



# Apontamentos Práticos

Aula 11/10/2021

Referências

Aula 18/10/2021

Referências

Aula 08/11/2021

MVC

Models

View

Controllers

Notas

Como fazer o ex4

Aula 06/12/2021

Database First

Ficha 6

Alinea C)

Aula 13/12/2021

Ficha 6

Alinea D)

Alinea E)

Alinea F)

Referências

Aula 20/12/2021

Alinea G)

Usando o X.PagedList

Usando o plugin to jquery

Ficha 7

Aula 03/01/2022

Ficha 8

Aula 04/01/2022

Autenticação

Autorização

Diretamente no Controller

Para cada método dentro do Controller  
Autorização no Controller e nos métodos outras  
Podemos definir perfis para utilizadores  
Podemos definir políticas

## Aula 11/10/2021

- Feito o exercício 1,2 e 3 da Ficha 1
- Classe abstrata é uma espécie de esqueleto que serve apenas para apresentar ao utilizador
  - Esta classe nunca se instancia e os métodos normalmente são todos `public`  
`abstract [tipo] [nome do método]`
  - O método especial `ToString()` tem de ser `public abstract override string ToString()` porque este método herda de um Objeto e como não vai ser instanciado nenhum temos de dar `override`
  - A classe `FiguraPlana` é uma classe abstrata
- Nos métodos das classes que derivam da classe abstrata usamos `public override [tipo] [nome do método]`
  - As classes `Circulo`, `Triangulo` e `Retangulo` são classes derivadas

## Referências

- <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override>
- <https://www.damirscorner.com/blog/posts/20131028-UsingAbstractOverrideInC.html>
- <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract>

## Aula 18/10/2021

- Ler a ficha2 na parte do Linq
- Existem duas formas para usar o Linq

- extension methods
- query syntax
- Com o linq extension methods temos de pegar na lista e vermos os métodos que temos , por exemplo o OrderBy
- Contar o numero de palavras numa lista de strings

```
// linq extension method
var resultado = ListaUm.Select(e => e.Split(" ").Lenght);

// linq query method
var resultado2 = from e in ListaUm select e.Split(" ").Lenght;
```

- O `.Split()` cria uma lista com as palavras usando o espaço como delimitador e usando o `Lenght` conseguimos saber o tamanho dessa lista sabendo assim o numero de palavras por string
- Quando quero filtrar os dados utilizo o `Where`
- Quando quero filtrar e transformar os dados utilizado o `Select`
- Integrando o **Where** com o **Select**

```
// linq com extensions methods
var solucao = ListaUm.Where(s => s.Contains("C#"))
    .Select(s => new
    {
        frase = s.Trim(), // remove espacos a mais
        primeira = s.Trim().Split(" ").First(), // vai buscar a primeira palavra
        // se tiver apenas uma palavra escreve "NAO TEM" caso contrario escreve a ultima palavra
        ultima = (s.Trim().Split(" ").Count() == 1 ? "NAO TEM" : s.Trim().Split(" ").Last());
    });

foreach(var x in solucao)
    Console.WriteLine("Frase: " + x.frase + "\n\tPrimeira Palavra: " + x.primeira + "\n\tUltima Palavra: " + x.ultima);
```

## Referências

- <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/query-syntax-and-method-syntax-in-linq>
- <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods>

## Aula 08/11/2021

- O Razor é a mistura entre C# e html

## MVC

- Permite ter a separação em 3 camadas que nos permite construir código escalável

## Models

- São os nossos dados
- É aquilo que define o que são os dados e o acesso a esses dados
- Por exemplo o modelo é a nossa classe que trabalha com uma BD

