Trabalho Prático Nº1

Programação Orientada a Objetos

Rafael Ramalho
Nº 16452 / ESI

Nelson Araújo
Nº 20743 / ESI

22 de dezembro de 2023

Resumo

C# é uma linguagem de programação orientada a objetos amplamente utilizada, especialmente no desenvolvimento de aplicativos Windows e serviços web. Destaca-se por sua forte tipagem, suporte a herança, encapsulamento e polimorfismo, fundamentais para a programação orientada a objetos. A linguagem oferece uma sintaxe clara e estruturada, facilitando o desenvolvimento e manutenção de código. A plataforma .NET, na qual o C# é executado, fornece uma ampla biblioteca de classes reutilizáveis, contribuindo para a eficiência no desenvolvimento de software. Além disso, a linguagem é compilada, o que resulta em um desempenho otimizado durante a execução do programa.

Índice

1 Introdução

Este relatório propõe uma exploração no desenvolvimento de soluções em C# para desafios de complexidade moderada, com o objetivo de consolidar conceitos fundamentais para a cadeira de Programação Orientado a Objetos. Ao longo deste relatório, serão identificadas classes, definidas estruturas de dados e implementados os processos necessários para realização do trabalho prático.

O âmbito deste trabalho não se restringe apenas à codificação em C#, estendendo-se à análise aprofundada de problemas reais, onde não só são aprimoradas as competências de programação, mas também a compreensão dos desafios enfrentados no desenvolvimento do software. Dividindo o trabalho em duas fases distintas, na primeira fase, são focadas as Classes identificadas, a Implementação dessas mesmas e as Estruturas de dados utilizadas, onde nesta abordagem é permitido uma análise mais aprofundada e sistemática, estabelecendo as bases sólidas para o desenvolvimento posterior das soluções em C# para a resolução de problemas. Já na segunda fase, é focada a implementação final das classes e serviços apresentados na fase 1, consolidando a teoria em prática e refinando ainda mais as soluções propostas. A aplicação demontra os serviços implementados, oferecendo uma visão tangível do funcionamento efetivo do software desenvolvido. Ao abordar os objetivos delineados, é procurado não apenas consolidar conceitos básicos, mas também aplicar esses conhecimentos de forma prática. Dessa forma, a experiência adquirida durante a Unidade Curricular vai além da simples assimilação de conteúdo, convertendo-se em habilidades concretas no campo do desenvolvimento de software.

Este relatório encontra-se dividido em 3 partes, sendo estas a Introdução, o Desenvolvimento e a Conclusão. Na 2ª parte irá estar apresentado o conteúdo principal deste trabalho, sendo estes também os objetivos do mesmo.

1.1 Descrição do Problema

O projeto visa desenvolver um sistema abrangente para uma loja virtual de jogos, abordando desafios importantes na gestão de produtos, transações e experiência do cliente. O núcleo do problema reside na criação de uma arquitetura eficiente que permita a representação adequada de jogos, a gestão de compras através de faturas personalizadas, a definição de perfis de clientes e a facilitação de um carrinho de compras intuitivo.

2 Desenvolvimento

Para a resolução do problema em questão, foram implementadas diversas classes em C#, que proporcionaram uma estrutura modular e organizada, facilitando assim compreensão e manutenção do código. As classes em si utilizadas, tinham cada a sua respetiva importância, como por exemplo a classe Jogo, responsável por gerir informações relacionadas aos jogos no sistema e a classe Fatura, onde esta tinha como objetivo a manipulação e armazenamento de dados relacionados às transações de compra.

Tendo em conta, a complexidade do trabalho, estas classes ainda poderão sofrer atualizações ou mesmo alterações, de forma a otimizar a implementação do projeto. Sendo assim, para a primeira fase deste trabalho, todas as classes estão representadas em baixo, tal como as suas estruturas e explicações.

2.1 Representação de Jogos (`Classe Jogo`)

2.1.1 Objetivo

Desenvolver a classe `Jogo` para representar as características fundamentais de um jogo, como nome, preço, plataforma e género. Esta classe é importante para a organização dos produtos na loja.

2.1.2 Estrutura da Classe

A classe possui quatro atributos principais:

- **Nome (`string`):** Armazena o nome do jogo.
- **Preço (`double`):** Armazena o preço do jogo.
- **Plataforma (`string`):** Armazena a plataforma para a qual o jogo está disponível.
- **Género (`string`):** Armazena o género do jogo.

