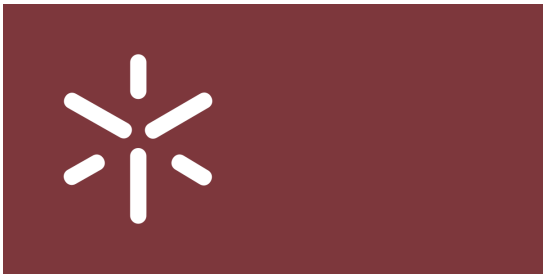


Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Programação Orientada a Objetos

Ano Letivo de 2024/2025

SpotifUM - Grupo 42

Maio, 2025

João Teixeira
A106836

Simão Mendes
A106928

Nelson Rocha
A106884

Índice

Lista de Figuras

1. Introdução

1.1. Contextualização

O projeto **SpotifUM**, desenvolvido no âmbito da unidade curricular de Programação Orientada aos Objetos (LEI e LCC, 2024/2025), tem como objetivo criar uma aplicação de gestão musical inspirada em plataformas de *streaming*, como, obviamente, o *Spotify*. A aplicação modela entidades como **músicas**, **álbuns**, ***playlists*** e **utilizadores**, permitindo uma interação personalizada consoante o tipo de utilizador.

Este projeto permite aplicar conceitos fundamentais da programação orientada a objetos, como **encapsulamento**, **herança**, **polimorfismo** e **persistência de dados**, promovendo o desenvolvimento de uma solução modular, robusta e extensível.

Para além de consolidar os conteúdos lecionados nas aulas teóricas e práticas, o projeto incentiva a análise crítica, a resolução de problemas reais e o trabalho colaborativo em equipa, sendo uma simulação fiel do processo de desenvolvimento de *software* no contexto profissional.

1.2. Objetivos

A aplicação *SpotifUM* visa:

- Gerir músicas, *playlists*, álbuns e utilizadores;
- Distinguir funcionalidades consoante o tipo de plano de subscrição (*PremiumBase*, *PremiumTop*);
- Reproduzir músicas e simular a escuta textual;
- Registar e analisar hábitos de reprodução;
- Gerar *playlists* personalizadas com base em preferências;
- Permitir guardar e carregar o estado da aplicação;
- Disponibilizar uma interface textual simples e funcional.

Com estes objetivos, visa-se a concretização de uma aplicação sólida e orientada a objetos, capaz de responder de forma eficiente aos requisitos funcionais definidos.

2. Arquitetura do Sistema

De seguida, apresenta-se uma versão reduzida do diagrama de arquitetura do sistema, que ilustra a estrutura das entidades e suas interações, bem como a organização geral do modelo:

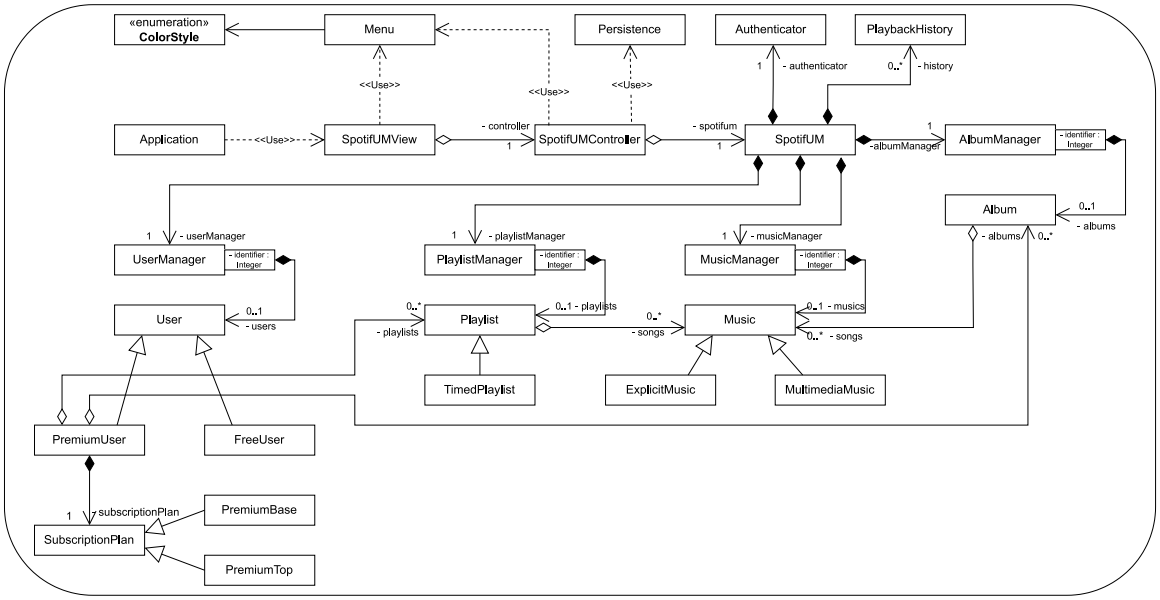


Figura 1: Diagrama de Classes UML Reduzido.

O diagrama completo pode ser encontrado em Anexo 1.

2.1. Descrição do Modelo

2.1.1. Entidades

As entidades que fazem parte do sistema incluem:

Utilizador (*User*)

A entidade *User* representa os indivíduos que interagem com o sistema. É implementada por uma classe abstrata, uma vez que a lógica de cálculo de pontos e a sua representação textual (*toString*) variam consoante o tipo de plano de subscrição associado.

Existem dois tipos principais de utilizadores:

- ***PremiumUser***: Utilizador com subscrição paga, que tem acesso a funcionalidades exclusivas como a reprodução de músicas sem ordem aleatória, criação de *playlists* específicas e uma biblioteca pessoal de álbuns e *playlists*. Para além dos atributos base de *User*, possui duas listas, uma de *playlists* e outra de álbuns, que representam a sua biblioteca pessoal. Importa salientar que estas entidades **estão associadas ao utilizador por agregação**, uma vez que, independentemente de quem as criou, pertencem ao sistema *SpotifUM*, e não ao utilizador. Além disso, cada *PremiumUser* tem associado um plano de subscrição, que determina as funcionalidades adicionais disponíveis.
- ***FreeUser***: Utilizador que utiliza o sistema gratuitamente, com acesso limitado, incluindo reprodução de música apenas em modo aleatório.

Planos de Assinatura (*SubscriptionPlan*)

A classe *SubscriptionPlan* está associada à entidade *PremiumUser* e representa o tipo de plano de subscrição que o utilizador possui, bem como as suas respetivas limitações. Trata-se de uma classe abstrata, cuja principal responsabilidade é delegar às suas subclasses o cálculo de pontos e benefícios associados.

