Left click – gestor de comércio eletrónico

Programação Orientada a Objetos

Trabalho Prático

Ano Letivo: 2024/2025

Licenciatura em Engenharia de Sistemas Informáticos

Regime Pós-laboral

Escola Superior de Tecnologia

Instituto Politécnico do Cávado e do Ave

Docente: Prof. Ernesto Casanova Discente: Miguel Silva Areal - Nº 29559

Índice

[Índice de figuras 1](#_Toc182314111)

[Introdução 2](#_Toc182314112)

[Análise do Problema 3](#_Toc182314113)

[Funcionalidades Principais 3](#_Toc182314114)

[Requisitos Técnicos 4](#_Toc182314115)

[Estrutura de Classes 5](#_Toc182314116)

[Armazenamento de Dados 5](#_Toc182314117)

[Descrição de Interfaces 5](#_Toc182314118)

[Conclusão 6](#_Toc182314119)

[Bibliografia 7](#_Toc182314120)

# Índice de figuras

# Introdução

O presente projeto faz parte da Unidade Curricular (UC) de Programação Orientada a Objetos (POO), lecionada no 1º semestre do 2º ano do curso de Engenharia de Sistemas

Informáticos, do Instituto Politécnico do Cávado e do Ave.

O trabalho prático divide-se em 2 partes, referentes a diferentes fases de

desenvolvimento, de forma a testar os conhecimentos dos alunos, tendo como objetivo

principal a aplicação prática do paradigma orientado a objetos na implementação de

possíveis soluções para problemas reais.

• Fase 1: Estrutura de Classes identificadas. Implementação essencial das classes.

Estruturas de dados a utilizar. Relatório do trabalho desenvolvido até à data (15 de

novembro de 2024).

• Fase 2: Implementação final das classes e serviços. Aplicação demonstradora dos

serviços implementados. Relatório final do trabalho realizado.

A linguagem de programação utilizada durante todo este projeto será a linguagem

lecionada, C#.

Para este projeto, denominado de ‘**Left-Click’**, o tema escolhido foi: Comércio

eletrónico (sistema que permita a gestão de uma loja online).

Como tal, o objetivo da solução é criar um sistema que seja capaz de gerir diversos

parâmetros que englobam uma loja online.

O projeto encontra-se num repositório público, na minha página no Github

(https://github.com/MiguelAreal/POO\_TP\_29559).

# Análise do Problema

O projeto tem como objetivo desenvolver um sistema de comércio eletrónico para a gestão de uma loja online que comercializa produtos de diferentes categorias e marcas.

O sistema deve permitir o registo de produtos, a gestão de inventário, a organização de categorias e marcas, o controlo de vendas e o registo de clientes.

De modo a atender às necessidades de uma loja online, o sistema deve ser intuitivo, com validações essenciais para a operação eficiente.

### Funcionalidades Principais

A partir do levantamento de requisitos, foram identificadas as funcionalidades fundamentais para o sistema:

* **Gestão de Produtos:** O sistema deve ser capaz de armazenar informações detalhadas sobre cada produto, incluindo nome, descrição, preço, stock, categoria e marca. A gestão de produtos deve permitir o registo, a edição e a eliminação de produtos.
* **Organização de Categorias e Marcas**: O sistema deve permitir que produtos sejam associados a categorias e marcas. Deve permitir que o administrador da loja crie e edite categorias (como "Eletrónicos", "Eletrodomésticos", etc.) e marcas (como "*Samsung*", "*Sony*"), facilitando assim a consulta de produtos no sistema por meio de filtros.
* **Gestão de Clientes**: O sistema deve armazenar informações sobre os clientes, incluindo nome, endereço, contacto telefónico, endereço de email, NIF e histórico de compras associado. Esse registo é essencial para fins de acompanhamento de clientes e análise de dados de vendas.
* **Processamento de Vendas**: É necessária a funcionalidade para registar as vendas realizadas na loja, incluindo a data, o cliente (se associado), os produtos adquiridos e o valor total.
* **Processamento de Compras**: Pela parte do próprio cliente é necessária a funcionalidade para registar as compras realizadas pelo mesmo, semelhante às vendas.
* **Gestão de Stocks:** À medida que os produtos são vendidos, o stock deve ser automaticamente atualizado. O sistema também deve alertar sobre produtos que estão com baixo stock, permitindo um melhor controle de inventário.
* **Campanhas de descontos:** Para atrair possíveis clientes e aumentar as vendas, o sistema deve permitir a criação de campanhas e promoções temporárias, com descontos aplicáveis a determinadas categorias.
* **Interface:** Além dos requisitos funcionais, a aplicação necessita de ter um visual intuitivo, esclarecedor e apelativo, de modo que seja confortável ao utilizador.

