



**Universidade do Minho**

Licenciatura em Engenharia Informática

# Programação Orientada aos Objetos

## Trabalho Prático

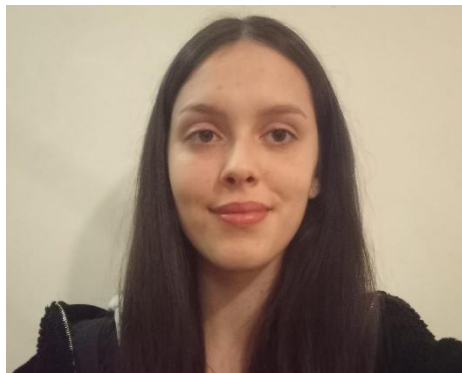
TP02

17 de Maio de 2025



Diogo Ribeiro

A106906



Carolina Martins

A107285



Filipa Gonçalves

A107329

# Resumo

No âmbito da Unidade Curricular de Programação Orientada aos Objetos, o grupo desenvolveu uma aplicação para gestão de músicas e listas de reprodução, permitindo aos utilizadores ouvir, organizar e explorar conteúdos musicais de forma prática. A aplicação possibilita a criação de novos utilizadores, a atualização do plano de utilização, a inserção de músicas e a criação de playlists associadas a utilizadores específicos. Além disso, determinados utilizadores podem aceder a uma biblioteca pessoal onde podem guardar playlists e álbuns como também controlar a reprodução das músicas, com opções para avançar ou retroceder na playlist. A aplicação também fornece uma maneira de consultar estatísticas detalhadas de utilização da mesma. Por fim, foi desenvolvido um sistema de recomendação que cria playlists personalizadas com base nos gostos de cada utilizador.

**Área de Aplicação:** Desenvolvimento no âmbito da Programação Orientada aos Objetos.

**Palavras-Chave:** Gestão de música, listas de reprodução e utilizadores, arquitetura de aplicação, JAVA, MVC, controlo de qualidade de código.

# Índice

<b>1 Introdução .....</b>	<b>3</b>
<b>2 Diagrama de classes .....</b>	<b>4</b>
<b>2.1 Entidades da Aplicação .....</b>	<b>4</b>
<b>2.2 Interrogações .....</b>	<b>9</b>
<b>2.3 Exceções.....</b>	<b>10</b>
<b>2.4 Persistência.....</b>	<b>10</b>
<b>3. Model-View-Controller .....</b>	<b>12</b>
<b>3.1 Implementação do MVC.....</b>	<b>12</b>
<b>4 Descrição da aplicação .....</b>	<b>14</b>
<b>4.1 UserFree .....</b>	<b>15</b>
<b>4.2 UserPremiumBase .....</b>	<b>16</b>
<b>4.3 UserPremiumTop .....</b>	<b>17</b>
<b>5 Conclusão.....</b>	<b>19</b>
<b>Anexos .....</b>	<b>20</b>
<b>Diagrama UML.....</b>	<b>20</b>

# 1 Introdução

Este relatório tem como objetivo abordar o projeto prático da Unidade Curricular de Programação Orientada a Objetos (POO) do ano letivo 2024/2025. Serão discutidas as decisões tomadas, as funcionalidades implementadas pelo grupo, bem como o método de raciocínio adotado para o desenvolvimento do projeto final. O objetivo deste trabalho consistiu no desenvolvimento de uma aplicação abrangente para a gestão de músicas e listas de reprodução de músicas que os utilizadores podem ouvir.

O enunciado do projeto estabeleceu requisitos progressivos, desde funcionalidades básicas de criação e gestão de utilizadores até recursos avançados, como a gerar playlist próprias para cada utilizador com base em certos critérios. Ao longo deste relatório, abordaremos as decisões de design, a arquitetura de classes adotada como os desafios enfrentados e as soluções implementadas para atender a todos os requisitos propostos.

## 2 Diagrama de classes

Nesta secção do relatório, será apresentada a construção do sistema com base no diagrama de classes. Apesar de existirem diversas ferramentas que poderiam ser utilizadas para auxiliar a criação da representação gráfica das classes e das suas relações empregando a notação UML (Unified Modeling Language), o grupo decidiu fazer esse processo manualmente.

O grupo considera que o diagrama de classes é uma parte crucial do design do sistema, uma vez que oferece uma visão estrutural da estrutura da aplicação. Descreve as classes que compõem o sistema, os seus atributos, métodos e as relações entre as mesmas.

Assim, ao fazer isto manualmente, consideramos que conseguimos ganhar um melhor entendimento das necessidades da aplicação e da forma como os diferentes componentes interagem entre si. Este processo exigiu uma análise mais aprofundada dos requisitos e uma reflexão cuidada sobre o papel de cada classe no sistema, o que contribuiu para um design mais consistente e alinhado com os objetivos do projeto. Além disso, permitiu ao grupo consolidar conhecimentos sobre modelação orientada a objetos, reforçando a importância de uma fase de planeamento sólida antes da implementação do código.

### 2.1 Entidades da Aplicação

Vamos começar por abordar as hierarquias de classes utilizadas para a representação das entidades da aplicação. Estas entidades estão centradas em classes que encapsulam a informação e o comportamento de elementos como utilizadores, músicas, playlists, álbuns e planos de subscrição.