2.1.3 Métodos da Classe

Construtor `Jogo` **Descrição:** Inicializa um novo jogo com base nas informações fornecidas.

Parâmetros:

- `string nome` - Nome do jogo.
- `double preco` - Preço do jogo.
- `string plataforma` - Plataforma para a qual o jogo está disponível.
- `string genero` - Género do jogo.

Funcionalidades: A classe `Jogo` proporciona uma maneira de representar jogos na loja virtual, abrangendo informações essenciais como nome, preço, plataforma e género. O construtor facilita a criação de instâncias de jogos com todas as informações necessárias.

2.2 Gestão de Compras (`Classe Fatura`)

2.2.1 Objetivo

Implementar a classe `Fatura` para representar e imprimir informações sobre as compras efetuadas pelos clientes. A `Fatura` desempenha um papel importante na emissão de documentos para os clientes e na contabilização das transações.

2.2.2 Estrutura da Classe

A classe possui três atributos principais:

- **Carrinho** (`CarrinhoDeCompras`)**:** Referência ao carrinho de compras associado à fatura.
- **DataCompra** (`DateTime`)**:** Armazena a data e hora em que a compra foi efetuada.
- **ValorTotal** (`double`)**:** Armazena o valor total da compra.

2.2.3 Métodos da Classe

Construtor `Fatura` **Descrição:** Inicializa uma nova fatura com base no carrinho de compras fornecido.

Parâmetro: `CarrinhoDeCompras carrinho` - Carrinho de compras associado à fatura.

Método `ImprimirFatura` **Descrição:** Imprime na consola os detalhes da fatura, incluindo data, cliente, itens comprados e valor total.

Funcionalidades: A classe `Fatura` automatiza a geração de faturas com base nas informações do carrinho de compras associado. O método `ImprimirFatura` exibe detalhes formatados da fatura na consola.

2.3 Perfil do Cliente (`Classe Cliente`)

2.3.1 Objetivo

Criar a classe `Cliente` para representar as informações essenciais de um cliente, como nome e endereço de e-mail. Esta classe é vital para associar informações específicas do cliente aos diversos elementos do sistema.

2.3.2 Estrutura da Classe

A classe possui dois atributos principais:

- **Nome (`string`):** Armazena o nome do cliente.
- **Email (`string`):** Armazena o endereço de e-mail do cliente.

2.3.3 Métodos da Classe

Construtor `Cliente` **Descrição:** Inicializa um novo cliente com um nome e endereço de e-mail específicos.

Parâmetros:

- `string nome` - Nome do cliente.
- `string email` - Endereço de e-mail do cliente.

Funcionalidades: A classe `Cliente` proporciona uma maneira de armazenar e acessar as informações básicas do cliente, como nome e e-mail.

2.4 Carrinho de Compras (`Classe CarrinhoDeCompras`)

2.4.1 Objetivo

Desenvolver a classe `CarrinhoDeCompras` para representar o carrinho de compras de um cliente. Esta classe armazena a lista de jogos selecionados pelo cliente e fornece métodos para adicionar jogos e calcular o valor total da compra.

2.4.2 Estrutura da Classe

A classe possui dois atributos principais:

- **Itens (`List<Jogo>`):** Armazena a lista de jogos adicionados ao carrinho.
- **Comprador (`Cliente`):** Armazena o cliente associado ao carrinho.

2.4.3 Métodos da Classe

Construtor `CarrinhoDeCompras` **Descrição:** Inicializa um novo carrinho de compras associado a um cliente específico.

Parâmetro: `Cliente comprador` - Cliente associado ao carrinho.

Método `AdicionarJogoAoCarrinho` **Descrição:** Adiciona um jogo à lista de itens no carrinho.

Parâmetro: `Jogo jogo` - Jogo a ser adicionado ao carrinho.

Método `CalcularTotal` **Descrição:** Calcula o valor total dos jogos presentes no carrinho.

Retorno: Valor total da compra.

