



Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Programação Orientada aos Objetos

Ano Letivo de 2024/2025

Relatório

Francisco Castro
a104433

Nuno Silva
a99432

Miguel Diegues
a107361

Abril, 2025

PPO

Data da Receção	
Responsável	
Avaliação	
Observações	

Relatório

Francisco Castro
a104433

Nuno Silva
a99432

Miguel Diegues
a107361

Abril, 2025

Índice

1. Resumo	1
2. Introdução	2
3. Modelação do Diagrama de Classes	3
3.1. Planos de subscrição	3
3.2. Utilizadores	4
3.3. Playlists	5
3.4. Músicas	6
3.5. Álbuns	7
3.6. Gestor de Estatísticas	8
3.7. Parsing	8
3.8. Estado	9
3.9. MVC	9
3.10. Final	10
4. Conceito do MVC	11
5. Interface	12
6. Funcionalidades	13
6.1. Funcionamento geral	13
6.2. Início da aplicação	13
6.3. Interação com Utilizadores	13
6.4. Interação com Playlists	14
6.5. Interação com Músicas	16
6.6. Interação com Álbuns	17
6.7. Interação com Plano de Subscrição	18
6.8. Estatísticas	19
6.9. Salvaguarda do estado	19
7. Conclusão	20

1. Resumo

O presente trabalho tem como objetivo o desenvolvimento de uma aplicação de gestão de músicas, playlists, álbuns e planos de subscrição, denominada SpotifUM. A aplicação permite gerir diferentes tipos de músicas, como reprodução, criação de playlists ou álbuns e acompanhamento de estatísticas de uso pelos utilizadores. A aplicação oferece planos de subscrição Free, PremiumBase e PremiumTop, com diferentes funcionalidades e permissões associadas a cada tipo de plano. Através dessa estrutura, a aplicação pode personalizar a experiência do utilizador com base no seu plano, controlando o acesso a playlists personalizadas, reprodução de músicas aleatórias e acesso a conteúdos explícitos.

Foram implementadas funcionalidades para criar e gerir utilizadores, músicas, playlists e planos de subscrição. A aplicação também permite registar a reprodução de músicas e atualizar as playlists de acordo com as preferências e histórico do utilizador. Além disso, o sistema calcula estatísticas como o número de reproduções e os pontos acumulados, dependendo do plano de subscrição do utilizador.

A aplicação ainda permite gerar playlists automáticas com base nas preferências do utilizador, como tipo de música, gênero e tempo de reprodução. Para utilizadores Premium, playlists personalizadas podem ser criadas e organizadas, enquanto utilizadores Free têm acesso apenas a playlists aleatórias.

Para além das playlists, ainda é possível criar álbuns, adicionar músicas a álbuns já existentes ou então visualizar as músicas presentes num álbum, tal como reproduzir o próprio álbum.

O sistema foi desenvolvido utilizando os princípios de Programação Orientada a Objetos (POO), com uma clara separação entre a camada de Modelo, Vista e Controller, seguindo o padrão MVC. A aplicação garante a flexibilidade para futuras expansões e a robustez necessária para garantir uma experiência contínua, além de permitir o armazenamento e recuperação dos dados através de serialização.

2. Introdução

Este relatório tem como objetivo apresentar o desenvolvimento da aplicação SpotifUM, um sistema de gestão de músicas, playlists, álbuns e planos de subscrição, criado no âmbito da Unidade Curricular de Programação Orientada a Objetos (POO). O projeto foi desenvolvido com o intuito de aplicar os conceitos de POO na prática, garantindo que a aplicação fosse organizada, modular e facilmente escalável. Durante o desenvolvimento, foram adotados princípios fundamentais de design de software, com foco na reutilização e manutenção do código.

A aplicação permite aos utilizadores gerir músicas, playlists e álbuns de forma personalizada, além de oferecer diferentes planos de subscrição que influenciam a experiência do utilizador. Através de planos como Free, PremiumBase e PremiumTop, os utilizadores podem aceder a funcionalidades variadas, como playlists automáticas, personalização de conteúdo e cálculos de pontos baseados na reprodução de músicas.

O objetivo deste trabalho foi a implementação de um sistema robusto que, além de gerenciar os dados dos utilizadores, possibilita a personalização e o controlo da experiência musical dentro da aplicação. Ao longo deste relatório, serão discutidas as escolhas de design feitas, a arquitetura adotada e os desafios enfrentados, assim como as soluções aplicadas para garantir que todos os requisitos do projeto fossem atendidos.

3. Modelação do Diagrama de Classes

A aplicação tem presente vários modelos, cada um com características únicas. Todos os modelos foram implementados de forma a se manter uma expansão de funcionalidades sustentável do programa.

3.1. Planos de subscrição

No SpotifUM, a gestão dos planos de subscrição é implementada utilizando o princípio da herança. A classe PlanoSubscricao é a superclasse que define os atributos e comportamentos comuns a todos os tipos de planos de subscrição. Ela contém informações como o nome do plano e a quantidade de pontos acumulados pelas reproduções de músicas.

As subclasses PlanoFree, PlanoPremiumBase e PlanoPremiumTop estendem a classe PlanoSubscricao e implementam ou sobreescrevem os métodos e comportamentos de acordo com as regras específicas de cada plano. Por exemplo, o cálculo de pontos é diferente para cada plano, e algumas permissões, como a capacidade de criar playlists personalizadas, estão restritas apenas aos planos Premium.

