

Gonçalo Fileno nº201180372

Bernardo Câmara nº2014198344

Relatório da Aplicação *RockStarInc*

**Introdução à Programação em Java**

**Acertar ao Rumo - 2023/2024 – 1ºTrimestre**

Janeiro de 2024

**Índice**

Introdução\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pag.3

Descrição de Classes\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pag.4

Diagrama de Classes\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pag.8

Descrição Gráfica e Funcionamentos dos Métodos\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pag.9

Conclusão\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pag.17

Anexo I (1º Diagrama UML)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pag.18

Anexo II (2º Diagrama UML)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Pag.19

**Introdução**

Este relatório abrange a implementação do primeiro projeto desenvolvido durante o curso "Acertar o Rumo" na disciplina de Introdução à Programação em Java.

O objetivo principal deste projeto consiste em aplicar os conhecimentos adquiridos ao conceber e executar uma aplicação de gestão de uma plataforma musical destinada a artistas e clientes. A sua execução envolve a aplicação de habilidades analíticas, planeamento estratégico, criatividade, resolução de problemas e colaboração efetiva em equipa.

A elaboração deste trabalho prático foi desenvolvida no programa *IntelliJ*, conforme praticado nas aulas ao longo do trimestre.

A nossa aplicação está estruturada de forma a poder ser utilizada por um Cliente ou por um Artista, sendo que ambos têm funcionalidades diferentes ao seu alcance.

O Cliente pode fazer a gestão das suas músicas, pode criar e eliminar Playlists, pode adicionar músicas a essas Playlists, pode adquirir novas músicas, pode pesquisar através do nome da Música ou do Artista dos grupos de música que tem à sua disposição e pode ainda carregar o seu saldo com que efetua compras.

Por sua vez o Artista pode criar as suas Músicas assim como os seus Álbuns, pode alterar o preço e a visibilidade das suas Músicas. Pode também pesquisar pelo nome da Música e por Género nas suas músicas, tendo também acesso a um conjunto de estatísticas relativas ao mesmo.

Os dados relativos à Aplicação são guardados num ficheiro de objetos, utilizado pela nossa aplicação para ir buscar os dados existentes e guardar as alterações que vão sendo feitas na utilização da mesma. A aplicação inicia fazendo a leitura desse mesmo ficheiro e guardando o objeto, que será́ enviado para as classes da interface gráfica.

**Descrição das Classes**

A nossa aplicação é constituída por um FrontEnd e um BackEnd, separados no nosso código por packages diferentes. O BackEnd é constituído pelas classes necessárias para criar os objetos e métodos que a nossa aplicação irá utilizar. Por sua vez, o FrontEnd é constituído pelas classes necessárias a criar os painéis que o utilizador vê e com os quais interage ao utilizar a mesma. Estas duas classes interagem e dependem uma da outra.

O código do BackEnd da nossa aplicação está estruturado com uma classe central ***RockstarInc*** que representa a aplicação “Rockstar Inc”. Esta possui como atributos informações como a lista de todos os *Artistas* presentes na aplicação, a lista de todos os *Clientes*, a lista com todas as *Playlists,* a lista com todos os *Álbuns,* a lista com todas as *Músicas,* e tem ainda como atributo um *Utilizador Atual* que guardará o utilizador que estiver a utilizar a aplicação. Cada uma das classes listadas anteriormente tem os seus atributos próprios. É na classe *RockstarInc* onde encontramos grande parte métodos utilizados para o funcionamento da aplicação.

A classe ***GrupoDeMúsicas***servirá como *classe* *mãe* das classes *Playlist* e classe *Álbum,* por isso a classe *GrupoDeMúsicas* tem como atributos um *Arraylist* de *Músicas* e uma *String* representando o nome desse grupo de músicas. Esta classe possui dois construtores, um que cria um *GrupoDeMúsicas* já recebendo um *ArrayList* de *Músicas* para lhe ser atribuído e outro construtor que não precisa de receber o mesmo, recebendo apenas o nome para atribuir ao *GrupoDeMúsicas*. Possue alguns métodos que permitem obter informação desta classe bem como um método para adicionar novas músicas ao *ArrayList* de *Músicas* presente nos atributos, este método verifica se a *Música* recebida já pertence ao *ArrayList* ou não antes de a adicionar.

A classe ***Album*** sendo *filha* da classe *GrupoDeMusicas* tem como atributos uma *String* que faz referência ao seu género musical, um objeto *Artista* que criou o álbum e uma *LocalDate* com a data de lançamento do álbum. Esta classe possui dois construtores, um que permite criar um *Album* recebendo um *ArrayList* de *Músicas* para lhe ser atribuído e outro que não recebe esse *ArrayList*, criando o *Album* vazio. Possuem ainda alguns métodos que permitem obter informação desta classe.

A classe ***Playlist*** sendo também *filha* da classe *GrupoDeMúsicas* tem atributo apenas um valor *booleano* que dita a visibilidade dessa mesmo Playlist. Possui também métodos que permitem aceder e alterar essa informação.