Todos os utilizadores são compostos pelos mesmos atributos, mas podem ter acesso a features diferentes conforme o seu PlanoSubscricao (*Free*, *PremiumBase* e *PremiumTop*). Assim, a decisão mais lógica para representar um utilizador foi implementar uma classe *User*. Esta classe implementa *Serializable*, o que permite que os objetos do tipo *User* possam ser convertidos para um formato que pode ser armazenado (no nosso caso, em ficheiros binários). A serialização é útil, por exemplo, para guardar o estado dos utilizadores entre sessões ou para enviar dados entre diferentes componentes da aplicação. Ao implementar *Serializable*, garantimos que os dados dos utilizadores podem ser facilmente persistidos e restaurados quando necessário.

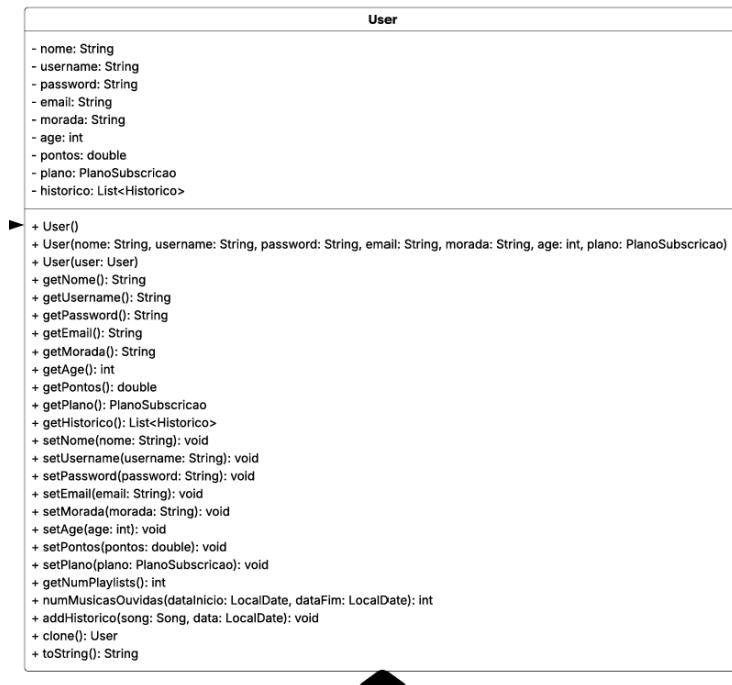


Figura 1 - Diagrama UML de User

A classe Song representa uma música no sistema e inclui atributos como nome, intérprete, editora, letra, pauta, género, duração e número de reproduções. Esta classe implementa a interface Serializable. Além disso Song atua como a superclasse principal no sistema, servindo como base para representar qualquer tipo de música. A partir dela derivam várias subclasses que especializam o comportamento de acordo com as características da música. Entre estas, destaca-se SongExplicit, uma das subclasses diretas de Song, que representa músicas com conteúdo explícito. Outra subclasse direta é SongMultimedia, que estende Song para adicionar suporte a diferentes tipos de conteúdo multimédia, como vídeos. Por sua vez, SongMultimedia também é uma classe pai: dela deriva SongMediaExplicit, uma subclasse que combina tanto a característica multimédia quanto o conteúdo explícito. Tanto SongExplicit como SongMediaExplicit implementam a interface Explicita, que define um comportamento comum para identificar músicas com linguagem ou conteúdo sensível. Esta estrutura promove uma hierarquia clara e modular, permitindo a reutilização e a extensão de código de forma organizada.

