Programação Visual
Trabalho de Laboratório nº 6

Objetivo	MVC – Introdução. Aplicações básicas com acesso a base de dados através do *Entity Framework* e utilização do pacote *Microsoft Identity*.
Programa	Pretende-se criar um *Site* de compras de supermercado. A aplicação deverá permitir aos utilizadores visualizar os produtos, onde as lojas se situam e adicionar produtos ao carrinho de compras. Deverá existir um administrador que faz a gestão de stock dos produtos.
Regras	Implementar o código necessário e testar no fim de cada nível. Use as convenções de codificação adotadas para a linguagem C# e para o modelo MVC.

Nível 1

- Crie uma aplicação **ASP.NET Core Web Application, MVC,** dê-lhe o nome de **MiniESTS** e escolha como método de autenticação **Individual Accounts**.
- Na diretoria **Models**, crie os seguintes modelos:
    - **Produto - ProdutoId, LojaId, Nome, Tipo, Preco, Unidades, Loja**
    - **Loja –LojaId, Stock, Localidade**

  Nota: **Stock** é uma lista de produtos. **Tipo** é um tipo enumerado com os valores **Congelados, Bebidas, Carnes, Frescos e Enlatados**.
- No **Tipo** inclua a anotação [EnumDataType(typeof(Tipo))] e no **Preco**, as anotações [Column(TypeName = "decimal(10,2)")] e [Range(0, 999.99), DataType(DataType.Currency)]
- Gere por *scaffolding*, os controladores MVC para os **Produtos** e para as **Lojas** selecionando a opção que inclui as vistas usando o ***Entity Framework***. Em ambos os casos, use o contexto **ApplicationDbContext** criado antes pelo **Microsoft Identity.**
- Adicione à barra de menus os links para a view *Index* dos controladores que criou e corrija as várias vistas do controlador **Home** de forma que estejam personalizadas para esta aplicação.
- Compile a aplicação e aplique as modificações na base de dados, executando os comandos de consola para criação e atualização da base de dados.
- Verifique que a aplicação está a funcionar com as alterações de visualização efetuadas. Não adicione nenhum produto ou loja.

Nível 2

- Defina um conjunto de dados iniciais para a aplicação. Neste caso, faça o *override* do método **onModelCreating** na classe de contexto e adicione 2 lojas e 5 produtos.
- Corrija agora os erros de funcionamento. Em particular, deve ter em atenção que quando criar ou editar um produto deve ser possível selecionar um valor de tipo usando uma caixa *dropdown.* Deve também visualizar a Localidade na lista de lojas e de produtos em vez do ID da loja e a lista de produtos nos detalhes da respetiva loja.

Nota: Em qualquer altura, pode apagar a base de dados através do *SQL Server Object Explorer* e a pasta de migrações e voltar a criar as migrações e a base de dados com os respetivos comandos de consola.

Programação Visual
Trabalho de Laboratório nº 6

Nível 3

- Pretende-se agora criar uma classe **Cliente** estendendo o *user* criado pelo pacote *Microsoft Identity* que foi instalado inicialmente na aplicação. Uma vez que não tem acesso a este código, adicione à pasta do projeto um novo *scaffold item* selecionando a opção *Identity*. Na janela de diálogo que abriu, escolha o *override* dos ficheiros *Account\Register, Account\Login, Account\Logout* e *Account\Manage\index* e o contexto da aplicação criado antes.
- No modelo, crie a classe **Cliente** que deriva do **IdentityUser** e acrescente-lhe a propriedade **Nome**.
- Assegure que a classe do contexto deriva de **IdentityDbContext<Cliente>**.
- Será necessário agora corrigir as vistas *Register* e *Index* relativas aos ficheiros que adicionou antes. Para isso siga este tutorial (apenas as alterações realçadas referentes ao nome): https://docs.microsoft.com/en-us/aspnet/core/security/authentication/add-user-data?view=aspnetcore-6.0&tabs=visual-studio&viewFallbackFrom=aspnetcore-6#add-custom-user-data-to-the-identity-db
  Atenção que não está a usar agora a classe **IdentityUser** dos utilizadores, mas sim a classe **Cliente**. Altere o tipo anterior nas classes associadas a estas vistas.
- Na classe **Program**, também é necessário substituir o **IdentityUser** por **Cliente**.
- Também na vista **_LoginPartial** necessita de ser atualizada com a nova classe **Cliente** em vez de **IdentityUser.**
- Crie também a classe de serviço **EmailSender** e adicione o respetivo serviço na classe **Program**.
- Teste criando registando um novo cliente.

Nível 4

- Para se poder ter os papeis de **Administrador** e **Cliente** e incluir alguns utilizadores nestes papeis crie, na pasta **Data**, a classe estática **SeedData** com o método
  `public static async Task Seed(UserManager<Cliente> userManager, RoleManager<IdentityRole> roleManager)`
  E os métodos privados para a criação dos papeis e dos utilizadores. O método **SeedData** deve ser chamado dentro do controlador **Home** por injeção de dependências. Este deve receber, nos parâmetros, os gestores de papeis e de utilizadores.
- Adapte agora os menus, controladores e vistas para que os utilizadores no papel de **Cliente** tenham apenas acesso à visualização da lista de lojas e dos detalhes de cada loja, sem possibilidades de criação, edição ou remoção dos mesmos. Remova também o acesso dos **Clientes** poderem ver o menu de lista de produtos e todas as funções associadas ao mesmo, apenas os **Administradores** podem aceder a esta página e respetivas funções.
- A adição, edição e remoção das lojas é, também, exclusiva dos **Administradores**.

Programação Visual
Trabalho de Laboratório nº 6

Nível 5

- Para completar as funcionalidades da aplicação, vamos criar agora a possibilidade de os clientes poderem criar uma lista de compras. Neste caso, nos detalhes da loja onde é listado também os produtos, deverá de aparecer um botão que permita adicionar um produto ao carrinho. No menu principal da aplicação deve existir igualmente uma opção para "Ver carrinho". Esta opção deve mostrar a lista de produtos no carrinho com opção de remover produtos.
  Sendo assim, comece por criar a classe **Carrinho** que irá relacionar produtos com os clientes.
  Para que funcione com os utilizadores criados defina esta classe de acordo com o seguinte código:

```
public class Carrinho
 {
     [Key]
     public int ProdutoClienteId { get; set; }

     [ForeignKey("Produto")]
     public int ProdutoId { get; set; }
     [ForeignKey("Cliente")]
     public string ClienteId { get; set; }
     public Produto Produto { get; set; }
     public int Quantity { get; set; }
     public double Preco{ get; set; }
 }
```

- A opção de menu para o carrinho deve ser mostrada apenas para quem estiver no papel de **Cliente**, tal como o botão para adicionar um produto ao carrinho. Os clientes também devem poder visualizar e apagar os produtos que pertencem exclusivamente ao seu carrinho.
- Teste a aplicação.

Desafio

- Adicione uma opção para "Comprar" no **Carrinho** onde os produtos selecionados no carrinho serão removidos e alterando as unidades do produto da loja.

Notas

Para os identificadores siga as convenções adotadas pelo C#, nomeadamente:

- A notação camelCase para o nome das variáveis locais e identificadores privados.
- A notação PascalCase para os nomes públicos dos métodos, classes e interfaces.
- Não utilize o símbolo '_' nos identificadores nem abreviaturas