2.5 Funcionalidades Esperadas

- A classe `Jogo` permite representar jogos na loja virtual, abrangendo informações essenciais como nome, preço, plataforma e género.
- A classe `Fatura` automatiza a geração de faturas com base nas informações do carrinho de compras associado.
- A classe `Cliente` proporciona uma maneira de armazenar e acessar as informações básicas do cliente, como nome e e-mail.
- A classe `CarrinhoDeCompras` permite adicionar jogos ao carrinho e calcular o valor total da compra.

2.6 Modelo MVC

Ao optar pelo padrão Modelo-Visão-Controlador (MVC) na arquitetura do software em questão durante o projeto, a iniciativa visou proporcionar uma organização eficiente, facilitar os testes e a manutenção. O MVC dividiu o sistema em três componentes principais: o Modelo, encarregado dos dados e da lógica de negócios; a Visão, responsável pela apresentação visual ao utilizador; e o Controlador, que gere a comunicação entre o Modelo e a Visão, controlando o fluxo de dados.

2.6.1 Arquitetura do Projeto

O projeto é estruturado em três camadas principais: **Models**, **Views** e **Controllers**. Cada camada é implementada como uma biblioteca dinâmica (DLL) separada para promover a modularidade e a reutilização de código. A interligação entre essas DLLs é importante para o funcionamento adequado do programa. Essa abordagem permite uma organização mais eficiente do software, facilitando a manutenção e promovendo uma maior modularidade, o que contribui para uma experiência de desenvolvimento mais fluída no contexto académico.

2.6.2 Camada de Models

A camada **Models** engloba as classes que representam a estrutura de dados da aplicação. Estas classes encapsulam a lógica de negócios e os dados que serão manipulados pelo sistema. A DLL correspondente a esta camada é responsável por disponibilizar as funcionalidades relacionadas aos modelos de dados, tornando assim esta abordagem a mais apropriada para uma gestão da lógica de negócios.

2.6.3 Camada de Views

A camada **Views** assume a responsabilidade pela apresentação dos dados ao utilizador, incluindo as interfaces gráficas e outros componentes de apresentação. A DLL associada a esta camada trata da interação com o utilizador e da exibição visual dos dados provenientes da camada **Models**. Esta abordagem contribui para uma experiência mais intuitiva por parte dos utilizadores, separando claramente as preocupações de apresentação das lógicas subjacentes.

2.6.4 Camada de Controllers

A camada **Controllers** desempenha o papel de intermediário entre as camadas de **Models** e **Views**. Esta coordena as interações entre a lógica de negócios (**Models**) e a interface do utilizador (**Views**). A DLL associada a esta camada gere os eventos do utilizador e traduz essas ações em operações na camada de **Models**. Essa abordagem, ao separar claramente as responsabilidades de gestão de eventos e coordenação de interações, promove uma arquitetura mais modular no âmbito acadêmico.

2.6.5 Interligação das DLLs

A interligação entre as DLLs revela-se fundamental para assegurar uma comunicação eficaz entre as diversas camadas. Essa ligação é essencial para garantir o correto funcionamento do sistema, possibilitando a troca de informações de maneira eficiente entre as camadas de **Models**, **Views** e **Controllers**. Sendo assim, esta abordagem, ao promover uma comunicação bem estabelecida entre as bibliotecas dinâmicas, contribui para a coesão do software.

- A DLL da camada **Models** é utilizada pela DLL da camada de **Controllers** para possibilitar o acesso aos modelos de dados e à lógica de negócios.
- A DLL da camada **Views** utiliza a DLL da camada de **Controllers** para interagir com a lógica de negócios e apresentar os dados apropriados.

Views

2.7 AdministradorView.cs

2.7.1 Descrição

A classe `AdministradorView` assume a responsabilidade de exibir o menu e facilitar as interações do administrador na loja de jogos.

2.7.2 Método ExibirMenuAdministrador

Este método apresenta um menu interativo ao administrador, permitindo a execução de diversas operações.

2.7.3 Opção 1 - Registar Administradores

Quando o administrador escolhe a opção 1, o método `RegistarAdministrador` é chamado.

Funcionalidade Este método permite o registo de novos administradores na loja.

Parâmetros

- `admin`: Instância do administrador atual.
- `administradores`: Lista de administradores registados.

2.7.4 Opção 2 - Adicionar Jogos

Quando o administrador escolhe a opção 2, o método `AdicionarJogos` é chamado.

Funcionalidade Este método permite a adição de novos jogos à lista de jogos disponíveis na loja.