A classe ***Utilizador***servirá como *classe* *mãe* das classes *Cliente* e classe *Artista,* por isso a classe *Utilizador* tem como atributos em formato *String* um *username*, um *nome* e uma *passe,* tudo pertencente a esse a Utilizador. Possui métodos que permitem aceder a alguma informação da classe. Possui ainda um método que faz a verificação da correspondência de uma passe inserida com a passe verdadeira do utilizador.

A classe ***Cliente*** sendo *filha* da classe *Utilizador*, tem como atributos um *ArrayList* de *Músicas* que faz referência ao Carrinho de Compras do *Cliente*, um *ArrayList* de *Músicas* que faz referência à Biblioteca de Músicas do *Cliente*, um *ArrayList* de *Playlists* que faz referência às *Playlists* pertencentes ao *Cliente*, um *Arraylist* de *Compras* que faz referência a todas as compras de músicas que o *Cliente* já fez, e uma variável do tipo *Double* onde ficará guardado o saldo do *Cliente*. Possui um método que criará uma *Playlist* aleatória, apenas recebendo o número de músicas que se quer nessa *Playlist* e de que género. Possui um método que permite verificar se uma *Playlist* tem um determinado número de *Músicas*. Possui um método que permite comprar *Músicas*, verificando o preço das mesmas, sendo que se as mesmas possuírem *Preço* são adicionadas ao Carrinho de Compras e se não possuírem são adicionadas diretamente à Biblioteca do *Cliente*. Possui um método que verifica se uma *Música* já se encontra na Biblioteca ou não, e um método idêntico para o Carrinho de Compras. Possui um método que limpa o Carrinho de Compras. Possui um método que verifica se uma *Música* já possui um *Rating* atribuído pelo *Cliente*. Possui um método que permite adicionar ou alterar o *Rating* dado pelo *Cliente* a uma música. Possui ainda diversos métodos que permitem aceder e alterar informação guardada nestes atributos.

A classe ***Artista***, sendo filha da classe *Utilizador*, tem como atributos uma *String* onde é guardado o seu PIN de acesso, um *ArrayList* de *Albuns*, que faz referência aos *Albuns* pertencentes ao *Artista*, um *ArrayList* de *Músicas*, que faz referência aos Singles do *Artista* (*Músicas* sem *Album* atribuído). Possui um método que permite verificar se um PIN introduzido equivale ao PIN verdadeiro do *Artista*. Possui um método que verifica se já existe um *Álbum* com o mesmo nome na lista de *Álbuns* do *Artista*. Possui um método que retorna um *Array* de *Strings* com o nome de todos os álbuns pertencentes ao *Artista*. Possui ainda diversos métodos que permitem aceder e alterar informação guardada nestes atributos.

A classe ***Compra*** guarda toda a informação relativa a uma compra efetuada por qualquer *Cliente*. Esta classe tem como atributos um objeto *Cliente*, que guarda o *Cliente* que efetua a compra, um *ArrayList* de *Músicas* onde ficam guardadas as *Músicas* que compõe aquela *Compra*, um *LocalDate* que faz referência à data em que a compra é efetuada e um *Double* com o valor total da *Compra* que foi efetuada. No construtor desta classe é transferido do Carrinho de Compras pertencente ao *Cliente* todas as músicas para o *ArrayList* de *Músicas* presente como atributo desta classe.

A classe ***Preço*** guarda toda a informação relativa ao preço que compõe no *ArrayList* de *Preços* presente na classe *Música*. Esta classe tem como atributos um *Double* que guarda o valor do *Preço* e uma *LocalDate* que guarda a data com que o respetivo valor foi atribuído ao preço. Possui ainda diversos métodos que permitem aceder a informação guardada nestes atributos.

A classe ***Rating*** guarda a informação relativa aos ratings atribuídos pelos *Clientes* às *Músicas* que possuem na sua biblioteca. Esta classe tem com atributos um obecto *Cliente* a quem pertence o respetivo *Rating*, um objeto *Música* a quem o *Rating* é atribuído e guardado num *ArrayList* de *Ratings* e um *Int* que guarda o valor da avaliação atribuída pelo *Cliente*. Possui ainda diversos métodos que permitem aceder e alterar informação guardada nestes atributos.

A classe ***Música*** guarda a toda a informação relativa às músicas presentes na aplicação. Esta classe tem como atributos uma *String* onde guarda o titulo atribuído à *Música* pelo *Artista*, tem um objeto Artista onde guarda toda a informação sobre o *Artista* que criou a respetiva *Música*, tem uma *String* onde guarda o tipo de género a que a *Música* pertence, tem um *ArrayList* de *Preços* onde ficam guardados todos os preços atribuídos à *Música* desde a sua criação, possui um *ArrayList* de *Ratings* onde são guardados todos os *Ratings* atribuídos pelos *Clientes* à respetiva *Música*, possui um valor *Booleano* que faz referência à visibilidade da respetiva *Música*, possui um objeto *Álbum* à qual pode pertencer ou não, possui um *Int* onde é guardado o número de vezes que a *Música* é adquirida, quer seja grátis ou não, possui ainda um valor *Double* onde é guardado todo o valor que é faturado através da compra da respetiva *Música*. Esta classe tem dois tipos de construtores, um que permite criar *Músicas* sem *Álbum* atribuído e outro e permite criar *Músicas* com *Álbum* já atribuído. Possui ainda diversos métodos que permitem aceder e alterar informação guardada nestes atributos.

