Relatório do trabalho da disciplina de Programação Orientada a Objetos

Shop Sharp

André Freitas - a25975 Nuno Castro – a4944

Engenharia Sistemas Informáticos

Novembro de 2023



Afirmo por minha honra que não recebi qualquer apoio não autorizado na realização deste trabalho prático. Afirmo igualmente que não copiei qualquer material de livro, artigo, documento web ou de qualquer outra fonte exceto onde a origem estiver expressamente citada.

André Freitas - a25975 Nuno Castro - a4944

Índice

[Introdução 3](#_Toc155107808)

[Implementação 4](#_Toc155107809)

[Descrição do problema 4](#_Toc155107810)

[Solução 4](#_Toc155107811)

[Implementação C# 4](#_Toc155107812)

[DAL 5](#_Toc155107813)

[BLL 8](#_Toc155107814)

[Carrinho BLL 10](#_Toc155107815)

[Venda BLL 11](#_Toc155107816)

[Classes 12](#_Toc155107817)

[Clientes 12](#_Toc155107818)

[Produtos 13](#_Toc155107819)

[Venda 13](#_Toc155107820)

[Marca 13](#_Toc155107821)

[Garantia 14](#_Toc155107822)

[Categoria 14](#_Toc155107823)

[Carrinho Compras 15](#_Toc155107824)

[Campanha 16](#_Toc155107825)

[Interface 16](#_Toc155107826)

[Menus 17](#_Toc155107827)

[Menu Inicial 17](#_Toc155107828)

[Menu Gestor 17](#_Toc155107829)

[Menu 18](#_Toc155107830)

[Métodos Aplicados no Interface 18](#_Toc155107831)

[Novo Registo 18](#_Toc155107832)

[Compra Produtos 20](#_Toc155107833)

[Editar Cliente 21](#_Toc155107834)

[Métodos de Gestão 22](#_Toc155107835)

[Login 23](#_Toc155107836)

Lista de figuras

[Figura 1 - Visão Geral Produto.DAL 5](#_Toc155103625)

[Figura 2 - Visão Geral dos ficheiros DAL 8](#_Toc155103626)

[Figura 3 - Classe Cliente e construtor 12](#_Toc155103627)

[Figura 4 – Classes Herança de Cliente 12](#_Toc155103628)

[Figura 5 - Classe Produto 13](#_Toc155103629)

[Figura 6 - Classe Venda 13](#_Toc155103630)

[Figura 7 - Classe Marca 14](#_Toc155103631)

[Figura 8 - Classe Garantia 14](#_Toc155103632)

[Figura 9 - Classe Categoria 14](#_Toc155103633)

[Figura 10 - Classes Herança de Categoria 15](#_Toc155103634)

[Figura 11 - Classe Carrinho Compras 15](#_Toc155103635)

[Figura 12 - Classe Campanha 16](#_Toc155103636)

# Introdução

No âmbito da disciplina de Programação Orientada a Objetos, dedicamos o nosso trabalho ao desenvolvimento de uma solução que consolida os temas aprendidos ao longo do semestre, assim como um desafio prático e atual como o comércio online. A escolha do tema *“(VI) Comércio eletrónico: sistema que permita a gestão de uma loja online.”.*

A nossa abordagem para o projeto envolve a implementação de uma arquitetura “NTIER”, uma estrutura que divide o sistema em camadas distintas, cada uma com responsabilidades específicas. A arquitetura “NTIER” compreende camadas dedicadas à apresentação, lógica de negócio e dados. A camada de apresentação concentra-se na interface do utilizador e na experiência de compra, a camada de lógica de negócios gere as regras e operações específicas do comércio eletrónico, enquanto a camada de dados lida com o armazenamento e recuperação de informações cruciais para o funcionamento da loja online.

No âmbito deste trabalho, a aplicação prática dos conceitos de Programação Orientada a Objetos incluirá a criação de classes, como ‘Produto´, ‘Cliente’ e ‘Venda’ e, para além disso, a utilização de listas será essencial para armazenar e manipular objetos, como a lista de produtos.

Dessa forma, este projeto não aborda apenas os aspetos teóricos da programação orientada a objetos, mas também proporciona uma oportunidade para aplicação prática desses conceitos na construção de um sistema funcional e adaptável para o ambiente desafiador do comércio eletrónico.

# Implementação

## Descrição do problema

Num mundo cada vez mais online, o desenvolvimento de uma solução capaz de lidar com a gestão de um comércio online, torna-se uma ferramenta crucial para o sucesso de qualquer negócio que desenvolva a sua atividade nestes moldes.

Empresas presentes em várias áreas de negócio e com diferentes capacidades financeiras, desde negócios a começar a atividade até empresas consolidadas no mercado recorrem a este tipo formato como meio de chegar aos clientes.

