Instituto Politécnico do Cávado e do Ave

Licenciatura em Engenharia de Sistemas Informáticos

Programação Orientada a Objetos

Trabalho Prático

Students :
Francisco Arantes - 23504

Teacher :
Luís G. Ferreira

20 de dezembro de 2023

Resumo

A presente aplicação de gestão de uma loja online desenvolvida em C# representa um sistema de uma loja virtual.A estrutura da aplicação baseia-se em conceitos de programação orientada a objetos, utilizando uma arquitetura modular que facilita a manutenção e expansão do sistema. As principais entidades, como produtos, clientes e carrinhos de compras, são geridas de forma organizada e eficaz. Outro destaque é a integração de exceções personalizadas. Em suma, a aplicação de gestão de loja online em C# oferece uma solução abrangente e escalável para otimizar os processos comerciais.

Conteúdo

Lista de Siglas

UC Unidade Curricular

POO Programação Orientada a Objetos

NIF Número de Identificação Fiscal

1 Introdução

Este trabalho da *(UC)* de *(POO)* foca a análise de problemas reais simples e a aplicação do Paradigma Orientado a Objetos na implementação de possíveis soluções.

O tema escolhido parte da criação de uma loja *online* com classes que representam produtos, clientes e pedidos. Posteriormente serão implementados métodos especiais como por exemplo: adicionar certo produto a um carrinho, calcular o preço total e processar pedidos de forma a gerir a loja *online*.

2 Objetivos do Projeto

Esta cadeira tem como objetivo consolidar os conceitos fundamentais do Paradigma Orientado a Objetos. A abordagem prática é enfatizada através da análise de problemas reais.

É dada uma ênfase particular ao desenvolvimento de habilidades de programação em *C#*, uma linguagem de programação muito utilizada no contexto do Paradigma Orientado a Objetos.

Este projeto tem como objetivo final fornecer as competências necessárias para aplicar os princípios do Paradigma Orientado a Objetos de maneira eficaz na resolução de problemas práticos e no desenvolvimento de *software* de qualidade.

3 Primeira Fase

Durante a Fase 1 do projeto, concentramo-nos na estrutura inicial, identificação e implementação essencial das classes para o sistema em desenvolvimento. Foram identificadas as classes que desempenharão funções fundamentais dentro do paradigma orientado a objetos. Foi utilizado o *Visual Paradigm* para alguns rascunhos de como seria o potencial diagrama de classes do sistema. 3, posteriormente foi usado o *Visual Studio* para a construção do diagrama de classes. 5

A implementação inicial destas classes foi realizada com sucesso, escolheu-se a estrutura de dados *array*, alinhando-se com as exigências específicas do projeto.

O relatório atual documenta o progresso até à data. Destacam-se as classes, detalhes sobre a implementação inicial e as estrutura de dado adotada. Adicionalmente, são apresentados desafios enfrentados durante esta fase, juntamente com as soluções propostas para superá-las.

Um dos principais focos desta fase foi o cumprimento dos prazos estabelecidos. A gestão eficiente do tempo foi aplicada de forma a garantir que as atividades planeadas fossem concluídas dentro do prazo definido.

3.1 Arquitetura do Projeto

3.1.1 Biblioteca de classes *User*

A figura 1 mostra parte do diagrama de classes do sistema de gestão da loja *online*, esta parte representa as classes presentes na biblioteca de classes dos utilizadores.

A *class User* realçada pelo retângulo vermelho representa um utilizador do sistema, as classe *Admin* e *Costumer* destacadas por o retângulo a verde, herdam as propriedades *email* e *password* da classe *User*.

Para além das classes destacadas foram criadas outras 3 classes, *Users, Admins* e *Costumers*, responsáveis por gerir os respetivos utilizadores com o uso da estrutura de dados escolhida (*array*).

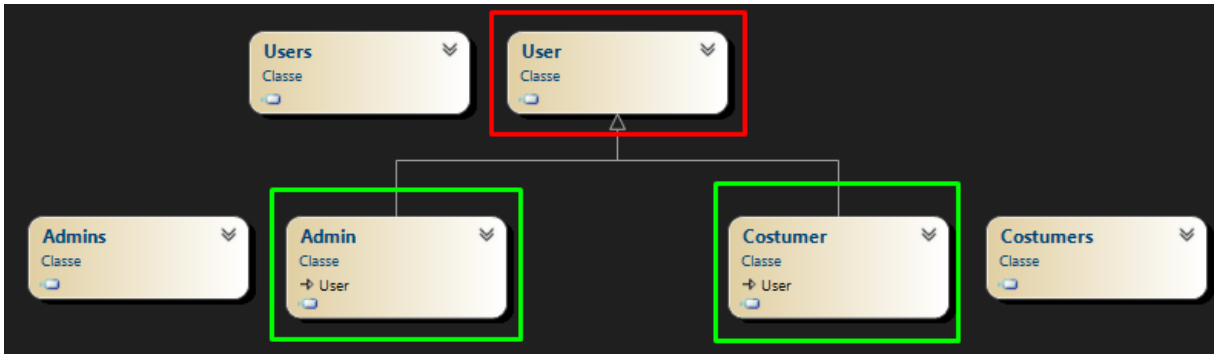

Figura 1: Relação entre *User* e *Costumer/Admin*

3.1.1.1 Classe *User* e *Users*

A figura 3, mostra as propriedades e métodos implementados para cada classe do tipo *User*.