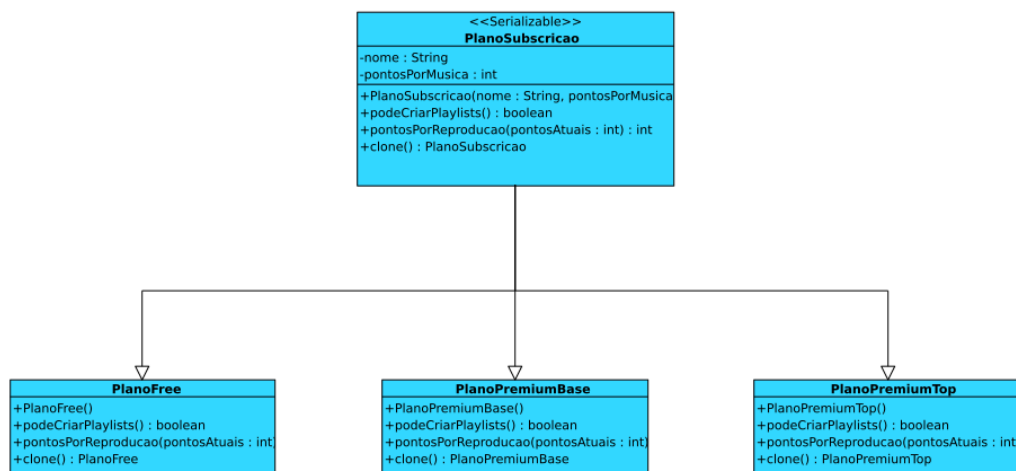


Figura 1: Plano Subscricao

Esta estrutura permite uma organização clara e extensível dos diferentes planos de subscrição, garantindo que as funcionalidades comuns sejam centralizadas na superclasse PlanoSubscricao, enquanto as diferenças específicas de cada plano são tratadas nas suas respectivas subclasses.

Para complementar a hierarquia de classes que representa os diferentes tipos de PlanoSubscricao, o sistema inclui a classe utilitária GestorPlanos, responsável por centralizar a lógica associada à gestão dos planos de subscrição dos utilizadores. Esta classe não representa um plano em si, mas disponibiliza métodos auxiliares para consultar os planos disponíveis, apresentando os respetivos tipos e benefícios, visualizar o plano atual de um utilizador e permitir-lhe alterar o seu plano para outro à sua escolha.

GestorPlanos
+consultarPlanosDisponiveis() : void
+verPlanoAtual(u : Utilizador) : void
+alterarPlanoAtual(u : Utilizador, opcao : int) : void

Figura 2: Gestor de Planos

3.2. Utilizadores

A classe Utilizador representa os utilizadores da aplicação SpotifUM e é uma parte fundamental do sistema, pois armazena as informações e comportamentos relacionados aos perfis dos utilizadores. Cada Utilizador tem informações básicas, como nome, email, morada, password e pontos acumulados com base nas interações da aplicação. Para além disso, está associado a um plano de subscrição, que define as permissões e benefícios de que usufrui no sistema.

Adicionalmente, o utilizador possui uma coleção de músicas associadas, representando aquelas com que interagiu, como por exemplo músicas reproduzidas. Estas músicas são utilizadas, entre outras finalidades, para o cálculo do total de reproduções efetuadas pelo utilizador. Para complementar a experiência musical, o utilizador também pode manter uma biblioteca pessoal de álbuns e playlists, que pode ser gerida através de métodos próprios que permitem adicionar novas entradas de forma controlada. A possibilidade de criar e gerir playlists, contudo, está reservada apenas a utilizadores com planos Premium, conforme definido pelas regras de cada plano.

<<Serializable>> Utilizador
-nome : String -email : String -morada : String -password : String -pontos : int -plano : PlanoSubscricao -musicas : List<Musica> -bibliotecaAlbuns : List<Album> -bibliotecaPlaylists : List<Playlist>
+Utilizador(nome : String, morada : String, email : String, password : String, pontos : int) +adicionarAlbumABiblioteca(a : Album) : void +adicionarPlaylistABiblioteca(p : Playlist) : void +adicionarPontos(pontos : int) : void +adicionarMusica(musica : Musica) : void +podeCriarPlaylists() : boolean +getTotalReproducoes() : int

Figura 3: Utilizador

Para complementar a estrutura que representa os diferentes perfis de utilizadores e os seus comportamentos, o sistema inclui a classe utilitária GestorUtilizadores, responsável por centralizar a lógica relacionada à gestão dos utilizadores da aplicação SpotifUM. Esta classe não representa um utilizador em si, mas fornece os mecanismos necessários para criar, autenticar, consultar e atualizar os dados dos utilizadores registados. Através de um mapeamento eficiente por email, o GestorUtilizadores garante o registo único de cada perfil, assegurando integridade e consistência na gestão dos dados.

Além de permitir o registo de novos utilizadores com associação imediata a um plano de subscrição (Free, Premium Base ou Premium Top), esta classe oferece funcionalidades como login com validação de credenciais, atualização de planos, verificação de existência e exibição dos perfis ativos. Assim, o GestorUtilizadores atua como intermediário entre a lógica de aplicação e os dados dos utilizadores, promovendo uma arquitetura modular, clara e de fácil manutenção.

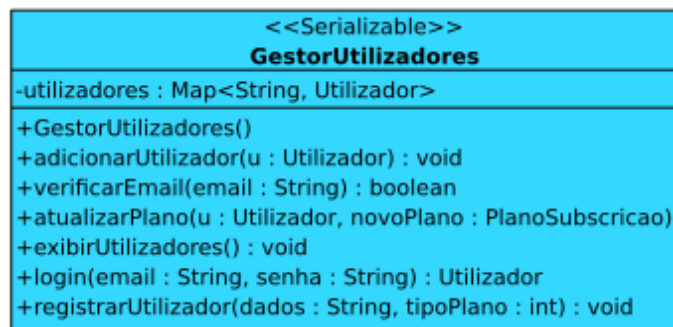


Figura 4: Gestor de Utilizadores

3.3. Playlists

A classe Playlist funciona como uma superclasse que representa o conceito geral de uma playlist, armazenando as características comuns a todas as playlists, como o nome da playlist, o seu criador e a lista de músicas que ela contém. Através dessa classe base, conseguimos agrupar comportamentos comuns, como adicionar/remover músicas à/da playlist, exibir as músicas contidas nela e reproduzir a playlist completa, bem como se é uma playlist pública ou privada.