## Solução

Desenvolvemos uma aplicação capaz de lidar com a gestão das diferentes partes do negócio, desde os clientes até ao registo de vendas.

1. Gestão de produtos: A gestão dos produtos é central para a operação da loja, cada produto é representado por uma classe dedicada, contendo informações vitais como nome e preço. A estrutura de gestão de produtos permite a adição e remoção de novos itens à lista disponível e várias tarefas relacionadas com a gestão dos produtos.
2. Gestão de clientes: Clientes são ativos fundamentais para qualquer empreendedorismo. A solução permite fazer a gestão destes ativos, permitindo criar uma ficha com os dados essenciais para o envio dos produtos, assim como métodos para manipular esses dados.
3. Gestão de vendas: A estrutura de gestão de vendas permite o registo detalhado de cada transação vinculando clientes aos produtos adquiridos.

# Implementação C#

A implementação do código procura ser o mais simples quanto possível, estando sempre bem documentado e utilizando a plataforma do GITHUB para controlo e desenvolvimento das versões.

Link GITHUB: <https://github.com/ItsShoon/ShopSharp/tree/master>

## DAL

Camada de acesso a dados, responsável por facilitar a interação entre a aplicação e o banco de dados, gerindo operações relacionadas com a leitura e gravação da informação gerada.

A nossa abordagem nesta camada da aplicação passou por separar o acesso aos dados das diferentes classes existentes. Para isso foi criado um arquivo para cada classe tornando o código simples e limpo, facilitando qualquer alteração ou modificação necessária à medida que a complexidade da aplicação vai aumentando.

O formato de dados utilizado é um ficheiro “.json”, abordado e explicado em contexto de aula, e nesta camada da aplicação tratamos da manipulação deste ficheiro, em tarefas com gravação e leitura dos dados presentes no mesmo.

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 1 - Visão Geral Produto.DAL

private const string produtoFilePath = "produtos.json";

- Declaração de uma constante ‘produtoFilePath’ responsável por armazenar o caminho para o

ficheiro JSON.

public List<Produto> LerProdutos()

public List<Cliente> LerClientes()

{

if (File.Exists(clienteFilePath))

{

var json = File.ReadAllText(clienteFilePath);

var settings = new JsonSerializerSettings

{

TypeNameHandling = TypeNameHandling.Auto

};

// Desserializa a lista de clientes, preservando o tipo específico de cada cliente.

var clientesComTipo = JsonConvert.DeserializeObject<List<JObject>>(json, settings);

var listaClientes = clientesComTipo.Select(item =>

{

var tipoCliente = item["Tipo"].ToObject<string>();

switch (tipoCliente)

{

case "Particular":

return item["Cliente"].ToObject<ClienteParticular>();

case "Empresarial":

return item["Cliente"].ToObject<ClienteEmpresarial>();

default:

return item["Cliente"].ToObject<Cliente>();

}

}).ToList();

return listaClientes;

}

return new List<Cliente>();

}

- Método responsável pela leitura dos dados existentes no ficheiro .JSON, sendo feita a desserialização do conteúdo do ficheiro para a lista clientes. Adicionalmente temos uma verificação da existência do ficheiro, que retorna uma lista vazia se o ficheiro não existir. No ficheiro DAL dos clientes, como existes duas subclasses de cliente, é feita uma análise adicional que verifica o tipo de cliente e devolver os dados do cliente para o tipo correto de cliente, empresarial ou Particular.

Nos restantes ficheiros DAL, não se verifica esta particularidade, portanto esta separação adicional não é necessária.

public void GravarClientes(List<Cliente> clientes)

{

var settings = new JsonSerializerSettings

{

TypeNameHandling = TypeNameHandling.Auto

};

// Prepara a lista de clientes com informações de tipo para serialização.

var clientesComTipo = clientes.Select(cliente =>

{

var tipoCliente = cliente is ClienteParticular ? "Particular" : cliente is ClienteEmpresarial ? "Empresarial" : "Comum";

return new JObject

{

["Tipo"] = tipoCliente,

["Cliente"] = JToken.FromObject(cliente, JsonSerializer.Create(settings))

};

}).ToList();

var json = JsonConvert.SerializeObject(clientesComTipo, Formatting.Indented);

File.WriteAllText(clienteFilePath, json);

Console.WriteLine("Cliente gravado com sucesso!");

}

- Método responsável pela gravação dos dados no ficheiro JSON, trata de receber a lista de clientes e, neste caso em específico, antes de proceder à gravação adiciona a informação sobre o tipo de cliente (comum, empresarial, particular). Em seguida recorrendo à biblioteca DAL transforma a lista numa STRING JSON formatada e escreve no arquivo.

