

> Ficha Prática Nº1 (ASP.NET CORE MVC – Introdução)

Notas:

- O ambiente de desenvolvimento selecionado para as aulas práticas é o *Visual Studio 2022*.
- As resoluções das fichas das aulas práticas não são para avaliação e, portanto, não é necessário efetuar qualquer entrega aos professores.

> Preparação do ambiente

1. Faça o download do *Visual Studio*:
 - a. Download disponível em <https://visualstudio.microsoft.com/>;
 - b. A versão “**community**” é o suficiente;
2. Execute o instalador e, nas opções, selecione a opção “*ASP.NET and Web development*” conforme imagem seguinte:

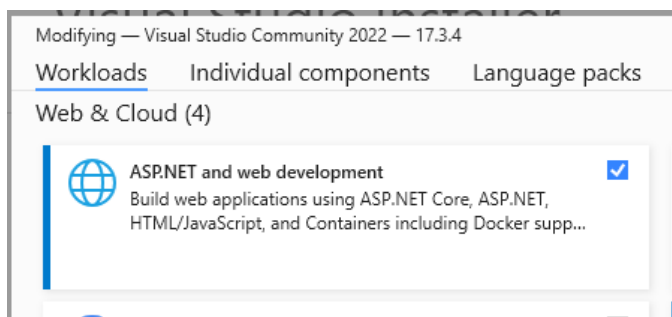


Figura 1- opções de instalação/modificação

3. Caso já tenha o *Visual Studio* instalado, verifique se tem esta opção ativa.
 - a. Para isso execute a aplicação “*Visual Studio Installer*”, clique na opção “*Modificar*” e verifique se a mesma está selecionada. Se não estiver selecionada, selecione e de seguida escolha a opção “*Instalar/Atualizar*”.

> Parte I – Introdução ao ASP.NET Core

ASP.NET Core é uma framework open-source, multi-plataforma, que permite construir aplicações web modernas (cloud-based /cloud-ready).

Nas aulas práticas vamos focar-nos no **ASP.NET Core** em conjunto com o padrão de arquitetura de software **MVC** (Model-View-Controller).

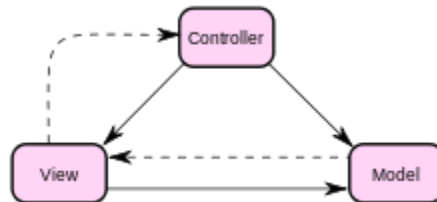


Figura 2 - Padrão MVC (wikipédia)

E, vamos construir uma aplicação web de raiz por forma a abordar as principais características, funcionalidades e possibilidades de desenvolvimento que a framework ASP.NET Core MVC fornece.

No final das aulas práticas os alunos deverão ser capazes de:

- Conhecer a arquitetura cliente servidor em ambiente web.
- Conhecer os fundamentos da ASP .NET Core MVC para criar aplicações web com acesso a base de dados.
- Usar o Entity Framework Core num projeto ASP .NET Core MVC.
- Usar o Identity para implementar mecanismos de segurança (Autenticação e Autorização) num projeto ASP.NET Core MVC.
- Usar os recursos do Bootstrap num projeto ASP .NET Core MVC.

Nas páginas seguintes e nas restantes fichas de laboratório são dados mais detalhes sobre a aplicação a desenvolver.

Documentação:

- **ASP.NET**
[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET](https://learn.microsoft.com/en-us/aspnet)
- **ASP.NET CORE**
[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/INTRODUCTION-TO-ASPNET-CORE?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-6.0)
- **ASP.NET CORE MVC**
[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/TUTORIALS/FIRST-MVC-APP/START-MVC?VIEW=ASPNETCORE-6.0&TABS=VISUAL-STUDIO](https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-6.0&tabs=visual-studio)

> Parte II – Criação de um projeto ASP.NET Core (MVC)

Abra o *Visual Studio 2022*. No menu **Ficheiro**, escolha a opção **Novo » Projeto**.

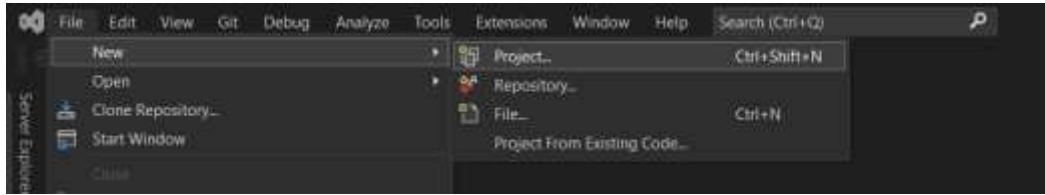


Figura 3 - Novo projeto

De seguida, nos filtros, escolha a linguagem de programação “C#” e nos tipos de projeto “Web”.

Selecione a opção “ASP.NET Core Web App (Model-View-Controller)” e clique em “Avançar”.

