*, portanto, será feito o uso dele para criar gráficos para os usuários, criar botões para troca entre gráficos, ativar funções para realizar *login*, entre outras funcionalidades.

Além do HTML, foi utilizado a linguagem de estilização CSS (*Cascading Style Sheets*), que permite definir o visual e o *layout* dos elementos em HTML, como os botões, caixas de texto, títulos etc. A partir do CSS, é possível alterar cores, tamanhos, fontes, espaçamento entre os botões e o posicionamento dos elementos no *dashboard* do projeto.

Utilizando essas duas linguagens, foi desenvolvido o código para criar um *dashboard* ao usuário de forma que seja possível visualizar os gráficos, e assim que o usuário desejar visualizar outro gráfico, trocar entre eles através do clique entre botões com o nome do gráfico a ser apresentado.

O *dashboard* começa apresentando o primeiro gráfico, que seria a popularidade das músicas do *top* músicas do usuário. A Figura 23 apresenta como o gráfico está sendo exibido ao usuário. Para que o gráfico fosse renderizado, foi necessário o uso da biblioteca Chart.js e do HTML. O gráfico apresenta a popularidade de 10 músicas que estão presentes no *top* músicas do usuário. O valor da popularidade varia de zero a cem, sendo que quanto maior esse valor, mais popular essa música é de acordo com a API do Spotify.

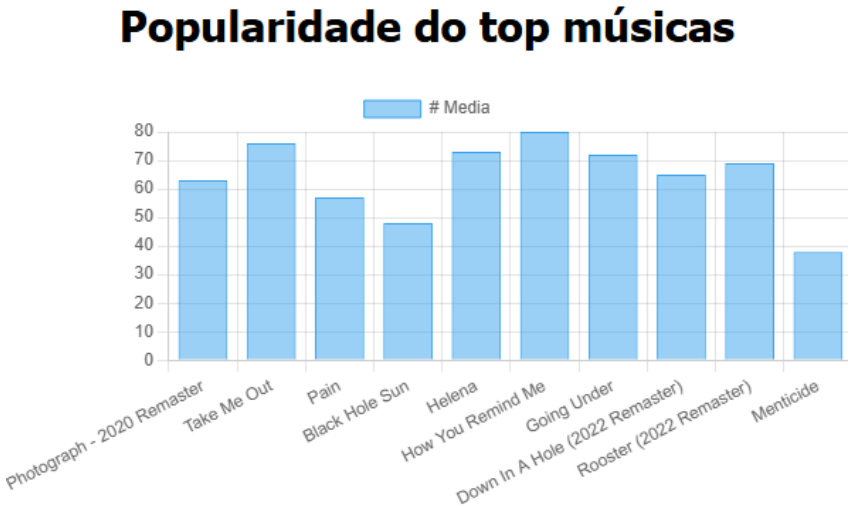


Figura 23. Gráfico sobre popularidade do *top* músicas renderizado na página em HTML utilizando a biblioteca Chart.js.

Em seguida, é apresentado o segundo gráfico, que apresenta ao usuário a popularidade dos *top* artistas do usuário de acordo com a API do Spotify. A Figura 24 mostra como o gráfico é exibido no navegador. O valor da popularidade varia de zero a cem, sendo que quanto maior esse valor, mais popular esse artista é de acordo com a API do Spotify.

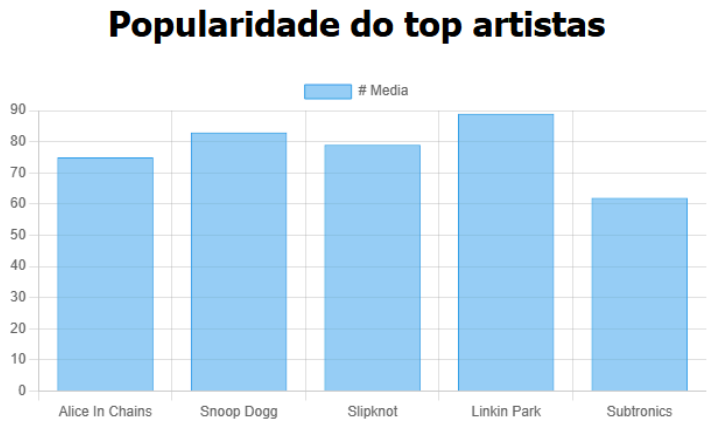


Figura 24. Gráfico sobre popularidade do *top* artistas renderizado na página em HTML utilizando a biblioteca Chart.js.