Atualmente, existem duas implementações concretas de *SubscriptionPlan*:

- ***PremiumBase***: Corresponde ao plano de subscrição básico, que permite o acesso às funcionalidades padrão associadas a um utilizador *Premium*.
- ***PremiumTop***: Trata-se de um plano mais avançado, que oferece benefícios adicionais, tais como a criação automática de *playlists* recomendadas com base no género musical preferido e no tempo disponível para audição.

Música (*Music*)

A Música constitui a unidade fundamental de conteúdo do sistema, sendo também a entidade com o maior número de variáveis de instância. Cada instância representa uma faixa musical com os seus respetivos metadados, como **título, artista, duração, género,** entre outros.

Existem dois tipos distintos de músicas, representados por subclasses da classe base *Music*:

- **ExplicitMusic**: Representa músicas que podem conter conteúdo explícito. A classificação pode ser feita manualmente ou automaticamente, através de um método que analisa a letra com base em duas listas de termos: uma de termos moderados e outra de termos severos. Esta análise determina uma restrição etária apropriada (0+, 13+ ou 18+). Adicionalmente, é registada a origem da classificação (por exemplo, sistema automático ou a entidade que classificou a música como explícita).
- **MultimediaMusic**: Representa músicas que incluem componentes multimédia, como vídeos. Esta classe armazena dados adicionais, como a resolução do vídeo associado e a presença ou ausência de legendas.

Playlist

As *Playlists* são coleções de músicas organizadas numa lista, podendo ser criadas tanto pelo administrador do sistema como por utilizadores com subscrição *Premium*. Para além da versão base, existe um tipo especializado de *playlist*:

- ***Playlists* temporizadas (*TimedPlaylists*):** Estas *playlists* impõem duas restrições principais — um tempo máximo total e um género musical específico. Qualquer música adicionada a uma *TimedPlaylist* deve pertencer ao género definido e, em conjunto com as restantes músicas já presentes, não pode ultrapassar o tempo máximo estabelecido.

As *Playlists* podem ser classificadas como públicas ou privadas, e podem estar associadas a um ou mais utilizadores. Quando uma *playlist* é criada por um utilizador *Premium* e colocada como pública, outros utilizadores *Premium* têm a possibilidade de a adicionar à sua própria biblioteca, promovendo a partilha e descoberta de conteúdo musical dentro da comunidade *Premium*.

Álbum (*Album*)

O Álbum representa uma coleção ordenada de músicas, armazenadas sob a forma de uma lista. Cada álbum contém apenas músicas de um único artista, definido aquando da sua criação. Esta entidade assume um papel central na organização do conteúdo musical do sistema. A criação de álbuns está exclusivamente reservada ao administrador da aplicação, garantindo a consistência e a qualidade dos dados. Apesar disso, os utilizadores com subscrição *Premium* podem adicioná-los à sua biblioteca pessoal para posterior audição.

Entrada de Histórico (*PlaybackHistory*)

A classe *PlaybackHistory* regista as músicas reproduzidas por um utilizador numa determinada data. Esta entidade permite ao sistema manter um histórico de reproduções, sendo essencial para funcionalidades como a geração de estatísticas de utilização ou recomendações personalizadas com base nos hábitos do utilizador. Cada entrada associa inequivocamente um utilizador, uma música e a respetiva data de reprodução.

2.1.2. Gestores

O sistema inclui quatro gestores: ***UserManager, MusicManager, PlaylistManager*** e ***AlbumManager***. Cada um destes gestores mantém um mapa, onde a chave corresponde ao identificador único (ID) e o valor é o próprio objeto gerido. Além disso, cada gestor mantém um contador interno que indica o número máximo de entradas atualmente sob a sua responsabilidade.

A criação destas classes surgiu como uma forma de evitar a centralização excessiva de responsabilidades na classe principal *SpotifUM*, que de outro modo acumularia um número excessivo de métodos. Os *managers* não se limitam a operações básicas como inserção, remoção e consulta de elementos. São também os pontos de entrada para funcionalidades mais complexas, como a **geração de estatísticas, criação dinâmica de *playlists*, e atualizações globais de estado** — por exemplo, quando um utilizador ouve uma música.

Importa referir que a entidade *PlaybackHistory* não possui um gestor dedicado, uma vez que é utilizada em contextos muito específicos. Por este motivo, encontra-se diretamente armazenada no *SpotifUM* sob a forma de uma lista.

2.1.3. SpotifUM

A classe ***SpotifUM*** é o núcleo central do sistema, integrando e gerindo todas as entidades, como *Users*, *Albums*, *Playlists*, e *Musics*. Esta coordena o acesso e a manipulação dos dados por meio dos gestores responsáveis, como *UserManager* e *AlbumManager*, e centraliza a execução de ações no sistema, como autenticação, controlo de estado e operações críticas de dados. A sua principal função enquanto classe é **servir de ponte que liga os diferentes componentes do sistema**, delegando tarefas específicas e permitindo uma comunicação fluída entre os módulos.

Além de gerir os dados, a classe *SpotifUM* também facilita a interação entre os módulos, assegurando que as funcionalidades sejam executadas de acordo com as regras propostas. Esta também gere operações essenciais, como a criação de novos utilizadores e a adição de músicas às *playlists*, delegando essas responsabilidades aos módulos apropriados.

Esta centralização permite flexibilidade na expansão do sistema, mantendo uma estrutura coesa e facilitando a manutenção e evolução do mesmo ao longo do tempo.

2.1.4. Outros Módulos

Em relação às restantes classes do sistema, estas desempenham papéis específicos e serão abordadas em detalhes nas seções seguintes. De forma resumida:

- ***SpotifUMController*** e ***SpotifUMView*** lidam com a lógica de controlo e a interface do utilizador, respetivamente.
- ***Authenticator*** é responsável pela autenticação e registo de utilizadores.
- ***Persistence*** garante o armazenamento persistente de dados.
- ***Application*** é apenas o ponto de partida do programa, responsável por iniciar o processo, sem manipular diretamente a lógica de execução.
- **Menu** contém os menus de navegação e as opções de interação do utilizador, como a leitura de comandos e a realização de *scans*.

Cada uma dessas classes será detalhada mais adiante no relatório, conforme as suas responsabilidades específicas no sistema.

2.2. Articulação do Modelo

A arquitetura do sistema é composta por diversas classes interligadas, sendo que a comunicação e organização dos dados são essenciais para o funcionamento eficiente da aplicação. A articulação entre as classes segue o princípio do encapsulamento e da delegação de responsabilidades, com uma clara separação de tarefas entre as entidades e os seus gestores.