Preferencialmente a começar pela classe *Users* realçada a vermelho, foi implementada a propriedade *AllUsers* que consiste em um *array* com todos os *users*, o limite de *users* é definido no inicio da class. De seguida, foram implementados métodos para adicionar e remover *users*. Foi também criado um construtor para inicializar uma nova instância da classe*Users* do tipo *array*.

Para a classe *User* realçada a verde, foi definido o *email* e a *password* e 2 construtores, que aceitam nenhum e 2 argumentos(*email* e *password*) respetivamente.

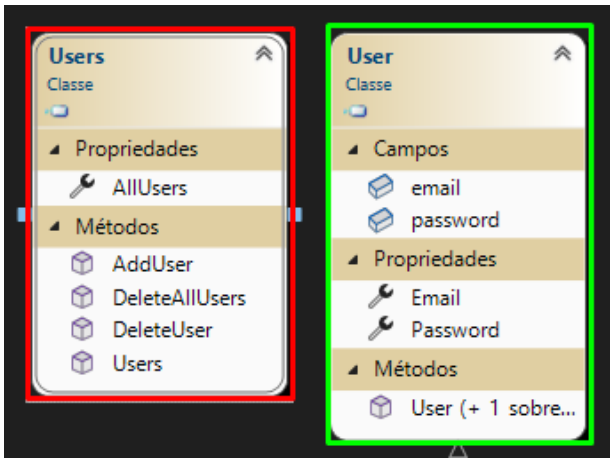

Figura 2: Propriedades e Métodos da classe *User* e *Users*

3.1.1.2 Classe *Costumer* e *Costumers*

A figura 2, mostra as propriedades e métodos implementados para cada classe do tipo *Costumer*.

Preferencialmente a começar pela classe *Costumer* realçada a verde, foram implementadas as propriedades, *Address*, NIF,*PhoneNumber*, *userName* e *zipCode* baseadas no registo do site da PcDiga 4.

Posteriormente foram implementados métodos para a classe *Costumer*, 2 construtores para inicializar membros estáticos e uma nova instância da class *Costumer* com valores padrão e mais outros 2 construtores que recebem parâmetros como, nome, email, password mais direcionado para um registo simples, e morada, código-zip, contacto telefónico e NIF mais direcionado para um registo ou personalização mais detalhada do perfil do cliente.

Para a classe *Costumers* realçada a vermelho, foi implementado novamente métodos para adicionar e remover clientes de uma estrutura de dados.

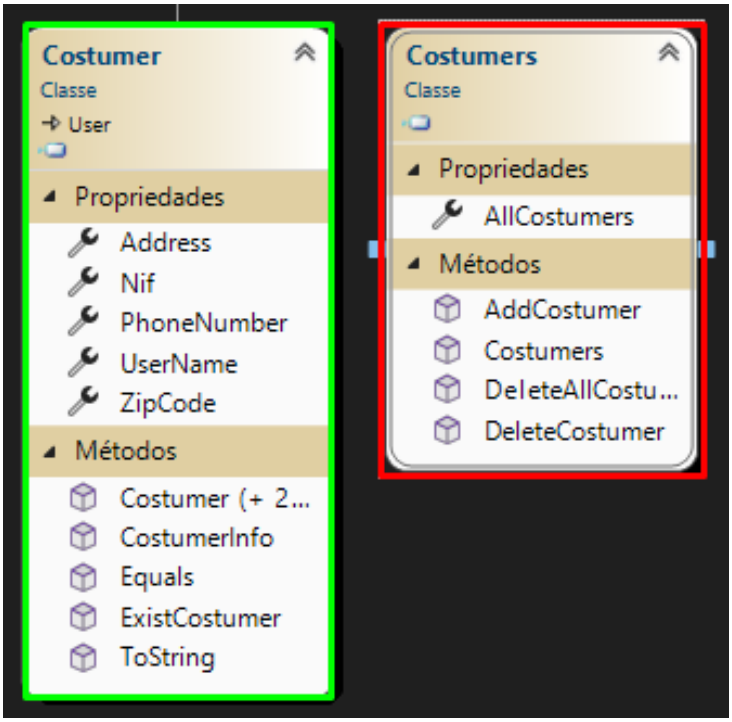

Figura 3: Propriedades e Métodos da classe *Costumer* e *Costumers*

3.1.1.3 Classe *Admin* e *Admins*

A figura 4, mostra as propriedades e métodos implementados para cada classe do tipo *Costumer*.

Preferencialmente a começar pela classe *Admin* realçada a verde, foram implementadas as propriedades, *AdminKey* responsável por atribuir um fator extra de segurança/privacidade ao *Admin* e *AdminUsername* que será o nome com que o administrador se irá destacar.