A partir da classe Playlist, foram criadas três subclasses, cada uma representando uma variação da playlist dependendo do plano de subscrição do utilizador. Essas subclasses implementam comportamentos específicos para atender às necessidades de cada tipo de plano, permitindo que as playlists se comportem de maneira diferente com base nas permissões do utilizador. Estas subclasses são PlaylistAleatoria, PlaylistFavoritos e PlaylistPersonalizada.

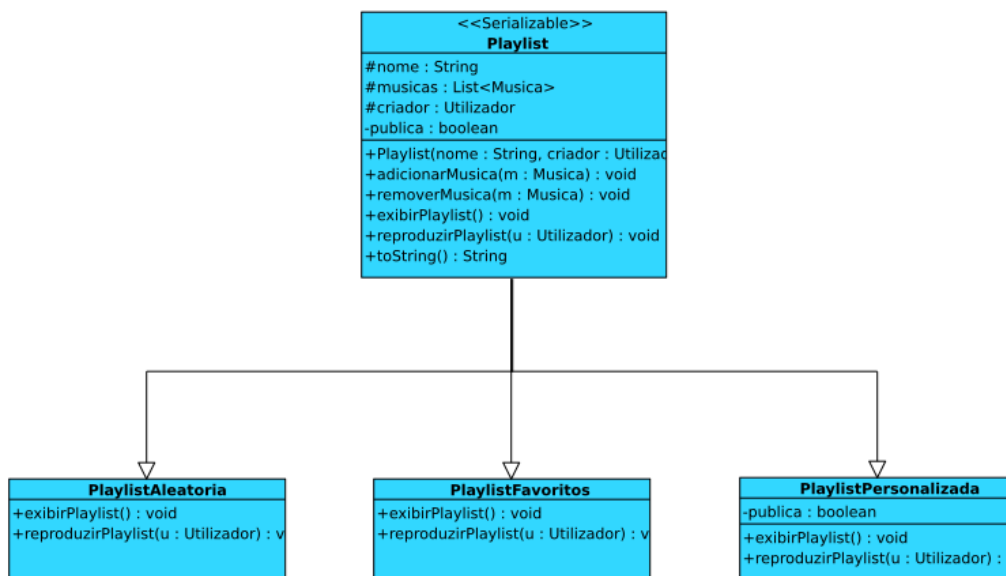


Figura 5: Playlist

Esta classe também retém o seu gestor, neste caso a classe utilitária GestorPlaylists. Esta classe centraliza toda a lógica associada à criação, manipulação e disponibilização de playlists, funcionando como um elo entre os utilizadores e as coleções musicais personalizadas ou geradas automaticamente.

Esta classe não corresponde a uma playlist específica, mas sim à infraestrutura que organiza e disponibiliza estas estruturas de forma eficiente. Através de mapeamentos internos, permite associar playlists aos respetivos criadores, distinguir entre coleções públicas e privadas, e garantir o acesso controlado às funcionalidades consoante o plano de subscrição do utilizador (por exemplo, apenas utilizadores Premium podem criar ou gerar playlists).

Entre as suas principais funcionalidades destacam-se a criação de playlists manuais e automáticas, podendo ser por género musical, por tempo de reprodução ou com base nos favoritos do utilizador, bem como a incorporação de playlists públicas por outros utilizadores, a reprodução e visualização de conteúdos e a gestão de músicas dentro das playlists. Esta classe contribui, assim, para uma experiência musical rica, flexível e adaptada ao perfil de cada utilizador, promovendo uma utilização intuitiva e organizada do sistema.



Figura 6: Gestor de Playlist

3.4. Músicas

A classe Musica representa uma música individual com diversos atributos que descrevem sua identidade (nome, intérprete, duração,...), bem como características específicas como a letra e a classificação de conteúdo explícito. As músicas podem ser associadas a playlists, álbuns e utilizadores, e a classe oferece métodos para manipular e exibir essas informações ao utilizador. Assim como ainda permite incrementar o número de reproduções da música e que um utilizador reproduza uma música através do seu nome e do seu intérprete.

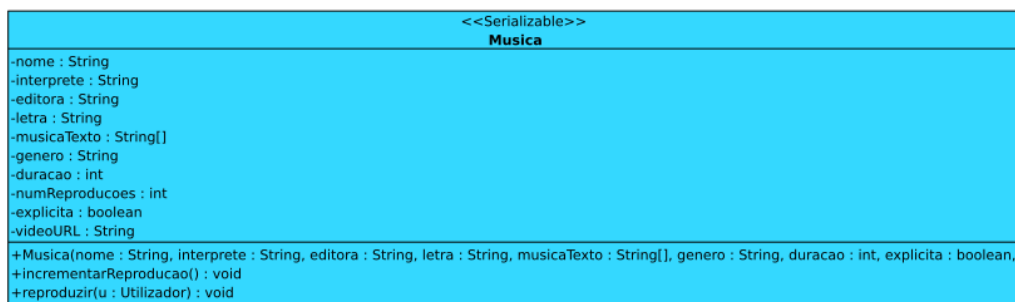


Figura 7: Musica

A classe utilitária GestorMusicas assume um papel central na gestão e organização das músicas disponíveis no sistema. Esta classe não representa uma música individual, mas sim o repositório geral que armazena, indexa e disponibiliza todas as instâncias da classe Musica, assegurando um acesso eficiente e estruturado a estas entidades.