Parâmetros

- `admin`: Instância do administrador atual.
- `jogos`: Lista de jogos disponíveis na loja.

2.7.5 Opção 3 - Repor Stock

Quando o administrador escolhe a opção 3, o método `ReporStock` é chamado.

Funcionalidade Este método permite que o administrador reponha o stock de jogos na loja.

Parâmetros

- `jogos`: Lista de jogos disponíveis na loja.

2.7.6 Opção 4 - Editar Jogo

Quando o administrador escolhe a opção 4, o método `EditarJogo` é chamado.

Funcionalidade Este método permite que o administrador edite as informações de um jogo específico na lista de jogos disponíveis na loja.

Parâmetros

- `jogos`: Lista de jogos disponíveis na loja.

2.8 ClienteView.cs

2.8.1 Descrição

A classe `ClienteView` é responsável por exibir o menu e facilitar as interações do cliente na loja de jogos.

2.8.2 Método ExibirMenuCliente

Este método apresenta um menu interativo ao cliente, permitindo a execução de diversas operações.

2.8.3 Opção 1 - Listar Jogos Disponíveis

Quando o cliente escolhe a opção 1, o método `ListarJogos` é chamado.

Funcionalidade Este método lista os jogos disponíveis na loja, exibindo informações relevantes.

Parâmetros

- `jogos`: Lista de jogos disponíveis na loja.

2.8.4 Opção 2 - Comprar Jogo

Quando o cliente escolhe a opção 2, o método `AdicionarAoCarrinho` é chamado.

Funcionalidade Este método permite que o cliente adicione jogos ao carrinho de compras.

Parâmetros

- `cliente`: Instância do cliente atual.
- `jogos`: Lista de jogos disponíveis na loja.

2.8.5 Opção 3 - Visualizar Carrinho

Quando o cliente escolhe a opção 3, o método `VisualizarCarrinho` é chamado.

Funcionalidade Este método exibe o carrinho de compras atual do cliente, mostrando os jogos selecionados e o total a ser pago.

Parâmetros

- `cliente`: Instância do cliente atual.
- `jogos`: Lista de jogos disponíveis na loja.

2.8.6 Opção 4 - Editar Perfil

Quando o cliente escolhe a opção 4, o método `EditarPerfil` é chamado.

Funcionalidade Este método permite que o cliente edite as informações do seu perfil, como nome, endereço, etc.

Parâmetros

- `cliente`: Instância do cliente atual.
- `clientes`: Lista de clientes cadastrados na loja.

2.8.7 Opção 5 - Apagar Conta

Quando o cliente escolhe a opção 5, o método `ApagarConta` é chamado.

Funcionalidade Este método permite que o cliente apague sua conta na loja, removendo todas as informações associadas a ela.

Parâmetros

- `cliente`: Instância do cliente atual.
- `clientes`: Lista de clientes cadastrados na loja.

Controllers

2.9 AdministradorController.cs

2.9.1 Descrição

A classe `AdministradorController` atua como um controlador para a interação entre o modelo (`Administrador`) e a visualização (`AdministradorView`) no contexto do administrador na loja de jogos.

2.9.2 Construtor

Este controlador possui um construtor que recebe instâncias do modelo e da visualização como parâmetros.

Parâmetros

- `administradorModel`: Instância do modelo do administrador.
- `administradorView`: Instância da visualização do administrador.

2.10 ClienteController.cs

2.10.1 Descrição

A classe `ClienteController` atua como um controlador para a interação entre o modelo (`Cliente`) e a visualização (`ClienteView`) no contexto do cliente na loja de jogos.

2.10.2 Construtor

Este controlador possui um construtor que recebe instâncias do modelo e da visualização como parâmetros.

Parâmetros

- `clienteModel`: Instância do modelo do cliente.
- `clienteView`: Instância da visualização do cliente.

Models

2.11 Administrador.cs

2.11.1 Descrição

A classe `Administrador` representa um administrador da loja de jogos e contém informações como nome, e-mail e senha.

2.11.2 Métodos

RegistrarAdministrador Este método permite o registo de novos administradores na loja.

Parâmetros

- `administradores`: Lista de administradores registados.

AdicionarJogo Este método permite adicionar novos jogos à lista de jogos disponíveis na loja.