Posteriormente foram implementados métodos, 2 construtores para inicializar membros estáticos e uma nova instância da class *Admin* com valores padrão e mais outros 2 construtores que recebem parâmetros como, email, password e chave mais direcionado para um *login in*, e nome, email, password e chave mais direcionado para um registo do administrador.

Para a classe *Admins* realçada a vermelho, foi implementado novamente métodos para adicionar e remover administradores de uma estrutura de dados.

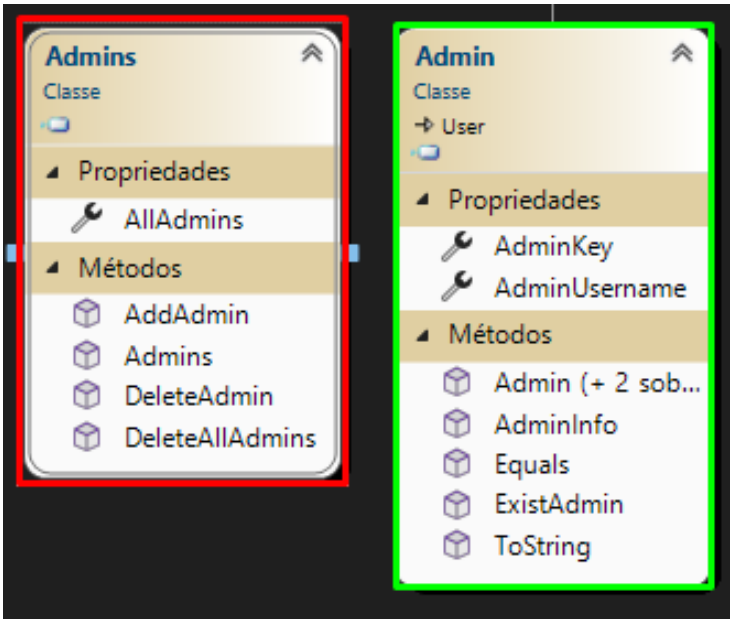

Figura 4: Propriedades e Métodos da classe *Admin* e *Admins*

3.1.2 Biblioteca de classes *Order*

A figura 5 mostra a segunda parte do diagrama de classes, mais especificamente as classes presentes na biblioteca de classes *Order*.

A *class Order* realçada pelo retângulo vermelho representa um pedido efetuado por um utilizador do sistema de uma certa quantidade de produtos, as classe *Payment* e *ShoppingCart* destacadas por o retângulo a verde, herdam as propriedades *orderId* e *totalAmmount* da classe *Order*.

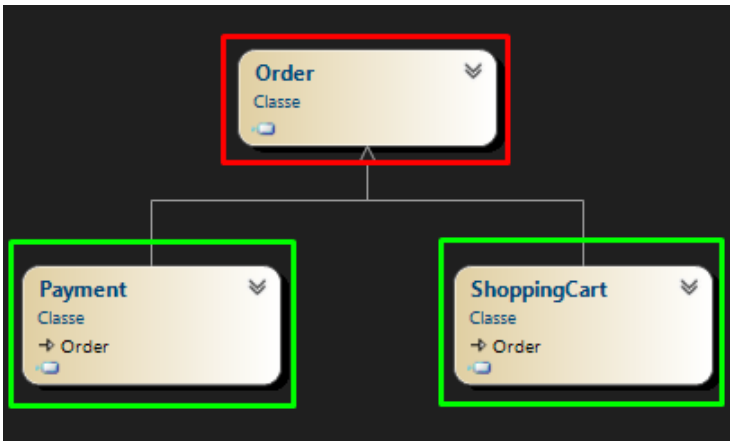

Figura 5: Relação entre *Order* e *Payment/ShoppingCart*

3.1.3 Biblioteca de classes *Product*

A *class Product* mostrada na figura 6, representa um produto selecionado por um utilizador/criado por um administrador do sistema.

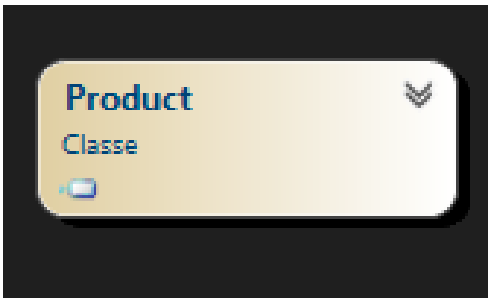

Figura 6: Classe implementada na biblioteca *Product*

3.2 Problemas encontrados

Para esta primeira fase, a maior dificuldade encontrada foi a estruturação e compreensão do negócio, ou seja, a capacidade de transpor as lojas *online* que estão bastante presentes no nosso dia a dia, em um projeto orientado a objetos. Pensar nas relações entre as classes que iria precisar de implementar, juntamente com as propriedades e métodos associados.

4 Segunda Fase

Durante a segunda fase do projeto, a estruturação e organização das classes foram alteradas relativamente á primeira fase do projeto.