Os restantes ficheiros .DAL da aplicação foram criados utilizando a mesma lógica sendo a sua construção igual à apresentada acima, com exceção da regra especifica para adicionar o tipo de cliente.

|  |
| --- |
| public Cliente ObterClientePorId(int clienteId)  {  var clientes = LerClientes();  return clientes.FirstOrDefault(c => c.ClienteID == clienteId);  }  public void EditarCliente(Cliente clienteEditado)  {  var clientes = LerClientes();  var clienteExistente = ObterClientePorId(clienteEditado.ClienteID);  if (clienteExistente != null)  {  var index = clientes.IndexOf(clienteExistente);  clientes[index] = clienteEditado;  GravarClientes(clientes);  }  } |

- Os métodos ‘ObterClientePorID’ e ‘EditarCliente’ estão relacionados com a manipulação da lista de clientes.

O primeiro recebe o ID de um cliente como parâmetro e retorna o primeiro cliente na lista, cujo ID corresponde ao fornecido. Utiliza a função ‘FirstorDefault’ para encontrar o primeiro cliente que atende à condição especificada. Se a condição não se verificar devolve ‘null’.

O segundo método recebe um objeto Cliente (representando o cliente editado) como parâmetro. Primeiro, ele lê a lista completa de clientes usando o método LerClientes. Em seguida, chama o método ObterClientePorId para encontrar o cliente existente na lista com o mesmo ID que o cliente editado.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Figura 2 - Visão Geral dos ficheiros DAL

Foram aplicados métodos semelhantes nos restantes ficheiros DAL existentes.

## BLL

Camada logica de negócio, responsável pelas regras do negócio da aplicação. Nesta camada temos o processamento, validação e execução das operações relacionadas às regras especificas do domínio da aplicação.

Foi utilizada a mesma abordagem da camada DAL, sendo separadas as regras de negócio associadas a cada classe pelas mesmas razões anteriormente referidas.

Utilizando como exemplo o BLL dos produtos, vamos demostramos abaixo os métodos comuns a todos os ficheiros.

private ProdutoDAL produtoDAL = new ProdutoDAL();

As regras de negócio necessitam estar relacionadas com o acesso a dados para poder interagir com os mesmos, sendo por isso, necessário declarar um campo para esse feito. É igualmente necessário criar o construtor da classe que recebe uma instância do DAL como parâmetro e atribui a um novo campo. Para isso é utilizado o trecho de código acima.

public List<Produto> ObterProdutos()

{

return produtoDAL.LerProdutos();

}

- Método responsável por chamar o método ‘LerProdutos’ existente na camada de DAL.

public Produto ObterProdutoPorId(int produtoId)

{

return produtoDAL.ObterProdutoPorId(produtoId);

}

- Método responsável por chamar o método ‘ObterProdutoPorId’ existente na camada de DAL.

public void AdicionarProduto(string nome, decimal preco, int stock, string nomeMarca, int garantiaMeses, string nomeCategoria, string tipoCategoria, string detalheCategoria)

{

// Obter a lista de produtos, marcas e categorias

var listaProdutos = produtoDAL.LerProdutos();

var listaMarcas = marcaBLL.ObterMarcas();

var listaCategorias = categoriaBLL.ObterCategorias();

// Obter o ID mais alto entre os produtos existentes

int novoProdutoID = listaProdutos.Count > 0 ? listaProdutos.Max(p => p.ProdutoId) + 1 : 1;

// Procurar a marca existente ou criar uma nova

Marca marcaExistente = listaMarcas.FirstOrDefault(m => m.Nome == nomeMarca);

if (marcaExistente == null)

{

marcaBLL.AdicionarMarca(nomeMarca); // Cria uma nova marca

listaMarcas = marcaBLL.ObterMarcas(); // Atualiza a lista de marcas

marcaExistente = listaMarcas.FirstOrDefault(m => m.Nome == nomeMarca);

}

// Procurar ou criar a categoria correspondente

Categoria categoriaExistente = listaCategorias.FirstOrDefault(c => c.Nome == nomeCategoria);

if (categoriaExistente == null)

{

switch (tipoCategoria)

{

case "Hardware":

categoriaExistente = new CategoriaHardware(nomeCategoria, detalheCategoria);

break;

case "Software":

categoriaExistente = new CategoriaSoftware(nomeCategoria, detalheCategoria);

break;

case "Gadgets":

categoriaExistente = new CategoriaGadgets(nomeCategoria, detalheCategoria);

break;

default:

throw new ArgumentException("Tipo de categoria inválido");

}

listaCategorias.Add(categoriaExistente);

categoriaDAL.GravarCategorias(listaCategorias);

}

// Cria e adiciona o novo produto com o ID incrementado

Produto novoProduto = new Produto(nome, preco, stock, marcaExistente, garantiaMeses, categoriaExistente)

{

ProdutoId = novoProdutoID

};

listaProdutos.Add(novoProduto);