## View

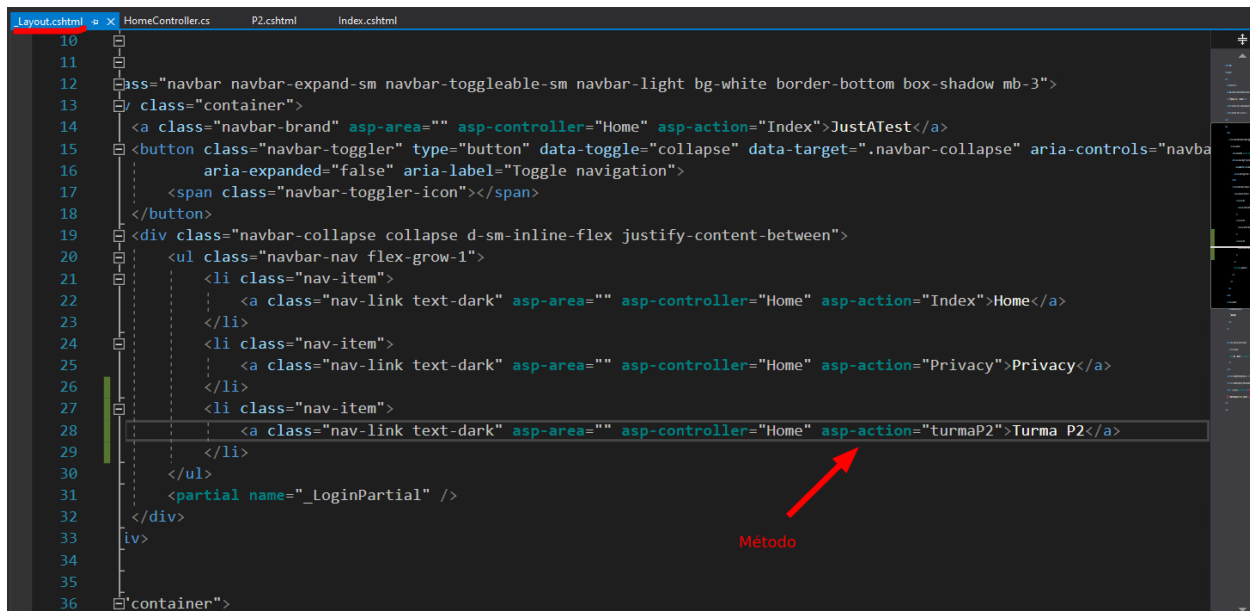
- Serve para mostrar ao utilizador a informação
- Dentro das vistas temos a vista em si e a pasta `shared` onde contém a estrutura da nossa aplicação
- Por omissão quando retornamos uma view, a view tem de ter o mesmo nome que o método, por isso se quisermos retornar uma vista com um nome diferente do método que queremos chamar, fazemos `return View("<nome do metodo>");` em vez de `return View();`

## Controllers

- É o intermediário entre o utilizador e o código
- Normalmente existe um controller e depois uma pasta nas vistas com esse mesmo nome
- Dentro do `HomeController.cs`

- O método Index devolve a vista Index
- Entao ele vai retornar o ficheiro index.cshtml que está em views/Home/
- O método Privacy devolve a vinda Privacy
- Entao ele vai retornar o ficheiro privacy.cshtml que está em views/Home/

## Notas



```

10
11
12 <div class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
13 <div class="container">
14 <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">JustATest</a>
15 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navba
16     aria-expanded="false" aria-label="Toggle navigation">
17     <span class="navbar-toggler-icon"></span>
18 </button>
19 <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
20 <ul class="navbar-nav flex-grow-1">
21 <li class="nav-item">
22 <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
23 </li>
24 <li class="nav-item">
25 <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
26 </li>
27 <li class="nav-item">
28 <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="turmaP2">Turma P2</a>
29 </li>
30 </ul>
31 <partial name="_LoginPartial" />
32 </div>
33 </div>
34
35
36 </div>

```

Método

- Operações CRUD é Create Read Update e Delete , são operações a uma base de dados

## Como fazer o ex4

- Criamos uma ASP.NET Core Web App MVC

## Additional information

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Target Framework ⓘ  
.NET 5.0 (Current) ▼