Foi optada o estilo de arquitetura de N camadas *N-Tier*, esta arquitetura divide uma aplicação em camadas lógicas e camadas físicas.

A implementação final das classes foi conseguida, foram também implementados serviços.

Este capitulo esta estruturado da seguinte forma:

- O Primeiro sub capitulo consta a arquitetura do projeto, a estruturação e organização dos ficheiros;
- No segundo capítulo é apresentada os problemas encontrados, dificuldades e indecisões encontradas ao longo do desenvolvimento do trabalho;

4.1 Arquitetura do Projeto

A arquitetura do projeto escolhida foi a *N-Tier*, foram implementadas 5 bibliotecas de classes e 1 Consola como é possivel observar na figura 7.

As camadas são uma forma de separar as responsabilidades e gerir as dependências. Cada camada tem uma responsabilidade específica. Uma camada superior pode utilizar serviços de uma camada inferior, mas não o contrário.

A camada *Aplication* é a aplicação consola *Console App*, responsável por apresentar para o utilizador num futura implementação por exemplo um *Windows Forms* a simular a Loja Online.

As seguintes camadas serão explicadas com mais pormenor nos próximos capítulos.

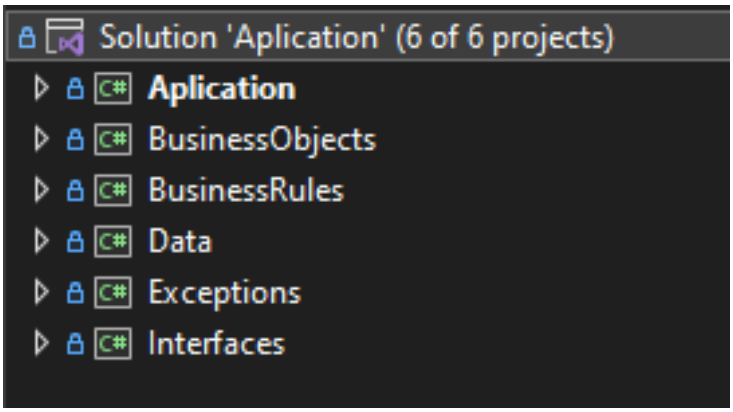

Figura 7: Camadas criadas

4.1.1 Biblioteca de Classes *Business Objects*

Esta camada, apresentada na figura 8, está responsável por armazenar todos os modelos dos objetos que serão necessários para a gestão da loja *online*.

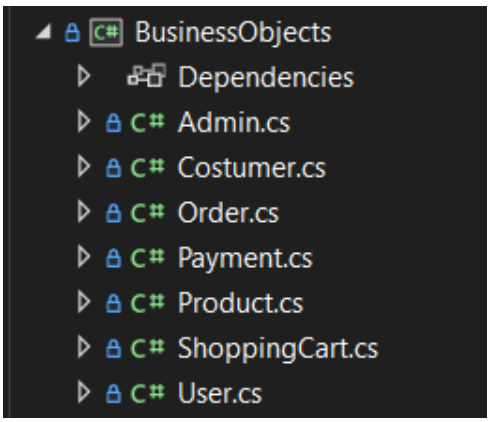

Figura 8: Classes implementadas

As entidades *Admin* e *Costumer* herdam de *User*, a aplicação poderá então ter 2 tipos de utilizadores *Administrados e Cliente*.

Essencialmente as classes *Admin* e *Costumer* herdaram 2 importantes propriedades e obrigatórias tanto para um Cliente como para um Administrador da aplicação, realçado na figura 9.

