Universidade do Minho

Licenciatura em Engenharia Informática

Unidade Curricular de

Programação Orientada aos Objetos

Ano Letivo de 2024/2025

SpotifUM

Beatriz Salgado Fernandes

(a100602)

João Silva Loureiro
(a100832)

POO

Índice

Introdução

No âmbito da unidade curricular de Programação Orientada aos Objetos, foi nos proposto o desenvolvimento de uma aplicação de gestão de conteúdos musicais e de utilizadores, designada por *SpotifUM*. Esta aplicação visa simular uma plataforma de streaming, onde é possível criar e reproduzir músicas, álbuns e playlists, tendo em conta diferentes perfis de utilizador e os seus respetivos planos de subscrição.

Ao longo deste relatório, iremos detalhar a estrutura da nossa aplicação, salientando as partes principais e as suas interações, bem como as funcionalidades implementadas e as decisões de design adotadas para a implementação das mesmas.

Exploraremos a estrutura do sistema, desde a definição dos utilizadores até a modelagem de como as reproduções são feitas e quem as pode efetuar. Além disso, discutiremos a abordagem utilizada para guardarmos as playlists relacionadas a um determinado utilizador, quer as suas playlists favoritas, quer as suas playlists criadas, ou mesmo os álbuns que queira ouvir.

Por fim, destacaremos os desafios enfrentados durante o processo de desenvolvimento, bem como as soluções adotadas para superá-los.

Arquitetura de Classes

Arquitetura implementada

Para uma melhor distribuição das funcionalidades e para um programa com um melhor encapsulamento, optamos por organizar o nosso sistema segundo uma estrutura MVC(Model-View-Controller).

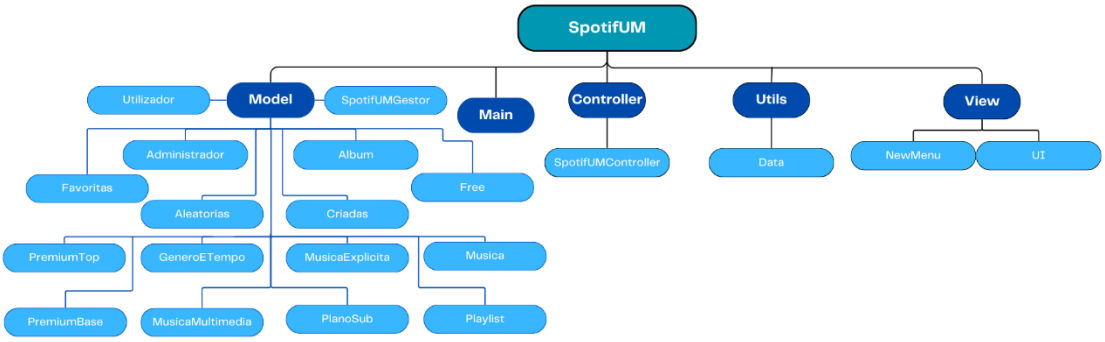

Figura 1: Arquitetura implementada

Como podemos ver através do esquema acima, o Model, contém todas as entidades de negócio, nomeadamente o Administrador, o Album, a Musica, o Utilizador, entre outros. O Controller é a classe SpotifUMController e esta é responsável pela lógica de aplicação e manipulação dos dados e a View é as classes UI e NewMenu, relativas à criação dos menus e da interação com o Utilizador.

Para além destas, adicionamos ainda o modulo utils, com a classe Data, que está encarregue da parte de guardar e carregar o estado do programa em ficheiro, com a informação existente em memória. Esta informação é guardada em formato binário e vai ser explicado mais abaixo.

Diagrama de Classes

Para conseguirmos organizar o nosso projeto e para conseguirmos ter uma ideia dos métodos que teríamos de definir e as relações que seriam estabelecidas entre as classes presentes, começamos por realizar um diagrama de classes, com auxílio da ferramenta Visual Paradigm.