# Requisitos Técnicos

***Left-Click*** será desenvolvido utilizando a plataforma *Windows Forms*, que oferece uma interface visual intuitiva para aplicações em C#. O projeto utilizará o .NET 8, a versão mais recente da *framework* que mantém compatibilidade com Windows Forms.

O sistema segue o modelo arquitetural ***MVC*** (Model-View-Controller), que organiza o código em três camadas principais: Model (dados e lógica), View (interface do utilizador) e Controller (controlador de fluxo). Esta estrutura permite uma separação de responsabilidades, facilitando a manutenção do código e possibilitando a expansão do sistema. A arquitetura MVC também contribui para uma maior organização e flexibilidade, permitindo a adição de novas funcionalidades sem comprometer a estrutura existente.

Para a persistência de dados, será utilizada a gravação em ficheiros *JSON*. Esta abordagem proporciona um armazenamento simplificado e facilita a manipulação de dados locais no projeto, enquanto mantém os dados legíveis e facilmente editáveis.

# Estrutura do Projeto

## Estrutura de Classes

As classes, ou modelos, estão bem organizados em termos de conteúdo. As classes principais são:

### Produto

Representa um objeto ao qual se pode manipular stock, e atribuir marca e categoria.

* ID identificador do produto
* Nome do produto
* ID de categoria associada
* ID de marca associada
* Preço
* Quantidade em Stock
* Data de Adição

### Marca

Representa um atributo pertencente a um produto.

* ID identificador de marca
* Nome de marca
* Descrição
* País de Origem
* Método para verificar se esta pode ser eliminada. (Não pode estar ativamente inserida num produto).

### Categoria

Representa um atributo pertencente a um produto.

* ID identificador de categoria
* Nome de categoria
* Descrição
* Data de criação
* Método para verificar se esta pode ser eliminada (Não pode estar ativamente inserida num produto).

### VendaCompra

Representa uma venda ou uma compra, constituída por vários produtos.

* ID identificador de venda
* ID de cliente (facultativo)
* NIF (facultativo)
* Lista do tipo *VendaItem*, o Acoplamento de vários itens a serem vendidos
* Quantidade, dada pela soma de quantidades de todos os itens na lista *VendaItem*
* Preço Total Bruto, dado pelo produto do preço unitário e número de unidades de cada produto vendido,
* Preço Total Líquido, contemplado de descontos.
* Data de venda
* Data de fim de garantia
* Método de pagamento

### ItemVenda

Representa cada item a ser vendido dentro de uma venda. A razão pela qual integra tanto o ID do atributo e Nome do atributo (Produto, Marca e Categoria) deve-se ao facto de, se posteriormente o atributo for eliminado ou alterado, ainda ser possível ver no histórico de vendas o valor que foi utilizado no momento da venda.