O código FrontEnd está dividido em três partes diferentes: a parte onde são apresentadas os painéis de login e de registo , a parte onde são apresentados todos os elementos inerentes ao *Cliente* e uma terceira parte onde são apresentados todos os elementos inerentes ao *Artista*. Todos os painéis desta aplicação são apresentados na mesma *JFrame* *Frame* que vai sendo redimensionada consoante o painel apresentado.

A *JFrame* ***Frame***é a responsável pela criação ou leitura do ficheiro de objetos “rockstar.dat”, quando esta é inicializada, e é responsável por escrever e guardar todos os dados das alterações efetuadas quando esta é fechada.

O *JPanel* ***LoginPanel*** é o painel onde é feita a autenticação de utilizadores da *Rockstar*. Este painel inclui campos para preenchar com o username e com a pass do utlizador, e também os butões Registo e Login.

O *JPanel* ***Registo***é o painel responsável pelo registo de utilizadores. Inclui os campos para inserir username, pass, nome e o pin. O campo de pin apenas fica visível se o *JRadioButton* com o texto artista estiver *selecionado*.

O *JPanel* ***LoginPin*** é o painel que *valida* o pin do artista.

O *JPanel* ***InterfaceCliente*** contém todos os elementos necessários para a utilização da aplicação por parte dos *Clientes*. Esta classe desempenha um papel crucial e oferece uma interface eficiente e intuitiva para explorar, adquirir e gerir as músicas que cada *Cliente* possui.

O *JPanel* ***ClientePlaylists*** é responsável por exibir as *Playlists* do *Cliente*. Esta classe oferece também funcionalidades para criar *Playlists* vazias e *Playlists* geradas de forma automática.

O *JPanel* ***TabelaCliente*** é o componente central e crucial para a interação do utilizador com as músicas da aplicação. Este painel oferece funcionalidades de visualização das músicas, de compra destas, de alteração de *Rating* e visualização do histórico de preços de cada música. Nesta tabela podem ser mostradas as músicas da loja, da biblioteca do cliente ou as músicas de *cada* uma das *Playlists* do *Cliente*.

O *JPanel* ***PanelCarrinho*** representa o carrinho de compras do utilizador e fornece componentes responsáveis por adicionar músicas, calcular o preço total da compra, realizar a compra e limpar o carrinho.

O *JPanel* ***PesquisaPanel*** fornece ao *Cliente* os meios necessários para este realizar pesquisas de músicas com base no artista ou no título, exibindo o resultado da pesquisa na *TabelaCliente*.

O JPanel ***InterfaceArtista*** contém todos os elementos necessários para a utilização da aplicação por parte dos *Artistas*. Esta é a classe central para a gestão das contas *Artista*.

O *JPanel* ***TabelaArtista*** é responsável pela exibição das músicas do *Artista*. Este painel fornece funcionalidades como alteração de títulos, preços e visibilidade, bem como a adição de músicas a álbuns.

O *JPanel* ***ArtistaAlbuns*** é responsável por criar e gerir álbuns de um *artista* na aplicação.

O *JPanel* ***EstatisticasArtista*** é a classe responsável por apresentar todas as estatísticas relacionadas à *Rockstar*. Ainda contém um butão responsável por mostrar os tops de artistas e de músicas vendidas.

O *JPanel* ***PesquisaArtista*** fornece ao *Artista* os meios necessários para este realizar pesquisas de músicas com base no título ou no álbum, exibindo o resultado da pesquisar na *TabelaArtista*.

**Diagrama de Classes**

Inicialmente foi entregue um diagrama de classes UML (anexo I) que sofreu algumas alterações ao longo do desenvolvimento do projeto. Estas foram, maioritariamente, ao nível da distribuição de alguns métodos pelas diferentes classes quando no primeiro diagrama se encontravam concentradas na classe *RockstarInc*.

Foram também criados atributos e métodos dentro de algumas classes que achámos necessários para uma correta e facilitada implementação no código para que todas as funcionalidades exigidas conseguissem ser atingidas.

As alterações mencionadas podem ser visualizadas no diagrama UML do anexo II.