// Atualizar as listas no arquivo

produtoDAL.GravarProdutos(listaProdutos);

}

- Método responsável por adicionar um novo produto na lista, começa por ler as listas atuais de Produtos, marcas e categorias e gera um novo id para o produto com base nos produtos anteriores. Procura igualmente se a marca inserida já existe e se não for o caso cria uma marca utilizando o método de adição do BLL das marcas, atualizando a lista das marcas, sendo o processo repetido para as categorias. No final o novo produto é adicionado à lista de produtos.

public void EditarProduto(Produto produto)

{

produtoDAL.EditarProduto(produto);

}

public void RemoverProduto(int produtoId)

{

produtoDAL.RemoverProduto(produtoId);

}

- Métodos responsáveis por chamar o método de edição e remoção dos produtos presentes na classe DAL. Recebe por parâmetro o id de um produto e chama os métodos correspondentes para tratar os dados.

Para além dos métodos comuns a todos os BLL presentes na solução, existem métodos para realizar tarefas exclusivas do BLL onde se inserem.

### Carrinho BLL

|  |
| --- |
| public void AdicionarAoCarrinho(CarrinhoDeCompras carrinho, int produtoId, int quantidade)  {  var produto = produtoBLL.ObterProdutoPorId(produtoId);  // Verifica se o produto existe e se há stock suficiente  if (produto != null && produto.Stock >= quantidade)  {  // Verifica se o item já existe no carrinho  var itemExistente = carrinho.Itens.FirstOrDefault(i => i.ProdutoId == produtoId);  if (itemExistente != null)  {  itemExistente.Quantidade += quantidade;  }  else  {  carrinho.Itens.Add(new ItemCarrinho { ProdutoId = produtoId, Quantidade = quantidade });  }  }  else  {  Console.WriteLine("Produto não disponível ou stock insuficiente.");  }  } |

- Método responsável por adicionar produtos a um carrinho de compras, verificando se o mesmo está disponível e se há stock suficiente para o pedido.

Recebe por parâmetro um objeto ‘CarrinhoDeCompras’, o ID de um produto e uma quantidade e vai à classe produto buscar as informações do produto com o ID correspondente e verifica se existe stock suficiente para a quantidade pretendida. De seguida é feita uma verificação se o produto já se encontra no carrinho, e se for o caso é aumentada a quantidade do mesmo, caso não esteja no carrinho é adicionado um novo produto.

Quando não se verificam as condições é apresentada uma mensagem de aviso.

|  |
| --- |
| public Venda FinalizarCompra(CarrinhoDeCompras carrinho, int clienteId)  {  // Obtém os produtos do carrinho e cria uma nova venda  List<Produto> produtosComprados = carrinho.Itens.Select(i => produtoBLL.ObterProdutoPorId(i.ProdutoId)).ToList();  Venda novaVenda = new Venda(  clienteBLL.ObterClientePorId(clienteId),  produtosComprados,  DateTime.Now  );  // Calcula o total da venda e atualiza o stock dos produtos  decimal totalVenda = vendaBLL.CalcularTotalVenda(novaVenda);  novaVenda.AtualizarTotal(totalVenda);  foreach (var item in carrinho.Itens)  {  var produto = produtoBLL.ObterProdutoPorId(item.ProdutoId);  produto.Stock -= item.Quantidade;  produtoBLL.EditarProduto(produto);  }  vendaBLL.AdicionarVenda(novaVenda);  LimparCarrinho(carrinho);  return novaVenda;  } |

- Método criado para concluir uma venda, onde é criada e guardada a informação da nova venda e atualizados os dados do produto vendido.

São passados por parâmetro o objeto ‘CarrinhoDeCompras’ e o ID do cliente que faz a compra, sendo de seguida criada lista de produtos comprados obtendo as informações sobre os produtos através do método ‘ObterProdutoPorID’ da classe ‘produtoBLL’. É criada a nova venda associando-a ao cliente, á lista de produtos e introduzindo a data atual como a data de venda.

Para finalizar é atualizado o stock de cada produto presente no carrinho, recorrendo ao método ‘EditarProduto’ e adicionada a nova venda à lista de vendas e retornada a nova venda.

### Venda BLL

- Método utilizado para calcular o valor total de uma venda, levado em consideração os produtos incluídos na venda e a aplicação de eventuais descontos de acordo com a campanha em vigor.