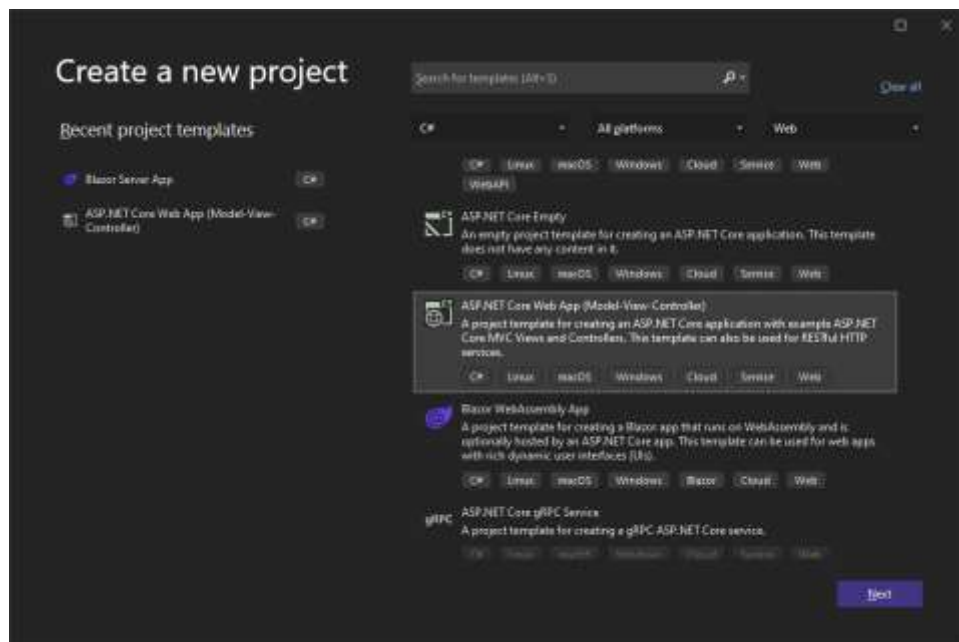
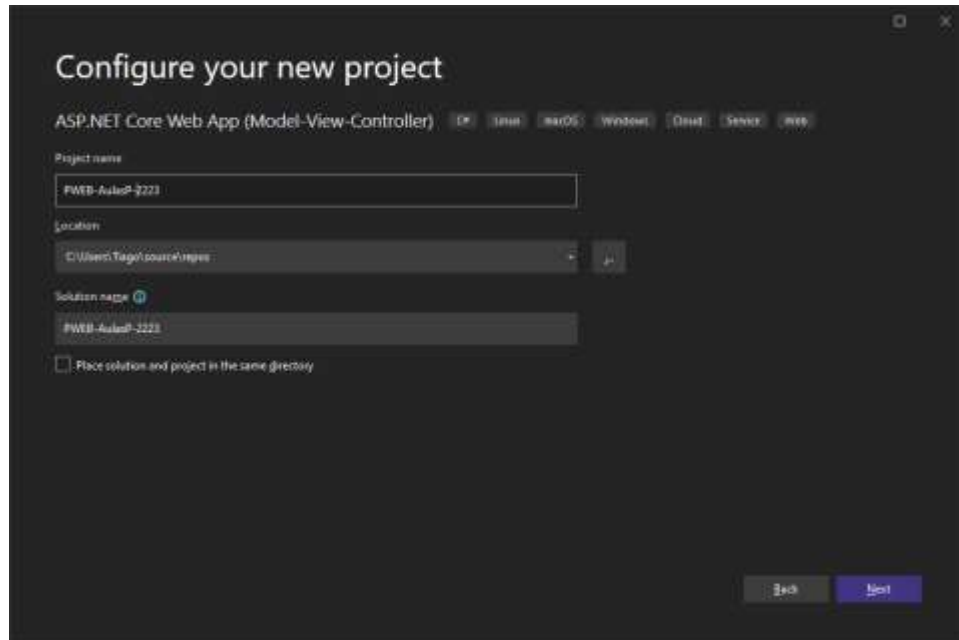


Figura 4 – Novo projeto - tipo de projeto

Na janela seguinte escreva o nome do projeto/solução e escolha o local onde quer guardar o mesmo.