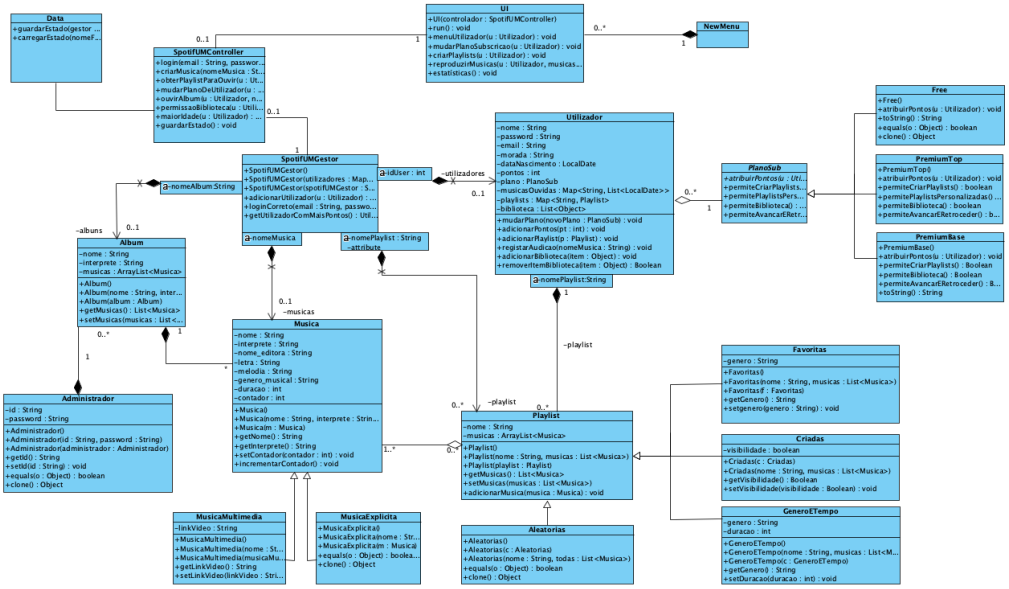

Figura 2: Digrama de Classes

Classes

Abordando agora as classes implementadas

Utilizador

Começando pela classe Utilizador, esta apresenta toda a informação relacionada com o cliente, ou seja, o seu nome, email, morada, password, plano de subscrição e os pontos que já ganhou conforme o plano que está inserido e as músicas que já ouviu. Para além disso, guarda ainda 2 Map's: um com o nome de todas as músicas que já ouviu e a respetiva data em que esta audição ocorreu e outro com todas as playlists que o mesmo já criou ou foram criadas para ele (no caso das geradas pelo sistema). Por último apresenta ainda uma biblioteca, que é uma lista de Objetos, que podem ser playlists ou albuns, onde são guardados, caso o utilizador possa e assim o deseje.

PlanoSub

A classe PlanoSub, tal como nome sugere é a classe responsável pelo gerenciamento do Plano de Subscrição do Utilizador. É nesta classe que os pontos são atribuídos ao utilizador e é ainda nesta classe, que são dadas as permissões de utilização de algumas funcionalidades do programa, que dependem do tipo de Plano ao qual o utilizador está subscrito, como por exemplo, se pode ou não criar uma playlist, se pode avançar ou retroceder uma musica, entre outros.

Esta classe é abstrata, já que não é instanciada diretamente uma vez que o utilizador tem que ter um tipo de plano específico e não um plano genérico. Nós decidimos usar herança, tendo então PlanoSub como classe principal e cada tipo de plano de subscrição como uma subclasse, já que desta forma se torna mais fácil adicionar um novo tipo de plano, futuramente. Neste momento temos 3 tipos de planos diferentes no nosso programa:

Free

Este tipo de plano é o plano mais básico, permitindo apenas ouvir sem a possibilidade de a avançar ou retroceder e pode ainda ouvir a playlist aleatória que mais à frente vamos referir.

PremiumBase

Este tipo de plano já apresenta mais algumas funcionalidades, nomeadamente consegue avançar e retroceder as músicas, pode criar playlists e tem ainda acesso a uma biblioteca pessoal, onde pode guardar playlists que queira ouvir mais tarde.