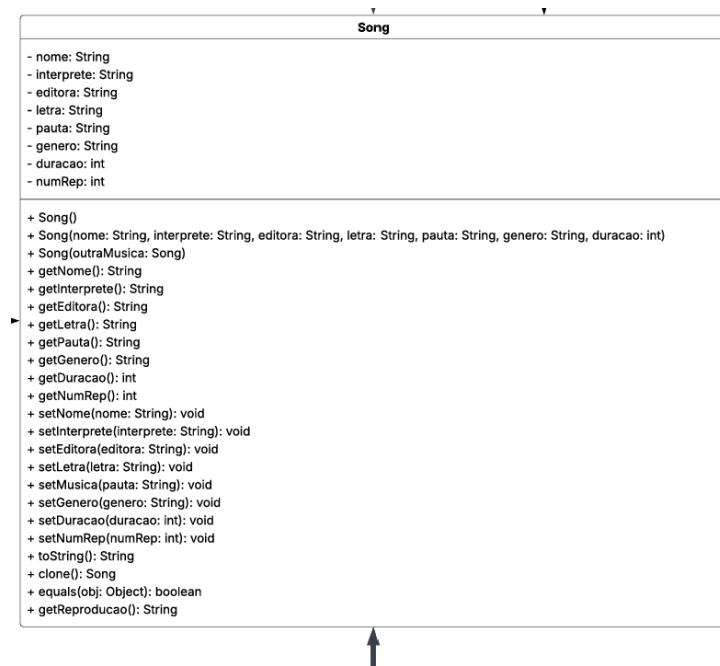


Figura 2 - Diagrama UML de Song

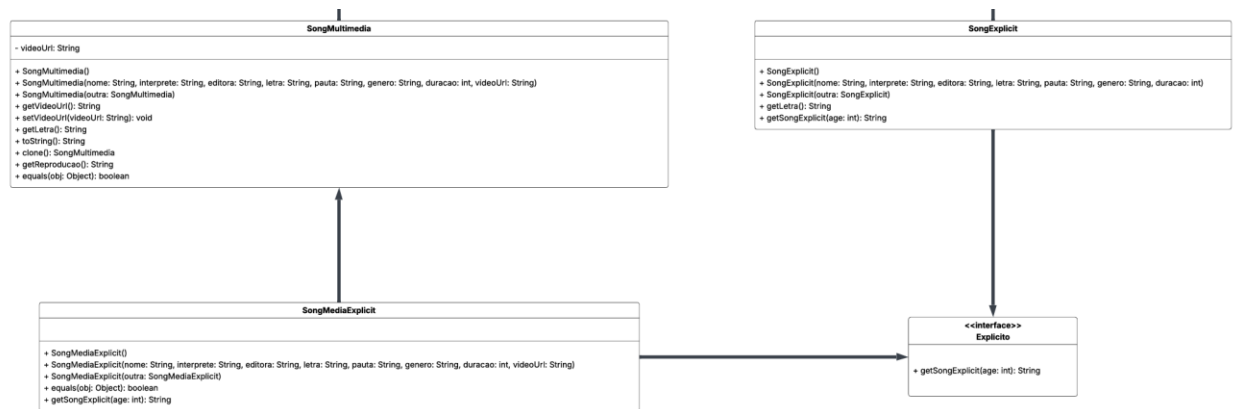


Figura 3 - Diagrama UML das subclasses de Song

A Playlist é uma classe abstrata que representa uma lista de músicas, pode ser pública ou privada. Por ser abstrata, não pode ser instanciada diretamente, servindo como base para diferentes tipos de playlists especializadas: PlaylistUser, PlaylistRandom e PlaylistTematica. Cada uma destas subclasses implementa funcionalidades específicas — por exemplo, playlists com tema e duração máxima (PlaylistTematica), ou playlists geradas aleatoriamente (PlaylistRandom). Esta abordagem permite uma estrutura extensível e reutilizável, alinhada com os princípios de herança e polimorfismo. Tal como outras entidades da aplicação, Playlist também implementa Serializable.

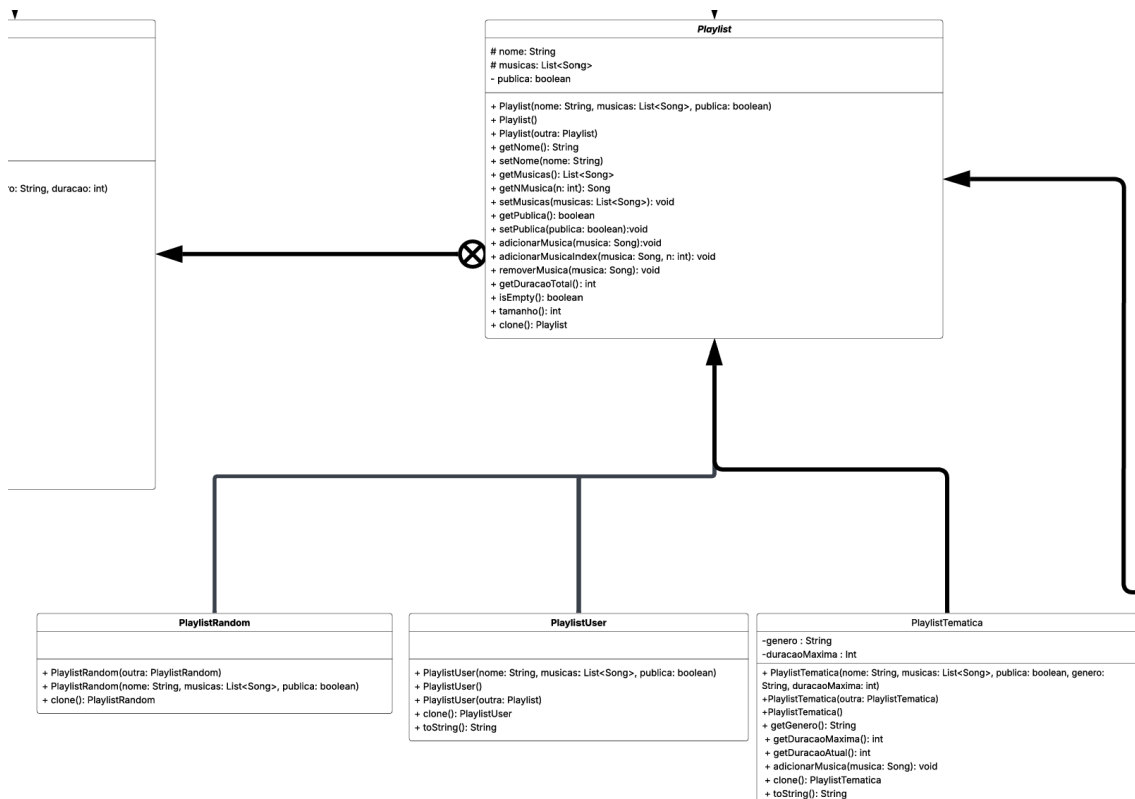


Figura 4 -Diagrama UML de Playlist

A classe *Album* agrupa várias músicas do mesmo artista sob um nome comum. Contém métodos que permitem adicionar, remover músicas e aceder ao conteúdo do álbum. Tal como outras entidades, a classe *Album* é serializável. Isto é especialmente útil para guardar álbuns criados ou modificados, mantendo a informação mesmo após a aplicação ser encerrada.