**Descrição Gráfica e Funcionamento dos Métodos**

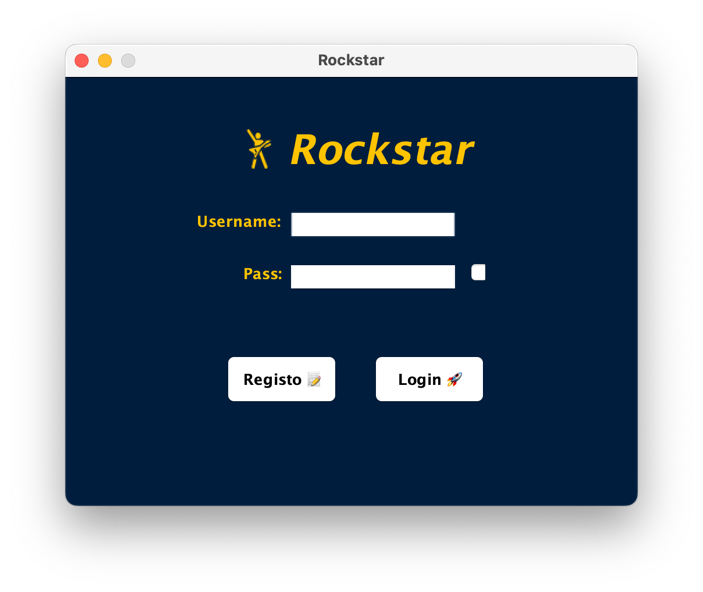
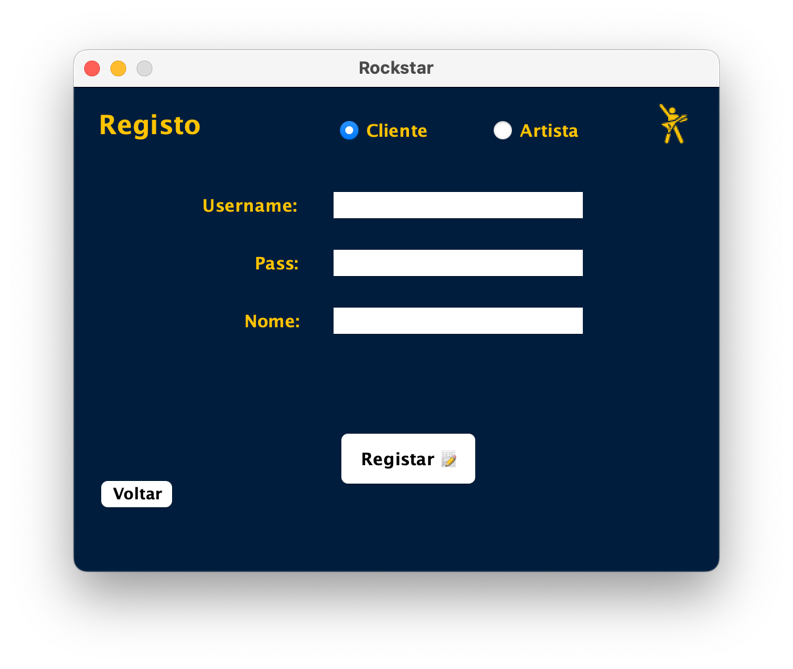


Figura - Painel de Login

A aplicação inicia sempre no menu *Login* onde é apresentado ao utilizador a possibilidade de fazer login ou de efetuar o seu registo caso ainda não se tenha registado na aplicação.



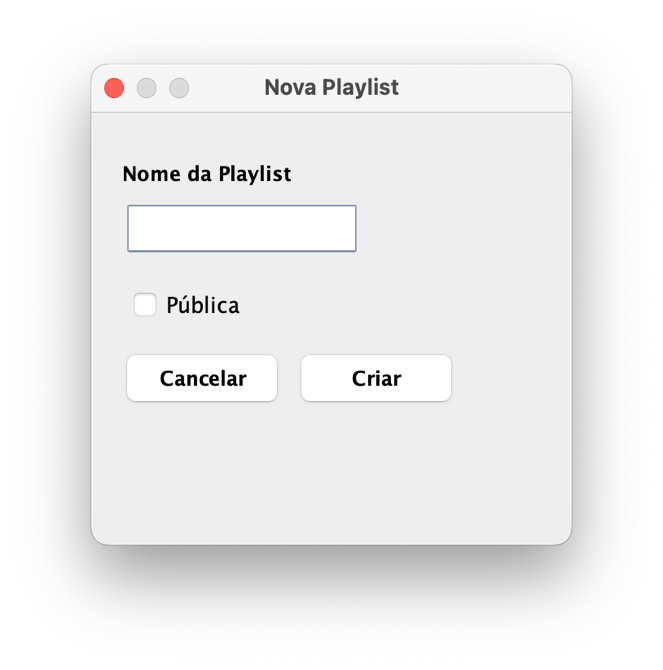
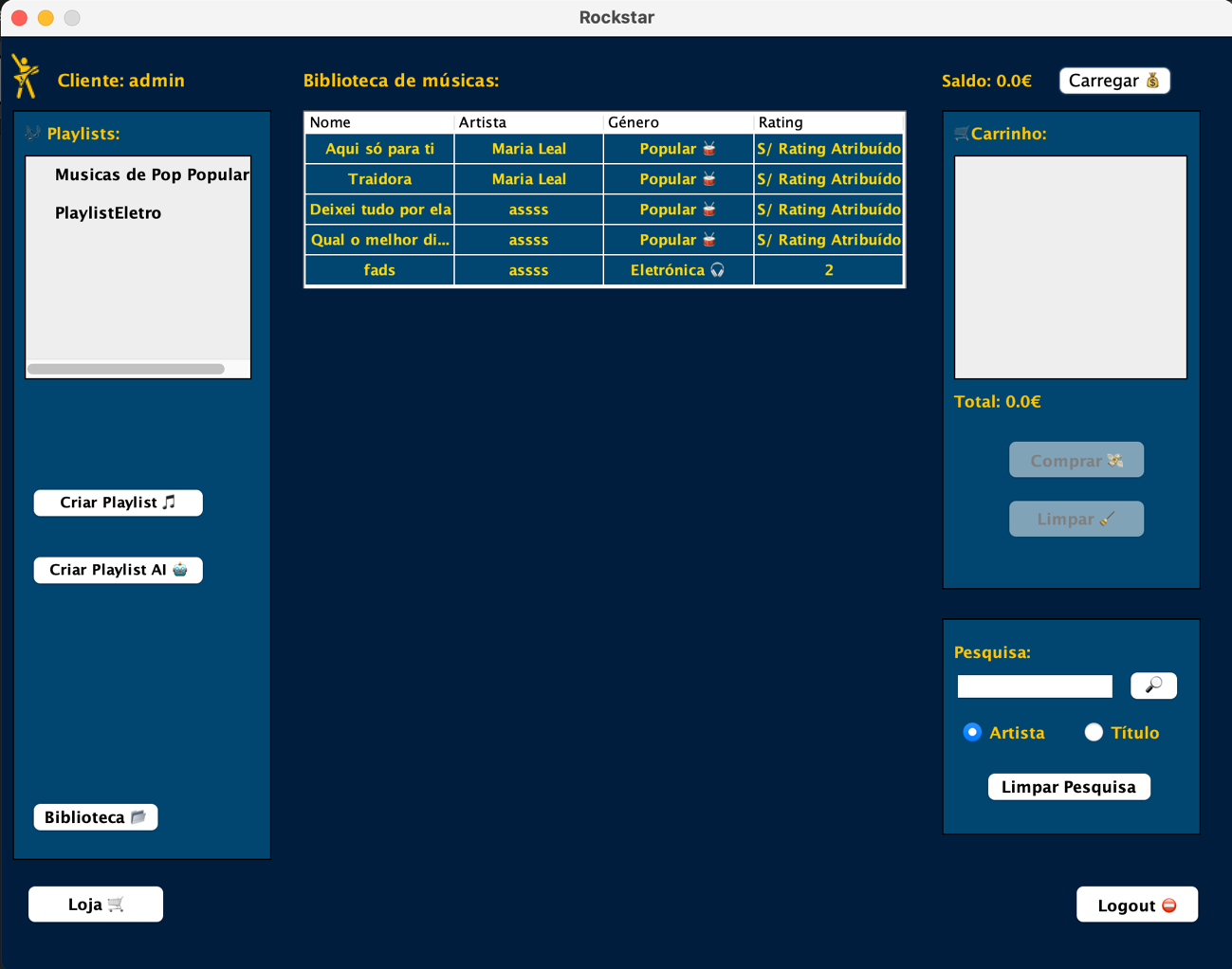
*Figura 2 - Painel de Registo*