PremiumTop

Este tipo de plano apresenta todas as funcionalidades do PremiumBase e ainda dá acesso a playlists geradas de forma automática pelo sistema, conforme certos parâmetros, sejam estes referentes ao gosto pessoal do utilizador ou a pedido do mesmo (tempo de duração, explicitas apenas).

Cada uma destas subclasses faz Override do método '*atribuirPontos (Utilizador u)*' da classe principal e atribui os seus pontos, como definido no enunciado. Para as permissões também é usado o Override.

Administrador

A classe Administrador, guarda o seu identificador e a sua password. Este é unicamente responsável por adicionar novos álbuns, com as suas respetivas músicas. Tem ainda acesso a todas as estatísticas do programa, nomeadamente qual é o intérprete mais escutado, qual é o utilizador com mais pontos, entre outros.

O login do administrador é feito de forma "estática" no sentido em que não criamos novos administradores, e usamos sempre as mesmas credenciais, que estão indicadas diretamente no código.

Album

A classe Album apresenta um nome, um interprete e uma lista de músicas. A existência de uma classe Album é obrigatória para podermos ter músicas, já que estas só existem no contexto de um álbum. Dai a ligação entre as classes Album e Musica, no diagrama de classes, ser uma associação de composição, visto que uma música só existe, se existir um álbum.

Musica

A classe Musica, para além de guardar as informações gerais da música (nome, interprete, género, duração, etc), guarda ainda um contador. Este atributo é usado para marcar o número de vezes que a música foi ouvida, sendo incrementado através do método '*incrementarContador()*', que é chamado pelo programa, quando a musica é reproduzida. O contador é usado, posteriormente, nas estatísticas.

Esta classe é a base para os tipos específicos de música: com conteúdo explícito ou com componente multimédia, sendo ela por si só um tipo. Falando agora das subclasses:

MusicaExplicita

A MusicaExplicita é um tipo de música que, tal como o nome sugere, possui linguagem explicita/imprópria. Para implementar esta lógica no nosso programa, acrescentamos o atributo *dataNascimento* ao utilizador, para que desta forma, quando um utilizador quer ouvir uma música deste tipo, o sistema verifica a idade do mesmo, através da sua data de nascimento, para perceber se o mesmo pode ou não a ouvir.

MusicaMultimedia

Quanto à MusicaMultimedia, esta representa uma música, mas com um vídeo associado. Para associar o vídeo à música, acrescentamos um atributo linkVideo, que é uma String.

Playlist

A Paylist, para além do seu próprio nome, tal como nos álbuns, apresenta ainda uma lista de músicas. É a estrutura base para qualquer tipo de lista de reprodução na aplicação SpotifUM, servindo tanto para playlists criadas pelo utilizador como para playlists geradas automaticamente.

Esta serve assim como superclasse comum, permitindo que diferentes tipos de playlists herdem os seus atributos e métodos, especializando-se conforme os objetivos específicos de cada caso. No nosso programa temos 4 tipos de playlists:

Aleatoria

A Aleatoria representa as playlists mais básicas. Este tipo de playlist é gerado pelo sistema, quando um utilizador não pagante (com plano Free) tenta aceder às suas playlists. Esta playlist fica com 5 músicas das presentes na base de dados do programa, sendo que estas são escolhidas aleatoriamente, usando um shuffle para baralhar todas as músicas e escolhendo as primeiras 5. O nome desta playlist é sempre iniciado por Aleatoria seguido do nome do utilizador que tem sessão iniciada.

Criadas

A classe Criadas modela playlists criadas manualmente pelos utilizadores, sendo que apenas os subscritos a planos Premium, sejam eles Top ou Base, tem acesso a esta funcionalidade. O utilizador escolhe o nome da playlist e as músicas que a compõem, bem como a ordem das músicas na playlist.

Este tipo de playlist introduz um novo atributo: a visibilidade, um booleano que indica se a playlist é pública/true (visível a todos os utilizadores) ou privada/false (acessível apenas pelo seu criador). Essa funcionalidade permite a partilha de gostos musicais dentro da comunidade da aplicação, sendo que diferentes utilizadores vão ter a possibilidade de ouvir playlists públicas de outros utilizadores. Neste momento só é possível tornar a playlist pública ou privada ao criá-la, uma melhoria possível seria poder mudar a visibilidade quando quisesse.