Parâmetros

- `jogos`: Lista de jogos disponíveis na loja.

ReporStock Este método repõe o stock de um jogo específico, adicionando uma quantidade adicional ao stock atual.

Parâmetros

- `jogos`: Lista de jogos disponíveis na loja.

EditarJogo Este método edita as informações de um jogo existente na lista de jogos.

Parâmetros

- `jogos`: Lista de jogos disponíveis na loja.

2.12 CarrinhoDeCompras.cs

2.12.1 Descrição

A classe `CarrinhoDeCompras` representa o carrinho de compras de um cliente na loja de jogos. Mantém uma lista de itens no carrinho e o cliente associado.

2.12.2 Propriedades

- `Itens`: Obtém ou define a lista de itens no carrinho.
- `Comprador`: Obtém ou define o cliente associado ao carrinho.

2.12.3 Construtor

- `CarrinhoDeCompras(Cliente comprador)`: Inicializa uma nova instância da classe CarrinhoDeCompras.

2.12.4 Métodos Públicos

`AdicionarJogoAoCarrinho` Adiciona um jogo ao carrinho com a quantidade especificada.

Parâmetros

- `jogo`: O jogo a ser adicionado ao carrinho.
- `quantidade`: A quantidade desejada do jogo.

`CalcularTotal` Calcula o total do carrinho somando os preços de todos os jogos no carrinho.

Retorno

- O valor total do carrinho.

`RemoverDoCarrinho` Remove um jogo específico do carrinho de compras do cliente.

Parâmetros

- `cliente`: O cliente que possui o carrinho.
- `jogos`: A lista de jogos disponíveis.

2.13 Cliente.cs

2.13.1 Descrição

A classe `Cliente` representa um cliente da loja de jogos, incluindo informações do perfil, histórico de compras e carrinho de compras.

2.13.2 Atributos

- `Nome`: Obtém ou define o nome do cliente.
- `Email`: Obtém ou define o endereço de e-mail do cliente.
- `Senha`: Obtém ou define a senha do cliente.
- `Carrinho`: Obtém ou define o carrinho de compras do cliente.
- `HistoricoCompras`: Obtém ou define o histórico de compras do cliente.

2.13.3 Métodos

Constructor

- `Cliente(string nome, string email, string senha)`: Inicializa uma nova instância da classe Cliente.

EditarPerfil

- `EditarPerfil(Cliente cliente, List<Cliente> clientes)`: Edita o perfil do cliente, permitindo a alteração de nome, e-mail ou senha.

ApagarConta

- `ApagarConta(Cliente cliente, List<Cliente> clientes)`: Apaga a conta do cliente, removendo seus dados.

AdicionarAoCarrinho

- `AdicionarAoCarrinho(List<Jogo> jogos)`: Adiciona jogos ao carrinho de compras do cliente.

AdicionarQuantidadeAoCarrinho

- `AdicionarAoCarrinho(Jogo jogo, int quantidade)`: Adiciona a quantidade especificada de um jogo ao carrinho de compras do cliente.

VisualizarCarrinho

- `VisualizarCarrinho(Cliente cliente, List<Jogo> jogos)`: Visualiza o carrinho de compras do cliente, permitindo a finalização da compra ou remoção de itens.

RealizarCompra

- `RealizarCompra(Cliente cliente, List<Jogo> jogos)`: Finaliza a compra, atualizando os estoques de jogos, registrando a compra no histórico do cliente e guarda as alterações nos Ficheiros.

RemoverDoCarrinho

- `RemoverDoCarrinho(List<Jogo> jogos)`: Remove um jogo específico do carrinho de compras do cliente.

GuardarHistoricoCompras

- `GuardarHistoricoCompras()`: Guarda o histórico de compras do cliente em um Ficheiro JSON.

CarregarHistoricoCompras

- `CarregarHistoricoCompras()`: Carrega o histórico de compras do cliente a partir de um Ficheiro JSON.

2.14 Compra.cs

2.14.1 Descrição

A classe `Compra` representa uma compra realizada por um cliente, incluindo a lista de jogos comprados e a data da compra.

2.14.2 Atributos

- `JogosComprados`: Obtém ou define a lista de jogos comprados nesta compra.
- `DataCompra`: Obtém ou define a data e hora da compra.