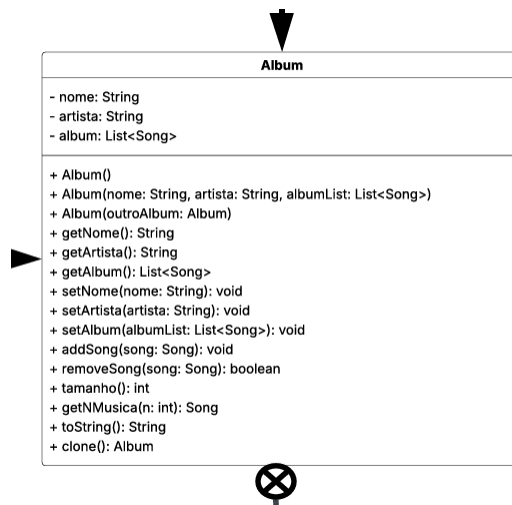


Figura 5 - Diagrama UML de Album



O sistema de subscrição foi modelado com base numa classe abstrata *PlanoSubscricao*, que define o comportamento comum a todos os planos (por exemplo, cálculo de pontos ou permissões de navegação na música). As classes *PlanoFree* e *PlanoPremium* herdam desta superclasse, enquanto os planos *PlanoPremiumBase* e *PlanoPremiumTop* herdam da classe abstrata *PlanoPremium* todas estas implementando comportamentos distintos de acordo com o tipo de subscrição. Esta abordagem permite aplicar o princípio do polimorfismo, facilitando o tratamento genérico de utilizadores com diferentes tipos de plano. A serialização destas classes assegura que o tipo de subscrição de cada utilizador é mantido ao longo do tempo.

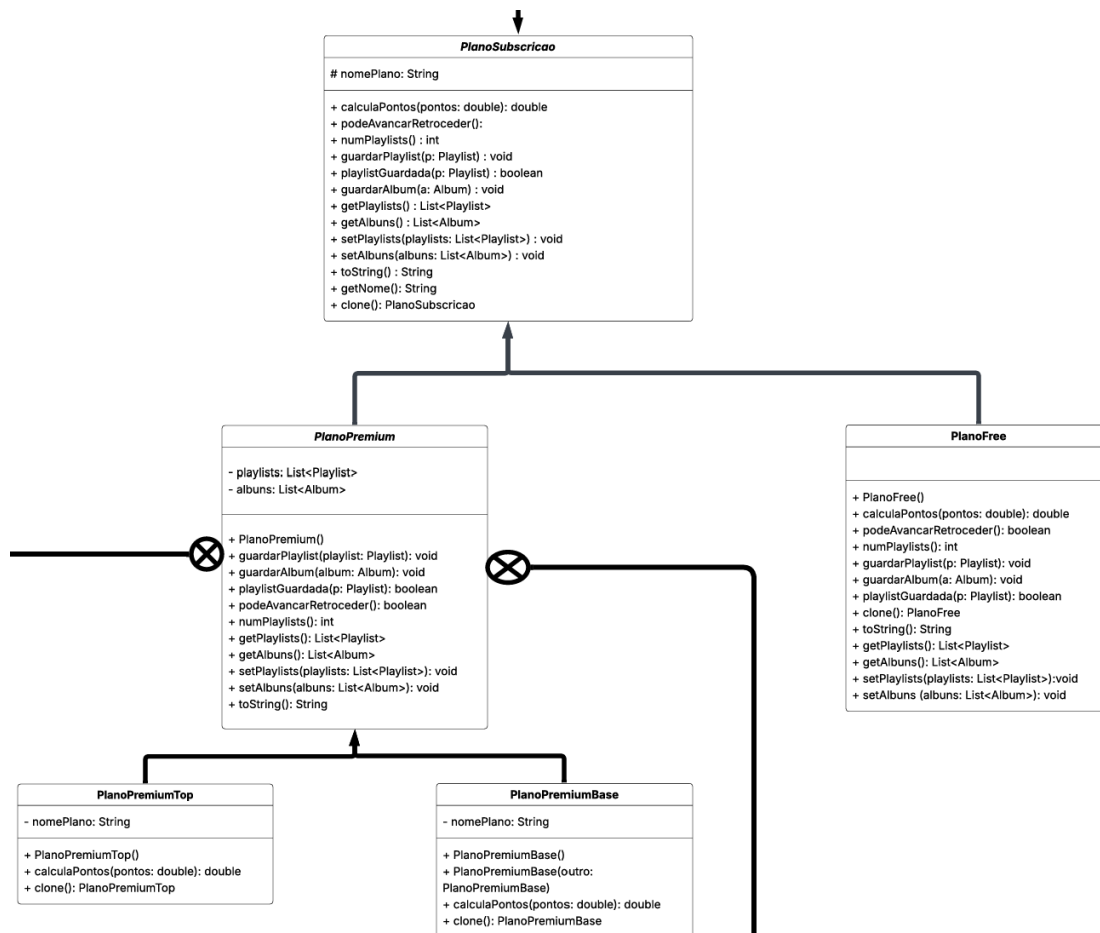


Figura 6 - Diagrama UML de Playlist

A classe *Historico* associa uma música a uma data específica, registrando que a mesma foi ouvida por um utilizador num dado momento. Este registo é essencial para funcionalidades como recomendações ou estatísticas. A classe é serializável, o que permite guardar os históricos de audição e manter a rastreabilidade das ações dos utilizadores ao longo do tempo, mesmo entre diferentes sessões da aplicação.