Caso opte por se registar é lhe aberto o menu de *Registo*, onde é apresentado um *radio button* que dá a possibilidade ao utilizador de se registar como *Cliente* ou *Músico*, caso escolha registar-se como *Cliente* são apresentados 3 (três) *Text Fields* onde é pedido que insira um *username*, uma *password* e um nome. Caso o utilizador escolha registar-se como *Artista* para além dos *Text Fields* referidos anteriormente é ainda apresentado mais um *Text Field* onde é pedido que o utilizador insira um código PIN. Ao clicar no botão ‘*Validar’* todos estes *Text Fields* passam por uma série de verificações. Para o *username* é chamado um método *verificarUsername*, este método verifica se o *username* já se encontra registado no sistema e se o mesmo é constituído por mais de 4 (caracteres). Para a *password* é chamado o método *verficarPass*, este método verifica se a pass é constituída por mais de 4 (quatro) caracteres. Para o nome é apenas verificado se o mesmo tem mais que 1 (um) caracter. No caso de registo de *Artista* onde é apresentado o *Text Field* para o código PIN, é chamado o método *verificarPIN*, onde se verifica se o código é constituído por apenas e somente 4 dígitos e apenas carateres numéricos.

De volta ao menu *Login*, após o preenchimento dos *Text* *Fields* de *username* e *password* e quando clicado o botão de *Login* é chamado o método *verificarUtilizador*, este método verifica se o *username* se encontra na lista de *Clientes* ou de *Artistas*, e caso encontre o mesmo, devolve o respetivo objeto do *Utilizador* correspondente ao *username* inserido, para que este possa ser gravado numa variável temporária de *utilizadorAtual*. Após a obtenção do objeto do *Utilizador* correspondente ao *username* inserido é chamado o método *verificaLoginPass* onde é feita a verificação se *password* inserida corresponde à *password* do objeto *Utilizador* que foi obtido. Depois de feitas estas verificações a aplicação verifica ainda se o objeto de *Utilizador* que foi devolvido pertence a uma instância de *Cliente* ou a *Artista*, e caso pertença a *Artista* a aplicação abre uma nova janela para que possa ser inserido o código PIN referente ao *Artista* em questão, é então chamado o método *verificarLoginPin* para verificar a correspondência do mesmo.

Caso seja um Cliente a efetuar login na aplicação, é lhe apresentado uma nova janela com toda a sua interface e funcionalidades atribuídas. Esta interface divide-se em 4 (quatro) painéis.