2.14.3 Construtores

- `Compra(List<Jogo> jogosComprados)`: Inicializa uma nova instância da classe Compra com a lista de jogos comprados e a data da compra.
- `Compra()`: Construtor padrão necessário para desserialização, inicializando a lista de jogos e a data da compra.

2.14.4 Propriedades

- `JogosComprados`: Obtém ou define a lista de jogos comprados nesta compra.
- `DataCompra`: Obtém ou define a data e hora da compra.

2.14.5 Métodos Públicos

- `RealizarCompra(Cliente cliente, List<Jogo> jogos)`: Finaliza a compra, atualizando a quantidade de jogos, registrando a compra no histórico do cliente e guardando as alterações nos arquivos.

2.15 GerirFicheiros.cs

2.15.1 Descrição

A classe `GerirFicheiros` contém métodos para carregar e Guardar informações relacionadas a clientes, administradores e jogos em Ficheiros JSON.

2.15.2 Métodos de Manipulação de Dados (JSON)

CarregarClientes

- `CarregarClientes()`: Carrega a lista de clientes a partir de um Ficheiro JSON.

GuardarClientes

- `GuardarClientes(List<Cliente> clientes)`: Guarda a lista de clientes em um Ficheiro JSON.

CarregarAdministradores

- `CarregarAdministradores()`: Carrega a lista de administradores a partir de um Ficheiro JSON.

GuardarAdministradores

- `GuardarAdministradores(List<Administrador> administradores)`: Guarda a lista de administradores em um Ficheiro JSON.

CarregarJogos

- `CarregarJogos()`: Carrega a lista de jogos a partir de um Ficheiro JSON.

GuardarJogos

- `GuardarJogos(List<Jogo> jogos)`: Guarda a lista de jogos em um Ficheiro JSON.

2.16 Programa.cs

2.16.1 Descrição

O ficheiro `Programa.cs` funciona como o ponto de entrada da aplicação e contém a função `Main`, que é o ponto de início da execução do programa.

2.16.2 Função `Main`

A função `Main` desempenha um papel importante na inicialização e execução do programa. Abaixo estão as principais responsabilidades da função `Main`:

- **Carregamento de Dados Iniciais:** O início da função `Main` é dedicado ao carregamento de dados iniciais essenciais para o funcionamento do programa. Isso inclui a carga de listas de clientes, administradores e jogos por meio da classe `GerirFicheiros`.
- **Exibição do Menu Principal:** Após o carregamento bem-sucedido dos dados, a função `Main` chama `ExibirMenu`, que é responsável por apresentar o menu principal da aplicação.
- **Autenticação de Utilizador:** Dentro da função `ExibirMenu`, quando a opção de login é escolhida, a função `Login` é invocada. Esta função autentica tanto clientes quanto administradores.
- **Processamento de Operações:** Com base na escolha do utilizador no menu principal, diversas operações são realizadas, como o registo de clientes.
- **Conservação de Dados Atualizados:** Após a conclusão de todas as operações, as listas atualizadas de clientes, administradores e jogos são guardadas por meio de chamadas aos métodos da classe `GerirFicheiros`.

2.16.3 Demonstração de Segmento de Código

A seguir apresenta-se um segmento simplificado da função denominada `Main`:

```
static void Main(string[] args)
{
    // Carregamento de dados iniciais
    List<Cliente> clientes = GerirFicheiros.CarregarClientes();
    List<Administrador> administradores = GerirFicheiros.
   CarregarAdministradores();
    List<Jogo> jogos = GerirFicheiros.CarregarJogos();

    // Exibi  o do menu principal e processamento de opera  es
    ExibirMenu(clientes, administradores, jogos);

    // Guarda os dados atualizados
    GerirFicheiros.GuardarClientes(clientes);
    GerirFicheiros.GuardarAdministradores(administradores);
    GerirFicheiros.GuardarJogos(jogos);

    Console.WriteLine("Operacoes concluidas com sucesso!");
}
```

Listing 1: Demonstração de Segmento de Código

Este segmento evidencia a estrutura fundamental da função `Main` e a sua importância vital no decorrer do ciclo de vida da aplicação.