Exemplo: **PWEB-AulasPraticas**



Configure your new project

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

Project name
PWEB-AulaP-2223

Location
C:\Users\Tigo\source\repos

Solution name
PWEB-AulaP-2223

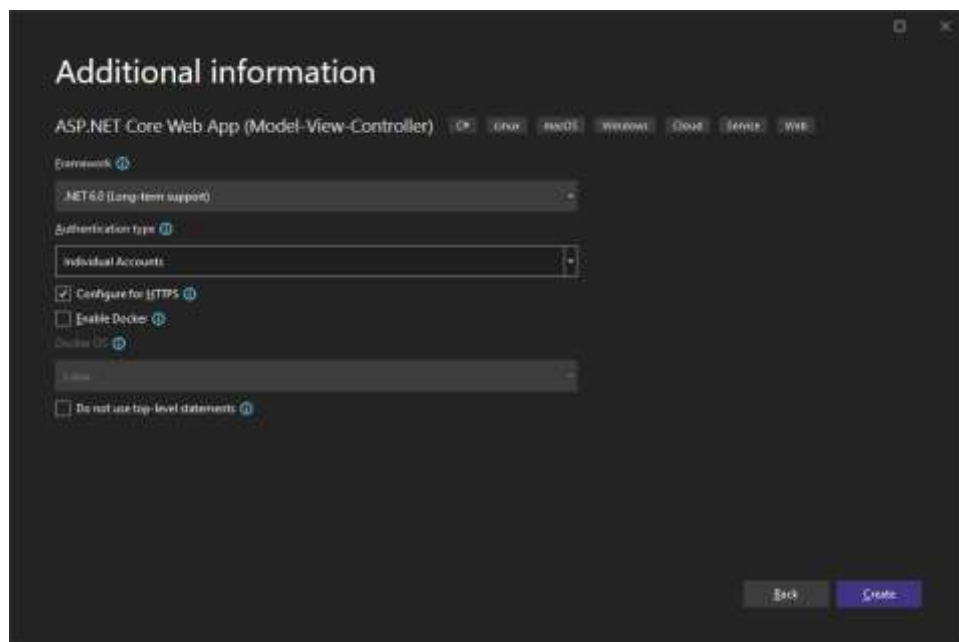
☐ Place solution and project in the same directory

Back Next

Figura 5 - Novo projeto - configuração inicial do projeto

Por fim, na janela seguinte, escolha as seguintes opções:

- Framework = **.NET 6.0 (Long term support)**;
- Autenticação = **Contas individuais**,



Additional information

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

Framework
.NET 6.0 (Long term support)

Authentication type
Individual Accounts

☒ Configure for HTTPS

☐ Enable Docker

Default UI
None

☐ Do not use top-level statements

Back Create

Figura 6 - Novo projeto - informação adicional

e clique no botão criar.

Depois do projeto criado o *Visual Studio* irá ficar com um aspeto semelhante à imagem seguinte, onde podemos ver uma área com acesso à documentação do ASP.NET Core, bem como um explorador da solução e do projeto criado.

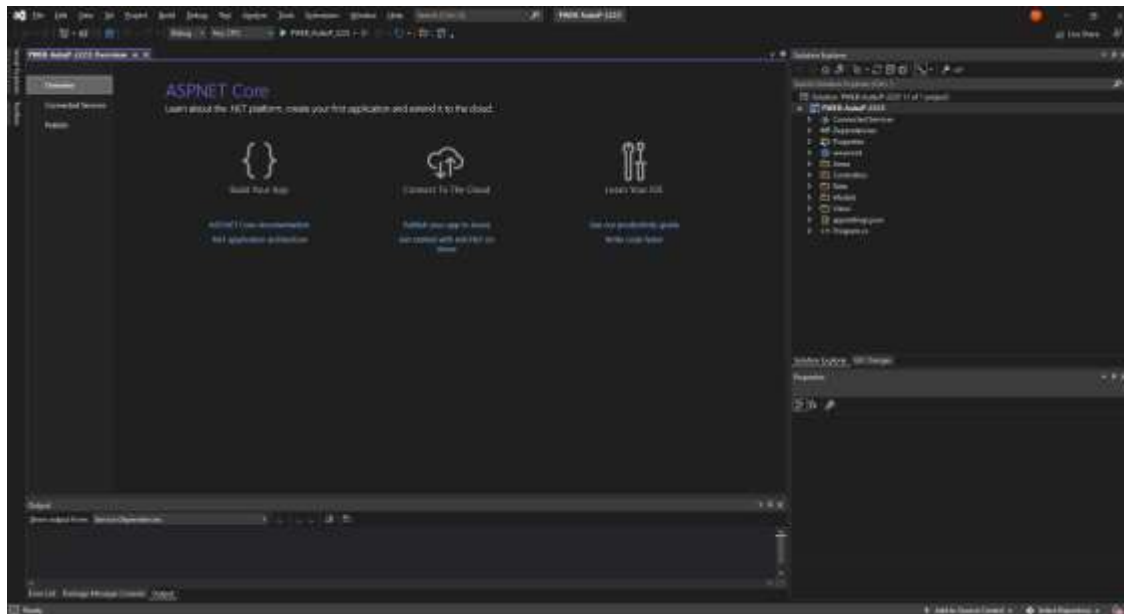


Figura 7 - Novo projeto - projeto criado - ambiente inicial

A estrutura de diretórios e ficheiros de um projeto ASP.NET Core MVC (.NET 6) é a seguinte:

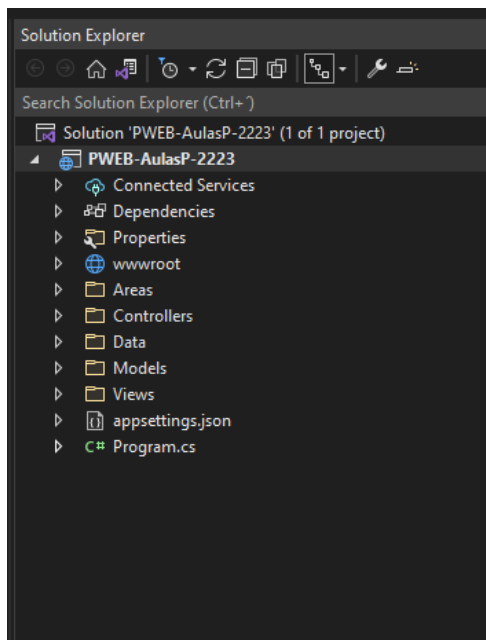


Figura 8 - Estrutura de diretórios

Durante as aulas iremos abordar em maior detalhe esta estrutura, mas resumidamente temos:

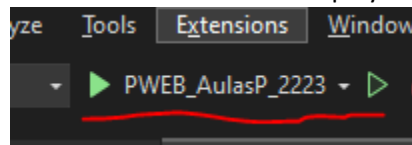
- **Connected Services**
Referências a serviços adicionados ao projeto, como por exemplo, serviços cloud Azure.
- **Dependencies**
Dependências do projeto e referências a pacotes Nuget instalados.
- **Wwwroot**
Ficheiros estáticos necessários ao projecto, CSS, Javascript, Imagens, etc.
- **Areas**
Organizar e particionar uma aplicação em grupos funcionais.
- **Controllers**
Os controllers são responsáveis por “receber” os pedidos do utilizador e pelo processamento dos mesmos. Devem ficar armazenados nesta diretoria.
- **Data**
Contexto de dados, Migrações / controlo versões
- **Models**
Modelos de dados (Classes C#)

- **Views**
Interface do Utilizador (vulgo HTML, mas não só).
- **Appsettings.json**
Configuração da aplicação web. Exemplo: acesso a base de dados, etc.
- **Program.cs**
Inicialização da aplicação. Configuração dos serviços necessários, definição do pipeline de manipulação de pedidos (componentes middleware).

> Parte III – Exercícios introdutórios

>> Execução da aplicação

1. Execute a aplicação web
 - a. Para executar com debug => **F5**
 - b. Para executar sem debug => **CTRL + F5**
 - c. Ou clicar num dos botões “play” no *visual studio*



>> Conceitos: Controllers, Views, Layouts / Templates

2. Altere a mensagem inicial de boas-vindas para:

Bem-vindo a PWEB 2022/23!

3. Altere o conteúdo da View “**Privacy**” para:

<p>Quando utiliza os nossos serviços, está a confiar-nos as suas informações. Compreendemos que é uma enorme responsabilidade e trabalhamos arduamente para proteger as suas informações e dar-lhe o controlo sobre elas.</p>

<p>A presente Política de Privacidade destina-se a ajudá-lo a compreender as informações que recolhemos, o motivo para o fazermos, e como pode atualizar, gerir, exportar e eliminar as suas informações.</p>

4. Altere o texto de copyright para o seu Nome e número de aluno.
5. Crie uma View chamada “**contacts**”:
 - a. No conteúdo da vista escreva “<h3>Brevemente disponível</h3>”
 - b. Adicione um link no menu superior com o texto “**Contacts**” e faça com que ao clicar no link seja mostrada a View “**contacts**”.
6. Adicionar o logotipo do ISEC
 - a. Faça o download do logotipo do ISEC (disponível no site do ISEC).
 - b. Adicione ao projeto o ficheiro PNG/JPG que fez download.
 - c. Insira a imagem no Header da página por forma a substituir o texto e link existente (nome da aplicação criada)

> Parte IV – Estrutura da aplicação web a criar durante as aulas

Durante as aulas práticas iremos desenvolver uma aplicação web cujo objetivo é dar suporte ao negócio de uma “Escola de Condução”. Isto é, vamos construir uma aplicação web que está dividida em diferentes áreas: uma área institucional (apresentação da escola, cursos, contactos, etc.), uma área reservada para os clientes (compra de cursos, agendamento de aulas, acesso a conteúdos exclusivos, etc.), uma área reservada aos formadores (calendário de aulas, alunos, conteúdos, etc.) e uma área de administração da aplicação.

Durante a resolução das próximas fichas iremos abordar detalhadamente cada uma das funcionalidades da aplicação.

Os próximos exercícios visam construir a estrutura base da aplicação – Estrutura “HTML” e modelo de dados.



Figura 9 - Exemplo de layout da aplicação a construir

>> Exercícios

>> Criar uma estrutura HTML semelhante à figura 9

1. Instalar o package NuGet “BootstrapIcons.AspNetCore”

Este package permite ter acesso aos ícons Bootstrap de uma forma simples.
 Exemplos Bootstrap 5.1 : <https://getbootstrap.com/docs/5.1/examples/>
 Mais informação em <https://github.com/Neilvannattu/BootstrapIcons.Net>

2. Edite o ficheiro Views/_ViewImports.cshtml e adicione a seguinte linha:

@addTagHelper *, BootstrapIcons.AspNetCore

3. Faça o download da seguinte folha de estilos e aplique-a no projeto:

- URL: <https://getbootstrap.com/docs/5.1/examples/carousel/carousel.css>
- Guarde o ficheiro “carousel.css” na diretoria “wwwroot\css”
- Edite o ficheiro _Layout.cshtml e inclua o ficheiro carousel.css