*Figura 3 - Interface Cliente*



*Figura 4 - Painel Criar Playlist*

O primeiro painel, do lado esquerdo, é onde é feita a gestão de *Playlists* pertencentes ao *Cliente* que se encontra autenticado. As várias *Playlists* que o *Cliente* possui são apresentadas num primeiro painel, sendo que as mesmas são clicáveis para que possa ser mostrado o conteúdo das mesmas no painel central. Ainda neste painel é apresentado um botão *‘Criar Playlist’* para que o *Cliente* possa criar uma nova *Playlist*, este botão faz abrir uma nova janela onde é pedido um nome para a nova *Playlist* e um *checkbox* para se a mesma é criada com visibilidade pública ou privada, e ao ser clicado o botão ‘*Criar’*, é criada uma nova instância de *Playlist* (ainda sem *Músicas* atribuídas), sendo essa *Playlist* automaticamente adicionada ao *Arraylist* de *Playlists* pertencentes ao utilizador e à *Arraylist* de *Playlists* totais presente na classe *RockstarInc*. Ao clicar no botão ‘*Criar* *Playlist* *AI’* é aberta uma nova janela para que o *Cliente* possa definir as condições para a criação desta nova *Playlist* inteligente, nesta nova janela é pedido ao *Cliente* um nome para a *Playlist*, um *checkbox* para definir a visibilidade com que a mesma é criada, um *Text* *Field* para definir o número de músicas a incluir nessa *Playlist* e uma lista *Dropdown* para definir o género da *Playlist* a ser criada. Ao clicar no botão ‘*Criar’* é chamado o método *criaPlaylistAI*, este método isola todas as músicas do genero selecionado pelo Cliente num *ArrayList* de *Músicas* temporário, *Músicas* estas apenas obtidas da biblioteca do *Cliente*, e depois de isoladas, aleatoriamente, selecionará o número de *Músicas* especificado pelo *Cliente* para a nova *Playlist* inteligente. Neste painel é ainda apresentado um botão ‘*Biblioteca’* que faz apresentar no painel central todas as *Músicas* adquiridas pelo *Cliente*.



*Figura 5 - Painel Criar Playlist AI*



*Figura 6 - Painel Dar Rating*

Uma imagem com texto, captura de ecrã, número

Descrição gerada automaticamenteO painel central da interface do *Cliente* funciona como ‘ecrã’ dos diversos Grupo de Músicas que o *Cliente* poderá selecionar. Este painel funciona sempre com a exibição de uma tabela onde são apresentados diversos atributos das músicas pertencentes ao *Grupo de Músicas* selecionado. Inicialmente a tabela de *Músicas* apresentada é referente à biblioteca de *Músicas* adquiridas pelo *Cliente*. Esta tabela pode ser alterada se o *Cliente* clicar numa das suas *Playlists*, ficando assim a tabela a apresentar as *Músicas* presentes nessa mesma *Playlist*. Ainda quando uma *Playlist* está a ser apresentada são disponibilizados ao *Cliente* autenticado dois botões que permitem alterar a visibilidade da *Playlist* clicada ou remover a mesma da *ArrayList* de *Playlists* presente em cada *Cliente*. Ao clicar no botão ‘*Loja’*, presente no painel esquerdo da interface, a tabela irá apresentar todas as *Músicas* presentes na aplicação para serem adquiridas. Ao mostrar as músicas presentes na biblioteca do *Cliente* é disponibilizado um *Pop-up Menu* que é acionado através do clique com o botão direito do rato em cima de uma *Música*, este *Pop-up Menu* oferece duas opções, a de adicionar a música clicada a uma *Playlist* ou de atribuir/alterar um *Rating* à *Música* selecionada. Ao selecionar a opção de adicionar a música a uma playlist é chamado o método *addMusica* que verifica se a mesma já faz parte da *Playlist* e a adiciona caso ainda não pertença à *Playlist*. Quando estão a ser apresentadas as *Músicas* presentes na Loja é disponibilizado um *Pop-up Menu* com duas opções. A primeira opção, Comprar Música, que chama o método *comprarMusica*, este método faz a verificação se *Música* clicada tem algum preço atribuído e caso tenha a mesma irá ser adicionada ao carrinho de compras do *Cliente* autenticado para posterior validação, se a *Música* for grátis a mesma irá ser diretamente adicionada à biblioteca de *Músicas* do *Cliente*. É também apresentado a opção de ver histórico de preços, ao clicar nesta opção será aberta uma nova janela onde são apresentadas todas as alterações de *Preço* que a uma *Música* sofreu desde a sua criação.