2.17 Regras de Negócio

No desenvolvimento do sistema, foram estabelecidas diversas regras de negócio para garantir a integridade e segurança das operações realizadas pelos utilizadores. Estas regras são essenciais para manter a consistência dos dados e proteger informações sensíveis. Abaixo, são detalhadas algumas das principais regras de negócio implementadas no sistema:

2.17.1 Autenticação do Cliente para Exclusão de Conta

Antes de permitir a exclusão de uma conta de cliente, é necessária a autenticação do utilizador. Para garantir a segurança, o sistema solicita que o cliente introduza a sua senha para confirmar a exclusão. Esta medida visa evitar a exclusão acidental ou não autorizada de contas.

```
1 Console.Write("Introduza a sua senha para confirmar a exclusao da
     conta: ");
2 string senhaConfirmacao = Console.ReadLine();
```

Listing 2: Código de Autenticação para Exclusão de Conta

2.17.2 Validação de Email Único para Clientes

Para evitar informações duplicadas e garantir a unicidade dos clientes no sistema, é verificado se o novo email fornecido já está em uso por outro cliente. Caso isso aconteça, o sistema solicita que o cliente escolha outro email.

```
1 if (clientes.Any(c => c.Email == novoEmail && c != cliente))
2 {
3 Console.WriteLine("Este email ja esta em uso. Escolha outro.");
4 }
```

Listing 3: Validação de Email Único para Clientes

2.17.3 Verificação de Email Único para Administradores

Semelhante à validação para clientes, a verificação de email único é aplicada também aos administradores. O sistema impede o registo de mais de um administrador com o mesmo email, prevenindo assim potenciais conflitos.

```
if (administradores.Any(a => a.Email == email))
{
Console.WriteLine("Ja existe um administrador registado com este
    email. Tente novamente com um email diferente.");
}
```

Listing 4: Verificação de Email Único para Administradores

2.17.4 Verificação de Email Único para Clientes

Da mesma forma, a verificação de email único é aplicada aos clientes. O sistema impede o registo de mais de um cliente com o mesmo email, contribuindo para a organização e integridade da base de dados de clientes.

```
if (clientes.Any(c => c.Email == email))
{
Console.WriteLine("Ja existe um cliente registado com este email.
    Tente novamente com um email diferente.");
}
```

Listing 5: Verificação de Email Único para Clientes

3 Conclusão

O projeto e análise das classes essenciais para a loja de jogos proporcionaram uma compreensão valiosa da estruturação de um sistema orientado a objetos. Cada classe, desde a representação dos jogos até à geração de faturas, desempenha um papel crucial na construção de uma aplicação funcional. Além disso, a flexibilidade do design das classes permite a fácil expansão e adaptação a futuras exigências sobre o tema. A abordagem orientada a objetos adotada demonstra a sua eficácia na criação de um sistema modular e escalável, fundamentais para enfrentar desafios dinâmicos do mercado de jogos online.

Para complementar as classes desenvolvidas, o sistema adota uma arquitetura Models-Views-Controllers (MVC) para proporcionar uma organização eficiente, modularidade e uma clara separação de responsabilidades. As classes e componentes foram estruturados de forma a refletir as melhores práticas de desenvolvimento, promovendo a reutilização de código e facilitando a manutenção. As regras de negócio implementadas no sistema visam garantir a integridade dos dados, a segurança das operações e a consistência das informações. A autenticação do cliente para a exclusão de conta, a validação de emails únicos para clientes e a verificação de emails únicos para administradores são exemplos de medidas implementadas para assegurar uma experiência confiável e protegida para os utilizadores. O uso de DLLs separadas para cada camada do MVC contribui para a modularidade do sistema, permitindo o desenvolvimento independente de cada componente. A interligação entre essas DLLs, garante uma comunicação eficiente entre as camadas.

Por fim, as classes desenvolvidas formam uma base sólida para a implementação de uma loja de jogos virtual, refletindo boas práticas de programação orientada a objetos. A iteracao continua e a estrutura do sistema oferecem uma interação intuitiva para os utilizadores, garantindo a segurança das operações e facilitando a expansão futura do sistema com a adição de novas funcionalidades.

4 Anexos

- Código fonte disponível em:
  `https://github.com/rafaelramalhoipvc/PooProjeto`
- Material adicional sobre Camadas e MVC:
  `https://github.com/luferIPCA/LESI-POO-2023-2024/tree/master/Aula%2013%20-%20Camadas/MVC`