Aplicação do paradigma de Orientação a Objetos para integração entre Web APIs

Milton Bravo

Abstract—Este artigo traz uma demonstração da aplicação dos conceitos e padrões da programação orientada a objetos em uma aplicação que integra dois serviços de streamings de música, permitindo a transferência dos dados de usuário entre eles, através de APIs fornecidas. Os dois aplicativos utilizados possuem APIs disponíveis para assinantes da plataforma, e disponibilizam documentações sobre as mesmas que serão discutidas posteriormente, pequenas diferenças de abordagem nas APIs REST fizeram com a utilização de uma interface para organizar e aplicar o polimorfismo se tornasse interessantemente recomendada. Foi implementada a transferência de playlists entre plataformas e a criação de playlists por lote, onde uma lista de músicas é adicionada a uma playlist em uma única operação.

Index Terms—POO, OOP, API, REST, RESTFul, Deezer, Spotify, objetos, classes.

1 Introdução

O paradigma da Programação Orientada a Objetos (POO ou OOP em inglês) começou a ser apresentado na década de 60 com a linguagem SIMULA, e foi consolidado na próxima década com o Smalltalk que introduziu o termo propriamente dito para descrever o paradigma. Hoje em dia a POO é fundamental para o desenvolvimento de sistemas grandes e complexos, pois seus conceitos e características como a abstração, a modularidade e escalabilidade otimizam o processo de criação e manutenção desses sistemas [1].

A motivação desse artigo é trazer a aplicação da POO em um contexto de resolução de problema real, bem como destacar suas características durante o desenvolvimento.

Nos dias atuais, o consumo de músicas, podcasts e até mesmo programas de rádio estão se concentrando cada vez mais em serviços de streaming que são SaaS (Software as a Service), onde os usuário pagam para ter acesso ao conteúdo mas não donos da mídia ou de uma cópia da mesma. Sendo assim, o compartilhamento desses conteúdos são possíveis apenas para usuários que utilizam o mesmo serviço, ou seja, se eu uso o serviço A não posso compartilhar uma playlist de musicas que eu fiz com uma pessoa que usa o serviço B.

No contexto desse artigo abordaremos esse problema aplicado aos serviços da Deezer e do Spotify, onde integraremos esses dois SaaS através de APIs (interface de programação de aplicação) fornecidas por essas empresas. O objetivo é fazer uma aplicação modularizada e escalável, onde é possível adicionar novos serviços de streamings sem muita dificuldade e também novas funcionalidades de forma rápida.

As duas APIs utilizam o padrão REST (do inglês *representational state transfer*, ou em português "transferência de estado representacional") que trafega sobre o protocolo HTTPS, esse padrão REST foi criado justamente para permitir a comunicação entre aplicações de forma segura, funcional e independente [2]. Elas seguem um padrão de arquitetura tipo servidor e cliente, onde a aplicação aqui desenvolvida é o cliente nesse contexto. Outra vantagem é que não é necessário a implementação de todas as funcionalidades da API, e sim apenas as que serão utilizadas no lado do cliente.

2 Metodologia

Algumas etapas foram necessárias para o desenvolvimento e execução das atividades de projeto da aplicação. Foi necessário um estudo prévio das tecnologias utilizadas, definição de escopos de desenvolvimento e uma organização de arquitetura das classes.

2.1 Estudo prévio

Antes de entrar nos aspectos do código desenvolvido, vale ressaltar que é necessário um estudo sobre as APIs dos streamings escolhidos, para entender quais são as operações ou requisições permitidas em cada uma bem como devem ser executadas. Também é necessário entender como são construídos os objetos de retorno e seus atributos. As documentações das APIs estão disponíveis on-line na página para desenvolvedores que desejam utilizá-las para integração [4] [5].

Ambas APIs utilizam do protocolo de autenticação OAuth2, que combina a utilização de tokens de acesso e credenciais do usuário [3]. Para isso, tanto no Deezer quanto no Spotify, deve ser feito o cadastro da aplicação para que seja fornecido um código identificador da aplicação e um código de acesso da aplicação ao servidor dos streamings. Após isso, é solicitado ao usuário da aplicação que permita o acesso aos seus dados, através de um token de acesso com escopos definidos e tempo de expiração de uma hora. Depois de uma hora é necessário realizar o login novamente. O pseudo código 1 ilustra como é feito esse

processo na aplicação.