Finalmente, é apresentado o terceiro gráfico (Figura 25), representando a média dos recursos e características quantitativas do *top* músicas do usuário. Os valores são apresentados de zero a um. Quanto mais próximo de zero o valor estiver, menos as músicas apresentam a característica específica, como exemplo a característica "*instrumentalness*", que quanto mais próximo de zero, menos a música possui ou aparenta possuir instrumentos nela.

Figura 25. Gráfico com a média dos recursos e características quantitativas das músicas renderizado na página em HTML utilizando a biblioteca Chart.js.

Após a construção dos gráficos, foi necessário criar uma estrutura para que esses gráficos pudessem ser obtidos de forma simples para o usuário. Como solução, foi criado uma caixa com botões que apresenta um nome de gráfico. Após o clique em algum desses botões, o gráfico que estava a ser apresentado deve ser substituído pelo gráfico que foi escolhido pelo usuário. Para que a caixa de botões seja apresentada ao usuário, é necessário clicar no botão "Gráficos", que fica acima do gráfico.

Além da opção da caixa de botões, no cabeçalho da página, é possível visualizar um botão "Fazer *login*". Ao clicar nesse botão, o usuário é redirecionado para o *login* do Spotify, que já foi apresentado anteriormente neste artigo. Após o *login*, o usuário é redirecionado de volta para o *dashboard*. Então os dados são obtidos do *backend* do projeto e os gráficos são

carregados ao usuário, e então é possível fazer a visualização das informações disponibilizadas. As Figuras 26, 27 e 28 apresentam como é o *dashboard* após a criação dos elementos mencionados acima e as possíveis interações com o usuário.

Figura 26. *Dashboard* apresentando no cabeçalho o botão para fazer *login* com a conta do Spotify.

Figura 27. *Dashboard* apresentando a interação com o botão de gráficos, que abre uma caixa de botões para escolher outro gráfico a ser apresentado.

Figura 28. Apresentação do gráfico popularidade dos artistas após clique em um dos botões na caixa de gráficos.

6. Conclusões

Esse trabalho teve como objetivo inicial o desenvolvimento de um sistema web para construir gráficos que disponibiliza *insights* para a realização de análises mais precisas sobre o tema música e para a transformação desses dados em informações de valor que podem servir de suporte para estudos ou pesquisas mais precisas. Durante o desenvolvimento do projeto, foi preciso realizar o estudo sobre integrações de sistemas para poder compreender melhor a API do Spotify e planejar como fazer a integração entre o Spotify e o *backend* em Spring, e a integração entre o *backend* e o *frontend* em Angular. Após a compreensão e maior entendimento sobre integração de sistemas, foi realizado um estudo aprofundado da API do Spotify em relação aos dados disponibilizados. Foi possível notar que a API seria capaz de retornar os dados para a criação de gráficos valiosos em relação a popularidade e características específicas de cada música.

Os trabalhos de Rea *et al.* (2021) e Silva (2017) foram de suma importância para trazer inspiração e referências relacionadas ao uso de dados de fontes externas e integração de sistemas, que são conceitos muito utilizados no desenvolvimento deste trabalho. A partir da leitura desses trabalhos e do estudo feito sobre integração de sistemas e da documentação do Spotify, foi possível a construção do *backend*, a integração dele com a API do Spotify e a integração da interface do usuário em Angular, sendo possível expor os dados das músicas recuperados da API do Spotify e disponibilizá-las em gráficos seguindo o conceito de visualização de informações. Relacionado ao desenvolvimento, houveram algumas limitações e mudanças durante o desenvolvimento do projeto, como a atualização da API do Spotify, que fez com que deixasse de ser possível recuperar os recursos das músicas. Após essa atualização, um dos gráficos que foram desenvolvidos passou a não ser capaz de disponibilizar as informações.

Também foi necessário muito tempo de desenvolvimento para o desenvolvimento do *backend* para realizar a conexão de forma correta com a interface em Angular, pois alguns problemas relacionados a sessão de usuários foram enfrentados durante o desenvolvimento do projeto, como não salvar qual usuário estava fazendo as requisições ao Spotify, ficando impossibilitado de receber os dados das músicas dos usuário específicos. O atraso com sessões e a atualização da API do Spotify impossibilitaram o desenvolvimento do planejamento e construção de filtros, fazendo com que essa parte do desenvolvimento fique para os trabalhos futuros relacionados a esse projeto.