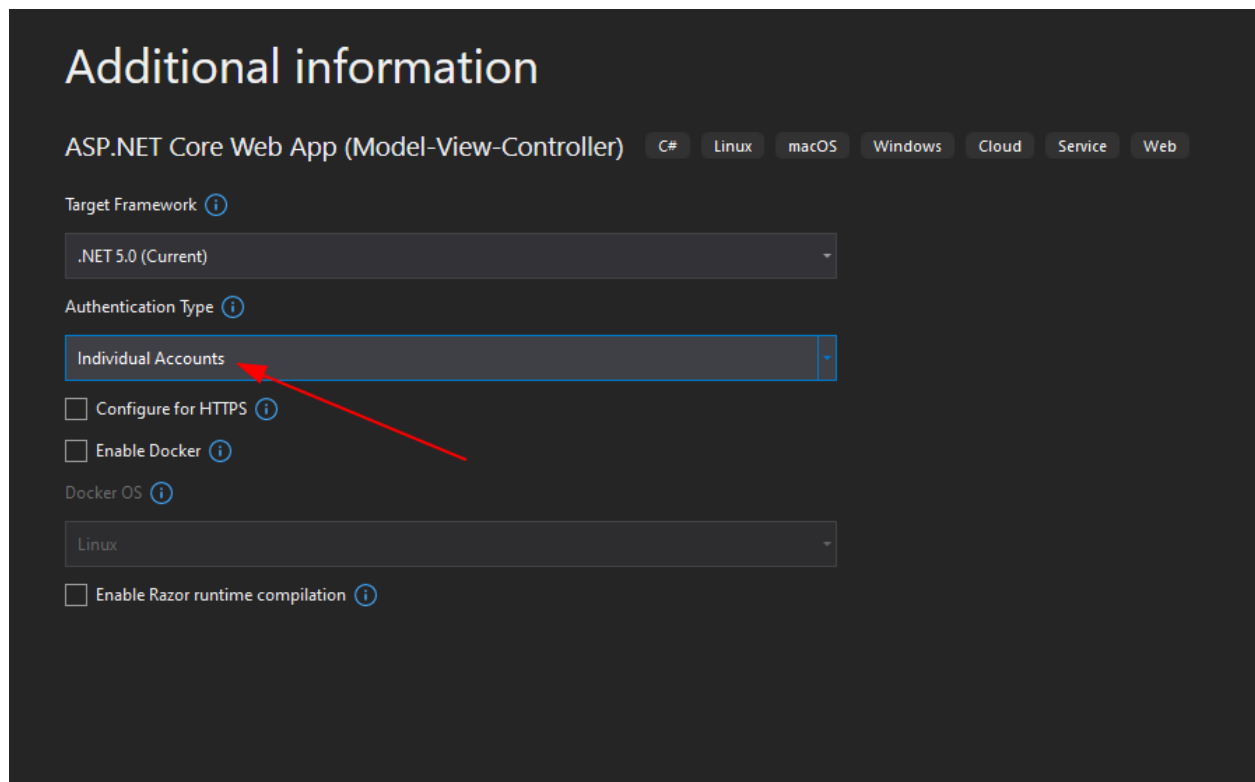
Authentication Type ⓘ  
Individual Accounts ▼

☐ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ  
Linux ▼

☐ Enable Razor runtime compilation ⓘ



- Ao criarmos uma Web App MVC convem começarmos pelos Models ( `Code First` )
- Entao criamos um Model vazio ( `UCs.cs` ) e colocamos os atributos dessa classe com um construtor

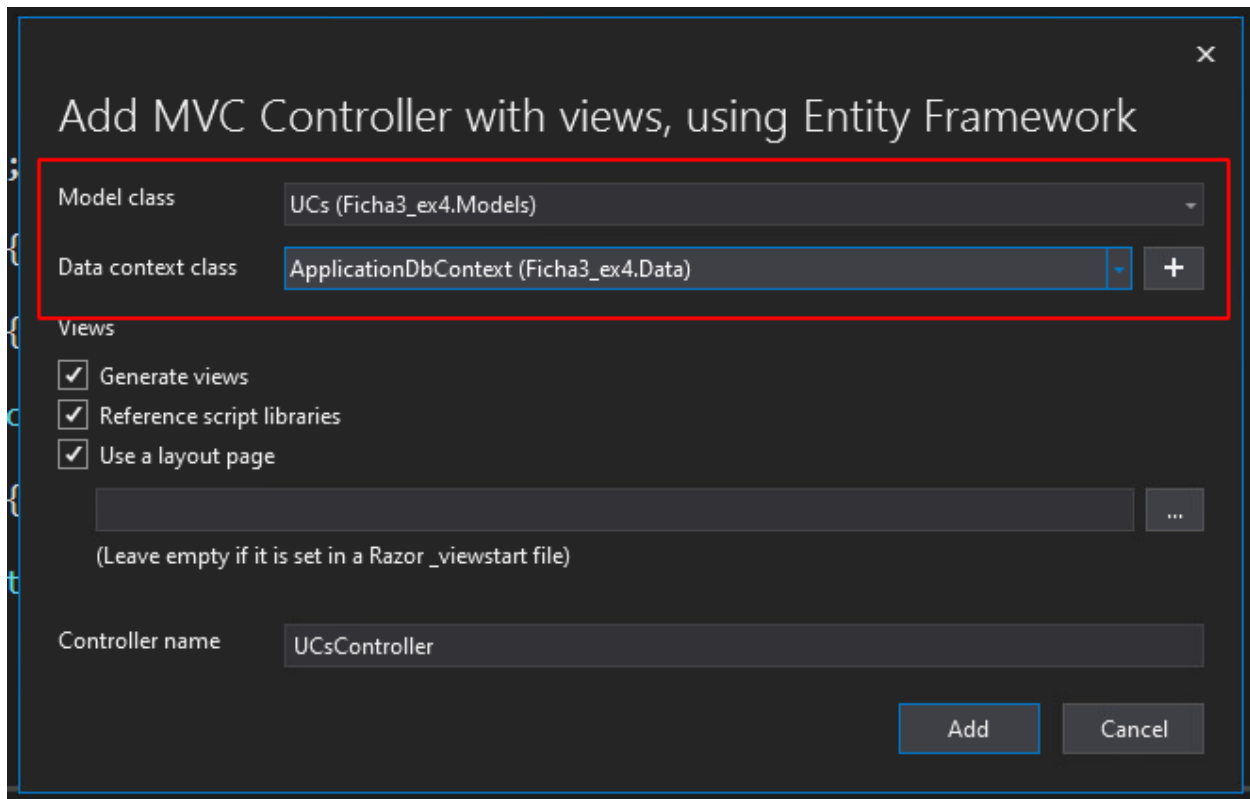
```

1 reference
public class UCs
{
    0 references
    public int Id { get; set; }
    0 references
    public string Nome { get; set; }
    0 references
    public string ECTS { get; set; }
    0 references
    public string Licenciatura { get; set; }
    0 references
    public string Ramo { get; set; }
    0 references
    public string Semestre { get; set; }

    0 references
    public UCs()
    {
    }
}