Agregação e Composição:

As relações entre as entidades, como *User*, *Playlist*, *Album* e *Music*, utilizam agregação e composição. Por exemplo, o *PremiumUser* pode ter várias *Playlists* e *Albums*, mas estas pertencem ao sistema, não ao utilizador (agregação). Já o *SubscriptionPlan* e o *PremiumUser* têm uma relação de composição, já que o plano de subscrição é parte integral do utilizador *Premium*. As *Playlists* e *Albums* possuem música por agregação, pois estas não precisam dos mesmos para existir, por mais que os álbuns tenham um mecanismo próprio para dar a ilusão do contrário.

Os gestores (*Managers*) gerem as suas respetivas entidades por composição, já que as entidades estão totalmente contidas dentro deles. O *SpotifUM*, sendo a aplicação central, agrega os gestores por composição, consolidando todos os dados no núcleo do sistema. Além disso, dados como a autenticação e o histórico também são geridos pelo *SpotifUM* por composição. Relativamente às partes ligadas à interação com o utilizador, como *SpotifUMView* e *SpotifUMController*, a interação é feita por agregação, pois estas dependem do *SpotifUM* (modelo) para funcionar, mas o *SpotifUM* não depende da parte interativa para existir.

Classes Finais e Métodos Estáticos:

Algumas classes, nomeadamente **Menu** e ***Persistence***, são finais. Estas classes não possuem variáveis de instância e tem todos os métodos definidos como *static*, uma vez que estes são independentes de qualquer estado ou dados específicos. Os métodos estáticos são utilizados nestas classes, permitindo que as operações sejam invocadas sem a necessidade de criar instâncias, o que torna o sistema mais eficiente. Ao utilizar métodos estáticos, o acesso aos métodos é feito de forma direta, assegurando consistência e simplificando a arquitetura, já que não há necessidade de instanciar objetos adicionais para executar estas operações.

2.3. Autenticação

A lógica de autenticação e atribuição de planos de subscrição assenta em princípios de segurança e controlo de acesso, garantindo que cada utilizador tenha credenciais válidas e permissões adequadas ao seu perfil.

O sistema começa por registar os utilizadores, associando a cada um um identificador único e uma palavra-passe encriptada. Esta encriptação é feita através de um algoritmo seguro (***hashing* SHA-256**), que transforma a palavra-passe num código irreversível, protegendo-a contra acessos indevidos. Desta forma, mesmo que os dados sejam comprometidos, as credenciais permanecem seguras.

Quando um utilizador tenta autenticar-se, o sistema **compara a encriptação da palavra-passe fornecida com a versão encriptada armazenada**. Se coincidirem, o *login* é bem-sucedido e o utilizador fica registado como ativo. Caso contrário, o acesso é negado. Esta verificação é essencial para impedir entradas não autorizadas.

Para além da autenticação básica, o sistema pode ser estendido para atribuir planos de subscrição diferenciados. Cada utilizador, após autenticado, poderá ter associado um nível de acesso (***Free***, ***Premium*** ou ***Admin***), determinando quais as funcionalidades disponíveis. Esta associação pode ser feita através de um mapeamento adicional, onde o ID do utilizador está ligado ao respetivo plano.

O utilizador ***Admin*** é criado automaticamente durante a inicialização do sistema, com um **ID pré-definido (-1) e uma palavra-passe padrão ("1234")**. Esta abordagem simplifica a configuração inicial, permitindo um acesso privilegiado imediato para fins de gestão do sistema. A existência deste perfil *Admin* assegura que, mesmo sem utilizadores registados, haja sempre uma conta com permissões elevadas disponível para administração, como a criação de outros utilizadores ou a definição de regras de acesso.

O processo de ***logout*** encerra a sessão, removendo o estado ativo do utilizador e garantindo que, sem nova autenticação, não há acesso ao sistema. Esta gestão de sessões é fundamental para manter a segurança, especialmente em ambientes partilhados.

2.4. Persistência

A lógica de persistência implementada permite guardar o estado completo da aplicação num ficheiro binário, preservando todos os dados entre execuções do programa. O sistema utiliza a **serialização padrão de Java** para converter o objeto principal da aplicação (*SpotifUM*) numa sequência de *bytes* que pode ser armazenada em disco.

Quando se invoca a operação de gravação, o objeto é transformado num formato binário e escrito num ficheiro especificado. Este processo **captura não apenas o objeto em si, mas toda a sua estrutura de dados interna**, mantendo as relações entre os diferentes componentes. Caso ocorra algum problema durante a gravação, como erros de I/O ou permissões, o sistema trata a exceção de forma elegante, apenas registando a mensagem de erro sem interromper o fluxo da aplicação.

Para recuperar o estado guardado, o sistema verifica primeiro a existência do ficheiro de persistência. Se encontrado, lê o fluxo binário e **reconstrói o objeto exatamente** como estava no momento da gravação. O processo inclui verificações de tipo para garantir que o conteúdo do ficheiro corresponde ao esperado, **prevenindo erros de desserialização**. Se o ficheiro não existir ou contiver dados inválidos, o sistema regista o problema e permite que a aplicação continue com um estado novo.

2.5. *Queries*

A plataforma oferece um conjunto de funcionalidades dedicadas à análise de dados, permitindo identificar tendências e padrões de utilização. Estas **consultas de estatísticas são fundamentais para compreender o comportamento dos utilizadores** e a popularidade dos conteúdos disponíveis.

No que diz respeito às músicas, o sistema consegue determinar **qual é a mais reproduzida**, analisando o contador de *plays* associado a cada faixa. Esta informação é obtida através de uma iteração sobre a coleção de músicas, comparando os valores de reproduções e mantendo o registo da faixa com o número mais elevado. O mesmo princípio é aplicado para identificar **o género musical mais popular**, sendo que, neste caso, o foco está no género da música mais reproduzida, em vez da faixa em si.

Para os intérpretes, a lógica é um pouco mais complexa, já que é necessário agregar o número total de reproduções de todas as músicas associadas a cada artista. O sistema utiliza um mapa para acumular estas contagens, somando os *plays* de cada música ao respetivo intérprete. No final, basta percorrer este mapa para encontrar **o artista com o maior número total de reproduções**.

No âmbito dos utilizadores, existem várias métricas disponíveis. A primeira permite identificar **quem tem mais pontos**, utilizando uma lista ordenada por pontuação. Outra consulta revela **o utilizador com mais tempo de escuta**, baseando-se no número total de reproduções associadas a cada perfil. Além disso, é possível determinar **quem criou mais *playlists***, contabilizando quantas listas de reprodução estão associadas a cada utilizador. Esta última funcionalidade recorre a um mapa para manter a contagem, atualizando os valores à medida que percorre todas as *playlists* existentes.