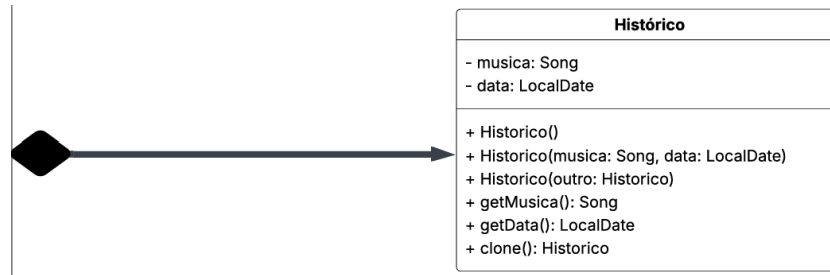


Figura 7 - Diagrama UML de Historico

## 2.2 Interrogações

A aplicação inclui um conjunto de interrogações que fornecem estatísticas e informações relevantes sobre a utilização do sistema, como por exemplo: a música mais ouvida, o intérprete mais popular, o utilizador com mais músicas ouvidas, entre outras. Estas interrogações estão implementadas na classe *Model*, o que faz sentido do ponto de vista arquitetónico, pois é nesta camada que se concentra toda a lógica de negócio e gestão dos dados da aplicação.

Entre as queries disponíveis, destacam-se:

- *musicaMaisOuvida()*: devolve a música com maior número de reproduções.
- *interpreteMaisOuvido()*: identifica o intérprete mais ouvido no sistema.
- *userMaisMusicasOuvidas()*: retorna o utilizador que ouviu mais músicas.
- *userMaisPontos()*: mostra o utilizador com maior pontuação acumulada.
- *generoMaisOuvido()*: determina o género musical mais popular.
- *numPlaylistsPublicas()*: conta o número total de playlists públicas.
- *userMaisPlaylists()*: identifica o utilizador com mais playlists criadas.

Colocar estas funcionalidades na classe *Model* permite centralizar o acesso e a manipulação de dados, respeitando o princípio da separação de responsabilidades. Assim, a interface gráfica (*View*) e os controladores (*Controller*) podem simplesmente invocar estas *queries* sem se preocuparem com os detalhes da implementação ou estrutura interna dos dados. Esta abordagem também facilita a manutenção e evolução da aplicação, permitindo alterar a lógica das interrogações sem impactar outras camadas do sistema.

```

+ musicaMaisOuvida(): String
+ interpreteMaisOuvido(): String
+ userMaisMusicasOuvidas(dataInicio: LocalDate, dataFim: LocalDate): String
+ userMaisPontos(): String
+ generoMaisOuvido(): String
+ numPlaylistsPublicas(): int
+ userMaisPlaylists(): String

```

Figura 8 -Queries no diagrama UML

## 2.3 Exceções

A aplicação define um conjunto de exceções personalizadas que servem para sinalizar situações específicas em que a execução normal do programa não pode continuar, normalmente por falta de dados. Estas exceções são importantes para garantir a robustez e a clareza do código, permitindo lidar de forma controlada com casos inesperados ou limites do sistema.

As exceções presentes são:

- *zeroSongsListen*: lançada quando não há músicas reproduzidas no sistema.
- *zeroGenresListen*: indica que nenhum género musical foi ouvido.
- *zeroInterpretesListen*: sinaliza que nenhum intérprete foi reproduzido.
- *zeroUsersWithPlaylist*: ocorre quando não há utilizadores com playlists.
- *zeroUserWithPoints*: usada quando não existe nenhum utilizador com pontos atribuídos.

A criação destas exceções específicas melhora a **legibilidade do código**, separa a lógica normal da lógica de erro, e ajuda a manter um fluxo de execução claro e previsível, mesmo em situações limite. Além disso, segue boas práticas de programação orientada a objetos, ao utilizar exceções como mecanismos formais para tratamento de erros.

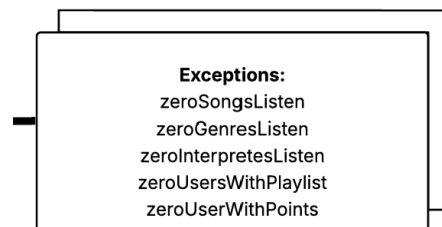


Figura 9 - Exceptions no diagrama UML

## 2.4 Persistência

Para garantir a continuidade da experiência do utilizador e a preservação da informação entre sessões, foi implementado um módulo de persistência que permite guardar e carregar o estado da aplicação. Este módulo utiliza serialização de objetos e escrita em ficheiros binários, permitindo que instâncias das principais entidades do sistema (como utilizadores, músicas, playlists e álbuns) possam ser armazenadas e posteriormente restauradas.

A classe responsável pela persistência disponibiliza métodos específicos para guardar e carregar cada tipo de entidade:

- *saveUsers(...)* e *loadUsers()* — para os dados dos utilizadores.
- *saveSongs(...)* e *loadSongs()* — para o catálogo de músicas.
- *savePlaylists(...)* e *loadPlaylists()* — para playlists.
- *saveAlbum(...)* e *loadAlbums()* — para álbuns musicais.

Esta abordagem assegura que todas as alterações realizadas durante a execução — como criação de novas playlists, reprodução de músicas ou alteração de planos — são registadas de forma duradoura. Ao reabrir a aplicação, o utilizador pode retomar a sua atividade exatamente no ponto onde a deixou.

A ligação entre a camada de persistência e a camada de modelo (Model) é feita através de uma dependência representada no diagrama por uma seta tracejada. O Model utiliza os métodos da classe de persistência para carregar os dados no início da execução e para os guardar no final ou em momentos estratégicos da aplicação (como após uma alteração significativa). Esta separação de responsabilidades permite manter o código modular, facilitando a manutenção e possíveis evoluções futuras no mecanismo de armazenamento.