```

- Depois adicionamos à BD esse modelo
  - Add-Migration "Inicial"
  - Update-Database
- Depois vamos criar um Controlador para fazer todas as operações CRUD, então criamos um MVC Controller with views, using entity Framework



- Depois disto, temos novamente de fazer uma nova migração
  - Add-Migration "UCs"
  - Update-Database

## Aula 06/12/2021

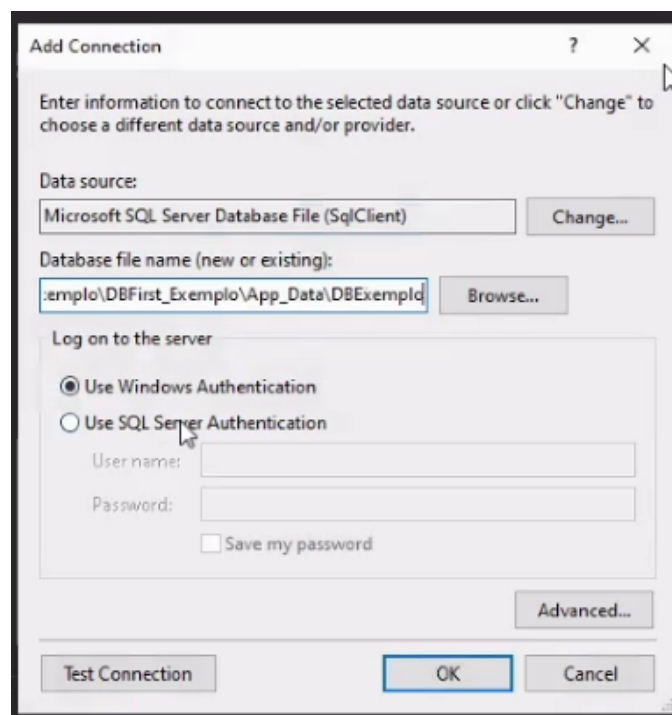
- Existe o conceito Code First
  - Maior controlo
  - Conseguimos gerir mais facilmente as alterações no código pelo sistema de versões
- Existe o conceito database First
  - Este conceito foi o usado nesta aula
  - Somos nós que criamos a base de dados e a base de dados é gerida à parte