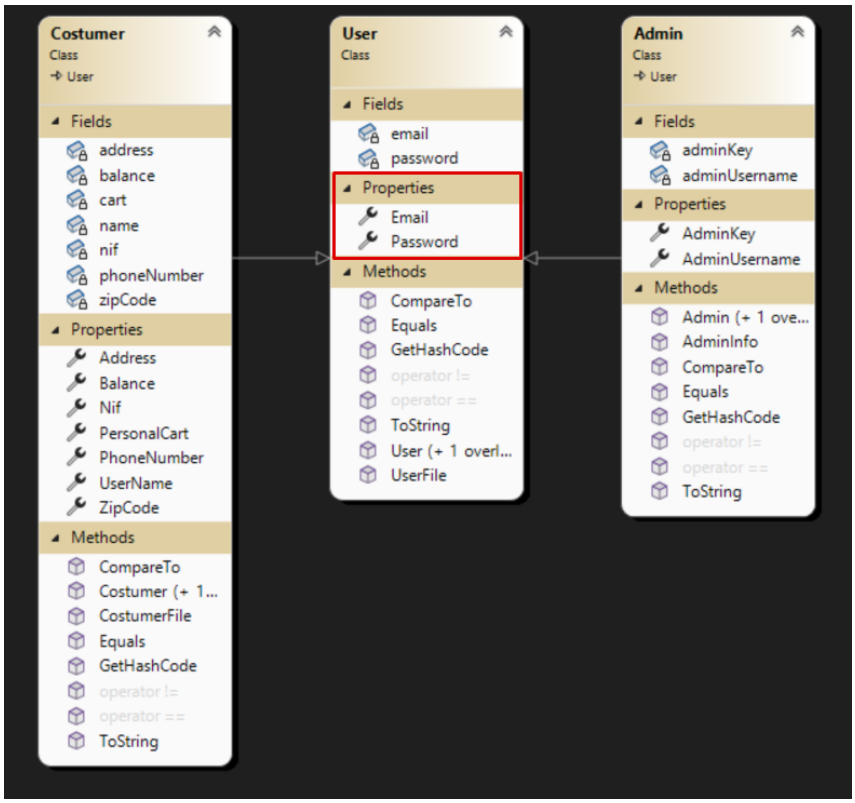

Figura 9: Propriedades e métodos herdados da classe User

Como é possível observar na figura 10, as classes *Product*, *ShoppingCart* e *Payment* deixaram de herdar da classe *Order*, mesmo assim, estão todas relacionadas com o processo de seleção e processamento da encomenda, há uma série de processos que acontecem quando fazemos as nossas compras online, selecionamos os produtos disponíveis, selecionamos os produtos para um carrinho de compras virtual e de seguida procedemos á encomenda, processo este que será tratado posteriormente na class *Orders*.

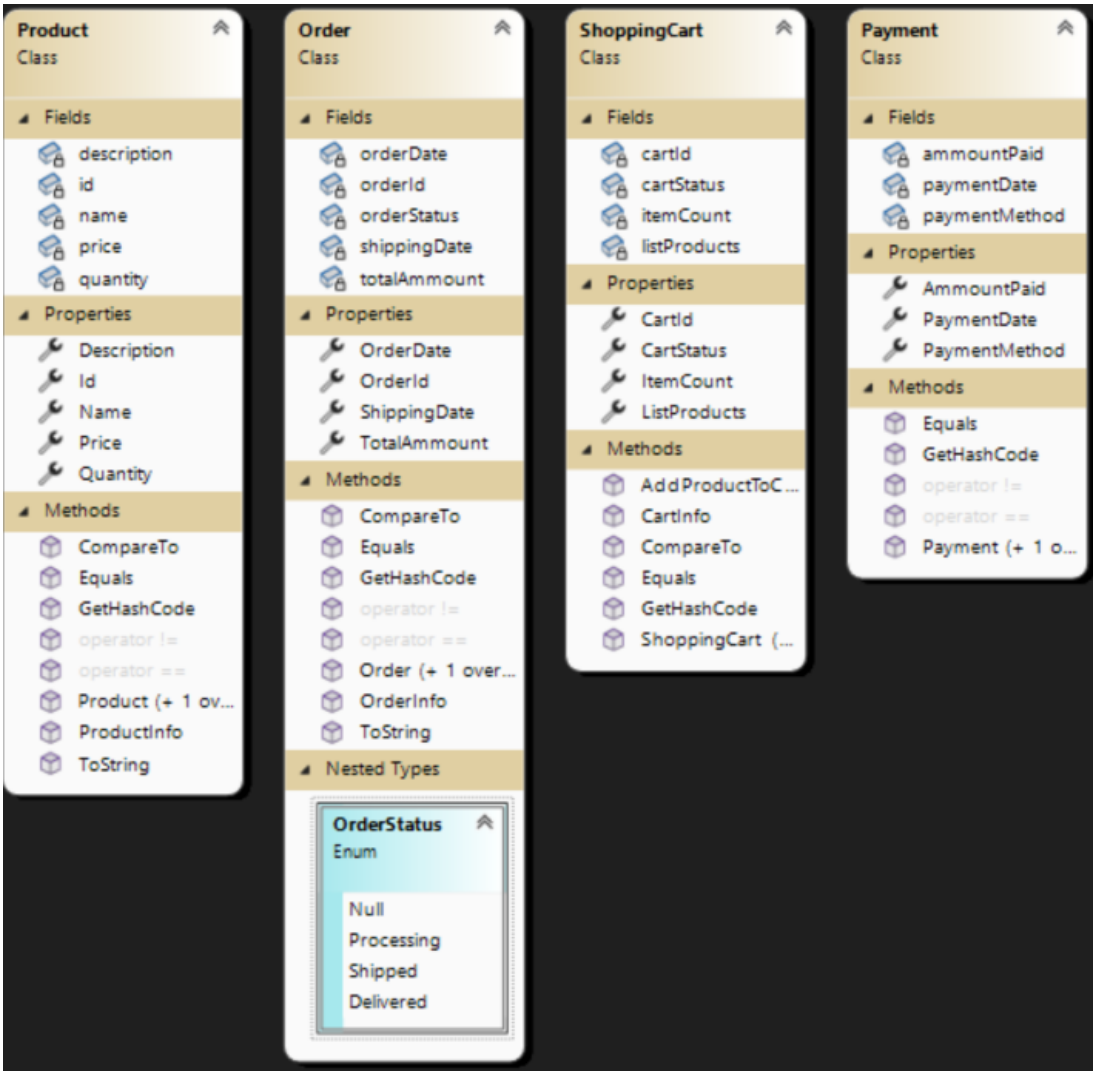

Figura 10: Propriedades e métodos herdados das classes mencionadas

4.1.2 Biblioteca de Classes *Data*

A biblioteca de classes *Data* tem um papel muito importante na gestão dos modelos (*Business Objects*) nesta biblioteca de classes converteu-se o antigo método de armazenamento dos objetos (Arreios) para Listas, elas fornecem uma alternativa dinâmica, redimensionável e fortemente tipada para arrays.