É iniciada uma variável com o valor a 0 para receber o valor total da compra. De seguida itera sobre os produtos presente na venda e verificar se existe alguma campanha ativa em que o produto esteja inserido. Se esta última condição se verificar é reduzido o valor do produto. Depois de verificar as condições e somar o valor é devolvida a variável com o valor da compra.

|  |
| --- |
| public decimal CalcularTotalVenda(Venda venda)  {  decimal total = 0;  foreach (var produto in venda.Produtos)  {  decimal precoProduto = produto.Preco;  if (venda.CampanhaAplicada != null &&  venda.CampanhaAplicada.EstaAtiva() &&  venda.CampanhaAplicada.ProdutosAssociados.Contains(produto.ProdutoId))  {  precoProduto -= precoProduto \* (venda.CampanhaAplicada.Desconto / 100m);  }  total += precoProduto;  }  return total;  } |

## 

## Classes

Foram criadas classes para os clientes, produtos e vendas sendo que cada uma contém os elementos necessários para a construção das listas necessárias para a solução. Utilizamos igualmente classes que herdaram atributos, como é o caso das classes clientes particulares e empresariais que herdam atributos da classe Cliente.

### Clientes

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

Descrição gerada automaticamente**

Figura 3 - Classe Cliente e construtor

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

Descrição gerada automaticamente**

Figura 4 – Classes Herança de Cliente

### Produtos



Figura 5 - Classe Produto

### Venda

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 6 - Classe Venda

### Marca

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 7 - Classe Marca

### Garantia

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

Figura 8 - Classe Garantia

### Categoria

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Figura 9 - Classe Categoria

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 10 - Classes Herança de Categoria

### Carrinho Compras

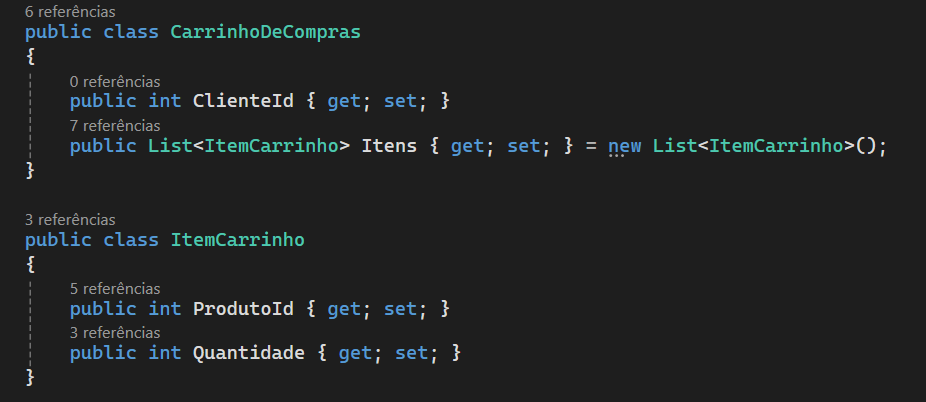


Figura 11 - Classe Carrinho Compras

### Campanha

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 12 - Classe Campanha

A classe campanha contém uma função para verificar se as campanhas se encontram ativas.

|  |
| --- |
| public bool EstaAtiva()  {  var agora = DateTime.Now;  return agora >= DataInicio && agora <= DataFim;  } |

# Interface

Camada de interação com o utilizador, onde são incluídos os métodos que necessitam de ação humana para os podermos utilizar.

Foram criados menus para facilitar a utilização da aplicação, e criados os métodos que através do menu permitem tarefas como listar, procurar, adicionar ou remover produtos e clientes, das listas anteriormente explicadas.

Nesta camada do programa só estão presentes os métodos necessários para a manipulação dos restantes métodos criados que foram criados nas restantes camadas da aplicação.