Por fim, a plataforma disponibiliza uma forma de contabilizar **todas as *playlists* públicas**, filtrando as que estão marcadas como tal e criando uma nova estrutura que contém apenas estas entradas. Esta funcionalidade é útil para permitir que os utilizadores descubram conteúdo partilhado pela comunidade.

2.6. Exceções

O sistema também implementa um conjunto de exceções personalizadas que permitem lidar com situações de erro específicas de forma clara e organizada. Cada exceção foi concebida para representar problemas distintos, facilitando a identificação e tratamento adequado de falhas em diferentes componentes da aplicação.

A ***AlbumException*** é lançada quando ocorrem problemas relacionados com a gestão de álbuns, como operações inválidas ou dados incorretos. Esta exceção pode ser instanciada com uma mensagem descritiva ou até mesmo com uma causa subjacente, o que ajuda a rastrear a origem do erro.

Para questões de autenticação, como credenciais inválidas ou tentativas de acesso não autorizado, a aplicação recorre à ***AuthenticatorException***. Esta exceção é crucial para garantir a segurança do sistema, alertando quando ocorrem falhas no processo de *login* ou validação de utilizadores.

A ***MusicException*** surge em cenários relacionados com músicas, como a tentativa de aceder a uma faixa inexistente ou executar operações não permitidas sobre ela. Esta exceção simplifica a deteção de problemas no módulo de gestão de músicas.

Quando há erros associados ao histórico de reproduções, como falhas ao recuperar ou atualizar registos, a ***PlaybackHistoryException*** entra em ação. Esta exceção ajuda a manter a integridade dos dados relacionados com as reproduções efetuadas pelos utilizadores.

A ***PlaylistException*** é utilizada para sinalizar problemas na criação, modificação ou acesso a *playlists*, como tentativas de adicionar músicas inválidas ou operações em listas que não existem. Esta exceção assegura que as *playlists* sejam geridas corretamente, evitando estados inconsistentes.

Por fim, a ***UserException*** é lançada quando ocorrem erros relacionados com a gestão de utilizadores, como a criação de perfis duplicados ou a tentativa de aceder a informações de um utilizador que não existe. Esta exceção é fundamental para manter a consistência dos dados dos utilizadores.

2.7. Interação com o Utilizador

O sistema *SpotifUM* foi desenvolvido seguindo o padrão de arquitetura ***Model-View-Controller* (MVC)**, que separa claramente as responsabilidades entre a lógica de negócio, a apresentação e o controlo de fluxo. Esta abordagem garante uma interação intuitiva com o utilizador, aliada a uma estrutura de código organizada e fácil de manter.

A ***View (SpotifUMView)*** é a camada responsável por toda a interação com o utilizador. Através da classe **Menu**, apresenta interfaces coloridas e formatadas no terminal, utilizando códigos **ANSI** para realçar títulos, opções e mensagens de erro. As cores são aplicadas de forma consistente: azul para menus, amarelo para opções, vermelho para ações críticas (como "Sair") e verde para *prompts* de *input*. Esta abordagem visual melhora significativamente a experiência do utilizador, tornando a navegação mais intuitiva.

O ***Controller (SpotifUMController)*** atua como intermediário entre a ***View*** e o ***Model***. Recebe as ações do utilizador (como selecionar uma música ou criar uma *playlist*) e coordena a execução das operações correspondentes no ***Model***. Por exemplo, quando um utilizador escolhe reproduzir uma música, o ***Controller*** valida permissões, solicita ao ***Model*** a execução e devolve o resultado à ***View*** para apresentação. Esta camada também trata da conversão entre *inputs* brutos (como *strings*) e os tipos necessários para a lógica de negócio, garantindo que dados inválidos não propaguem para o ***Model***.

O ***Model (SpotifUM)*** contém toda a lógica de negócio e gestão de dados. Aqui residem as operações críticas, como autenticação, gestão de *playlists*, reprodução de músicas e geração de estatísticas. As exceções personalizadas, abordadas na secção anterior, permitem que o ***Controller*** traduza erros técnicos em mensagens compreensíveis para o utilizador.

A interação flui de forma cíclica:

1. A ***View*** exibe um menu formatado e recolhe a escolha do utilizador;
2. O ***Controller*** processa a escolha, invocando métodos do ***Model***;
3. O ***Model*** executa a operação e retorna o resultado ao ***Controller***;
4. A ***View*** apresenta o resultado ao utilizador, com cores e formatação adequadas (como mensagens de sucesso a verde ou erros a vermelho).

Recursos como ***Autosave*** (que persiste o estado do ***Model*** em ficheiro) e **histórico de operações** são geridos de forma transparente para o utilizador, demonstrando como o **MVC** facilita a adição de funcionalidades complexas sem comprometer a usabilidade. A separação clara de responsabilidades permite, por exemplo, modificar a interface (como adicionar novas cores) na ***View*** sem afetar a lógica de negócio no ***Model***.

2.8. Geração de *Playlists* Recomendadas

A geração de *playlists* recomendadas tem como objetivo criar listas de reprodução personalizadas para cada utilizador, com base nas suas preferências musicais e outras restrições, como tempo máximo ou conteúdo explícito. O processo analisa factores como o género musical, os artistas preferidos e a frequência com que o utilizador ouve determinadas músicas, gerando automaticamente uma *playlist* adaptada aos seus gostos.

Existem três tipos de *playlists* recomendadas:

- ***Playlist* com preferências musicais do utilizador**: Gerada com base nas músicas que o utilizador mais ouve, respeitando os seus gostos musicais.
- ***Playlist* com restrição de tempo**: Similar à anterior, mas com um limite de duração. O sistema escolhe as músicas dentro desse tempo máximo.
- ***Playlist* com músicas explícitas**: Criada apenas com músicas explícitas, com base nas preferências do utilizador que optou por este tipo de conteúdo.

O método começa por analisar o histórico de reprodução do utilizador, verificando as músicas que foram reproduzidas e criando mapas de frequência para géneros e artistas. Com isso, é possível identificar as preferências do utilizador, como os géneros musicais e artistas que mais escutou. O primeiro critério para seleção das músicas é o género, priorizando aquelas com maior frequência no histórico de reprodução do utilizador. Se houver músicas com o mesmo género, o segundo critério de comparação será a frequência do artista, priorizando aqueles que o utilizador mais ouviu.