Funcionando como uma camada intermediária entre as entidades musicais e as ações realizadas sobre elas, o GestorMusicas permite registar novas músicas, aceder a músicas específicas por nome ou por combinação de nome e intérprete, verificar a existência de uma determinada música e obter listas completas de todas as músicas registadas. Além disso, oferece suporte à reprodução de músicas, encapsulando a lógica de interação entre uma música e o utilizador que a ouve — incluindo o sistema de atribuição de pontos consoante o plano de subscrição.

<<Serializable>>
GestorMusicas
-musicas : Map<String, Musica>
+GestorMusicas() +adicionarMusica(musica : Musica) : void +verificarMusica(nome : String, interprete : String) : boolean +reproduzirMusica(u : Utilizador, m : Musica) : void

Figura 8: Gestor de Musicas

3.5. Álbuns

A classe Album representa um conjunto organizado de músicas pertencentes a um mesmo artista. Cada álbum é identificado por atributos essenciais como o título, o nome do artista e a lista de músicas que o compõem. A classe permite a adição e remoção de músicas, bem como a reprodução sequencial de todas as faixas contidas no álbum, delegando essa funcionalidade a cada instância de Musica. Além disso, um álbum pode ser associado a um ou mais utilizadores, permitindo que os mesmos ouçam e interajam com as músicas de forma organizada.

<<Serializable>>
Album
-titulo : String -artista : String -musicas : List<Musica>
+Album(titulo : String, artista : String) +adicionarMusica(musica : Musica) : void +removerMusica(musica : Musica) : void +reproduzir(utilizador : Utilizador) : void +toString() : String

Figura 9: Album

Para complementar os álbuns, temos a classe utilitária GestorAlbuns que não representa um álbum individual, mas sim a infraestrutura que organiza, armazena e disponibiliza múltiplos álbuns existentes no sistema, permitindo uma gestão eficiente e modular destas coleções.

Cada álbum é identificado pelo seu título e associado a uma coleção de músicas, podendo ser criado, atualizado ou consultado através das funcionalidades oferecidas pelo gestor. O GestorAlbuns assegura o correto mapeamento entre os títulos dos álbuns e as suas instâncias correspondentes, permitindo a adição dinâmica de faixas, a recuperação de álbuns já existentes e a listagem das músicas contidas em cada um.

Adicionalmente, esta classe facilita a interação entre os utilizadores e os álbuns, permitindo a reprodução sequencial de um álbum por parte de um utilizador específico. Essa funcionalidade assegura que as métricas de utilização e os dados de reprodução possam ser registados corretamente, em conformidade com o plano de subscrição de cada utilizador.

<<Serializable>>
GestorAlbuns
-albuns : Map<String, Album>
+GestorAlbuns() +adicionarMusicaAlbum(nomeAlbum : String, m : Musica) +exibirMusicasDeAlbum(nomeAlbum : String) : void +exibirAlbuns() : void +reproduzirAlbum(u : Utilizador, titulo : String) : void

Figura 10: Gestor de Albuns

3.6. Gestor de Estatísticas

A classe GestorEstatisticas assume o papel de componente analítico do sistema, oferecendo funcionalidades que permitem consultar diferentes métricas relacionadas com a atividade dos utilizadores, o comportamento das músicas e o estado das playlists. Ao ser construída com uma referência ao Estado e ao GestorPlaylists, consegue aceder a todos os dados necessários para calcular estatísticas relevantes e úteis ao longo da execução.

Através das operações implementadas nesta classe podemos obter variados resultados, tais como: música mais reproduzida, intérprete mais escutado, género musical mais reproduzido, etc.. Para além disso, a classe também permite alterar dinamicamente o plano de subscrição de um utilizador, oferecendo diferentes opções que refletem a hierarquia de planos existente no sistema.

GestorEstatisticas
-gp : GestorPlaylists
-estado : Estado
+GestorEstatisticas(estado : Estado, gp : GestorPlaylists)
+musicaMaisReproduzida() : Musica
+interpreteMaisEscutado() : String
+utilizadorMaisReproducoes() : Utilizador
+utilizadorMaisPontos() : Utilizador
+generoMaisReproduzido() : String
+utilizadorMaisPlaylists() : Utilizador
+alterarPlanoAtual(utilizador : Utilizador, opcaoPlano : int)
+exibirEstatisticas(estado : Estado) : void
+numeroPlaylistsPublicas() : int

Figura 11: Gestor de Estatísticas

Com um design orientado à análise, esta classe permite centralizar toda a lógica de cálculo estatístico num único ponto, mantendo os restantes componentes do sistema mais limpos e especializados nas suas responsabilidades principais. Esta separação de preocupações torna o código mais modular e facilita futuras extensões ou alterações ao sistema de estatísticas sem comprometer outras áreas da aplicação.

3.7. Parsing

O objetivo principal desta classe seria efetuar o carregamento e interpretação dos dados provenientes de ficheiros de textos (txt), inseridos na pasta input/. Estes dados que correspondem a utilizadores, músicas, playlists e álbuns são convertidos em objetos, que são posteriormente registados no estado da aplicação.

Parsing
+parserUtilizador(linha : String, tipoPlano : int) : Utilizador
+parseUtilizadoresInput(estado : Estado) : List<Utilizador>
+parserMusica(linha : String) : Musica
+parseMusicasInput(estado : Estado) : List<Musica>
+parserPlaylist(linha : String, estado : Estado) : Playlist
+parsePlaylistsInput(estado : Estado) : List<Playlist>
+parserAlbum(linha : String, estado : Estado) : Album
+parseAlbunsInput(estado : Estado) : List<Album>

Figura 12: Parsing

Além do carregamento dos dados, esta classe é também responsável pela criação, atualização e remoção de objetos no estado da aplicação. Estas operações são executadas com base em instruções contidas nos ficheiros de input, sendo processadas de forma semelhante entre si: identificando o tipo de entidade, localizando-a (caso exista) e aplicando a ação correspondente.