* ID identificador de produto o Tem como finalidade depois manipular o stock do produto.
* Nome de Produto o Incluído para ser possível visualizar o produto vendido num histórico, mesmo que o produto tenha sido eliminado ou alterado posteriormente à venda.
* Preço Unitário
* ID identificador de categoria o Incluído para na altura da venda, ser possível aplicar o desconto da campanha.
* Nome de categoria o Incluído para ser possível visualizar o a categoria a que pertencia o produto vendido num histórico, mesmo que a categoria tenha sido eliminada ou alterada posteriormente à venda.
* Nome de marca
* Número Unidades vendidas o Número de unidades vendidas de cada produto.
* Percentagem de desconto aplicável

### Utilizador

Representa um utilizador no sistema, com características que tornam a sua utilização dinâmica, composto por:

* ID identificador de utilizador
* Nome
* Contacto
* Morada
* NIF
* Data de nascimento (facultativo)
* Data de adição no sistema
* Palavra-passe
* IsAdmin, que representa se o utilizador é um funcionário, ou um cliente.

### Campanha

Representa atributo de desconto de preço aplicável a categorias específicas.

* ID identificador de campanha
* Nome de campanha
* Percentagem de desconto aplicável
* Data de início de campanha
* Data de fim de campanha
* ID de categoria a que se aplica a campanha

Em consequência, existem outras classes modelo que servem como transformação da classe inicial, nomeadamente para exibição nas views:

* ProdutoViewModel
* VendaCompraViewModel
* CampanhaViewModel

Estas classes, servem apenas como base de exibição de dados, devido ao facto de nas próprias classes, a referência a outros objetos ser feita através de ID, similar a uma base de dados. Foi organizado desta forma para que não haja repetição de dados nos respetivos ficheiros JSON.

Desta forma, ao carregar os dados para a *view*, é feita a associação de dados, e mostrados os dados de forma correta, por exemplo, a assimilação de ID de Marca a Nome de Marca, para cada produto.

# Estrutura de Controladores

Neste projeto, os controladores desempenham um papel essencial na aplicação da arquitetura MVC funcionando como intermediários entre o modelo e a interface gráfica (View). Estes controladores são responsáveis por carregar, manipular e exibir dados, além de coordenar as ações do utilizador com as funcionalidades do sistema.

A implementação utiliza um controlador base genérico, o *BaseController* que encapsula comportamentos comuns, como o carregamento de dados, filtros, e ações de CRUD (Create, Read, Update, Delete).

Este controlador é estendido por classes específicas para cada entidade, como *ProdutoController*, *ClienteController*, e *VendaController*, que adicionam funcionalidades particulares conforme as regras de negócio.

### Controlador Base (BaseController<T, TView>)

O BaseController centraliza funcionalidades genéricas e reutilizáveis, como:

* Carregamento de Dados: Carrega itens de um ficheiro JSON e sincroniza com a view.
* Filtro: Realiza buscas dinâmicas usando LINQ, permitindo filtrar itens por propriedades específicas.
* CRUD Básico: Implementa a lógica genérica para adicionar e remover itens do repositório.
* Mapeamento de Propriedades: Facilita a tradução entre nomes depropriedades exibidos (em tabelas, por exemplo) e os atributos reais das classes.

O controlador utiliza o repositório genérico (BaseRepo<T>) para persistir dados e garantir consistência entre as operações realizadas na memória e o armazenamento em disco.

### Controladores Específicos

Cada entidade principal possui um controlador que herda de BaseController, adicionando lógica específica, como verificações de regras de negócio ou processamento de dados para exibição:

#### ProdutoController

Este controlador é responsável por gerir produtos. Ele traduz IDs de marcas e categorias em nomes para exibição na interface gráfica, utilizando ViewModels. Também permite a atualização automática de stock com base em vendas ou edições.

#### CampanhaController

Gere campanhas de desconto. Realiza traduções de IDs de categorias associadas a campanhas e valida a consistência temporal dos dados, como a sobreposição de datas de campanhas.

#### CategoriaController

Além de funcionalidades básicas, verifica se uma categoria pode ser removida, consultando os produtos associados. Caso a remoção não seja possível, exibe um aviso ao utilizador.

*MarcaController*

Semelhante ao CategoriaController, este valida se uma marca pode ser excluída, verificando sua associação com produtos existentes.

#### UtilizadorController

Gere os dados de utilizadores, incluindo a classificação como administrador ou cliente. Essa distinção é importante para funcionalidades como permissão de visualização de dados.