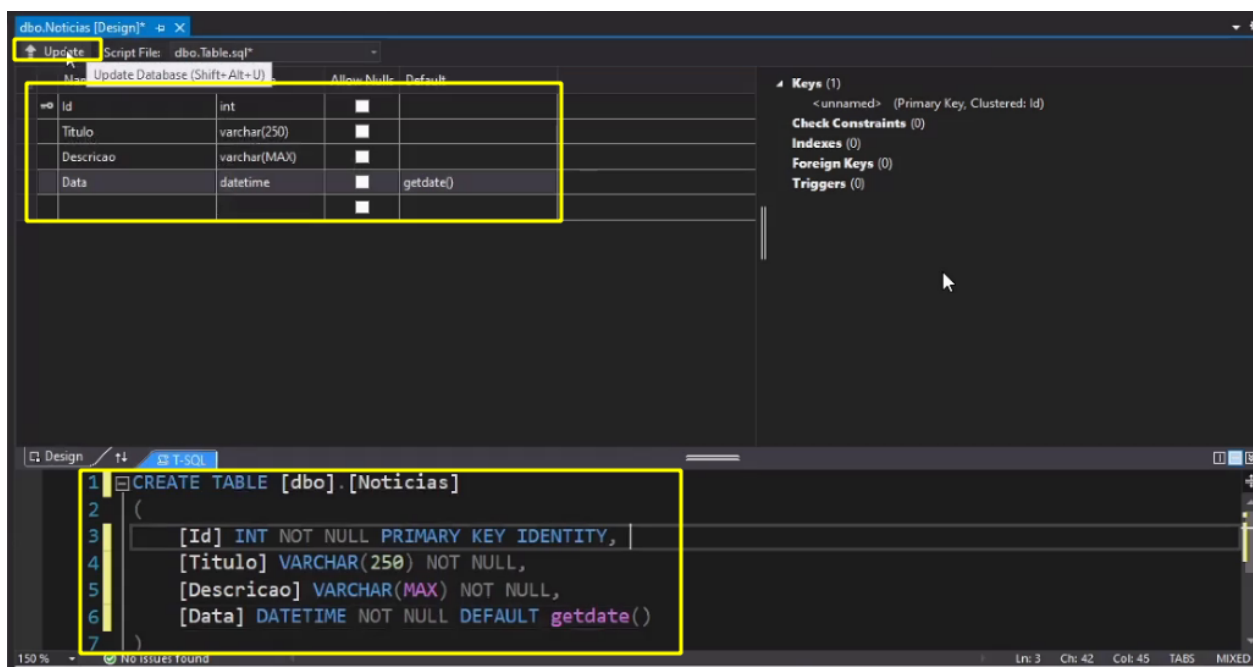
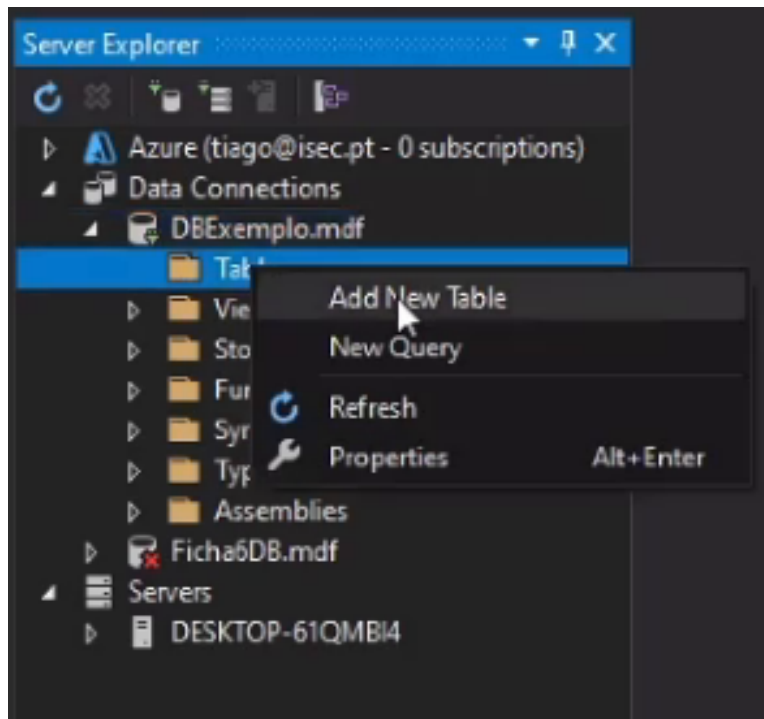
- Tem o problema de controlo de versões, uma vez que é mais trabalhoso e mais difícil

## Database First

- Criamos um projeto MVC
- Criamos uma base de dados dentro da pasta `App_Data`
  - Depois vamos a `Server Explorer` > `Data Connections` > `Change` > `Microsoft SQL Server Database File`