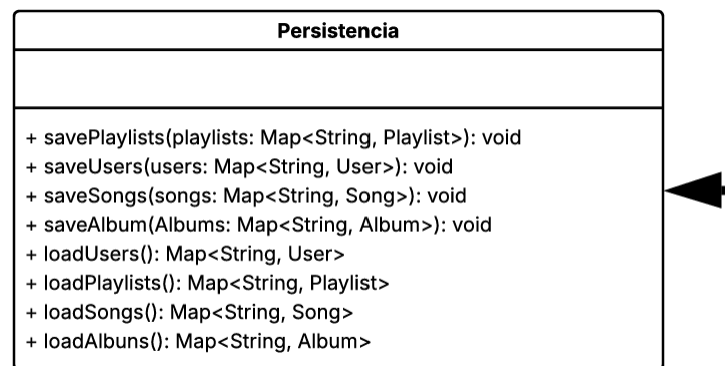


Figura 10 - Persistência no diagrama UML

### 3. Model-View-Controller

Nesta secção, abordamos a utilização do padrão arquitetural MVC (Model–View–Controller) e a forma como foi aplicado na nossa aplicação. O padrão MVC é uma abordagem amplamente adotada no desenvolvimento de software, que promove a separação de responsabilidades entre três componentes principais: a Model, que gere os dados e a lógica de negócio; a View, que trata da interface com o utilizador; e o Controller, que atua como intermediário entre as duas anteriores, coordenando as interações e o fluxo de dados.

A adoção desta estrutura modular traz diversas vantagens, como a facilidade de manutenção, a escalabilidade do sistema e a organização clara do código. No nosso projeto, o padrão MVC permitiu desenvolver cada componente de forma relativamente independente, facilitando testes, alterações e futuras extensões. A Model concentra toda a lógica da aplicação e os dados persistentes, a View apresenta menus e recolhe input do utilizador, e o Controller gere as ações solicitadas, invocando os métodos apropriados no Model e atualizando a View.

Apesar das vantagens, a implementação do padrão MVC também trouxe alguns desafios, como a necessidade de garantir uma comunicação eficaz entre os componentes e evitar dependências excessivas. Ainda assim, os benefícios superaram largamente as dificuldades, contribuindo para uma aplicação bem estruturada, coesa e de fácil evolução.

#### 3.1 Implementação do MVC

A aplicação foi desenvolvida com base no padrão Model-View-Controller (MVC), refletido claramente na organização das classes do projeto. Esta separação estrutural permite isolar responsabilidades e tornar o sistema mais modular e de fácil manutenção.

- A classe Model é o centro da lógica de negócio e da gestão de dados. Nela estão implementadas todas as funcionalidades principais da aplicação, como a criação de utilizadores, playlists, músicas, álbuns e também as várias queries estatísticas (como `musicaMaisOuvida()` ou `userMaisPontos()`). A Model mantém em memória as principais tabelas da aplicação (`userTable`, `songTable`, `playlistTable`, `albumTable`) e interage com a camada de persistência para guardar e carregar o estado da aplicação.
- A classe View representa a interface textual com o utilizador. É responsável por apresentar menus, recolher input e chamar métodos apropriados no Controller de acordo com as opções escolhidas. A View nunca interage diretamente com o Model, respeitando a separação de camadas.
- A classe Controller atua como intermediário entre a View e o Model. Recebe comandos da View, processa-os e invoca os métodos adequados no Model, retornando depois os resultados para serem exibidos. Contém a lógica de controlo da aplicação, como a reprodução de músicas, manipulação de playlists e criação de planos de subscrição.

Esta implementação respeita os princípios fundamentais do MVC: a lógica de apresentação (View), controlo (Controller) e dados (Model) estão separadas, o que permite, por exemplo, alterar a interface (substituir o menu textual por uma GUI) sem afetar a lógica interna da aplicação. A existência de dependências bem definidas entre as camadas (por exemplo, a View tem uma referência ao Controller, e o Controller ao Model) reforça esta arquitetura e facilita a evolução futura do projeto.

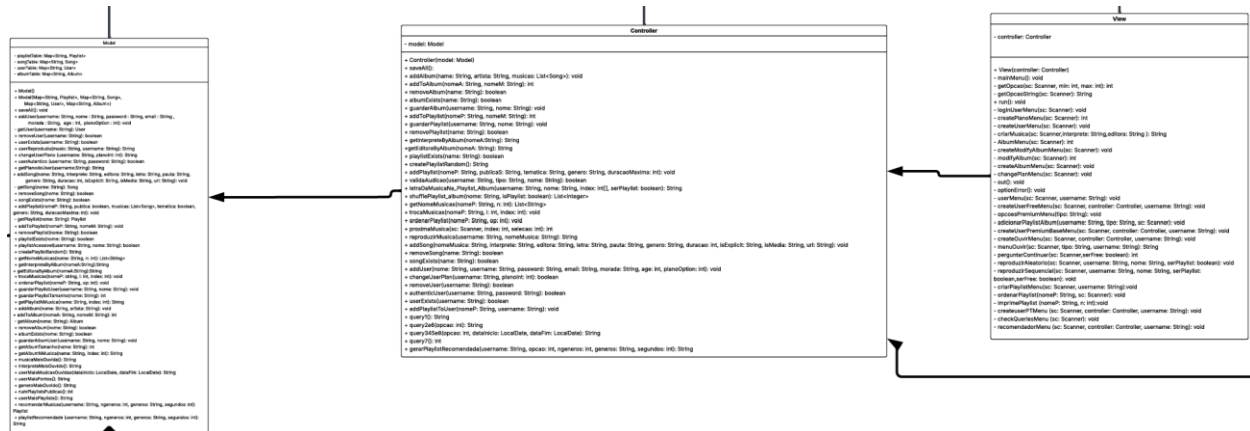


Figura 11 - MVC

## 4 Descrição da aplicação

A aplicação foi concebida como uma plataforma de gestão e reprodução musical, onde os utilizadores podem criar contas, ouvir músicas, criar playlists e álbuns, consultar estatísticas e muito mais. Toda a interação com o utilizador é realizada através de menus textuais, organizados e apresentados pela classe View.