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - PWEB Aulas 2022/23</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
  <link rel="stylesheet" href="~/css/carousel.css" asp-append-version="true" />
</head>
```

4. Edite o ficheiro “_Layout.cshtml” e substitua o conteúdo da tag “BODY” por:

```
<body class="d-flex h-100 text-center bg-white p-0 m-0">
  <div class="cover-container d-flex w-100 h-100 p-0 mx-auto flex-column">
    <header class="p-7 text-white">
      <div class="container">
        <nav class="navbar navbar-expand-sm navbar-togglerable-sm navbar-dark mb-3
fixed-top" style="background: rgba(0, 0, 0, .2);">
          <div class="container" >
            <a class="navbar-brand text-warning" href="#"><svg bootstrap-
icon="SignTurnRightFill" class="text-warning" width="48" height="48" aria-
label="SignTurnRightFill"></svg> <strong>PWEB</strong></a>
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarTogglerDemo02" aria-controls="navbarTogglerDemo02" aria-expanded="false"
aria-label="Toggle navigation"><span class="navbar-toggler-icon"></span></button>
            <div class="collapse navbar-collapse" id="navbarTogglerDemo02">
              <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                <li class="nav-item"><a class="nav-link text-white" asp-area=""
asp-controller="Home" asp-action="Index">INÍCIO</a></li>
                <li class="nav-item"><a class="nav-link text-white" asp-area=""
asp-controller="Home" asp-action="Contactos">CONTACTOS </a></li>
              </ul>
              <partial name="_LoginPartial" />
            </div>
          </div>
        </nav>
```



```

        </div>
    </header>

    <main class="px-0 mx-0">
        @RenderBody()
    </main>
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2023 - PWEB Aulas - <a asp-area="" asp-controller="Home" asp-
action="Privacy">Privacy</a>
        </div>
    </footer>
</div>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>

```

5. Crie o menu superior por forma a conter as seguintes opções:

“INÍCIO”
 “QUEM SOMOS”
 “CURSOS”
 “CONTACTOS”

6. Edite a View “Home/Index.cshtml” e substitua o seu conteúdo por:

```

@{
    ViewData["Title"] = "Home Page";
}
<div id="carouselDestaques" class="carousel slide m-0" data-bs-ride="carousel">
    <div class="carousel-indicators">
        <button type="button" data-bs-target="#carouselDestaques" data-bs-slide-to="0"
class="active" aria-label="Slide 1" aria-current="true"></button>
        <button type="button" data-bs-target="#carouselDestaques" data-bs-slide-to="1" aria-
label="Slide 2" class=""></button>
        <button type="button" data-bs-target="#carouselDestaques" data-bs-slide-to="2" aria-
label="Slide 3" class=""></button>
    </div>
    <div class="carousel-inner">
        <div class="carousel-item active banner_01">
            <div class="container">
                <div class="carousel-caption">
                    <h1 class=" text-warning">Conquista a tua liberdade!</h1>
                    <p class=" text-white">Carro, Mota, Camião, tu escolhes, nós ensinamos.</p>
                    <p><a class="btn btn-lg btn-warning " href="#">Junta-te a nós</a></p>
                </div>
            </div>
        </div>
    </div>
</div>

```

```

<div class="carousel-item banner_06">
  <div class="container">
    <div class="carousel-caption text-white">
      <h1>Formação Teórica.</h1>
      <p>
        Acesso exclusivo a conteúdos no nosso portal!<br />
        Todas as questões do IMTT, Testes, Estatísticas, Comunicação com o teu
instrutor.
      </p>
      <p><a class="btn btn-lg btn-success" href="#">Regista-te!</a></p>
    </div>
  </div>
</div>
<div class="carousel-item banner_02">
  <div class="container">
    <div class="carousel-caption text-end">
      <h1>Os melhores instrutores.</h1>
      <p>Temos os melhores instrutores do país.</p>
      <p><a class="btn btn-lg btn-primary" href="#">Conhecer</a></p>
    </div>
  </div>
</div>
<div>
  <button class="carousel-control-prev" type="button" data-bs-target="#carouselDestaques"
data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-target="#carouselDestaques"
data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
<div class="container bg-white">
  <div class="pricing-header p-3 pb-md-4 mx-auto text-center">
    <h1 class="display-4 fw-normal">Cursos & Preços</h1>
    <p class="fs-5 text-muted">Na nossa escola podes aprender a conduzir desde uma scooter
até um avião, consoante a tua vontade e disponibilidade financeira. Temos planos para todas as
bolsas. Podes comprar um curso completo ou podes comprar packs de aulas até estares pronto para
o exame.</p>
  </div>
  <div class="row row-cols-1 row-cols-md-3 mb-3 text-center">
    <div class="col">
      <div class="card mb-4 rounded-3 shadow-sm">
        <div class="card-header py-3">
          <h4 class="my-0 fw-normal">Condução defensiva</h4>
        </div>
        <div class="card-body">
          <h1 class="card-title pricing-card-title">€ 2000<small class="text-muted
fw-light"></small></h1>
          <ul class="list-unstyled mt-3 mb-4">
            <li>Aulas prioritárias</li>
            <li>1 instrutor dedicado</li>
            <li>Automóveis e Motas</li>
            <li>Instrutor virtual</li>
            <li>Centro de apoio online exclusivo</li>
          </ul>
          <button type="button" class="w-100 btn btn-lg btn-outline-primary">Saber
mais</button>
        </div>
      </div>
    </div>
  </div>
</div>

```