A plataforma Spotify tem uma particularidade que apenas usuários cadastrados podem utilizar a aplicação que foi cadastrada. Há um limite de 25 usuários que podem ser cadastrados, caso queira disponibilizar a aplicação para o público geral é necessário realizar um processo de solicitação e aprovação por parte da equipe do Spotify.

2.2 Escopo de desenvolvimento

As limitações das ferramentas de streaming são, dentre outras, o compartilhamento entre plataformas distintas e também a carga massiva de músicas variadas em uma playlist. Então o escopo principal de desenvolvimento é a transferência de playlists entre o Deezer e o Spotify, e carga de um arquivo com títulos de músicas para a criação de uma playlist.

2.3 Projeto da aplicação

Entrando agora na estruturação do código desenvolvido, detalharemos cada classe e pacotes utilizados e suas principais funções semânticas:

- Pacote Interface: Contém a classe *StreamingInterface* que é a interface que define os métodos públicos que devem ser implementados para cada streaming a fim de realizar as operações envolvendo as APIs de cada uma.
- Pacote MusicLibray: Contém as classes *Music* e *Playlist*, que definem as estruturas de dados locais, que recebem os dados de músicas e playlists que foram obtidos das APIs dos streamings. Os atributos são todos privados, porém alguns possuem *getters* que garantem somente a leitura dos mesmos. Nesse pacote é definida a agregação entre *Music* e *Playlist*, onde uma instância de Playlist possui uma lista de objetos Music.
- Pacote Streamings: Contém a classe *Streaming*, que possui alguns atributos protegidos comuns aos streamings a serem trabalhados, e também métodos públicos relacionados à manipulação local dos dados e troca de mensagens entre classes. É construída também a agregação entre *Streaming* e *Playlist*, onde uma instância de Streaming possui uma lista de objetos Playlist.
- Deezer: Contém a classe *Deezer* que implementa *StreamingInterface* e herda de *Streaming*. Seus atributos são todos privados e contém dados para autenticação nas chamadas de API.
- Spotify: Contém a classe *Spotify* que implementa *StreamingInterface* e herda de *Streaming*. Seus atributos são todos privados e contém dados para autenticação nas chamadas de API.
- app.py: Contém a função *appRun* que executa de fato da aplicação, onde é estruturado o menu do aplicativo e realizada a instanciação das classes Deezer e Spotify. Aqui também são verificadas as entradas de usuário e ocorre as tratativas de exceções.

Na figura 1, pode ser observado o diagrama de classes UML que mostra a estrutura das classes do projeto e como se dá o relacionamento entre elas.

A principal questão e desafio do projeto é a orquestração da obtenção dos dados, manipulação dos dados e envio dos dados para os streamings.

Para a transferência de playlists entre plataformas, é necessário obter os dados das playlists localmente, buscar por esses dados na outra plataforma para verificar a disponibilidade e trazê-los localmente, e depois estruturar e enviar esses dados para a outra plataforma seguindo o padrão esperado por eles.

Para o envio de músicas em lote para uma playlist em ambas as plataformas, é feita a leitura de um arquivo com os títulos das músicas e criação de uma playlist local, buscar por esses dados na plataforma desejada para verificar a disponibilidade e depois estruturar e enviar esses dados para a plataforma seguindo o padrão esperado por eles.

Esses dois processos são basicamente dependentes dos pseudo códigos 2 e 3.

Os seguintes conceitos foram utilizados no desenvolvimento do projeto:

- Herança
- Polimorfismo
- Encapsulamento
- Agregação
- Interface

O desenvolvimento em Python foi realizado utilizando o aplicativo Visual Studio Code com o interpretador python versão 3.12.3. O projeto completo da aplicação pode ser visto no repositório do GitHub [6]. Para uma melhor experiência aconselha-se ativar a extensão *Live Server* para que as instruções de coleta dos códigos de acesso sejam exibidos propriamente.

2.4 Pseudo Códigos

Algorithm 1 Autenticação

Require: $appId$, $secretKey$, $apiScopes$

1: Usando $appId$, solicita ao usuário para fazer o login
2: Obtém o código ($code$) enviado ao usuário
3: Utilizando $appId$, $secretKey$ e $code$ obtém o token de acesso ($token$)
4: Utilizando $token$ obtém o perfil do usuário

Algorithm 2 Construção das Playlists locais

Require: $userId$, $token$

1: **Conteudo** ← Utilizando $userId$ e $token$, obtém a lista de playlists do streaming
2: **for** $playlist$ em $Conteudo$ **do**
3: Cria playlist localmente
4: **Conteudo** ← Utilizando $playlistId$ e $token$ obtém as múscas da playlist
5: **for** $musica$ em $Conteudo$ **do**
6: Cria música localmente
7: Adiciona $musica$ na $playlist$
8: **end for**
9: **end for**