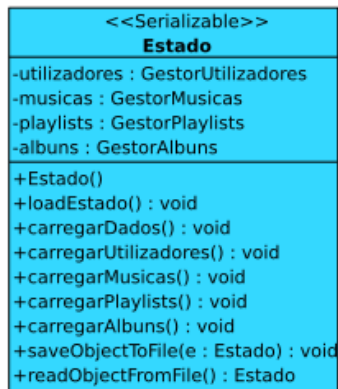
Durante todo o processo, são aplicados mecanismos de validação para garantir a integridade dos dados. Situações como formatos inválidos, campos obrigatórios em falta ou entidades não encontradas são

tratadas de forma a evitar comportamentos inesperados, sendo ignoradas ou devidamente registadas para análise posterior.

3.8. Estado

A classe estado representa o pilar da aplicação, funcionando como o ponto central onde reside toda a informação relevante ao longo da execução. É através desta classe que se mantém em memória os dados estruturados de utilizadores, músicas, playlists, álbuns e estatísticas, cada um gerido por componentes próprios (os respetivos gestores), mas acessíveis de forma unificada. Sempre que o programa é iniciado, o Estado tenta recuperar a última versão guardada desses dados a partir de um ficheiro de armazenamento persistente. Se não for possível restaurar esse estado anterior, então os dados são lidos dos ficheiros de texto localizados na pasta de entrada e convertidos para objetos válidos que passam a integrar o estado atual.

Esta centralização do acesso e gestão dos dados garante uma maior coesão e organização do projeto, permitindo que outras partes da aplicação interajam com os dados de forma simples e consistente. Além disso, a classe está preparada para guardar o estado atualizado em disco sempre que ocorrem alterações, o que assegura a persistência entre sessões e evita perdas de informação.



O tratamento de erros é igualmente cuidado: problemas durante o carregamento ou escrita do estado são detetados e comunicados de forma clara, permitindo que o programa prossiga com uma inicialização limpa sempre que necessário. Esta capacidade de adaptação e fiabilidade reforça ainda mais o papel do Estado como elemento estruturante e essencial ao bom funcionamento da aplicação.

3.9. MVC

A arquitetura do SpotifUM assenta na adoção clara do padrão Model-View-Controller (MVC), permitindo uma separação estruturada entre a lógica de dados, a apresentação e o controlo da aplicação. Neste contexto, as classes Interactive e Controlador assumem papéis distintos mas complementares dentro da camada de controlo, sendo ambas fundamentais para garantir a ligação eficaz entre o utilizador e os dados manipulados pelo sistema.

A classe Interactive, localizada no pacote menus, assume a responsabilidade de gerir o ciclo de vida da aplicação do ponto de vista do utilizador. É esta classe que controla a execução do programa desde o momento em que é iniciado, apresentando os diferentes menus com base no estado atual do sistema e nas permissões do utilizador autenticado. Atua como o elo direto entre o terminal e o sistema interno, captando as escolhas do utilizador através da classe Menu e estruturando a navegação entre diferentes funcionalidades. Contudo, esta classe não executa diretamente operações sobre os dados nem toma decisões de negócio; em vez disso, delega essas responsabilidades à classe Controlador.

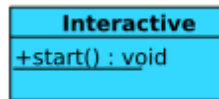


Figura 14: Interactive

A classe Controlador é a verdadeira ponte entre a camada de apresentação e o modelo. Ao receber os pedidos vindos da Interactive, é ela que interpreta as intenções do utilizador e executa as ações apropriadas sobre os gestores de domínio. Para tal, o Controlador mantém acesso direto ao estado global da aplicação, através do objeto Estado, que encapsula todas as estruturas relevantes do modelo. Assim, o Controlador serve como um encapsulador de lógica de mediação e encaminhamento, assegurando que a camada de apresentação nunca interage diretamente com o modelo, cumprindo rigorosamente a separação imposta pelo padrão MVC.



Figura 15: Controlador

A colaboração entre as classes Interactive e Controlador é fundamental na camada Controller do SpotifUM, assegurando a separação entre a lógica da aplicação e a interface com o utilizador. A Interactive gere o fluxo da aplicação e a interação com o utilizador, enquanto o Controlador executa as ações solicitadas, comunicando com os gestores do modelo. Esta divisão permite um sistema modular, mais fácil de testar, manter e evoluir, mantendo a lógica de negócio isolada da apresentação e assegurando a clareza estrutural imposta pelo padrão MVC.

3.10. Final

Finalmente, foram estabelecidas todas as relações entre as várias representações apresentadas. O diagrama completo, que não é apresentável no espaço reservado para este ficheiro, ficará disponibilizado como anexo externo na mesma diretoria do relatório atual.

4. Conceito do MVC

No SpotifUM, optámos por usar o padrão Model-View-Controller (MVC), uma abordagem amplamente utilizada no desenvolvimento de software que promove a separação de responsabilidades entre diferentes componentes do sistema. Esta divisão permite tornar o código mais modular, reutilizável e fácil de manter. No contexto deste projeto, o modelo MVC foi adaptado à estrutura do sistema da seguinte forma:

Na camada **Model**, concentramos toda a lógica e estrutura de dados: as classes Utilizador, Musica, Playlist, Album e PlanoSubscricao (com as respetivas subclasses) — definem o que é cada objeto, enquanto os gestores (GestorUtilizadores, GestorMusicas, GestorPlaylists, GestorAlbuns, GestorPlanos) tratam de criar, consultar e atualizar esses objetos, tal como disponibilizar funções de acesso e manipulação dos dados ao restante sistema.