#### VendaCompraController

Este controlador é o mais complexo, pois gere vendas/compras compostas por múltiplos itens. Ele agrega dados de produtos e clientes para exibição, calcula totais e períodos de garantia, e traduz métodos de pagamento para texto legível.

#### Interações com as Views

Os controladores interagem diretamente com a interface gráfica por meio do método abstrato ExibeItensNaView, que é implementado em cada controlador específico para adaptar os dados ao formato necessário para exibição. Isso permite que os dados, armazenados de forma otimizada no repositório (com IDs em vez de nomes), sejam apresentados de forma legível ao usuário, com todas as associações resolvidas.

* Exemplo: O ProdutoController converte os IDs de marca e categoria em nomes, enquanto exibe informações como preço, quantidade em stock e data de adição no DataGridView.

# Armazenamento de Dados

Para o armazenamento de dados, o projeto utiliza ficheiros JSON. Esta escolha permite uma forma simplificada de persistir dados e facilita a visualização e edição dos mesmos em formato de texto.

O uso de JSON oferece uma vantagem inicial em termos de simplicidade, embora tenha limitações em relação a sistemas de bases de dados mais completos. Estratégia de Armazenamento:

* Cada entidade (como Produto, Cliente, Venda, etc.) é guardada num ficheiro JSON separado, armazenando uma coleção de objetos dessa entidade específica.
* Durante o carregamento dos dados, o sistema lê esses ficheiros JSON e converte-os para objetos do modelo, que são manipulados pelo sistema.

# Descrição de Interfaces Visuais

O projeto utiliza Windows Forms para a construção das interfaces de utilizador, ou Views. Proporciona uma experiência visual para o utilizador e facilita o uso do sistema. Cada formulário corresponde a uma funcionalidade principal, e a interface é organizada para garantir uma navegação intuitiva e clara entre as diferentes operações.

## Interfaces Visuais:

### MainForm:

Form principal do sistema, que fornece acesso às funcionalidades principais, como gestão de produtos, clientes, vendas e campanhas, através de um menu em estilo de “separadores”.

### ChildForm:

Form filho, secundário do sistema.

Criado como template a outros forms, para evitar criar diferentes formulários com quase a mesma funcionalidade e aspeto, melhorando assim a manutenção do projeto.

Projetado com foco na simplicidade, usando componentes como DataGridView para exibir listas de produtos, utilizadores e vendas de forma organizada, e caixas de seleção para facilitar a busca e filtragem. Permite consulta de itens consoante a coluna da DataGridView escolhida.

### MainForm:

Form principal do sistema, que fornece acesso às funcionalidades principais, como gestão de produtos, clientes, vendas e campanhas, através de um menu em estilo de “separadores”.

Bibliotecas Utilizadas

Além da biblioteca *default* do .NET, que inclui classes essenciais para manipulação de dados, JSON e interface gráfica, o projeto utiliza bibliotecas adicionais conforme necessário.

Aqui está uma lista de bibliotecas aplicadas:

* Newtonsoft.Json: Para manipulação de JSON, usada para leitura e gravação de dados em ficheiros JSON.
* System.Windows.Forms: Para criação das interfaces gráficas no Windows Forms, fornecendo componentes como Button, TextBox, DataGridView, entre outros.
* System.Linq: Para facilitar consultas e manipulação de coleções, muito útil em operações com filtros e ordenação de dados na camada de controlador.
* MetroFramework: Controlos diferentes do estilo “Metro UI”, da Microsoft, para aplicações Windows Forms. Permite alterar o estilo e dar um aspeto mais moderno e personalizável a aplicações.
* ValidationLibrary: Biblioteca criada por mim, que tem como finalidade o auxílio na validação de introdução de dados em formulários em Windows Forms.

Estas bibliotecas contribuem para tornar o desenvolvimento mais eficiente, simplificando a manipulação de dados e desenvolvimento de interfaces.

# Bibliografia