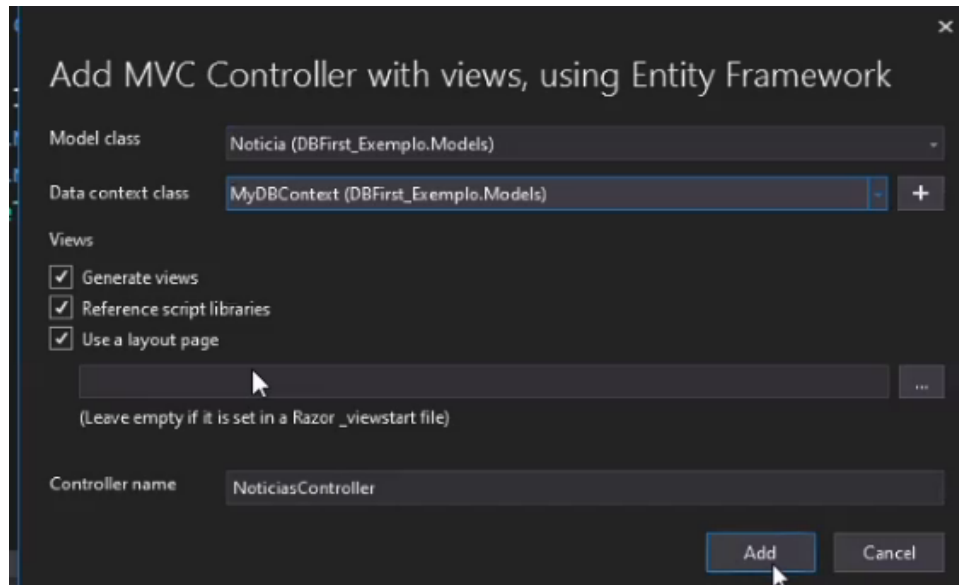


- Depois vamos criar uma tabela

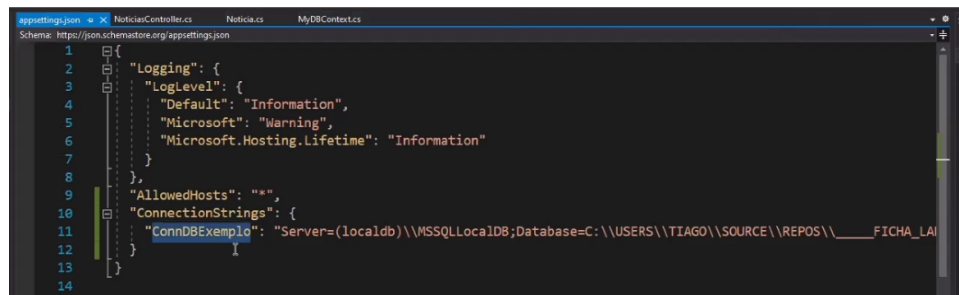


- Depois temos de ir ao **Package Manager Console** e colocar os seguintes comandos
  - `Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design -Version 5.0.2`
  - `Install-Package Microsoft.EntityFrameworkCore.Tools -Version 5.0.12`
  - `Install-Package Microsoft.EntityFrameworkCore.SqlServer -Version 5.0.12`

- E por fim usamos o Scaffold para criar os nossos models
  - `Scaffold-DbContext "Server=<caminho do localdb>;Database=<caminho da base de dados>" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Verbose`
- Depois temos de criar os controllers
  - Para isso utilizo o MVC Controller with views, using Entity Framework



- Depois é preciso ir ao nosso ficheiro gerado pelo Scaffold com o nome `DbContext.cs` nos models, copiar a string que começa com `Server=<caminho do localdb>;Database ....`, ir ao `appsettings.json` e acrescentar isto ...



- Depois vamos ao `Startup.cs` e vamos ao método `ConfigureServices` e adicionamos isto ...

```
// This method gets called by the runtime. Use this method to add services to the container.
public void ConfigureServices(IServiceCollection services)
{
    var connection = Configuration.GetConnectionString("ConnDBExemplo");
    services.AddDbContext<MyDbContext>(options => options.UseSqlServer(connection));

    services.AddControllersWithViews();
}
```

## Ficha 6

### Alinea C)

- Pra criar o botão de pesquisa criou-se um `Html.BeginForm` no `_Layout.cshtml` de modo a passar o input do utilizador para depois ser feita uma query à base de dados
- No `Html.BeginForm` dizemos o método onde vamos executar (`Index`), qual o controlador (`Categorias`) e qual o tipo de método (`Get`)

```
</div>
@using (Html.BeginForm("Index", "Categorias", FormMethod.Get))
{
    @Html.TextBox("search")<input type="submit" value=" Procurar" class="btn btn-primary" />
}
```

- Depois no `CategoriasController` criamos um método index que vai pesquisar na base de dados

```
public async Task<IActionResult> Index(string search)
{
    if (search != "")
    {
        return View(await _context.Categorias.Where(x => x.NomeCategoria == search || search == null).ToListAsync());
    }
    return View(await _context.Categorias.ToListAsync());
}
```

## Aula 13/12/2021

- Continuação da ficha 6 a partir da alinea c)