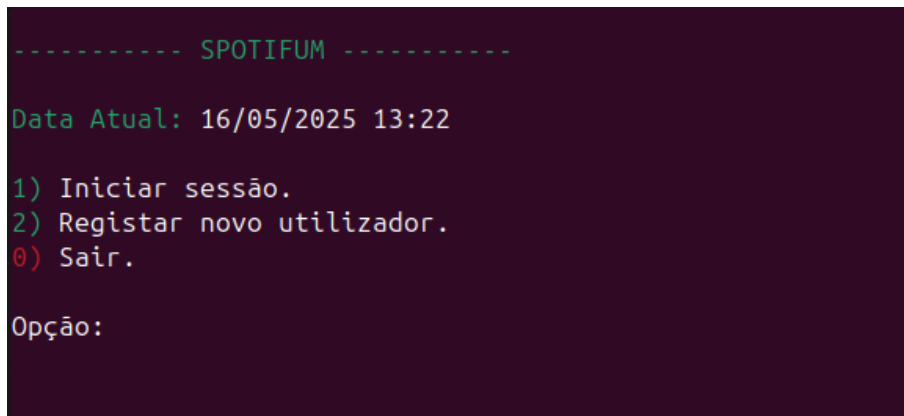
A camada **Controller** atua como intermediária entre os dados e a interface de interação, sendo composto pelas classes Interactive e Controlador. A Interactive gere o loop de execução, apresentando o menu adequado e captando a escolha do utilizador. Depois, passa essa escolha ao Controlador, que vai chamar o gestor certo do Model, obter os resultados (seja uma lista de playlists ou o sucesso de uma operação) e devolvê-los à Interactive, para que o menu final os apresente de forma clara.

Na camada **View** esta é responsável pela formatação e apresentação dos resultados ao utilizador. Dentro dela temos unicamente a classe Menu, cuja única função é apresentar ao utilizador as opções disponíveis e receber as entradas de forma textual. Todos os métodos de Menu — desde menuInicial até aos submenus de utilizador, playlists, músicas, álbuns e planos. O seu único propósito é exibir texto, tornando-se assim o ponto de contacto entre o utilizador e o resto da aplicação.

Desta forma, mantemos cada peça do SpotifUM no seu lugar: a lógica de dados está isolada, a camada de apresentação concentra-se exclusivamente na interface, e o controller é o único a entrar em contacto com ambos. Isso torna o código muito mais simples de evoluir e testar.

5. Interface

A nossa interface segue um estilo de interação com a aplicação através da escolha de opções. Tal como é possível observar na imagem seguinte, existem várias opções para o utilizador escolher, sendo que existe uma função responsável por desenhar cada tipo de interface.

A terminal window with a dark purple background and light green text. The text displays the title 'SPOTIFUM' between two dashed lines, the current date and time 'Data Atual: 16/05/2025 13:22', a numbered list of three options: '1) Iniciar sessão.', '2) Registar novo utilizador.', and '0) Sair.', and a prompt 'Opção:' at the bottom.

```
----- SPOTIFUM -----  
  
Data Atual: 16/05/2025 13:22  
  
1) Iniciar sessão.  
2) Registar novo utilizador.  
0) Sair.  
  
Opção:
```

Figura 16: Interface Menu Inicial

6. Funcionalidades

6.1. Funcionamento geral

A aplicação segue uma abordagem sequencial através de menus de linha de comando, que nos permite navegar pelas restantes funcionalidades da aplicação. Esta estrutura de menus simplifica a interação do utilizador com o sistema.

6.2. Início da aplicação

Para garantir que a aplicação seja construída e executada de forma correta, o processo de compilação e execução foi automatizado utilizando a ferramenta Make. O uso de Make permite a gestão eficiente das etapas de compilação, limpeza e execução, garantindo que todos os arquivos necessários sejam criados e que o estado da aplicação seja sempre limpo antes de iniciar uma nova execução.

6.3. Interação com Utilizadores

Para adicionar utilizadores, terão de ser dados como input ao programa algumas características, como no exemplo abaixo:

```
----- REGISTAR NOVO UTILIZADOR -----  
Insira os dados no formato: nome;email;morada;password  
nuno;nuno@email.com;Guimaraes;nunopass
```

Figura 17: Registo de utilizador

De seguida, temos que especificar o tipo de plano de subscrição:

```
Escolha o tipo de plano:  
1) Free  
2) Premium Base  
3) Premium Top
```

Figura 18: Registo de utilizador 2

Após isso teremos a confirmação de sucesso do registo do Utilizador ou a falha do mesmo.

Antes de falar sobre todas as funcionalidades do nosso programa este é o menu principal da nossa aplicação:

```
Bem-vindo, nuno!

----- MENU UTILIZADOR -----

1) Playlists.
2) Musicas.
3) Albuns.
4) Ver planos de subscrição.
5) Consultar estatísticas.
0) Terminar sessão.
Opção: █
```

Figura 19: Menu Principal

6.4. Interação com Playlists

```
----- GERIR PLAYLISTS -----

1) Criar nova playlist.
2) Reproduzir playlist.
3) Adicionar música a uma playlist.
4) Remover música de uma playlist.
5) Gerar uma playlist automática.
6) Ver todas as playlists.
7) Ver músicas de uma playlist.
8) Ver biblioteca pessoal de playlists.
9) Guardar playlist pública na biblioteca.
10) Remover playlist da biblioteca (não és o criador).
0) Voltar
```