```

        <div class="col">
            <div class="card mb-4 rounded-3 shadow-sm">
                <div class="card-header py-3">
                    <h4 class="my-0 fw-normal">Automóvel + Moto</h4>
                </div>
                <div class="card-body">
                    <h1 class="card-title pricing-card-title">€ 1500<small class="text-muted
fw-light"></small></h1>
                    <ul class="list-unstyled mt-3 mb-4">
                        <li>1 instrutor dedicado</li>
                        <li>Veículos novos</li>
                        <li>Centro de apoio online exclusivo</li>
                    </ul>
                    <button type="button" class="w-100 btn btn-lg btn-primary">saber
mais</button>
                </div>
            </div>
        </div>
        <div class="col">
            <div class="card mb-4 rounded-3 shadow-sm border-primary">
                <div class="card-header py-3 text-white bg-primary border-primary">
                    <h4 class="my-0 fw-normal">Automóvel</h4>
                </div>
                <div class="card-body">
                    <h1 class="card-title pricing-card-title">€ 1000<small class="text-muted
fw-light"></small></h1>
                    <ul class="list-unstyled mt-3 mb-4">
                        <li>Aulas partilhadas</li>
                        <li>Veículos partilhados</li>
                        <li>Testes online</li>
                    </ul>
                    <button type="button" class="w-100 btn btn-lg btn-primary">saber
mais</button>
                </div>
            </div>
        </div>
    </div>
</div>

```

7. Pesquise 6 imagens alusivas ao tema (escola de condução), faça o download das mesmas e adicione ao projeto na diretoria “wwwroot\img” com o nome banner_01.jpg banner_02.jpg, etc.

8. Edite o ficheiro de estilos “**site.css**” e adicione as seguintes classes de estilo.