Em seguida, o código filtra as músicas disponíveis na plataforma, removendo aquelas que já foram ouvidas pelo utilizador e, caso o filtro de *explicit content* esteja ativo, seleciona apenas músicas explícitas. Depois, as músicas são ordenadas com base nos critérios de género e artista e são selecionadas até atingir o número máximo de músicas especificado. O código também verifica se a duração total das músicas não ultrapassa o limite máximo de tempo (se especificado). O processo termina quando a *playlist* gerada contém as músicas recomendadas de acordo com as preferências do utilizador.

3. Descrição da Aplicação

A aplicação *SpotifUM* foi desenvolvida com uma interface textual intuitiva, organizada em menus hierárquicos que guiam o utilizador através das funcionalidades disponíveis. A interação é realizada via terminal, com opções numéricas e formatação colorida para melhorar a experiência. Abaixo, descrevem-se as principais funcionalidades e a forma de as aceder.

3.1. Funcionalidades Implementadas

3.1.1. Gestão de Utilizadores

O sistema *SpotifUM* implementa um processo de registo e autenticação robusto, garantindo acesso diferenciado conforme o tipo de subscrição do utilizador.

Registo de Utilizadores

Ao iniciar a aplicação, o utilizador é direcionado para um menu de autenticação (Figura 2), onde pode efetuar o registo como *FreeUser* ou *PremiumUser*. O registo exige os seguintes dados obrigatórios:

- ***Password*** (armazenada de forma encriptada);
- **Nome do utilizador;**
- **Idade;**
- ***Email;***
- **Morada;**
- **Tipo de conta.**

Para utilizadores ***Premium***, o registo inclui opções adicionais, como a seleção do tipo de subscrição (*PremiumBase* ou *PremiumTop*), que determinam funcionalidades exclusivas (ex: acesso a *playlists* temporizadas ou recomendações avançadas). A Figura 2 ilustra o formulário completo de registo, destacando a diferenciação entre os tipos de conta.

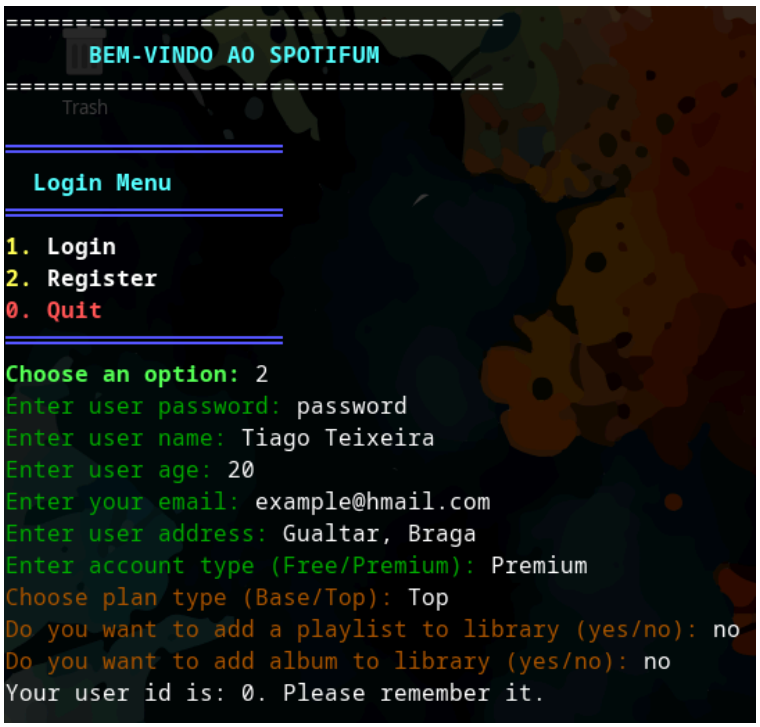


Figura 2: Menu de Registo.

Autenticação e Acesso

Após o registo, o utilizador recebe um *User* ID único, necessário para futuros *logins*. O processo de autenticação (Figura 3) é simplificado: o utilizador insere o *User* ID e a *password*, validados pelo sistema através de encriptação SHA-256. Em caso de sucesso, é redirecionado para o menu correspondente ao seu tipo de subscrição.

Figura 3: Menu de Autenticação do Utilizador criado.

Funcionalidades por Tipo de Subscrição

- ***FreeUser:***
  - Acesso limitado a *playlists* públicas;
  - Reprodução aleatória;
  - Visualização básica de estatísticas de reprodução.
- ***PremiumUser:***
  - Criação e gestão de *playlists* personalizadas (públicas ou privadas);
  - Acesso a álbuns e a uma biblioteca pessoal;
  - Reprodução sem restrições;
  - Geração de *playlists* recomendadas com base em preferências.

A Figura 4 apresenta o menu completo de um utilizador *PremiumTop*, que inclui funcionalidades adicionais, como a criação automática de *playlists* com restrições de tempo e género.

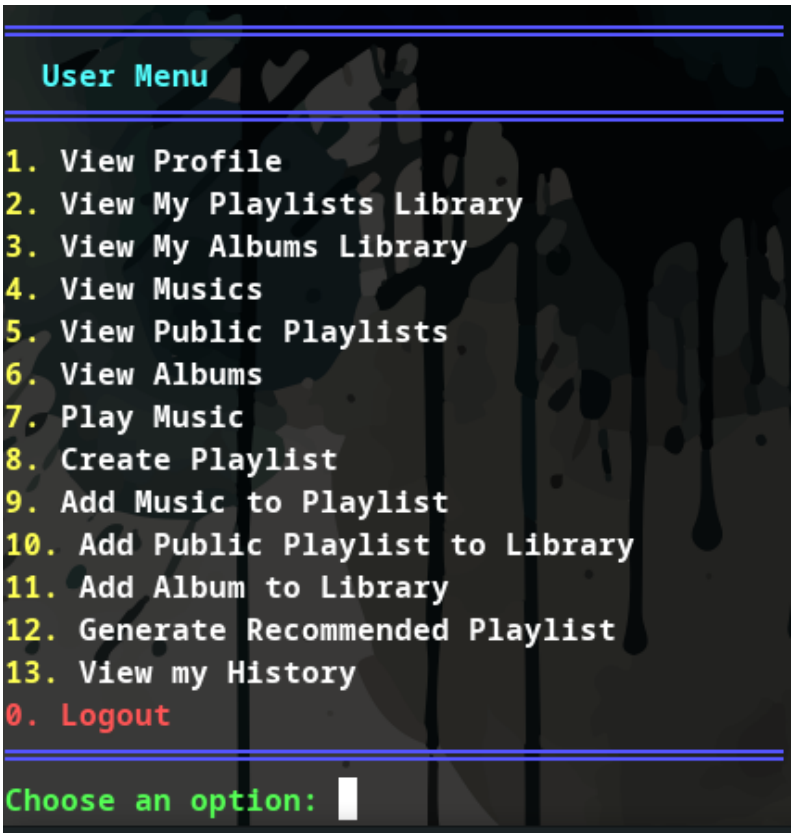


Figura 4: Menu de Utilizador *PremiumTop*

Encadeamento Lógico

1. Registo: Coleta de dados básicos e seleção de subscrição (Figura 2).

2. Autenticação: Validação segura de credenciais (Figura 3).

3. Acesso Personalizado: Menu adaptado ao tipo de utilizador (Figura 4), garantindo que as funcionalidades estejam alinhadas com o plano de subscrição - acessível tanto após o registo quanto após o login.