## Menus

### Menu Inicial

Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamente

Figura 13 - Menu Inicial

### Menu Gestor

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

Descrição gerada automaticamente

Figura 14 - Menu Gestor

### Menu

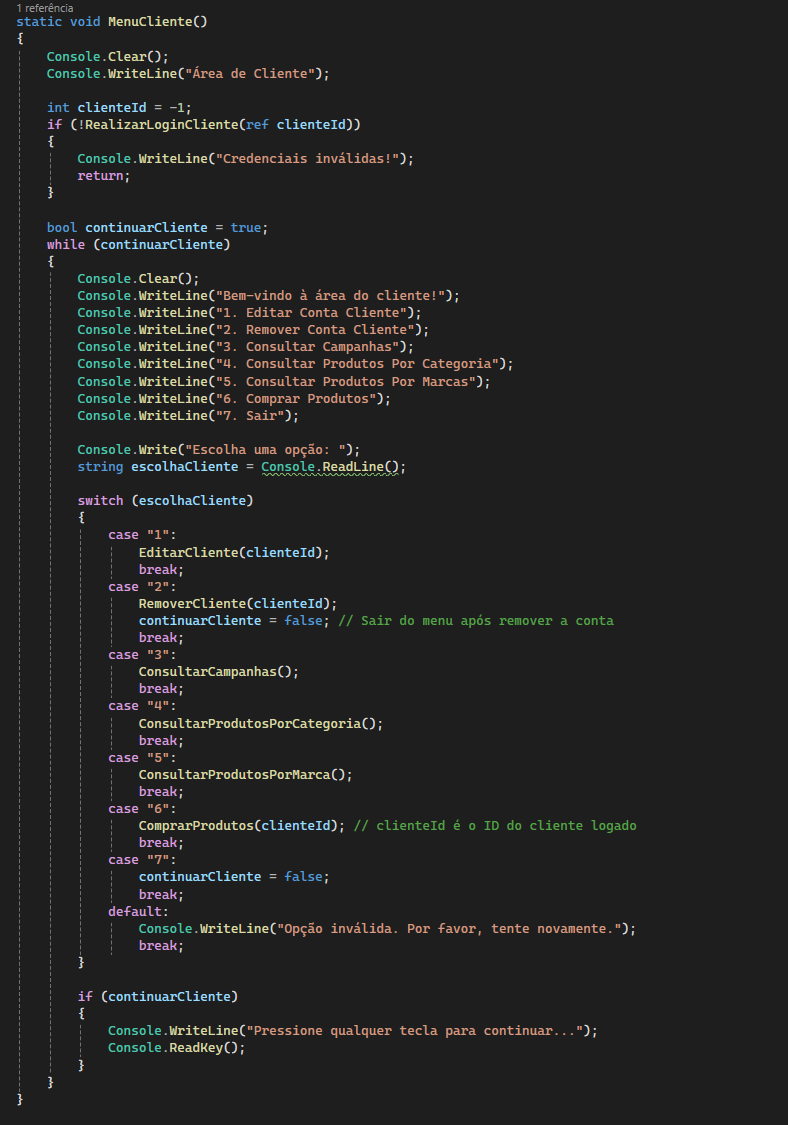


Figura 15 - Menu Cliente

## Métodos Aplicados no Interface

### Novo Registo

|  |
| --- |
| static void RegistarContaCliente()  {  Console.Clear();  Console.WriteLine("Registar nova conta de cliente");  ClienteBLL clienteBLL = new ClienteBLL();  List<Cliente> clientesExistentes = clienteBLL.ObterClientes();    Console.Write("Nome: ");  string nome = Console.ReadLine();  string email;  do  {  Console.Write("Email: ");  email = Console.ReadLine();  if (clientesExistentes.Any(c => c.Email == email))  {  Console.WriteLine("Este email já está registado. Por favor, tente outro.");  }  } while (clientesExistentes.Any(c => c.Email == email));  Console.Write("Palavra-passe: ");  string palavraPasse = Console.ReadLine();  Console.Write("Contacto: ");  string contacto = Console.ReadLine();  Console.Write("Morada: ");  string morada = Console.ReadLine();  string nif;  do  {  Console.Write("NIF: ");  nif = Console.ReadLine();  if (clientesExistentes.Any(c => c.NIF == nif))  {  Console.WriteLine("Este NIF já está registado. Por favor, tente outro.");  }  } while (clientesExistentes.Any(c => c.NIF == nif));    Console.WriteLine("Tipo de cliente (1 para Particular, 2 para Empresarial): ");  string tipoCliente = Console.ReadLine();  if (tipoCliente == "1")  {  Console.Write("Cartão de Crédito: ");  string cartaoCredito = Console.ReadLine();  clienteBLL.AdicionarClienteParticular(nome, email, palavraPasse, contacto, morada, nif, cartaoCredito);  }  else if (tipoCliente == "2")  {  Console.Write("Nome do Responsável: ");  string nomeResponsavel = Console.ReadLine();  clienteBLL.AdicionarClienteEmpresarial(nome, email, palavraPasse, contacto, morada, nif, nomeResponsavel);  }  else  {  Console.WriteLine("Tipo de cliente inválido.");  return;  }  Console.WriteLine("Conta de cliente registada com sucesso!");  }  #endregion Registo de Contas Cliente  }  } |

- Método para fazer o registo de um novo cliente na plataforma, inicia por criada uma instância de clienteBLL para lidar com a camada de negócios dos clientes.

Começamos por obter a lista de clientes existente, e de seguida solicitamos as informações de registo de um novo cliente, onde é validado que o email e o nif são únicos (requisitos obrigatórios).

De seguida é perguntado qual o tipo de cliente e preenchidos os atributos únicos da subclasse correspondente.

Por fim o cliente é adicionado à lista.