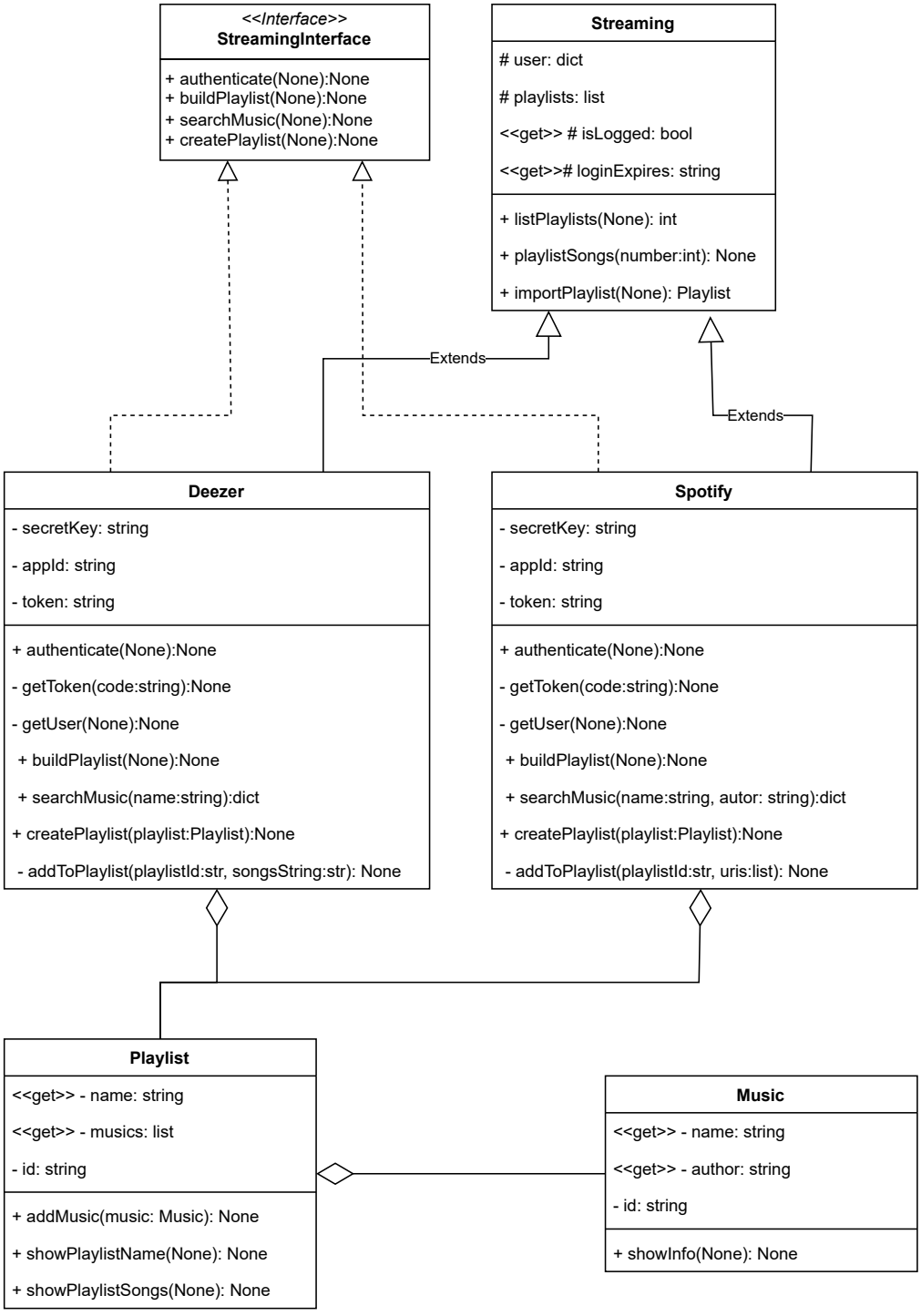

Fig. 1. Diagrama de classes UML do projeto

Algorithm 3 Criação da Playlists no streaming

Require: $userId$, $token$, $playlistLocal$

Utilizando $userId$, $token$ e $playlistLocal$ cria a playlist no streaming
for $musica$ em $playlistLocal$ **do**
    $musicasEncontradas \leftarrow$ Pesquisa a $musica$ no streaming
    if $musicasEncontradas$ é vazio **then**
        Registra que a música não foi encontrada
    else
        $listaMusicas \leftarrow$ Primeira música de $musicasEncontradas$
    end if
end for
Envia $listaMusicas$ para a playlist criada
Exibe quantas músicas não foram encontradas

Esses pseudo códigos representam uma visão superficial dos processos e englobam mais de uma função ou métodos propriamente dito.