Este fluxo assegura uma experiência intuitiva, respeitando os princípios de segurança e personalização definidos na arquitetura do sistema.

3.1.2. Gestão de Músicas e Álbuns

O sistema *SpotifUM* permite uma gestão estruturada de músicas e álbuns, garantindo organização e consistência no catálogo musical.

Adição e Edição de Músicas

Apenas administradores podem adicionar músicas ao sistema, assegurando a qualidade dos metadados. Cada música inclui:

- **Título;**
- **Artista** principal;
- **Editora;**
- ***Lyrics*;**
- **Som** (Notas musicais);
- **Género musical;**
- **Duração** (formato MM:SS);
- **Álbum** a adicionar a música;

- Classificação de **conteúdo explícito ou multimédia** (se aplicável).

Para músicas multimédia (*MultimediaMusic*), são registados dados adicionais, como resolução de vídeo e disponibilidade de legendas. A Figura 5 ilustra a interface de adição de músicas, onde o administrador define os campos obrigatórios e opcionais.

Figura 5: Interface de Adição de Músicas.

Criação de Álbuns

Álbuns são coleções ordenadas de músicas de um único artista, criadas exclusivamente por administradores. Durante a criação (ou adição de uma dada música a um álbum) (Figura 6), o sistema valida que todas as músicas pertençam ao mesmo artista e gera automaticamente um ID único para o álbum.

Figura 6: Interface de Criação de Álbuns.

Consulta e Filtragem

Utilizadores podem listar as músicas, os álbuns, e as *playlists* para visulizar o conteúdo. A Figura 7 mostra os resultados de uma listagem das *playlists* disponíveis para o utilizador em questão.

Figura 7: Resultados de Reprodução por *Id.*

3.1.3. Criação e Partilha de *Playlists*

A funcionalidade de *playlists* é central para a personalização da experiência do utilizador.

Tipos de *Playlists*

- **Públicas/Privadas:** Definidas durante a criação. *Playlists* públicas são visíveis a todos os utilizadores.
- **Temporizadas (*TimedPlaylist*):** Restringem a duração total e o género das músicas.

Processo de Criação

PremiumUsers acessam o menu Criar *Playlist* (Figura 8), onde selecionam o tipo, definem restrições. O sistema valida automaticamente as regras (ex: género único para *TimedPlaylist*).

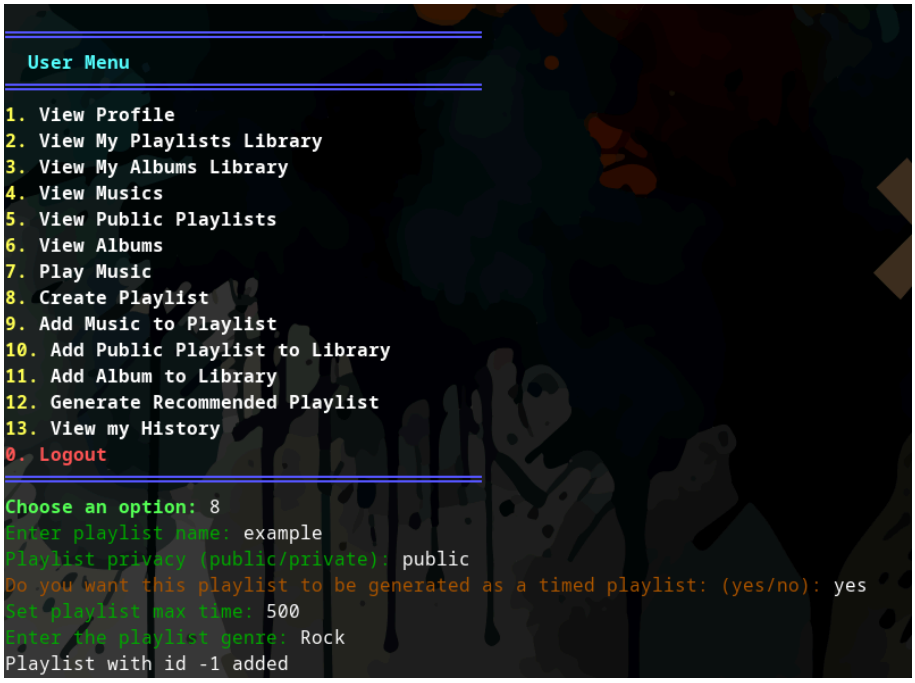


Figura 8: Interface de Criação de *Playlist*.

3.1.3.1. Reprodução de Músicas e Histórico

A reprodução de músicas é adaptada ao tipo de subscrição, com funcionalidades exclusivas para *PremiumUsers*.

Simulação de Reprodução

Ao selecionar uma música, o utilizador visualiza detalhes em tempo real (Figura 9):

- Título e artista;
- *Lyrics* da música.

Figura 9: Interface de Reprodução de Músicas.

Histórico de Reproduções

Todas as reproduções são registadas em *PlaybackHistory*. *PremiumUsers* podem acessar estatísticas detalhadas (Figura 10), como:

- Músicas mais ouvidas (a que for mais frequente no histórico);
- Quando ouviram a música;