### Compra Produtos

|  |
| --- |
| static void ComprarProdutos(int clienteId)  {  CarrinhoBLL carrinhoBLL = new CarrinhoBLL();  CarrinhoDeCompras carrinho = new CarrinhoDeCompras();  VendaBLL vendaBLL = new VendaBLL();  bool continuarCompra = true;  while (continuarCompra)  {  Console.WriteLine("Digite o ID do produto para adicionar ao carrinho ou 0 para finalizar:");  int produtoId = Convert.ToInt32(Console.ReadLine());  if (produtoId == 0)  {  break;  }  Console.WriteLine("Digite a quantidade do produto:");  int quantidade = Convert.ToInt32(Console.ReadLine());  // Adiciona o produto ao carrinho  carrinhoBLL.AdicionarAoCarrinho(carrinho, produtoId, quantidade);  }  // Finaliza a compra  Venda novaVenda = carrinhoBLL.FinalizarCompra(carrinho, clienteId);  Console.WriteLine($"Compra finalizada com sucesso! ID da Venda: {novaVenda.VendaId}, Total: {novaVenda.Total}"); |

- Método criado para finalizar a compra de um produto, que permite adicionar um produto ao carrinho de compras, especificar a quantidade necessária e finalizar a compra.

Os produtos são adicionados ao carrinho através de um ciclo while, onde o utilizador fornece o ID dos produtos e a quantidade desejada. A adição é feita através do método ‘AdicionarAoCarriho’ do BLL do carrinho.

Por fim é finalizada a compra através da chamada do método ‘FinalizarCompra’ do BLL carrinho, passando o carrinho e o ID do cliente. Calcula o total da compra usando a intância VendaBLL e atualiza o objeto venda com o valor total.

|  |
| --- |
| static void ConsultarProdutosPorMarca()  {  MarcaBLL marcaBLL = new MarcaBLL();  List<Marca> marcas = marcaBLL.ObterMarcas();  Console.Clear();  Console.WriteLine("Marcas Disponíveis:");  foreach (var marca in marcas)  {  Console.WriteLine($"{marca.MarcaID}: {marca.Nome}");  }  Console.Write("Escolha uma marca pelo ID: ");  int marcaId = Convert.ToInt32(Console.ReadLine());  var marcaEscolhida = marcas.FirstOrDefault(m => m.MarcaID == marcaId);  if (marcaEscolhida == null)  {  Console.WriteLine("Marca inválida.");  return;  }  ExibirProdutosPorMarca(marcaEscolhida.Nome);  }  static void ExibirProdutosPorMarca(string nomeMarca)  {  ProdutoBLL produtoBLL = new ProdutoBLL();  List<Produto> produtos = produtoBLL.ObterProdutos().Where(p => p.Marca.Nome == nomeMarca).ToList();  Console.Clear();  Console.WriteLine($"Produtos da marca '{nomeMarca}':");  foreach (var produto in produtos)  {  Console.WriteLine($"ID: {produto.ProdutoId}, Nome: {produto.Nome}, Preço: {produto.Preco}");  }  Console.WriteLine("Pressione qualquer tecla para voltar.");  Console.ReadKey();  } |

- Métodos criados para formar uma funcionalidade que permite consultar e exibir produtos com base na escolha de uma marca.

É criada uma instância da classe MarcaBLL e de seguida obtida a lista de marcas disponíveis através do método para obtenção das marcas.

Após o utilizador selecionar o ID da marca que deseja é chamado o segundo método para exibir os produtos da marca correspondente.

Este método obtém a lista dos produtos filtrados pela marca pretendida e mostra-os no ecrã.

A mesma lógica foi aplicada com a classe das categorias e das campanhas.

### Editar Cliente

|  |
| --- |
| static void EditarCliente(int clienteId)  {  ClienteBLL clienteBLL = new ClienteBLL();  Cliente cliente = clienteBLL.ObterClientePorId(clienteId);  if (cliente == null)  {  Console.WriteLine("Cliente não encontrado.");  return;  }  // Atualizar informações comuns a todos os clientes  Console.Write("Nome atual: " + cliente.Nome + ". Novo nome (deixe em branco para manter): ");  string novoNome = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(novoNome))  {  cliente.Nome = novoNome;  }  Console.Write("Email atual: " + cliente.Email + ". Novo email (deixe em branco para manter): ");  string novoEmail = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(novoEmail))  {  cliente.Email = novoEmail;  }  Console.Write("Nova palavra-passe (deixe em branco para manter): ");  string novaPalavraPasse = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(novaPalavraPasse))  {  cliente.PalavraPasse = novaPalavraPasse;  }  Console.Write("Morada atual: " + cliente.Morada + ". Nova morada (deixe em branco para manter): ");  string novaMorada = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(novaMorada))  {  cliente.Morada = novaMorada;  }  Console.Write("NIF atual: " + cliente.NIF + ". Novo NIF (deixe em branco para manter): ");  string novoNif = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(novoNif))  {  cliente.NIF = novoNif;  }  // Atualizar informações específicas para Cliente Particular  if (cliente is ClienteParticular clienteParticular)  {  Console.Write("Cartão de Crédito atual: " + clienteParticular.CartaoCredito + ". Novo cartão (deixe em branco para manter): ");  string novoCartaoCredito = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(novoCartaoCredito))  {  clienteParticular.CartaoCredito = novoCartaoCredito;  }  }  // Atualizar informações específicas para Cliente Empresarial  else if (cliente is ClienteEmpresarial clienteEmpresarial)  {  Console.Write("Nome do Responsável atual: " + clienteEmpresarial.NomeResponsavel + ". Novo nome (deixe em branco para manter): ");  string novoNomeResponsavel = Console.ReadLine();  if (!string.IsNullOrWhiteSpace(novoNomeResponsavel))  {  clienteEmpresarial.NomeResponsavel = novoNomeResponsavel;  }  }  // Salvar as alterações  clienteBLL.EditarCliente(cliente);  Console.WriteLine("Informações da conta atualizadas com sucesso.");  } |