Favoritas

A classe Favoritas representa playlists geradas com base no género musical mais ouvido por um utilizador. Durante a sua criação, é possível filtrar se a playlist deve incluir apenas músicas explícitas ou não.

Este tipo de playlist introduz um novo atributo, o genero, que indica o género favorito do utilizador, que é obtido tendo em conta o género dominante detetado no histórico de audições do utilizador.

A lógica de geração considera ainda a idade do utilizador, gerando playlists apenas com músicas não explícitas para utilizadores menores de idade, promovendo segurança e controlo de conteúdo.

GeneroETempo

A GeneroETempo, vai um bocado de encontro à Favoritas. Esta também gera playlists tendo em conta o género favorito do utilizador, mas, ao contrário das Favoritas, nestas o utilizador escolhe o tempo total de duração, em segundos, da playlist que quer que seja gerada. Introduz então os atributos, genero e duracao, sendo que o genero é obtido da mesma forma que já foi referido acima.

Para atingir o tempo de duração pedido pelo utilizador, o sistema vai adicionando músicas do género favorito, até atingir o valor pretendido.

SpotifUMGestor

A classe SpotifUMGestor atua como o núcleo de dados e lógica de negócio da aplicação SpotifUM. É responsável por armazenar, gerir e fornecer acesso aos principais recursos da aplicação: utilizadores, músicas, álbuns e playlists públicas. Estes são todos guardados em estruturas Map.

Funciona como uma classe de serviço, centralizando todas as operações de manipulação e consulta que não pertencem diretamente a nenhuma entidade em particular, garantindo separação de responsabilidades e promovendo encapsulamento. É a peça que conecta a camada de lógica de aplicação com os dados, promovendo uma organização limpa e coerente.

UI

A UI, é responsável por toda a interação com o utilizador, implementando a interface gráfica da aplicação SpotifUM. Utiliza a classe NewMenu, fornecida pelo professor, para criar menus interativos dinâmicos, com opções personalizadas e os respetivos handlers. Não realiza qualquer lógica de negócio, delegando todas as operações ao SpotifUMController.

A nossa interface gráfica é super simples. Organiza-se em menus distintos para utilizadores e administradores, mostrando menus com opções numeradas, e suporta todas as funcionalidades da aplicação, desde a criação de contas à audição de música. É um exemplo claro de separação de responsabilidades e aplicação prática da arquitetura MVC.

Abaixo vamos mostrar um exemplo de como estes menus aparecem.

SpotifUMController

A classe SpotifUMController funciona como o intermediário entre a View (UI) e o Model (SpotifUMGestor), sendo responsável por coordenar as operações da aplicação, validar permissões, controlar o fluxo de dados e aplicar regras de negócio. Esta separação garante um sistema modular, mais fácil de manter e expandir.

É a camada da arquitetura responsável por ligar a apresentação à lógica da aplicação, seguindo o padrão de desenvolvimento MVC (Model-View-Controller). Mantém uma referência direta ao SpotifUMGestor, sobre o qual delega todas as operações de leitura e escrita de dados.

Ao encapsular esta lógica no Controller, a UI pode concentrar-se apenas em apresentar menus e recolher inputs, sem precisar de conhecer a estrutura interna do modelo de dados.

Data

Por último, a classe Data é responsável por guardar e carregar o estado da aplicação, permitindo que a informação (utilizadores, músicas, playlists, etc.) seja persistente entre sessões. Funciona como uma classe utilitária (utility class), ou seja, todos os seus métodos são static, não sendo necessário criar instâncias de Data para a utilizar.

Esta usa serialização binária. Todas as classes implementam Serializable, o que permite guardar e recuperar facilmente o estado da aplicação como um todo. Esta abordagem simplifica o armazenamento e recuperação de dados, mantendo o sistema robusto e persistente.

É chamada pela Main para carregar o estado ao inicializar o programa e pelo SpotifUMController para guardar atualizações no estado.

Funcionalidades

Estando agora explicado todos as classes que temos, podemos passar para a visualização do trabalho criado. Abaixo mostramos imagens dos nossos menus e de algumas funcionalidades implementadas a decorrer.