Figura 20: Menu Playlists

Para a criação de uma nova Playlist apenas é necessário inserir o nome desejado para a criação da mesma e decidir se este é pública ou privada. Para adicionar uma música a uma Playlist é aberto um menu com opções das músicas existentes, após selecionar a música pretendida é necessário o nome da Playlist, o mesmo se aplicando à remoção de uma música de uma Playlist. Para gerar uma Playlist automática o utilizador pode escolher se quer uma Playlist de Favoritos, Por género e tempo ou Apenas explícitas.

```
Tornar playlist pública? (s/n): n
Playlist{nome='teste', numMusicas=0, public= false}
✓ Playlist 'teste' criada. Pública? false
Pressione Enter para continuar...
```

Figura 21: Criar Playlist

```
----- GERAR PLAYLIST AUTOMÁTICA -----
1) Favoritos
2) Por gênero e tempo
3) Apenas explícitas
Opção: 2
Gênero musical: Rock
Duração máxima (segundos): 500
Nome da nova playlist: Rock
✓ Playlist 'Rock' (Rock, ≤500s) gerada com 2 músicas.
Pressione Enter para continuar...
```

Figura 22: Gerar Playlist Automática

Na opção de ver todas as Playlists é mostrada uma lista que apresenta todas as playlists criadas e o seu criador. Para ver as músicas de uma Playlist apenas é preciso especificar o nome da Playlist e aparecerão todas as músicas que contém essa Playlist.

```
----- GERIR PLAYLISTS -----
1) Criar nova playlist.
2) Reproduzir playlist.
3) Adicionar música a uma playlist.
4) Remover música de uma playlist.
5) Gerar uma playlist automática.
6) Ver todas as playlists.
7) Ver músicas de uma playlist.
8) Ver biblioteca pessoal de playlists.
9) Guardar playlist pública na biblioteca.
10) Remover playlist da biblioteca (não és o criador).
0) Voltar

6
ChillHits (criador: miguel)
Top10 (criador: ines)
MIW (criador: nuno)
Pressione Enter para continuar...
```

Figura 23: Ver Todas as Playlist

Por fim, as últimas três opções servem para visualizar as playlist possuídas pelo utilizador e para guardar/remover qualquer playlist pública existente na sua biblioteca de utilizador.

```
Biblioteca de nuno
· MIW (PlaylistPersonalizada)
· teste (PlaylistPersonalizada)
· Rock (PlaylistPersonalizada)
Pressione Enter para continuar...
```

Figura 24: Biblioteca do Utilizador

6.5. Interação com Músicas

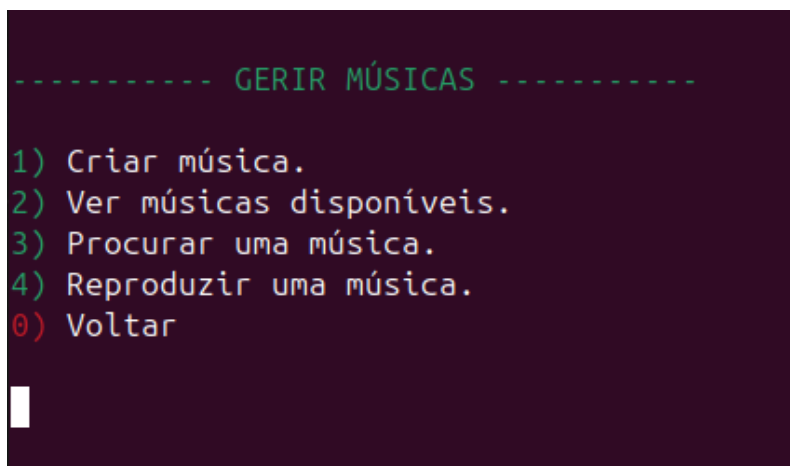
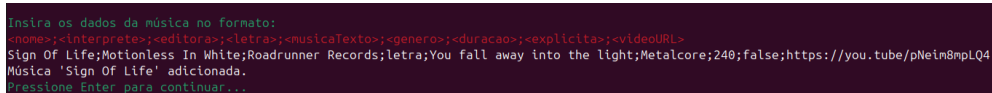


Figura 25: Menu Músicas

Neste menu temos diferentes formas de manipular e visualizar músicas. Por exemplo podemos criar uma nova música para a aplicação inserindo uma sequência de parâmetros válidos, tal como é possível visualizar na imagem abaixo.



```
Insira os dados da música no formato:  
<nome>;<interprete>;<editora>;<letra>;<musicaTexto>;<genero>;<duracao>;<explicita>;<videoURL>  
Sign Of Life;Motionless In White;Roadrunner Records;letra;You fall away into the light;Metalcore;240;false;https://you.tube/pNein8mpLQ4  
Música 'Sign Of Life' adicionada.  
Pressione Enter para continuar...
```

Figura 26: Criar Música

Ao escolher a opção para ver ou procurar músicas disponíveis, irá aparecer uma lista de géneros onde após a escolha aparecerá as músicas correspondentes a esse género que foram carregadas por um ficheiro txt e que estão no sistema. Ao procurar uma música precisas de selecionar a música que andas a procura dentro das opções e após aparecerá todas as informações acerca dessa música.

Por fim, ao reproduzir uma música, também irá aparecer uma lista de géneros seguido de uma lista de músicas do género escolhido, e ao selecionar a opção pretendida teremos a confirmação de que a música foi reproduzida e informará o utilizador de quantos pontos este ganhou e os seus pontos totais serem mostrados.