Outra melhoria importante a ser implementada no futuro é a análise e gestão de tempo de resposta da API para requisições, pois apesar da integração com a API do Spotify ter sido bem sucedida, a latência de resposta é um problema que pode ser enfrentado durante requisições maiores ou em momentos que a API recebe mais requisições. Uma solução seria implementar o uso de cache para dados que são frequentemente utilizados pelo usuário para garantir uma resposta mais ágil pelo sistema.

Durante o desenvolvimento do trabalho, foi possível aprender sobre a integração de sistemas e como ela é importante para interação entre diferentes sistemas e reutilização de funcionalidades, deixando os *softwares* mais flexíveis e práticos de desenvolver. Porém, também foi possível notar através desse trabalho que a dependência em outros sistemas pode afetar diretamente os resultados de um projeto, como foi relatado na Seção 5 Desenvolvimento, em que a API do Spotify deixou de disponibilizar dados cruciais para um dos gráficos que seriam apresentados ao usuário. Além disso, foram articulados conhecimentos assimilados nas disciplinas de Algoritmos e Programação, Linguagem de

programação I, II e III, Estruturas de Dados I, Arquitetura de *Software*, Programação Orientada a Objetos, Desenvolvimento Web, Desenvolvimento de Sistemas Web, Inglês Técnico, Inglês Técnico Avançado.

Referências

ABRAMUS (ed.). Consumo de Música no Brasil. [S. l.], 26 abr. 2019. Disponível em: < https://www.abramus.org.br/noticias/16444/consumo-de-musica-no-brasil/ >. Acesso em: 2 dez. 2023.

ANGULAR. Angular Documentation. Disponível em: < https://angular.dev/overview >. Acesso em: Out. 2024.

APPLE. Apple Music. Disponível em: < https://www.apple.com/apple-music/ >. Acesso em: Out. 2024.

APPLE. Apple Music API. Disponível em: < https://developer.apple.com/documentation/applemusicapi >. Acesso em: 4 dez 2023.

BATISTA, N. Angular o que é. Disponível em: < https://www.alura.com.br/artigos/angular-js?srsltid=AfmBOooZHk8yUJm4RIdI0gCoifJ-DS-2Nlap5PL1oQ6Q56J2zHjb4r5Z >. Acesso em: Out. 2024.

CARD, S.K.; MACKINLAY, J.D. E SHNEIDERMAN, B. Information Visualization. In: CARD, S.K.; MACKINLAY, J.D. E SHNEIDERMAN, B. (eds.) Readings in Information Visualization - Using Visualization to Think. San Francisco, Morgan Kaufmann Publ., 1999. pp. 1-34.

CHARTJS. Docs Chart.js. Disponível em: < https://www.chartjs.org/docs/latest/ >. Acesso em: 4 dez. 2024.

COHEN, L. Obrigada pelos 100 Milhões. Disponível em: < https://blog.youtube/intl/pt-br/news-and-events/obrigada-pelos-100-milhoes/ >. Acesso em: Out. 2024.

CUNHA, F. Sistema Web: o que é e como funciona. Disponível em: < https://www.mestresdaweb.com.br/tecnologias/sistema-web-o-que-e-e-como-funciona >. Acesso em: abril 2024.

CURRY, D. SoundCloud Revenue And Usage Statistics (2024). Disponível em: < https://www.businessofapps.com/data/soundcloud-statistics/ >. Acesso em: Out. 2024.

DA SILVA, S.; DA SILVA, F.; MARACCI, F.; PAZOTI, M. O uso de Web Services para integração de sistemas ERP com plataforma e-commerce Magento. Colloquium Exactarum, [S.l.], v. 9, n. 1, p. 79-94, 2017. Disponível em: < https://journal.unoeste.br/index.php/ce/article/view/1583 >. Acesso em: maio 2024.

DE ALMEIDA, José Raimundo Paim. Os benefícios cientificamente comprovados da música para estudar e trabalhar. [S. l.], 8 ago. 2023. Disponível em: <

https://ufrb.edu.br/bibliotecacecult/noticias/382-os-beneficios-cientificamente-comprovados-da-musica-paraestudar-e-trabalhar >. Acesso em: 2 dez. 2023.

FOWLER, Martin. Patterns of Enterprise Application Architecture. 1. ed. Boston: Addison-Wesley, 2002.

GUEDES, M. O que é OAuth 2. Disponível em: < https://www.treinaweb.com.br/blog/o-que-e-oauth-2 >. Acesso em: abril 2024.