MenuPrincipal

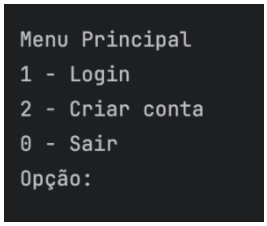

Figura 3: MenuPrincipal

CriarConta

```
Opção: 2
Nome: Jessica
Email: jessica2003@gmail.com
Password: jessica123
Morada: Rua do largo N13, Viseu
Data de nascimento (dd-MM-yyyy): 23-04-2003
Conta criada com sucesso.
```

Figura 5: Criar conta

```
Opção: 2
Nome: Diana
Email: jessica2003@gmail.com
Password: diana123
Morada: Rua Sampaio N19, Guarda
Data de nascimento (dd-MM-yyyy): 22-04-2005
Já existe uma conta com esse email.
```

Figura 4: Criar conta

Não é possível criar uma nova conta com um email repetido, ou seja, com um email que já tenha uma conta no sistema.

Login como administrador

```
Menu Principal
1 - Login
2 - Criar conta
0 - Sair
Opção: 1
Email: administrador
Password: administrador
Login como administrador realizado com sucesso!

Menu Administrador
1 - Adicionar álbum (e músicas)
2 - Estatísticas
0 - Sair
Opção:
```

Figura 6: Login como administrador

Para fazer o login como administrador são sempre usadas as credenciais "administrador", "administrador". Para efeitos práticos, supusemos que estas credenciais seriam dadas aos administradores e apenas eles saberiam e iriam entrar com elas. Como o administrador serve pura e simplesmente para adicionar álbuns ou ver as estatísticas da aplicação, a nosso ver não seria necessário ter contas diferentes para administradores.

Login como utilizador

```
Menu Principal
1 - Login
2 - Criar conta
0 - Sair
Opção: 1
Email: jessica2003@gmail.com
Password: jessica123
Login com sucesso!

Menu Utilizador
1 - Meu Perfil
2 - Playlists Públicas
3 - Albuns
4 - Criar uma playlist
5 - Ouvir musica
0 - Sair
Opção:
```

Figura 7: Login como utilizador

O login do administrador é feito com o email e a password que ele mesmo definiu ao criar a sua conta. Após o login bem-sucedido aparece então o menu de utilizador com as funcionalidades internas do programa.

```
--- Perfil ---
Nome: Jessica
Email: jessica2003@gmail.com
Morada: Rua do largo N13, Viseu
DataNascimento:2003-04-23
Pontos: 0
Plano: Free


1 - As minhas Playlists
2 - Biblioteca
3 - Criar Playlist Personalizada por Género
4 - Criar Playlist Personalizada por Género e apenas com músicas explicitas
5 - Criar Playlist Personalizada por Género e Tempo
6 - Mudar Plano de Subscrição
0 - Sair
Opção:
```

Figura 8: Perfil do Utilizador

Como dá para ver, através das figuras 7 e 8, mesmo sendo um utilizador não pagante (Plano de Subscrição Free), aparecem na mesma as opções de todas as funcionalidades. Ao selecionar uma funcionalidade ao qual não tem acesso, aparece uma mensagem (como se fosse uma publicidade) a dizer ao utilizador que não tem acesso à ferramenta e para mudar de plano de subscrição, caso queira.

```
Opção: 2
O seu Plano de Subscrição não tem acesso a esta ferramenta. Caso deseje, pode mudar o seu Plano de Subscrição no seu perfil.
```

Figura 9: Mensagem mostrada quando utilizador não tem permissão

Durante o desenvolvimento da aplicação SpotifUM, foi-nos sugerido que o sistema apenas apresentasse nos menus as funcionalidades permitidas para o tipo de plano do utilizador (por exemplo, esconder a opção de criar playlists a utilizadores com plano Free).

Porém, optámos por não esconder essas opções, mesmo que o utilizador não tenha acesso imediato a elas. Esta decisão foi tomada com base em duas razões principais.