3 Resultados

As funcionalidades apresentadas no escopo de desenvolvimento foram implementadas em sua completude. A transferência de playlists ocorreu nas duas direções conforme esperado bem como a importação do arquivo de músicas.

O menu construído permite fazer a reautenticação caso o tempo de atividade passe de uma hora, que é o tempo padrão de vida do token de acesso. Para esse controle de tempo fica exibida na tela o horário do ultimo login em cada plataforma.

Os algorítimos para realizar a busca de músicas em cada plataforma possui alguma limitações e particularidades, dessa maneira em alguns casos ocorre de alguma música não ser encontrada na plataforma concorrente, ou então ser retornada uma falsa correspondência. Isso pode ocorrer pois a música pode estar cadastrada com títulos diferentes ou mesmo não estar cadastrada, para o caso de não ser encontrada. Ou então quando há diferentes músicas com títulos semelhantes assim é retornada uma falsa correspondência.

Como a importação do arquivo de músicas não necessariamente precisa de duas contas conectadas, a aplicação permite a utilização com apenas uma plataforma conectada, porém libera apenas essa operação e trava a transferência de playlists.

4 Conclusão

A realização desse trabalho permitiu o desenvolvimento prático de diferentes conceitos estudados em sala de aula em um contexto real para a resolução de um problema. Foram estudadas novas tecnologias como as APIs REST e o protocolo de autenticação OAuth2, que inclusive tem grande aplicabilidade em diferentes problemas reais.

No estudo das APIs das plataformas foi notado a importância de uma documentação bem estruturada e acessível. A plataforma Spotify possui uma documentação robusta, rica em exemplos e bem detalhada, assim o entendimento dos objetos e métodos disponíveis na mesma foi rápido. A documentação disponibilizada pela Deezer parece ter sido desenvolvida a mais tempo, e assim não tem uma interface amigável e falta detalhamento principalmente na explicitação dos métodos disponíveis para consulta.

Um desafio encontrado foi a diferença de estruturação dos dados a serem enviados para as APIs. No caso do Spotify, os dados são enviados separadamente nos componentes da API, então é enviado um objeto de Header e um objeto Data com os dados por exemplo. Já no Deezer, todos os componente são enviados codificados na própria URL da API, assim é necessário uma maior atenção na formatação dos dados.

Podemos perceber a vantagem da utilização do paradigma da POO, quando notamos que para adicionar uma nova plataforma de streaming basta apenas criar uma nova classe que implementará os métodos da interface *StreamingInterface* seguindo os padrões da API dessa nova plataforma e ajustar o menu para a inclusão.

Como pontos de melhoria podem ser apontados, o desenvolvimento de uma interface gráfica para a utilização do usuário e uma melhor tratativa para os casos de falsas correspondências e de nenhum resultado encontrado.

Assim, os objetivos técnicos propostos foram alcançados bem como o desenvolvimento das habilidades requeridas. Notou-se a todo momento as vantagens da utilização da POO, e como a abstração e modularidade no código facilita a sua implementação.

References

[1] Aulas 1 a 5 POO. Gabriela Nunes. [https://drive.google.com/drive/u/0/folders/1AqLn6-fU5QIMP97CLzqVPXdH_u-0Gf4J] Acessado em: 21 de Junho de 2024
[2] Amazon Web Services. O que é uma API RESTful? [https://aws.amazon.com/pt/what-is/restful-api/]. Acessado em: 21 de Junho de 2024
[3] Aaron Parecki. OAuth 2.0 Simplified [https://www.oauth.com/]. Acessado em: 22 de Junho de 2024
[4] Deezer for developers. [https://developers.deezer.com/api]. Acessado em: 20 de Junho de 2024
[5] Spotify for developers - Web API. [https://developer.spotify.com/documentation/web-api]. Acessado em: 20 de Junho de 2024
[6] Git Repository - Playlist Management. Milton Bravo. [https://github.com/milbravo/PlaylistManagement]. Acessado em: 22 de Junho de 2024