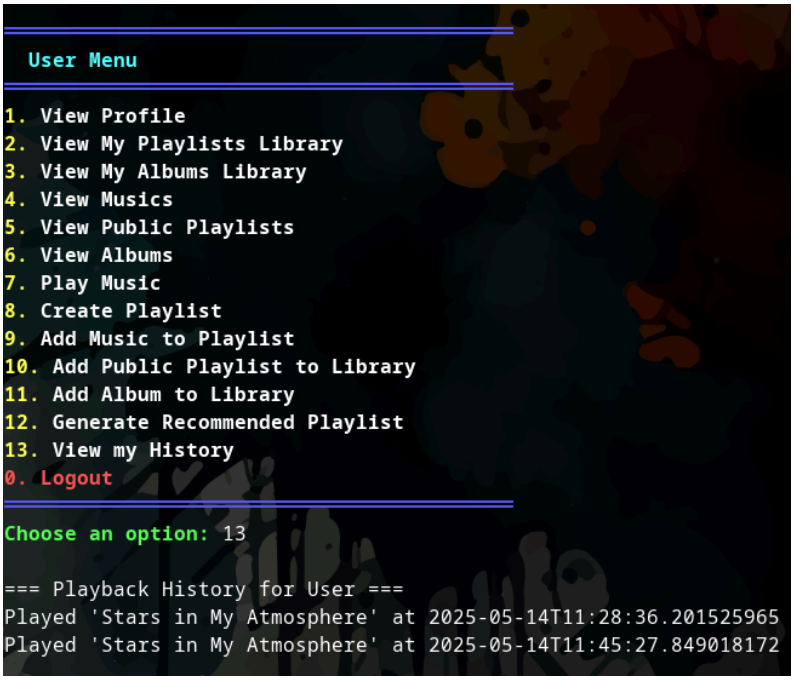


Figura 10: Histórico de Reprodução.

3.1.4. Geração de *Playlists* Recomendadas

O sistema utiliza algoritmos para criar *playlists* personalizadas com base em hábitos de escuta.

Critérios de Recomendação

- **Preferências do Utilizador:** Género e artistas mais ouvidos.
- **Restrições de Tempo:** Duração máxima definida pelo utilizador (no caso de escolher *TimedPlaylist*).
- **Conteúdo Explícito:** Filtro opcional para incluir/excluir músicas explícitas.

Processo de Geração

No menu Recomendações (Figura 11), o utilizador define parâmetros. O sistema combina dados do histórico, catálogo musical e restrições para gerar uma *playlist*.

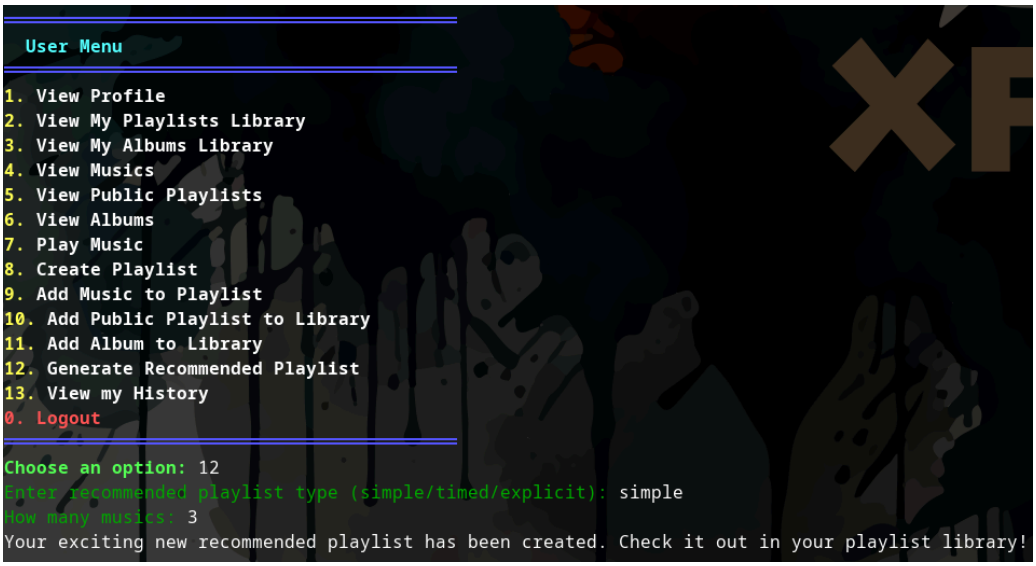


Figura 11: Criação de uma *Playlist* Recomendada.

3.2. Fluxo de Interação com o Utilizador

A interação na *SpotifUM* segue um modelo hierárquico de menus, projetado para garantir usabilidade e clareza.

Estrutura de Menus

- **Menu Inicial:**
  - **Opções:** *Login*, *Register*, *Quit*.
  - **Formatação:** Cores ANSI destacam ações críticas (ex: vermelho para Sair).

Menu Principal:

- **FreeUser:** Acesso a Reproduzir Música, *Playlists* Públicas, Estatísticas Básicas.
- **PremiumUser:** Inclui Criar *Playlist*, Biblioteca Pessoal, Recomendações, variando com o plano de subscrição.
- **Administrador:** Adiciona Gestão de Conteúdo e Carregamento/Salvaguarda do Estado (Figura 12).

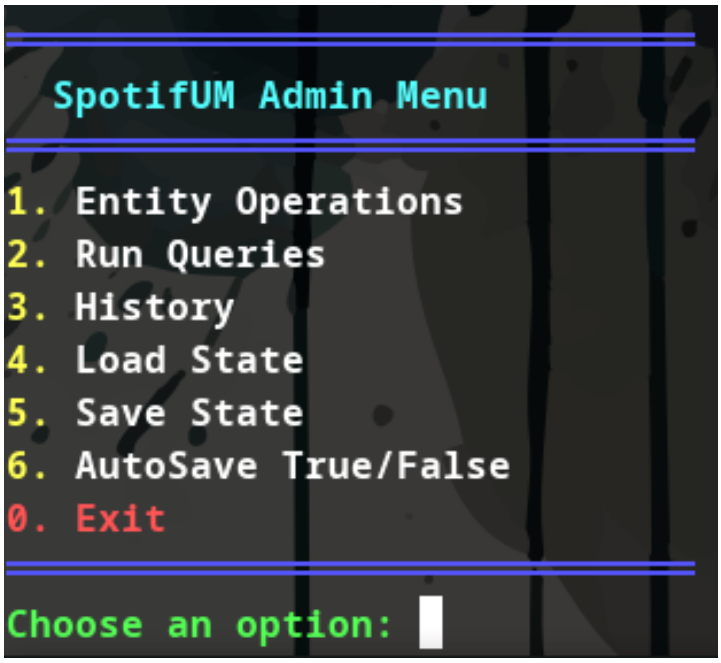


Figura 12: Menu de Administrator.

Feedback em Tempo Real

- **Cores e Mensagens:**
  - **Verde:** Confirmação de ações (ex: "*Playlist* '*Rock* Clássico' criada!").
  - **Vermelho:** Erros (ex: "*Invalid input. Please enter a valid integer:*").
- **Validação Dinâmica:**
  - **Restrições** (ex: idade do utilizador deve ser um *Integer*) são verificadas durante a inserção de dados (Figura 13).

Figura 13: Validação do tipo de dados da idade do utilizador.

3.3. Persistência de Dados

A aplicação utiliza serialização Java para guardar o estado completo do sistema, incluindo utilizadores, músicas e *playlists*.