Para começar, de uma perspetiva de negócio e usabilidade faz muito mais sentido. Ao mostrar todas as funcionalidades, mesmo que algumas estejam bloqueadas, o utilizador ter visibilidade sobre o que pode desbloquear ao mudar de plano. Isto cria incentivo para fazer upgrade e melhora a perceção de valor da aplicação. Caso contrário, um utilizador com plano Free nunca teria consciência de que existe a possibilidade de criar playlists ou adicionar à biblioteca— e, portanto, não teria motivação para mudar de plano.

Em segundo, a forma como implementámos o sistema de permissões baseia-se em métodos polimórficos definidos nos vários tipos de plano (permiteCriarPlaylists(), permiteBiblioteca(), etc.). Desta forma, o código da UI ou do Controller não precisa de saber qual é o tipo específico de plano (Free, Premium, etc.), apenas pergunta ao objeto do tipo PlanoSub se determinada funcionalidade é permitida. Isto facilita a adição de novos tipos de plano no futuro, bastando implementar os métodos apropriados na nova subclasse, sem alterar o resto do sistema.

Reproduzir Música

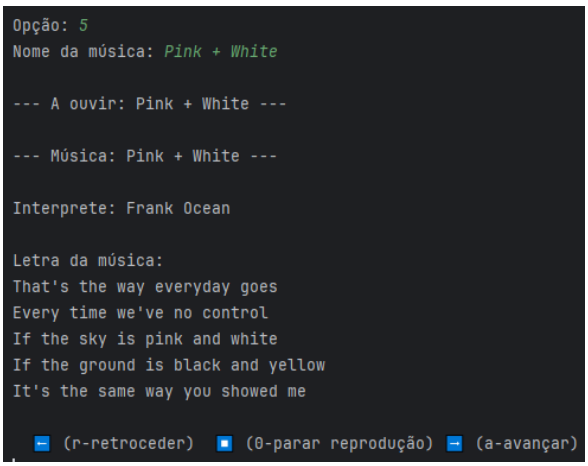

Figura 10: música a ser reproduzida

Conclusão

O desenvolvimento do projeto SpotifUM permitiu-nos consolidar os principais conceitos de Programação Orientada aos Objetos, através da implementação de uma aplicação funcional que simula um sistema de gestão e reprodução de música.

Consideramos que a arquitetura do sistema foi bem conseguida, com uma separação clara entre a interface do utilizador (UI), a lógica de controlo (Controller) e o modelo de dados (Model), alinhando-se com os princípios do padrão MVC. A utilização de herança, encapsulamento, polimorfismo e abstração está presente de forma coerente ao longo das várias componentes.

Um dos pontos fortes do nosso trabalho é a modularidade e extensibilidade do código: foi possível adicionar novas funcionalidades (como a gestão de pontos, playlists favoritas ou personalizadas) sem necessidade de reestruturar a base existente. A implementação de subclasses específicas como MusicaExplicita, Favoritas, ou GeneroETempo demonstra a vantagem de usar herança e composição para lidar com requisitos distintos. A persistência dos dados através de serialização binária também permitiu garantir a continuidade entre sessões, o que se revelou útil em termos práticos.

Durante o desenvolvimento deste projeto, enfrentámos algumas dificuldades, inicialmente, na interpretação do enunciado, nomeadamente na forma como a biblioteca do utilizador devia funcionar ou como as regras dos planos de subscrição se aplicavam, houve desafios a nível de organização da lógica entre a UI e o Controller, sendo necessário reestruturar partes do código para seguir os princípios de POO mais rigorosos e a criação do diagrama UML exigiu reflexão sobre as relações entre classes, especialmente com a existência de subclasses e métodos polimórficos.

Como melhorias futuras, gostaríamos de implementar uma interface gráfica mais desenvolvida e aparatosa, de forma a melhorar a experiência de utilização, adicionar testes automáticos às classes principais e otimizar a forma como a seleção e pesquisa de músicas é feita, incluindo a possibilidade de ordenação ou filtros avançados, bem como pôr as playlists geradas pelo sistema a aparecer e desaparecer de forma automática, por exemplo, uma nova a cada mês, em vez de ser o utilizador a decidir que a quer.