A classe View é responsável por toda a interface da aplicação. Ela recebe input do utilizador através da consola e apresenta as opções disponíveis, navegando entre menus com base nas escolhas feitas. A View é construída com uma referência ao Controller, o que lhe permite aceder às funcionalidades da aplicação sem interagir diretamente com a Model, respeitando o padrão arquitetural MVC.

Entre as principais funcionalidades da View, destacam-se:

- *mainMenu()* e *run()*: gerem o menu principal da aplicação e o seu ciclo de execução, oferecendo opções como iniciar sessão, registar utilizadores e sair.
- *loginUserMenu(...)*: apresenta o menu de login, permitindo a autenticação do utilizador com username e password.
- *createUserMenu(...)*, *createUserFreeMenu(...)*, *createUserPremiumBaseMenu(...)*: permitem a criação de novos utilizadores e a utilização desses menus com diferentes planos de subscrição (Free, Premium Base, etc.).
- *userMenu(...)*: fornece acesso ao menu principal do utilizador autenticado, com opções para ouvir música, criar playlists, consultar estatísticas, entre outras.
- *createPlanoMenu(...)*: permite ao utilizador escolher o tipo de plano durante o registo.
- *createOuvirMenu(...)* e *menuOuvir(...)*: apresentam as opções disponíveis para ouvir músicas, playlists ou álbuns.
- *criarMusica(...)* e *createAlbumMenu(...)*: guiam o utilizador na criação de novas músicas e álbuns.
- *adicionarPlaylistAlbum(...)*: permite guardar playlists ou álbuns na conta do utilizador (dependendo do plano).
- Métodos como *playSong(...)*, *print(...)*, *out()* e *optionError()* tratam da reprodução de músicas, impressão de mensagens, saída da aplicação e tratamento de opções inválidas.

Através desta interface, o utilizador pode explorar todas as funcionalidades do sistema de forma guiada e intuitiva. O fluxo de menus foi desenhado para ser claro e modular, facilitando a navegação e o acesso a todas as operações disponíveis na aplicação.

```

BEM VINDO/A AO SPOTIFUM !
-----
1. Iniciar Sessão
2. Criar conta
3. Criar ou Modificar Álbum
4. Estatísticas da App
5. Mudar de plano
6. Fechar o programa
Prima o número correspondente à opção que deseja executar:

```

Figura 12- Menu de entrada do SPOTIFUM

```

BEM VINDO/A AO SPOTIFUM !
-----
1. Iniciar Sessão
2. Criar conta
3. Criar ou Modificar Álbum
4. Estatísticas da App
5. Mudar de plano
6. Fechar o programa
Prima o número correspondente à opção que deseja executar:
2
  Digite o seu Nome:
user1
  Crie um username único:
user
  Crie uma password:
pass1234
  Digite o seu Email:
email
  Digite a sua Morada:
morada
  Digite a sua Idade:
19

  Escolha o seu plano:
1.Plano Free
2.Plano Premium Base
3.Plano Premium Top
1
A sua conta foi criada com sucesso user1

```

Figura 13 - Exemplo da criação de um novo utilizador

## 4.1 UserFree

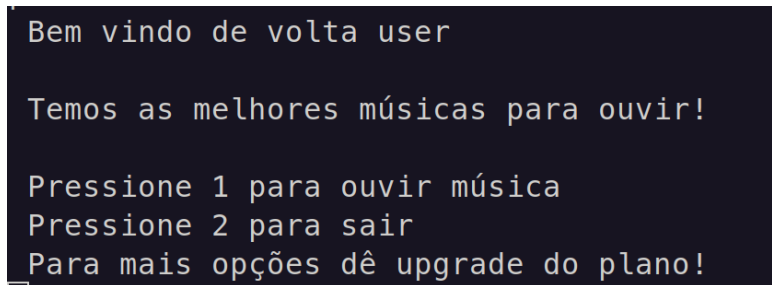
Os utilizadores com plano Free possuem acesso limitado às funcionalidades da aplicação, em comparação com os planos Premium. Uma das principais restrições impostas a estes utilizadores é a impossibilidade de escolher diretamente as músicas que pretendem ouvir. Em vez disso, podem apenas ouvir música em modo aleatório.



Para suportar esta limitação, foi criada a classe *PlaylistRandom*, uma subclasse de *Playlist*, que representa uma lista de músicas ordenadas aleatoriamente. Sempre que um utilizador *Free* pretende ouvir música, é chamada a função *createPlaylistRandom()*, que gera dinamicamente uma nova instância de *PlaylistRandom* contendo todas as músicas do sistema, dispostas por uma ordem aleatória. Esta playlist é criada em tempo de execução e utilizada apenas para aquela sessão de audição específica.

Como o objetivo desta playlist é apenas fornecer uma experiência de audição aleatória e temporária, ela não é considerada uma playlist formal criada pelo utilizador. Assim, cada nova tentativa de ouvir música por parte de um *UserFree* resulta numa nova *PlaylistRandom*, garantindo sempre uma sequência diferente de músicas.