## Ficha 6

### Alinea D)

- Para criar a Dropdownlist fez-se um `Html.DropDownList` na view `index.cshtml` dos Produtos
- `asp-controller` é o nome do controller
- `asp-action` é o nome do método a executar
- O Primeiro parâmetro do `Html.DropDownList` é o Id , depois damos cast para um `IEnumerable` do `ViewData["IdCategoria"]`

```
<form asp-controller="Produtos" asp-action="Index" method="get">
  Filtrar por categoria <td><Html.DropDownList("Id", (IEnumerable<SelectListItem>)ViewData["IdCategoria"])</td>
  <input type="submit" value="Filtrar" />
</form>
```

- É mais facil em ver de usar o Html helper tag, usar os tag helpers com o ViewBag

```
<form asp-controller="Produtos" asp-action="Index" method="get">
  <b>Filtrar por Categoria</b>
  <select name="id" asp-items="ViewBag.IdCategoria"></select>
  <input type="submit" value="Filtrar" />
</form>
```

- No controller, passamos o parâmetro `id`
- Criamos o `ViewData["IdCategoria"]` com uma `selectList` contendo o Id e o Nome da categoria a partir da BD
- Fazemos um IF a ver se foi selecionada alguma categoria na drop down list, se sim, fazemos um `Where` procurando aquele ID escolhido

```
// GET: Produtos
3 references
public async Task<IActionResult> Index(int? id)
{
    var lista_produtos = _context.Produtos.Include(p => p.IdCategoriaNavigation);
    ViewData["IdCategoria"] = new SelectList(_context.Categorias, "IdCategoria", "NomeCategoria");
    if (id != null){
        return View(await lista_produtos.Where(x => x.IdCategoria == id).ToListAsync());
    }
    return View(await lista_produtos.ToListAsync());
}
```

## Alinea E)

- Foi criada uma classe chamada `CategoriaComContagem`

```

namespace Ficha6.Models
{
    1 reference
    public class CategoriaComContagem
    {
        1 reference
        public int IdCategoria { get; set; }
        2 references
        public string Nome { get; set; }
        2 references
        public int QuantProdutos { get; set; }

        0 references
        public string NomeComQuantidade
        {
            get { return $"{Nome} ({QuantProdutos.ToString()})"; }
        }
    }
}

```

- Depois no controller dos Produtos, foi criado um *extension method* para agrupar e contar os produtos por categoria
- Basicamente agrupamos por IdCategoria, selecionamos esse grupo, criamos um novo objeto `CategoriaComContagem` e definimos os campos.

```

// GET: Produtos
3 references
public async Task<IActionResult> Index(int? id)
{
    var lista_produtos = _context.Produtos.Include(p => p.IdCategoriaNavigation);

    var lista = lista_produtos.GroupBy(e => e.IdCategoria).
        Select(x => new CategoriaComContagem() {
            IdCategoria = x.Key,
            QuantProdutos = x.Count(),
            Nome = _context.Categorias.FirstOrDefault(c => c.IdCategoria == x.Key).NomeCategoria
        });

    ViewData["IdCategoria"] = new SelectList(lista, "IdCategoria", "NomeComQuantidade");

    //ViewData["IdCategoria"] = new SelectList(_context.Categorias, "IdCategoria", "NomeCategoria");
    if (id != null){
        return View(await lista_produtos.Where(x => x.IdCategoria == id).ToListAsync());
    }
    return View(await lista_produtos.ToListAsync());
}

```

## Alinea F)

- Na view `Index.cshtml` dos produtos, criamos um select com as opções de Ascendente e Descendente

```
<b>Ordenar</b>
<select>
  <option value="asc">Ascendente</option>
  <option value="desc">Descendente</option>
</select>
```

- Depois foi criada a class `Ordenacao.cs` para ajudar no output do texto

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace Ficha6.Models
7  {
8      3 references
9      public class Ordenacao
10     {
11         2 references
12         public string Valor { get; set; }
13         2 references
14         public string Texto { get; set; }
15     }
16 }
```

- No `ProdutosController.cs` foi adicionado o argumento `string ordenacao` e depois foi adicionado o código para a ordenação

```

/*COM ORDENACAO*/
3 references
public async Task<IActionResult> Index(int? id, string ordenacao)
{
    var lista_produtos = _context.Produtos.Include(p => p.IdCategoriaNavigation);

    var lista = lista_produtos.GroupBy(e => e.IdCategoria).
        Select(x => new CategoriaComContagem() {
            IdCategoria = x.Key,
            QuantProdutos = x.Count(),
            Nome = _context.Categorias.
                FirstOrDefault(c => c.IdCategoria == x.Key).NomeCategoria
        });