Na figura 11 podemos observar o padrão que seguiu a implementação destas classes, realçado pelo retângulo vermelho podemos observar as listas criadas para gerir os dados, no retângulo verde podemos observar alguns dos métodos *CRUD* implementados, por fim mas menos importante, realçado pelo retângulo azul estão os método responsável pela escrita e leitura de ficheiros.

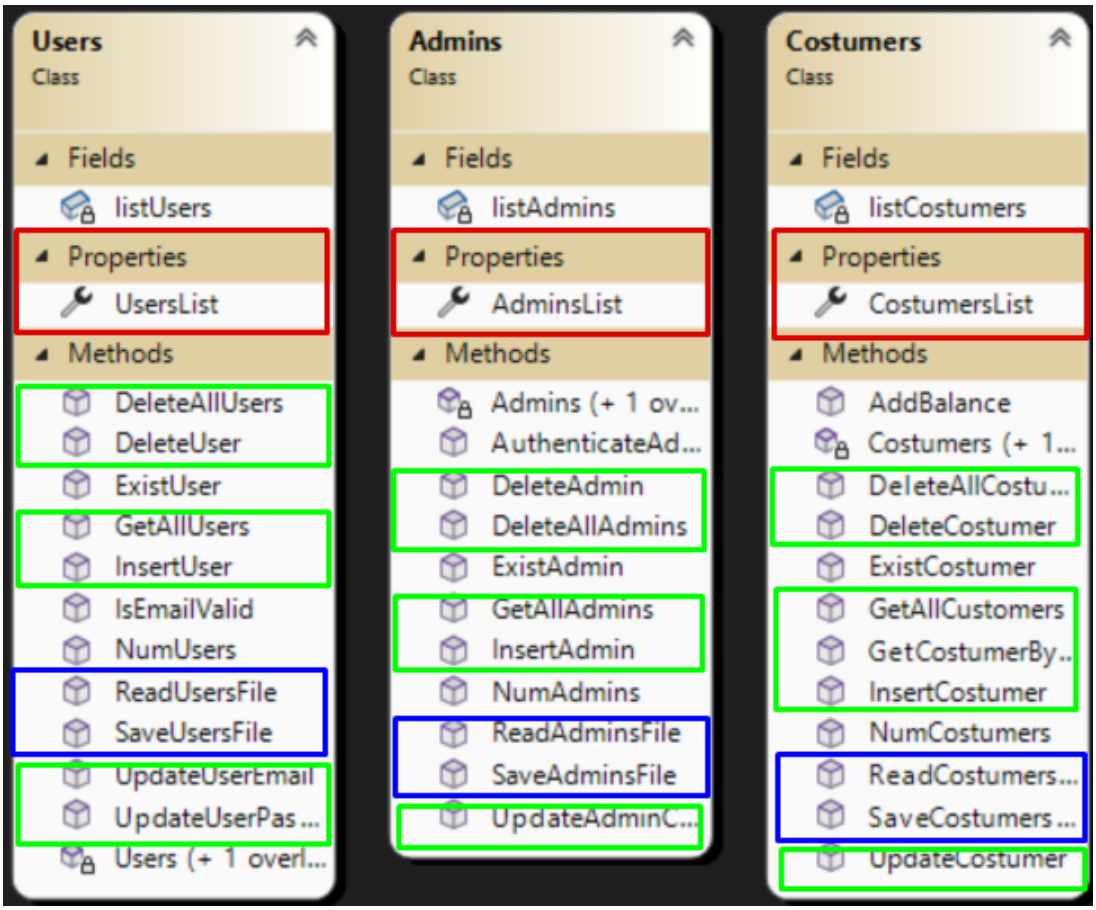

Figura 11: Propriedades e métodos herdados das classes mencionadas

4.1.3 Biblioteca de Classes *Business Rules*

Após termos os métodos bem definidos e as respetivas classes responsáveis por gerir os modelos implementadas, podemos começar a pensar na implementação nas regras de negócio previamente definidas.

Uma loja online pode conter inúmeras regras de negócio, para este trabalho implementei algumas que achei necessário e uma mais valia, tais como, verificar a inserção de um utilizador, verificar a inserção de um cliente, verificar a adição de um produto a um carrinho, verificar a correta adição de saldo na conta do utilizador, verificar a correta leitura e escrita dos ficheiros entre outras.