- Método criado para permitir ao cliente editar as informações do seu perfil com base no id do cliente fornecido como parâmetro.

Após obter as informações da lista de cliente, solicita ao utilizador os novos dados e atualiza os campos do objeto cliente, se o cliente for particular ou empresarial pede para atualizar os dados adicionais.

É chamado o método ‘EditarCliente’ para salvar as alterações e exibida a mensagem de confirmação.

### Métodos de Gestão

|  |
| --- |
| static void ListarMarcas(MarcaBLL marcaBLL)  {  var marcas = marcaBLL.ObterMarcas();  if (marcas.Count == 0)  {  Console.WriteLine("Não existem marcas registadas.");  }  else  {  foreach (var marca in marcas)  {  Console.WriteLine($"ID: {marca.MarcaID}, Nome: {marca.Nome}");  }  }  }  static void AdicionarMarca(MarcaBLL marcaBLL)  {  Console.Write("Insira o nome da nova marca: ");  string nomeMarca = Console.ReadLine();  marcaBLL.AdicionarMarca(nomeMarca);  Console.WriteLine("Marca adicionada com sucesso!");  }  static void EditarMarca(MarcaBLL marcaBLL)  {  Console.Write("Insira o ID da marca que deseja editar: ");  int id = Convert.ToInt32(Console.ReadLine());  var marca = marcaBLL.ObterMarcaPorId(id);  if (marca == null)  {  Console.WriteLine("Marca não encontrada.");  return;  }  Console.WriteLine($"Editar Marca: {marca.Nome}");  Console.Write("Novo nome: ");  marca.Nome = Console.ReadLine();  marcaBLL.EditarMarca(marca);  Console.WriteLine("Marca editada com sucesso!");  }  static void RemoverMarca(MarcaBLL marcaBLL)  {  Console.Write("Insira o ID da marca que deseja remover: ");  int id = Convert.ToInt32(Console.ReadLine());  var marca = marcaBLL.ObterMarcaPorId(id);  if (marca == null)  {  Console.WriteLine("Marca não encontrada.");  return;  }  marcaBLL.RemoverMarca(id);  Console.WriteLine("Marca removida com sucesso!");  } |

- Métodos criados para a gestão dos dados presentes nas listas de marcas, produtos, categorias, campanhas e clientes.

1. O método de listar é responsável mostrar todas marcas disponíveis, percorre a lista das marcas e mostra o ID e nome da marca.

2. O método adicionar é responsável por criar uma marca, pede os dados ao utilizador e chama o método ‘AdiconarMarca’ presente no BLL.

3. O método de editar recebe uma nova marca através do ID e procura se a mesma existe na lista, se for o caso pede para atualizar os dados e através do método ‘editarMarca’ do BLL altera a marca presente na lista.

4. O método de remover a marca recebe uma marca através do ID e se a marca existir chama o método ‘RemoverMarca’ do BLL correspondente para eliminar a marca.

### Login

|  |
| --- |
| static bool RealizarLoginCliente(ref int clienteId)  {  Console.WriteLine("Por favor, insira suas credenciais de login.");  Console.Write("Email: ");  string email = Console.ReadLine();  Console.Write("Palavra-passe: ");  string palavraPasse = Console.ReadLine();  ClienteBLL clienteBLL = new ClienteBLL();  List<Cliente> clientes = clienteBLL.ObterClientes();  var clienteLogado = clientes.FirstOrDefault(c => c.Email == email && c.PalavraPasse == palavraPasse);  if (clienteLogado != null)  {  clienteId = clienteLogado.ClienteID; // Atribuir o ID do cliente autenticado  return true;  }  return false; // Login falhou  } |

- Método criado para os clientes realizarem login, é pedido e lido o email e a password do cliente, de seguida é procurado o cliente na lista de clientes e verificado se os dados do cliente estão corretos, se os dados forem corretos é devolvido ‘true’ e é feito o login, senão é, exibida uma mensagem de erro.