```
.banner_01 {
    background-image: url('/img/banner_01.jpg');
    background-size: cover; background-repeat: no-repeat; background-position: 50% 50%;
}
.banner_02 {
    background-image: url('/img/banner_02.jpg');
    background-size: cover; background-repeat: no-repeat; background-position: 50% 50%;
}
.banner_03 {
    background-image: url('/img/banner_03.jpg');
    background-size: cover; background-repeat: no-repeat; background-position: 50% 50%;
}
.banner_04 {
    background-image: url('/img/banner_04.jpg');
    background-size: cover; background-repeat: no-repeat; background-position: 50% 50%;
}
.banner_05 {
    background-image: url('/img/banner_05.jpg');
    background-size: cover; background-repeat: no-repeat; background-position: 50% 50%;
}
.banner_06 {
    background-image: url('/img/banner_06.jpg');
    background-size: cover; background-repeat: no-repeat; background-position: 50% 50%;
}
```

Execute a aplicação. O resultado deverá ser semelhante à Figura 9 (excetuando as imagens).

9. Faça as alterações necessárias para que cada opção do menu apresente a vista correspondente:
- No menu superior, para cada opção, altere a tag-helper **asp-action** para a ação correspondente.
 - Crie as vistas para cada opção do menu.
 - Crie os métodos necessários no controlador **Home** por forma a obter as vistas pretendidas quando os utilizadores clicam nas opções do menu.

>> Modelo de domínio // Classes em C#

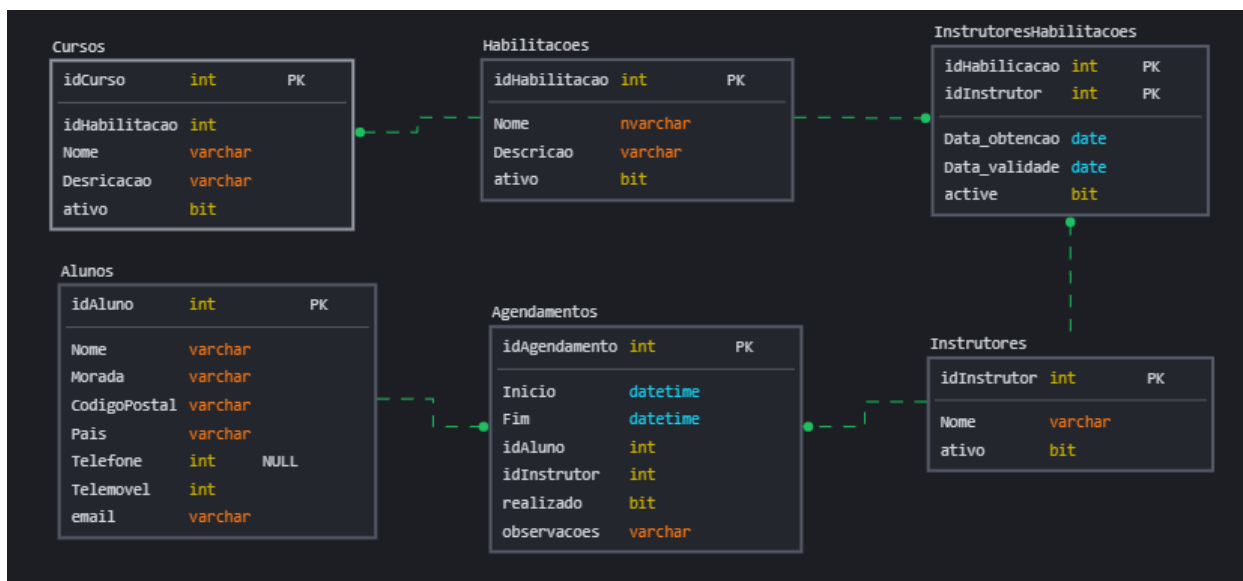
Modelo de domínio - classes c# que representam a lógica do negócio.

Negócio: **Vender cursos (escola de condução)**

Dados que são necessários para implementar a lógica do negócio:

- Cursos - nome, preço, descrição, resumo, imagem, tipo de curso, disponível
- Alunos – nome, idade, morada, email
- Instrutores – nome, idade, etc.

10. Construa as classes c# que representem o seguinte modelo de dados



> Ficha Prática Nº2 - Entity Framework Core

Objetivos:

- Introdução à manipulação de dados com o Entity Framework Core.
- Rever e consolidar os conceitos de Controller, Views e Models
- Introdução aos ViewModels, ViewBag, ViewData
- O que é o _ViewStart e como se define que _Layout utilizar

> Parte I – Conceitos (<https://learn.microsoft.com/en-us/ef/core/>)

>> Entity Framework (EF)

- Conjunto de tecnologias ADO.NET que dão suporte ao desenvolvimento de aplicações orientadas a dados.
- É um ORM (object-relational mapper) – Mapeamento objeto-relacional, que permite trabalhar com dados relacionais na forma de objetos .NET específicos do domínio.
- Foco na lógica do negócio e não no acesso aos dados, ou seja, elimina a maioria do código necessário relativo ao acesso e à manipulação dos dados (T-SQL, PL/SQL, etc.)

>> Entity Framework Core (EF Core)

- Desenvolvida de raiz
- Open source - <https://github.com/dotnet/efcore>
- Multiplatform
- Suporta diferentes tipos de bases de dados - <https://learn.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli>

>> Model

- Na EF o acesso aos dados é realizado através de um Model.
- Um Model é composto por classes (entity classes) e um objeto de contexto que representa a ligação à base de dados.
- Este objeto de contexto permite ler e atualizar dados (inserir, atualizar, apagar) da base de dados.
- A EF Core permite duas abordagens na criação de modelos:
 - Code First – Primeiro criamos os modelos (classes POCO) e depois geramos a base de dados com base nestas classes.
 - Database First – Os modelos (classes) são gerados com base numa base de dados existente.

- Exemplo de um Model:

```
public class Aluno
{
    public int Id { get; set; }
    public string Nome { get; set; }
    public string Morada { get; set; }
    public DateTime DataNascimento { get; set; }
}
```

>> Data Annotations

- Atributos aplicados às classes ou a propriedades das classes.
- Fornecem meta-dados adicionais sobre as classes ou as suas propriedades
- Não são específicos do Entity Framework Core – Fazem parte do .NET Core Framework.
- No ASP.NET MVC Core é possível utilizar estes atributos para validação dos Models
- Dois tipos de atributos:
 - Data Modeling Attributes
 - Validation Related Attributes

>> Chaves Primárias

- O EF depende de cada entidade ter “uma chave” que é usada para identificar a entidade.
- No modelo **Code First** está definida uma convenção - **chaves implícitas**, em que o **Code First** irá procurar uma propriedade chamada “Id”, ou uma combinação do **nome de classe** e “Id”, como “**Cursold**”. Essa propriedade será mapeada para uma coluna do “tipo” chave primária na base de dados.
- Se optar por não seguir a convenção das chaves implícitas então é necessário definir qual a propriedade que corresponde à chave primária através da utilização do atributo `[Key]`.

>> Chaves Compostas

- O EF suporta chaves compostas.
- Neste caso é necessário indicar quais as propriedades que compõem a chave `[Key]` e qual a ordem `[Column(Order=1)]` pela qual a chave composta é formada.