Este mecanismo respeita as restrições funcionais dos utilizadores com plano *Free*, enquanto oferece uma experiência musical contínua e variada.



```
Bem vindo de volta user

Temos as melhores músicas para ouvir!

Pressione 1 para ouvir música
Pressione 2 para sair
Para mais opções dê upgrade do plano!
```

Figura 14 - Menu do UserFree

## 4.2 UserPremiumBase

Os utilizadores com o plano *PremiumBase* têm acesso a um conjunto de funcionalidades mais alargado em comparação com o plano *Free*. Para além de poderem ouvir música de forma aleatória, estes utilizadores podem também guardar álbuns na sua conta, criar playlists publicas ou privadas, permitindo que outros utilizadores as descubram e explorem.

Para implementar estas funcionalidades, optou-se por não criar uma entidade separada para representar uma biblioteca de conteúdos (como álbuns ou playlists guardadas). Em vez disso, todas as informações relacionadas com os conteúdos pessoais de um utilizador *Premium* foram encapsuladas dentro do próprio plano de subscrição (*PlanoPremiumBase*). Assim, cada instância de plano guarda internamente as playlists e os álbuns associados ao utilizador que o possui.

Esta abordagem apresenta várias vantagens, como a simplicidade da estrutura de dados, a centralização da lógica de acesso a conteúdos guardados e a possibilidade de diferenciar facilmente os comportamentos entre diferentes tipos de plano (por exemplo, *PremiumBase* vs. *PremiumTop*) através de herança e polimorfismo. A lógica de verificação e manipulação de

conteúdos guardados é, assim, gerida diretamente nas subclasses do plano, respeitando os princípios da programação orientada a objetos.

Desta forma, o *PlanoPremiumBase* não é apenas um identificador de permissões, mas uma entidade funcional que armazena e gere os conteúdos multimédia guardados pelos utilizadores com esse tipo de subscrição.

```
Bem vindo de volta user

|||||

Pressione 1 para ouvir

Pressione 2 para adicionar uma playlist à sua biblioteca

Pressione 3 para adicionar um album à sua biblioteca

Pressione 4 para criar uma playlist

Pressione 5 para sair
```

Figura 15 - Menu do UserPremiumBase

### 4.3 UserPremiumTop

O plano *PremiumTop* oferece aos utilizadores a experiência mais completa e personalizada da aplicação. Para além das funcionalidades disponíveis nos planos *Free* e *PremiumBase* — como ouvir música aleatória, criar e guardar playlists, guardar álbuns — os utilizadores *PremiumTop* têm acesso a duas funcionalidades exclusivas: playlists geradas automaticamente pela aplicação e um sistema de recomendação personalizado.

As playlists geradas automaticamente são criadas pela aplicação com base em critérios gerais, e estão disponíveis apenas para utilizadores com este plano. Adicionalmente, o utilizador pode recorrer ao recomendador inteligente, que permite criar uma playlist personalizada de acordo com as suas preferências. O processo é interativo: o utilizador é questionado para indicar critérios como a duração desejada da playlist, o(s) género(s) musical(is) preferido(s) e se pretende incluir ou excluir músicas com conteúdo explícito.

Com base nessas escolhas, a aplicação seleciona músicas do catálogo que correspondem aos critérios fornecidos e constrói uma playlist única e ajustada ao gosto do utilizador. Esta funcionalidade é possível graças à integração entre os dados do sistema (Model), a lógica de controlo (Controller) e o tratamento interativo feito na View.

Este tipo de personalização reforça o valor do plano PremiumTop, oferecendo uma experiência musical adaptada e exclusiva, diferenciando claramente este tipo de utilizador dos restantes perfis da aplicação.

```
Bem vindo de volta user  
||||||||||||||||||||||||||||||  
Pressione 1 para ouvir  
Pressione 2 para adicionar uma playlist à sua biblioteca  
Pressione 3 para adicionar um album à sua biblioteca  
Pressione 4 para criar uma playlist  
Pressione 5 para gerar uma playlist  
Pressione 6 para sair
```

Figura 16 - Menu do UserPremiumTop

## 5 Conclusão

Concluimos assim que foi desenvolvida uma aplicação sólida para a gestão de músicas, utilizadores e playlists. Esta permite o registo de diversos *user* com vários planos de subscrição cada um com os seus *perks*. Inclui funcionalidades essenciais como a criação de utilizadores, músicas, playlists e a geração automática de playlists para certos utilizadores.

Para além das funcionalidades atualmente implementadas, é possível considerar diversas melhorias futuras que visam otimizar o desempenho e tornar a aplicação mais modular. Uma dessas melhorias passa pela reorganização da aplicação em diferentes views, separando de forma mais clara a interface de cada tipo de utilizador ou funcionalidade (ex: uma view para gestão de álbuns, outra para estatísticas, etc.). Esta modularização facilitaria a manutenção do código, permitiria um crescimento mais estruturado da aplicação e potenciaria a reutilização de componentes.

Outra melhoria possível seria a introdução de mecanismos mais eficientes de gestão de dados em memória, como a utilização de estruturas de dados mais adequadas ou até cache local para evitar acessos redundantes aos mesmos objetos.

Consideramos que todos os requisitos propostos foram cumpridos, com a adoção de boas práticas de programação orientada a objetos, tratamento robusto de erros e persistência do estado da aplicação em ficheiro. O resultado final é uma ferramenta completa e versátil para a gestão de música e dos seus ouvistes.

## Diagrama UML