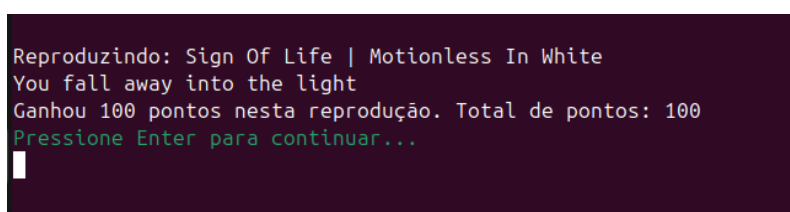


Figura 27: Reproduzir Música

6.6. Interação com Álbuns

```
----- GERIR ÁLBUNS -----  
1) Criar álbum.  
2) Adicionar música a um álbum.  
3) Ver músicas de um álbum.  
4) Reproduzir álbum.  
0) Voltar
```

Figura 28: Menu Álbuns

Em relação aos álbuns, é possível criar e adicionar músicas, onde na criação de um novo álbum é pedido ao utilizador para inserir o título e o artista dono desse álbum. Já para adicionar uma nova música primeiro é pedido para escolher o álbum ao qual será adicionado, posteriormente será mostrado uma lista de músicas do artista do álbum, na qual feita uma escolha a música será adicionada ao álbum. Para ver uma música funciona de forma semelhante, mas ao selecionar o álbum pretendido, exibirá de imediato as músicas presentes nesse álbum.

```
----- CRIAR ÁLBUM -----  
Insira os dados no formato: titulo;artista  
Reincarnate;Motionless In White  
Álbum 'Reincarnate' criado.  
Pressione Enter para continuar...
```

Figura 29: Criar Álbum

Por fim podemos reproduzir um álbum. O programa irá reproduzir todas as músicas dentro do álbum e contabilizar os pontos para o utilizador.

```
🎧 Reproduzindo álbum: Infamous de Motionless In White  
Reproduzindo: Sinematic | Motionless In White  
Sinematic scenes replay my mind  
Fighting battles left behind  
Every frame a scar so deep  
In this darkness I can't sleep  
Ganhou 102 pontos nesta reprodução. Total de pontos: 202  
Pressione Enter para continuar...  
█
```

Figura 30: Reproduzir Álbum

6.7. Interação com Plano de Subscrição

```
----- PLANOS SUBSCRIÇÃO -----  
  
1) Consultar planos.  
2) Ver plano atual.  
3) Alterar plano atual.  
0) Voltar  
  
█
```

Figura 31: Menu Plano Subscrição

Neste menu temos opções mais simples onde podemos consultar a lista de todos os planos de subscrição já mencionados anteriormente, onde também informará a quantidade de pontos recebidos ao reproduzir uma música.

```
----- PLANOS SUBSCRIÇÃO -----  
  
1) Consultar planos.  
2) Ver plano atual.  
3) Alterar plano atual.  
0) Voltar  
  
1  
----- PLANOS DISPONÍVEIS -----  
1) Plano Free - Pontos por música: 5  
2) Plano Premium Base - Pontos por música: 10  
3) Plano Premium Top - Pontos por música: 100 + 2.5% dos pontos atuais  
-----  
Pressione Enter para continuar...
```

Figura 32: Consultar Planos

Por outro lado, temos mais duas opções. Uma para ver o plano atual do utilizador e outra onde possibilita alterar o plano à escolha, se tiver sucesso haverá uma mensagem de confirmação.

```
----- PLANOS SUBSCRIÇÃO -----  
  
1) Consultar planos.  
2) Ver plano atual.  
3) Alterar plano atual.  
0) Voltar  
  
2  
Plano atual: PremiumTop (Pontos por música: 100)  
Pressione Enter para continuar...  
  
█
```

Figura 33: Ver Plano Atual

6.8. Estatísticas

Relativamente às estatísticas, ao entrar nesta opção o utilizador deparar-se-á com uma tela onde teremos várias informações em relação ao programa, onde este analisa os seus dados guardados e exibe diversas estatísticas, desde a música mais reproduzida, até aos pontos do próprio utilizador.

```
Música mais reproduzida: Sign Of Life
Intérprete mais escutado: Motionless In White
Utilizador que mais reproduziu músicas: bruno
Utilizador com mais pontos: nuno
Tipo de música mais reproduzido: Metalcore
Número de playlists públicas: 3
Utilizador com mais playlists: nuno
Pontos do utilizador nuno: 202

Pressione Enter para continuar...
```

Figura 34: Exibição de Estatísticas

6.9. Salvaguarda do estado

Para guardar o estado do programa em disco, recorreremos à funcionalidade de serialização de classes do Java. Para tal, foi criado um **Estado**, que é responsável por criar (caso ainda não exista) estado.obj e guardar as informações relativas ao programa (Utilizadores, Músicas, Playlists, Álbuns). Na classe Estado, também está definido um método para ler o estado do ficheiro, e carregá-lo no sistema. Com esta funcionalidade, o utilizador pode reter os seus dados. Também é possível popular a aplicação com utilizadores, músicas, playlists e álbuns, que é feita através de um ficheiro txt, que é carregado quando iniciámos a aplicação pela primeira vez sendo que depois é guardado em objeto no estado.

```
Bem-vindo, nuno!

----- MENU UTILIZADOR -----

1) Playlists.
2) Musicas.
3) Albuns.
4) Ver planos de subscrição.
5) Consultar estatísticas.
0) Terminar sessão.
Opção: 0
Sessão terminada.
Estado guardado com sucesso.
```

Figura 35: Estado Guardado

7. Conclusão

Em suma, o desenvolvimento do SpotifUM permitiu consolidar diversos conceitos fundamentais da programação orientada a objetos, bem como aplicar boas práticas de organização e estruturação de software, nomeadamente através da adoção do padrão Model-View-Controller (MVC). A aplicação cumpre o seu objetivo de oferecer uma gestão eficiente de músicas, playlists, álbuns e planos de subscrição, garantindo uma experiência personalizada para cada utilizador consoante o plano em vigor. A separação clara entre camadas facilita a manutenção, extensibilidade e testabilidade do sistema, permitindo que futuras evoluções — como novas funcionalidades ou alterações nos planos — possam ser integradas de forma controlada e eficaz. O SpotifUM revela-se, assim, uma aplicação robusta, modular e preparada para crescer.