O retângulo vermelho e azul da figura 12 mostra o nome das regras de negócio mencionadas neste capítulo.

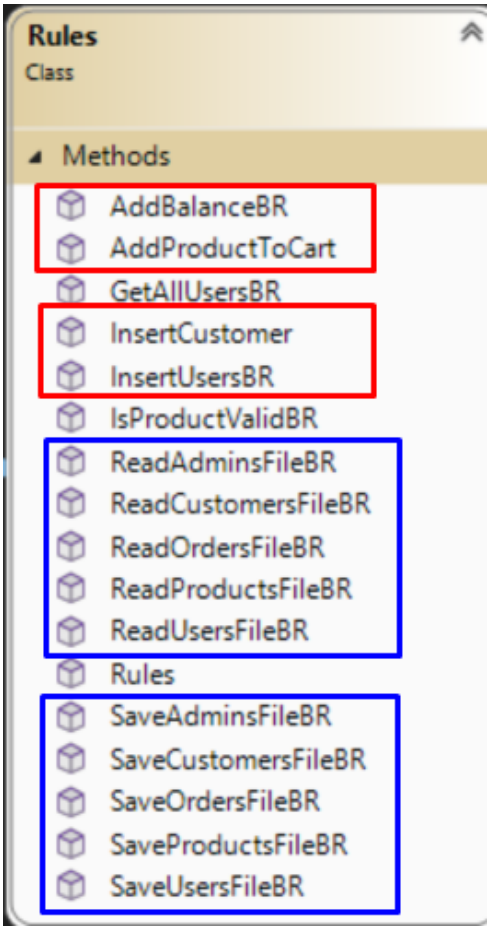

Figura 12: Regras de negócio implementadas

4.1.4 Biblioteca de Classes *Exceptions*

Quando usamos um tipo de exceção personalizado, podemos escrever uma mensagem especial para lidar com essa exceção. Também podemos monitorizar na aplicação algum tipo específico de exceção e notificar o utilizador quando este for detetado.

Para este projeto foram implementadas algumas exceções consideradas importantes, exceções de leitura e escrita de ficheiros realçadas pelo retângulo vermelho, utilizadores já existentes e não encontrados realçados pelo retângulo verde. como é possível observar na figura 13.

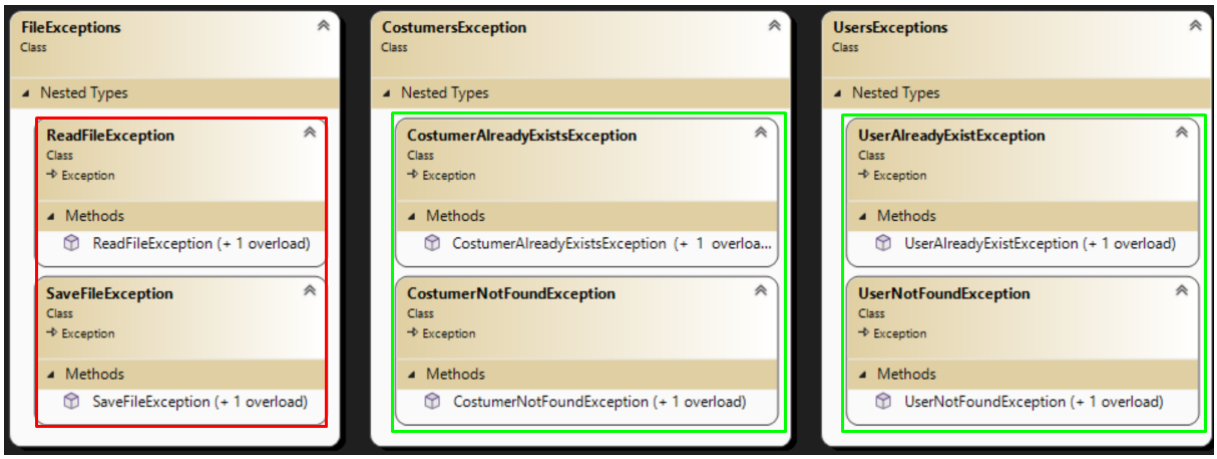

Figura 13: Exceções para ficheiros e utilizadores

4.1.5 Biblioteca de Classes *Interfaces*

Uma das principais vantagens do uso de interfaces em C-sharp é fornecer alternativa para implementar heranças múltiplas, nesta biblioteca foram implementadas interfaces para cada classe implementada na biblioteca dos Dados (*Data*).