*Figura 7 - Painel Histórico de Preços*

Uma imagem com texto, captura de ecrã, multimédia, aparelho

Descrição gerada automaticamente

Já no painel da direita começa por ser apresentado no seu topo a informação com o saldo para compras que o *Cliente* possui, é também apresentado um botão ‘*Carregar’* que faz abrir uma nova janela onde é pedido ao *Cliente* um valor com que deseja carregar o seu saldo, ao clicar no botão ‘*Carregar’* é chamado o método *carregarSaldo* que adiciona a quantia introduzida no *Text Field* da janela ao atual saldo do *Cliente*. Também no painel da direita é apresentado o carrinho de compras do *Cliente*, neste carrinho de compras vão sendo apresentadas as *Músicas* e respetivo *Preço* que o *Cliente* pretende adquirir da loja de *Músicas*, é também apresentado o valor total do somatório dos *Preços* das *Músicas* presentes no carrinho. Ainda dentro da interface do carrinho de compras são apresentados dois botões, um botão ‘*Comprar’*, quando clicado este método chama o construtor do objeto *Compra*, este construtor começa por verificar se o *Cliente* possui saldo suficiente para efetuar a *Compra*, se o mesmo possuir saldo suficiente o construtor irá passar todas as *Músicas* presentes no *ArrayList* de *Músicas* presentes no carrinho de compras de compras do *Cliente* para a biblioteca do mesmo. Posteriormente desconta do saldo o valor da respetiva *Compra* e guarda num *ArrayList* de *Compras* a respetiva *Compra* efetuada com a lista de *Músicas* compradas, o valor total da compra e a data em foi efetuada. É também apresentado um botão ‘*Limpar’* que chama o método *limparCarrinho*, método este que remove todas as *Músicas* presentes no *ArrayList* de *Músicas* referente ao carrinho de compras do *Cliente*.

*Figura 8 - Painel de Carregamento de Saldo*

Por último temos o quarto painel, o painel de Pesquisa, este painel permite ao *Cliente* pesquisar em qualquer *Grupo de Músicas* que esteja a ser mostrado no painel central. É possível ao *Cliente* pesquisar por dois atributos das *Músicas* diferentes, conforme o *Radio Button* que esteja *selecionado*, se for *‘Artista’* o selecionado e ao clicar no botão *‘Lupa’* irá ser chamado o método *pesquisaArtista*, que verificará o que se encontra inserido no *Text Field* e comparará o com a *String* do atributo a pesquisar, se for *‘Titulo’* o selecionado e ao clicar no botão *‘Lupa’* irá ser chamado o método *pesquisaTitulo*, que verificará o que se encontra inserido no *Text Field* e comparará o com a *String* do atributo a pesquisar.

*Figura - Interface Artista*

Caso seja um *Artista* a efetuar login na aplicação, é lhe apresentado uma nova janela com toda a sua interface e funcionalidades atribuídas. Esta interface divide-se em 4 (quatro) painéis.



Figura 9 - Interface Artista

O primeiro painel, do lado esquerdo, é onde é feita a gestão de *Albuns* pertencentes ao *Artista* que se encontra autenticado. Os várias *Albuns* que o *Artista* possui são apresentadas num primeiro painel, sendo que os mesmos são clicáveis para que possa ser mostrado o conteúdo dos mesmos no painel central. Ainda neste painel é apresentado um botão *‘Adicionar Album’* para que o *Artista* possa criar um novo *Album*, este botão faz abrir uma nova janela onde é pedido um nome para o novo *Album* e uma *Dropdown List* para que possa ser selecionado o género do *Album* a criar, e ao ser clicado o botão ‘*Criar’*, é criada uma nova instância de *Album* (ainda sem *Músicas* atribuídas), sendo esse *Album* automaticamente adicionado ao *Arraylist* de *Albuns* pertencentes ao utilizador e à *Arraylist* de *Albuns* totais presente na classe *RockstarInc*. Ao clicar no botão ‘*Adicionar Música’* é aberta uma nova janela para que o *Artista* possa definir os parâmetros para a criação desta nova *Música*, nesta nova janela é pedido ao *Artista* um nome para a *Música*, uma *dropdown* *list* para definir o genero com que a mesma é criada, outra *dropdown list* para definir o *Album* onde essa nova *Música* será inserida, um *text field* para inserir o preço a atribuir à *Música* e um *checkbox* para definir a visibilidade com que a mesma criada. Ao clicar no botão ‘*Criar’* é chamado o construtor da *Música*, este construtor tem em conta os parâmetros especificados pelo utilizador na nova Janela e adiciona a nova instância da *Música* a um *Album* caso seja especificado, ou ao *ArrayList* de *Músicas* *Singles* caso o utilizador especifique *Singles* como destino. Neste painel é ainda apresentado um botão ‘*As Minhas Músicas’* que faz apresentar no painel central todas Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamenteas *Músicas* pertencentes ao *Artista*.