Funcionalidades de Persistência

- **Save e Auto-Save:**
  - O estado é automaticamente salvo (se a opção *autosave* estiver habilitada pelo *admin*) após operações críticas (ex: criação de *playlist*).
  - Opção manual disponível no menu de Administrador (Figura 12).

Estrutura de Armazenamento

- **Ficheiro Binário:** Contém objetos serializados de *SpotifUM*, *UserManager*, *MusicManager*, etc.
- **Segurança:** Dados sensíveis (ex: *passwords*) são armazenados como *hashes* SHA-256.
- **Ficheiro de Estado:** Dados do estado do sistema são armazenados num ficheiro de estado, que pode, posteriormente, ser carregado.

4. Conclusão

O *SpotifUM* demonstra a aplicação prática de princípios de Programação Orientada a Objetos, desde o encapsulamento de entidades (ex: *PremiumUser*, *TimedPlaylist*) até à modularidade proporcionada pelo padrão MVC. A arquitetura escalável permite futuras expansões, como integração com APIs externas ou adição de funcionalidades sociais.

Desafios e Soluções

1. Complexidade de *Playlists* Temporizadas:
- A validação de restrições (género e tempo) exigiu a implementação de iteradores personalizados e métodos de verificação em tempo real.

2. Persistência de Dados:
- A serialização de coleções complexas (ex: `Map<User, List<Playlist>>`) foi resolvida com a implementação de *Serializable* em todas as entidades.

Potenciais Melhorias Futuras
- **Interface Gráfica (GUI):** Substituir o terminal por uma interface intuitiva com bibliotecas como *JavaFX*.
- **Recomendações Baseadas em IA:** Utilizar algoritmos de *machine learning* para personalizar *playlists*.
- **Integração Multidispositivo:** Permitir sincronização entre dispositivos via autenticação *OAuth*.

Em síntese, o projeto não apenas cumpriu os requisitos académicos, mas também simulou desafios reais de desenvolvimento de *software*, preparando-nos para cenários profissionais mais complexos.

5. Bibliografia

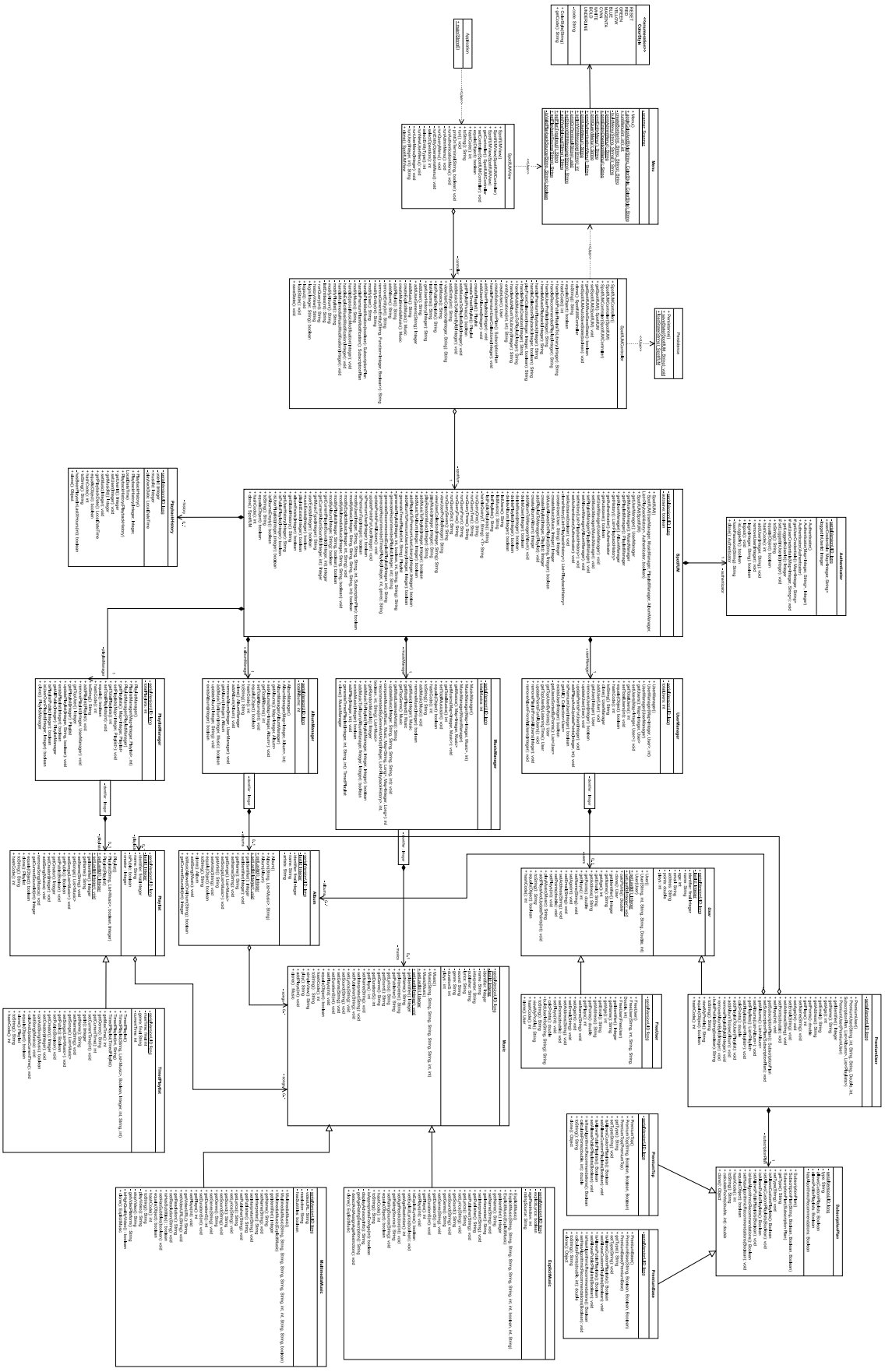
[1] Oracle Corporation, «Java Platform, Standard Edition 8 API Specification». [Online]. Disponível em: https://docs.oracle.com/javase/8/docs/api/

[2] António Nestor Ribeiro, «Slides da Unidade Curricular de Programação Orientada aos Objectos».

[3] Kunal Nalawade, «How to Write Unit Tests in Java». [Online]. Disponível em: https://www.freecodecamp.org/news/java-unit-testing/

[4] Oracle Corporation, «How to Write Doc Comments for the Javadoc Tool». [Online]. Disponível em: https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

6. Anexos



Anexo 1: Diagrama de Classes UML Completo.