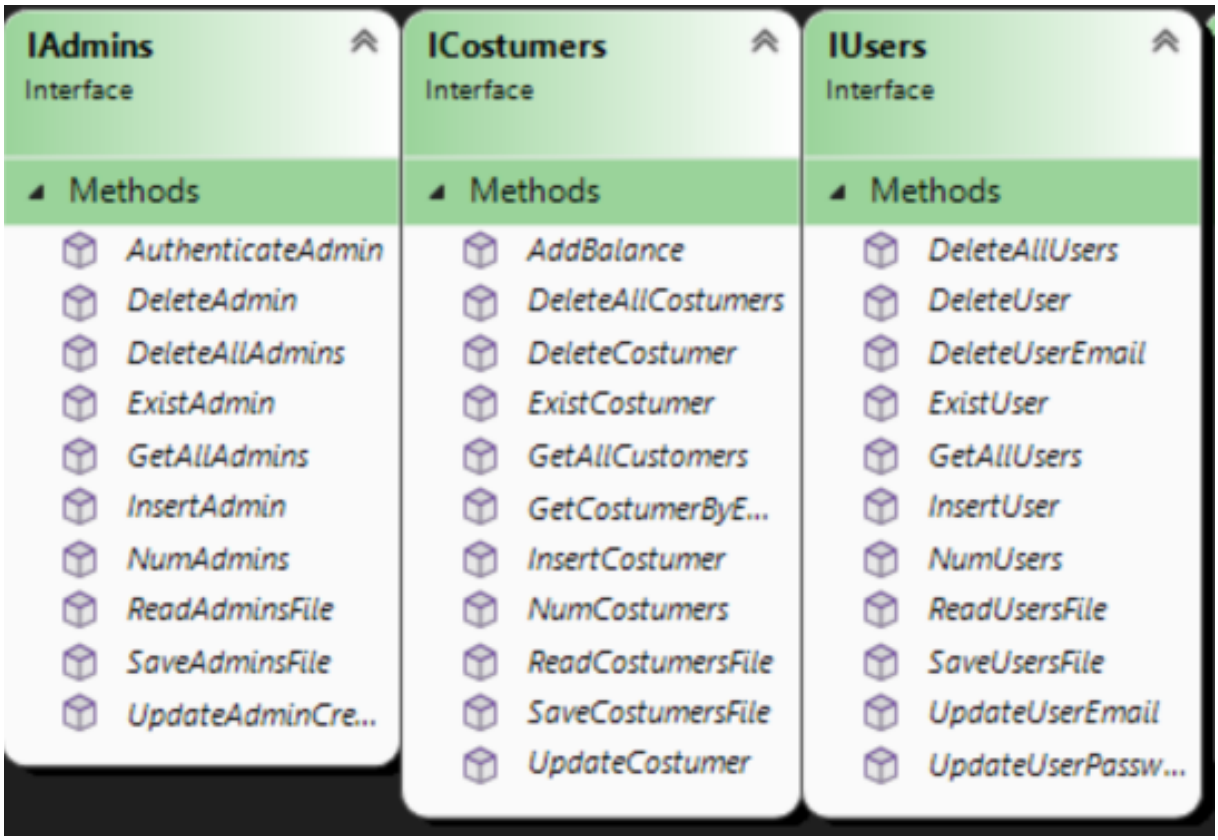

Figura 14: Algumas das interfaces implementadas

4.2 Problemas encontradas

Uma das maiores dificuldades durante o desenvolvimento deste trabalho foi perceber qual a melhor forma para relacionar as classes *Costumer, Order, Product, ShoppingCart* e *Payment*, eventualmente percebi que o Cliente (*Costumer*) poderá ter como propriedade uma instância de um carrinho (*ShoppingCart*) e uma instância de uma encomenda (*Order*).

Após o término da primeira fase do trabalho, percebi que seria uma mais valia usar uma arquitetura para estruturar melhor o projeto, esta transição originou-me problemas no compilador do *Visual Studio* á medida que eram criadas novas Bibliotecas de Classes.

5 Resultados

- Implementação dos objetos de negócio;
- Implementação de regras de negócio;
- Implementação de classes responsáveis por gerir os objetos de negócio;
- Implementação de exceções;
- Implementação de interfaces;
- Implementação de uma aplicação demonstradora dos serviços implementados;
- Relatório concluído dentro do prazo limite;

6 Conclusão

Em suma, na primeira fase explorou-se e analisou-se o tema de uma Loja *Online*, as classes foram identificadas e implementadas essencialmente e o prazo estipulado foi cumprido.

Houve a tentativa de implementar mais métodos para as classes *Product, Payment, ShoppingCart* e *Shipping* de modo a, enriquecer o trabalho e a consolidação dos conceitos básicos de POO.

Apesar disto, foi demonstrado como pretendido os conceitos básicos de POO.

Na segunda fase explorou-se a arquitetura N-tier, de forma a, dividir a aplicação em camadas lógicas e camadas físicas.

Houve a tentativa de implementar mais métodos de forma a enriquecer as regras de negócio e testes de forma a testar as implementações mais criticas.

7 Bibliografia

1. GitHub do Professor Luís G. Ferreira: LESI-POO-2023-2024;
2. GitHub pessoal com conceitos básicos de C#: CS-Programming-Language;
3. Visual Paradigm: Instalação
4. PcDiga: Website
5. Visual Studio: Instalação
6. Conceitos N-Tier: Microsft N-Tier guide
7. Nullable Warnings: Microsoft nullable warnings guide
8. Sebenta de C sharp do professor;
9. Sebenta de C sharp de Joe Mayo;