Figura 10 - Painel Criação de Album

Figura 12 - Painel Alterar Título

Figura 11 - Painel Alterar Preço

*Figura - Painel Criar Album*

Uma imagem com texto, captura de ecrã, software, multimédia

Descrição gerada automaticamenteO painel central da interface do *Artista* funciona como ‘ecrã’ dos diversos Grupo de Músicas que o *Artista* pode selecionar. Este painel funciona sempre com a exibição de uma tabela onde são apresentados diversos atributos das músicas pertencentes ao *Grupo de Músicas* selecionado. Inicialmente a tabela de *Músicas* apresentada é referente à biblioteca de *Músicas* pertencentes ao *Artista*. Esta tabela pode ser alterada se o *Artista* clicar num dos seus *Albuns*, ficando assim a tabela a apresentar as *Músicas* presentes nesse mesmo *Album*. Ao mostrar as músicas presentes na biblioteca do *Artista* é disponibilizado um *Pop-up Menu* que é acionado através do clique com o botão direito do rato em cima de uma *Música*, este *Pop-up Menu* oferece quatro opções, a de alterar a visibilidade da *Música* clicada, alterar o título da *Música* clicada, alterar o preço da *Música* clicada e adicionar a *Música* clicada a um *Album* previamente criado. Ao selecionar a opção de alterar a visibilidade, é automaticamente atribuído o valor inverso ao que estava guardado no atributo da Música referente à sua visibilidade. Ao selecionar a opção Alterar Título é aberta uma nova janela onde o utilizador pode inserir um novo título a atribuir à *Música* clicada. Ao selecionar a opção Alterar Preço é aberta uma nova janela onde o utilizador pode inserir um novo preço a atribuir à *Música* clicada. Ao selecionar a opção *Adicionar a Album* o mesmo apresenta uma lista dos *Albuns* disponíveis para adicionar a *Música* clicada, ao clicar num álbum para ser adicionada o método verifica se a *Música* clicada já tem um *Album* atríbuido e só adiciona caso a mesma não tenha qualquer *Album*.

Figura 13 - Painel de Criação de Música

*Figura - Painel Alterar Titulo*

*Figura - Painel Alterar Preço*

Uma imagem com texto, captura de ecrã, software, Sistema operativo

Descrição gerada automaticamente No painel à direita da interface do Artista é apresentado um painel com diversas estatísticas da aplicação. É possível visualizar estatísticas referentes a: total de utilizadores na aplicação, total de músicas na aplicação, valor total do somatório do preço das *Músicas* presentes na aplicação, valor total de faturação decorrente da compra de todas as *Músicas* na aplicação e total de *Albuns* presentes na aplicação por género. É também apresentado um botão ‘Ver Tops’ que faz surgir uma nova janela onde é possível visualizar as cinco Músicas mais vendidas da aplicação e os cinco Artistas com mais vendas na aplicação.

*Figura 14 - Painel de Tops de Vendas*

Por último temos o quarto painel, o painel de Pesquisa, este painel permite ao *Artista* pesquisar em qualquer *Grupo de Músicas* que esteja a ser mostrado no painel central. É possível ao *Artista* pesquisar por dois atributos das *Músicas* diferentes, conforme o *Radio Button* que esteja *selecionado*, se for *‘Titulo’* o selecionado e ao clicar no botão *‘Lupa’* irá ser chamado o método *pesquisaTitulo*, que verificará o que se encontra inserido no *Text Field* e comparará o com a *String* do atributo a pesquisar, se for *‘Album’* o selecionado e ao clicar no botão *‘Lupa’* irá ser chamado o método *pesquisaAlbum*, que verificará o que se encontra inserido no *Text Field* e comparará o com a *String* do atributo a pesquisar.

**Conclusão**

Consideramos que este projeto foi um trabalho muito desafiador e ao mesmo tempo interessante, principalmente por ter sido o nosso primeiro contacto a sério com interface gráfica. Conseguimos implementar as funcionalidades propostas e por isso consideramos ter sido um trabalho bem-sucedido.

Com a realização dos mockups na fase intermédia do projeto ficámos um pouco apreensivos de como iriamos conseguir implementar graficamente o que idealizávamos visto que a nossa experiência com interface gráfica era pouca ou nenhuma. Ficámos muito satisfeitos quando no final da aplicação voltámos a olhar para os nossos mockups iniciais e pudemos notar que pouco ou nada diferem da nossa aplicação final.

Temos noção que o tempo que despendemos a implementar a nossa interface gráfica, para que ficasse mais agradável e user-friendly nos roubou algum tempo que poderíamos ter despendido no backend da nossa aplicação, com a utilização de mais métodos polimórficos e eliminação de alguns atributos desnecessários com a aplicação de diferentes métodos. Ainda assim achámos a lógica da interface gráfica bastante interessante, pois a mesma obrigou-nos a muita pesquisa e á criação de métodos para que conseguíssemos aquilo que desejávamos.

Foi na criação dos diagramas UML que pudemos constatar que na fase intermédia do projeto pouca noção tínhamos dos métodos que iríamos utilizar e por onde ficariam distribuídos. Ao olhar para o diagrama UML final conseguimos ver que conseguimos seguir a sugestão inicial do professor, distribuindo grande parte dos métodos que se encontravam concentrados na classe *Rockstar.* Agradavelmente constatámos também que não necessitámos de mais nenhuma classe das que inicialmente idealizámos para a nossa aplicação.

Ao analisar o código pudemos notar alguma falta de uniformização na aplicação de alguns métodos de verificação de dados, onde algumas vezes são aplicados na interface gráfica e outras nas classes das classes correspondentes. Este será um aspeto que teremos atenção num próximo projeto que nos seja atribuído.

De modo geral, ficámos satisfeitos com o resultado final da nossa aplicação e finalizamos este projeto com a certeza de que evoluímos muito nos últimos meses mas que ainda há muito o que aprender.

**Anexo I:**

Uma imagem com texto, diagrama, Esquema, Paralelo

Descrição gerada automaticamente

**Anexo II:**

Uma imagem com texto, diagrama, Esquema, Desenho técnico

Descrição gerada automaticamente