IBM. O que é Java Spring Boot. Disponível em: < https://www.ibm.com/br-pt/topics/java-spring-boot >. Acesso em: Out 2024.

IFPI. IFPI Global Music Report 2019. [S. l.], 2 abr. 2019. Disponível em: < https://www.ifpi.org/ifpi-global-music-report-2019/ >. Acesso em: jul. 2024.

MOZILLA. CSS. Disponível em: < https://developer.mozilla.org/pt-BR/docs/Web/CSS >. Acesso em: 4 dez. 2024.

MOZILLA. HTML. Disponível em: < https://developer.mozilla.org/pt-BR/docs/Web/HTML >. Acesso em: 4 dez 2024.

MULTISHOW. O Impacto das Novas Tecnologias no Mundo da Música. [S. l.], 10 dez. 2019. Disponível em: < https://gente.globo.com/o-impacto-das-novas-tecnologias-no-mundo-da-musica/ >. Acesso em: 2 dez. 2023.

NASCIMENTO, H. A. D. DO; FERREIRA, C. B. R. Uma introdução à visualização de informações. Visualidades, Goiânia, v. 9, n. 2, 2012. DOI: 10.5216/vis.v9i2.19844. Disponível em: < https://revistas.ufg.br/VISUAL/article/view/19844 >. Acesso em: 4 mar. 2025.

PEDRON, K. Spotify Web API usando Postman: Fazendo requests para API do Spotify via Postman. Disponível em: < https://medium.com/@pedron.ketlin/spotify-web-api-usando-postman-9cea5371e4e5 >. Acesso em: abril 2024.

POSTMAN. Postman documentation overview. Disponível em: < https://learning.postman.com/docs/introduction/overview/ >. Acesso em: abril 2024.

REA, Antônio N.; DA SILVA, Douglas A. C.; PINTO, Mateus M. Vaz; DA SILVA, Leandro A. Classificação e análise de músicas para medição de popularidade. Revista Adelpha, São Paulo, 8 dez. 2021. Disponível em: < https://adelpha-api.mackenzie.br/server/api/core/bitstreams/b013c368-d191-49b7-8996-45ac5e8ed28e/content >. Acesso em: maio 2024.

SA, L. JSON: confira o guia completo. 2023. Disponível em: < https://www.hostgator.com.br/blog/json-guia-completo/ >. Acesso em: maio 2024.

SAUDATE, A. APIs REST: seus serviços prontos para o mundo real. [S .l.]. Editora Casa do Código. 2021.

SILVEIRA, P.; SILVEIRA, G.; LOPES, S.; MOREIRA, G.; STEPPAT, N.; KUNG, F. Introdução à Arquitetura e Design de Software: uma visão sobre a plataforma Java. São Paulo: Editora Casa do Código. 2012.

SPOTIFY. Web API. Disponível em: < https://developer.spotify.com/documentation/web-api >. Acesso em: 3 dez. 2023.

SPOTIFY. Introducing some changes to our Web API. Disponível em: < https://developer.spotify.com/blog/2024-11-27-changes-to-the-web-api >. Acesso em: Nov. 2024.

SPOTIFY. Spotify Reports Fourth Quarter 2023 Earnings. [S. l.], 6 fev. 2024. Disponível em: < https://newsroom.spotify.com/2024-02-06/spotify-reports-fourth-quarter-2023-earnings/ >. Acesso em: Set. 2024.

SPRING. Spring Boot Documentation. Disponível em: < https://spring.io/projects/spring-boot >. Acesso em: Out. 2024.

SYDLE. Integração de sistemas: saiba a importância, tipos e desafios. Disponível em: < https://www.sydle.com/br/blog/integracao-de-sistemas-6140d39a84679b13bf127a93 >. Acesso em: março 2024.

WENDEL, J. Deezer FY23 Results: Strong performance and momentum; reiterating financial targets for 2024 & 2025. Disponível em: < https://newsroom-deezer.com/2024/02/deezer-fy23-results-strong-performance-and-momentumreiterating-financial-targets-for-2024-2025/ >. Acesso em: Out. 2024.

Documento Digitalizado Público

Anexo I - artigo - TCC

Assunto: Anexo I - artigo - TCC
Assinado por: Andre Constantino
Tipo do Documento: Relatório
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Andre Constantino da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 14/03/2025 00:46:54.

Este documento foi armazenado no SUAP em 14/03/2025. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1966136
Código de Autenticação: 8fab95abb8