    ViewData["IdCategoria"] = new SelectList(lista, "IdCategoria", "NomeComQuantidade");

    var opcoesDeOrdenacao = new List<Ordenacao>() {
        new Ordenacao{Valor="ASC", Texto="Preço Ascendente" },
        new Ordenacao{Valor="DESC", Texto="Preço Descendente" }
    };

    ViewData["Ordenacao"] = new SelectList(opcoesDeOrdenacao, "Valor", "Texto");

    //ViewData["IdCategoria"] = new SelectList(_context.Categorias, "IdCategoria", "NomeCategoria");

    // se o utilizador escolher uma categoria
    if (id != null){
        if (ordenacao == "desc")
        {
            return View(await lista_produtos.Where(x => x.IdCategoria == id)
                .OrderByDescending(p => p.Preco).
               ToListAsync());
        }
        else
        {
            return View(await lista_produtos.Where(x => x.IdCategoria == id)
                .OrderBy(p => p.Preco).
                ToListAsync());
        }
    }

    return View(await lista_produtos.Where(x => x.IdCategoria == id).ToListAsync());
}

if (id == null) // se o utilizador não escolher uma categoria
{
    if(ordenacao == "desc")
    {
        return View(await lista_produtos.OrderByDescending(c => c.Preco).ToListAsync());
    }
    else
    {
        return View(await lista_produtos.OrderBy(c => c.Preco).ToListAsync());
    }
}

return View(await lista_produtos.ToListAsync());
}

```

## Referências

- <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/built-in/?view=aspnetcore-3.1>



# Aula 20/12/2021

- Continuação da ficha 6

## Alinea G)

- Podemos fazer a paginação de 2 maneiras
  - Do lado do servidor (c#)
    - Cada coisa vão ser novos pedidos ao servidor e toda a programação é feita por nós
    - `X.PagedList`
  - Do lado do cliente (jquery)

## Usando o X.PagedList

- Ver o video às 3h20 do stor

## Usando o plugin to jquery

- Ver o video as 3h55 do stor

## Ficha 7

- nao ha gets e sets nas classes de metadata

# Aula 03/01/2022

## Ficha 8

- Convem usarmos DB no TP e fazer uma relação entre o ficheiro e a casa em si
- Convem usarmos um ICollection nos modelos em vez de uma List
- Podemos fazer a relação com o Id ou com a classe toda, mas mais vale usar a classe toda

# Aula 04/01/2022

# Autenticação

- Convém criarmos uma classe nova que vai ser a classe dos utilizadores e depois herdar a IdentityUser nessa classe e acrescentarmos o que quisermos à classe dos utilizadores.
  - Temos de mudar algumas coisas e depois criar uma nova migração e atualizar a BD
  - Temos de alterar no `Startup` tudo o que tiver Identity para o nome da nova classe, na view `_loginpartial` também e no `applicationdbcontext`
- Se quisermos alterar as páginas que o Identity fornece(Manage your account,login,register, etc) temos de gerar nós as páginas do Identity
  - Ver o video da aula 9h58PM no pc do stor
- Conseguimos adicionar o `PersonalData` com o data anotations no nosso modelo, para assim dizer que esses dados sao dados do utilizador e estao de acordo com o GPDR e isto é obrigatório

# Autorização

## Diretamente no Controller

- Posso bloquear uma View por exemplo usando o `[Authorize]` antes do `public class CategoriasController:Controller`
  - Isto faz com que todo o acesso a este controller só possa ser acedido a quem tiver feito login

## Para cada método dentro do Controller

- Posso bloquear apenas um método dentro do controler, é só meter o `[Authorize]` antes da definição do método

## Autorização no Controller e nos métodos outras

- Posso usar autorizações globalmente no controller e utilizar por exemplo o `[AllowAnonymous]` num método

## Podemos definir perfis para utilizadores

- Podemos dizer que em todo o meu controller só os utilizadores que tenham o perfil Admin é que podem aceder ao Controller

```
10 [using Microsoft.AspNetCore.Authorization;
11
12 namespace Ficha_08_P5.Controllers
13 {
14     [Authorize(Roles = "Admins")]
15     public class CategoriasController : Controller
16     {
17         private readonly ApplicationDbContext _context;
18
19         public CategoriasController(ApplicationDbContext context)
20         {
21             _context = context;
22         }
23     }
```

- As roles estão guardadas na tabela **AspNetRoles**
- Para criarmos roles
  - ver as 10h45
- Ouvir e apontar o que ele diz as 11h05pm

## Podemos definir políticas

- Mas isto é meio meh