```
public class Passport
{
    [Key]
    [Column(Order=1)]
    public int PassportNumber { get; set; }
    [Key]
    [Column(Order = 2)]
    public string IssuingCountry { get; set; }
    public DateTime Issued { get; set; }
    public DateTime Expires { get; set; }
}
```

> Parte II – Exercícios

- CRIAR MODELOS
- GERAR, ANALISAR E APLICAR MIGRAÇÕES
- ANALISAR A BASE DE DADOS
- GERAR, ANALISAR E MODIFICAR CONTROLLERS E VIEWS QUE TRABALHEM COM OS MODELOS CRIADOS

>> Modelos, Migrações e Base de Dados

1. Crie o Modelo POCO que representa um Curso – com as seguintes propriedades:

- Id – `int`
- Nome – `string`
- Disponível – `bool`

2. Altere o ficheiro de contexto e mapeie este modelo

2.1. Edite o ficheiro `Data\ApplicationDbContext.cs` e adicione o seguinte código:

```
public DbSet<Curso> Cursos{ get; set; }
```

3. Crie a Migração Inicial

3.1. Abra a consola do Package Manager (Package Manager Console) e execute o seguinte comando

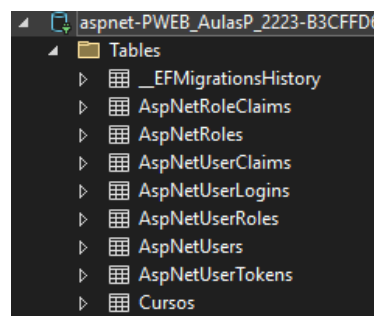
```
PM> Add-Migration MigracaoInicial
```

3.2. Analise o ficheiro (`*_MigracaoInicial.cs`) de migração gerado pelo comando anterior na diretoria `\data\Migrations\`

3.3. Execute o seguinte comando na Consola do Package Manager

```
PM> Update-Database
```

Foi gerada uma base de dados SQL Server com as seguintes tabelas:



__EFMigrationsHistory

histórico de migrações – nome das migrações aplicadas

AspNet*

Tabelas necessárias para o Identity (tema abordado mais tarde)

Cursos

Modelo mapeado na classe de contexto

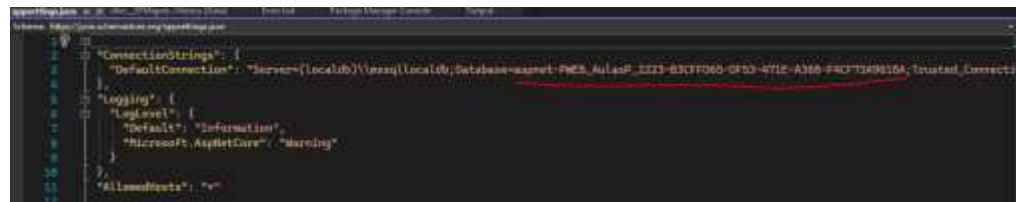
- Abra a base de dados criada e verifique se a estrutura gerada é igual à apresentada no ponto anterior.

Abra o “Server Explorer” e clique no botão – nova ligação e de seguida escolha o ficheiro com a base de dados



Notas:

- O nome da base de dados gerada está definida no ficheiro *appsettings.json*



- A base de dados é criada na home dir do utilizador, por exemplo em sistemas Windows:

c:\users\nome_do_utilizador\

- No exemplo acima o caminho para o ficheiro com a base de dados é:

c:\users\nome_do_utilizador\aspnet-PWEB_AulasP_2223-B3CFFD65-DF53-471E-A368-F4CF73A9618A.mdf

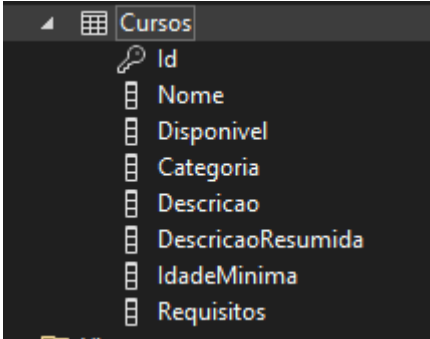
- Adicione os seguintes registos na tabela Cursos:

Id	Nome	Disponivel
1	Categoria AM - Ciclomotor (idade mínima 16 anos)	True
2	Categoria A1 - Motociclo - 11kw/125cc (idade mínima 16 anos)	True
3	Categoria A2 - Motociclo - 35kw (idade mínima 18 anos)	True
4	Categoria A - Motociclo (idade mínima 24 anos)	True

5	Categoria B1 - Quadriciclo (idade mínima 16 anos)	True
6	Categoria A1B1 - Motociclo + Quadriciclo (idade mínima 16 anos)	True
7	Categoria A2B - Ligeiro + Motociclo - 35kw (idade mínima 18 anos)	True
8	Categoria AB - Ligeiro + Motociclo (idade mínima 24 anos)	True
9	Categoria B - Ligeiro Caixa Automática (idade mínima 18 anos)	True
10	Categoria A1B - Motociclo 125cc + Ligeiro (idade mínima 18 anos)	True

6. Altere a classe **Curso** – adicione as seguintes propriedades:
 - Categoria - `string`
 - Descricao - `string`
 - DescricaoResumida - `string`
 - Requisitos - `string`
 - IdadeMinima - `int`
7. Crie uma migração com o comando `Add-Migration`.
8. Analise o ficheiro criado com a migração.
9. Atualize a base de dados com o comando `Update-Database`.
10. Verifique as alterações efetuadas na base de dados.

A tabela de Cursos ficou com as colunas apresentadas na imagem à direita.

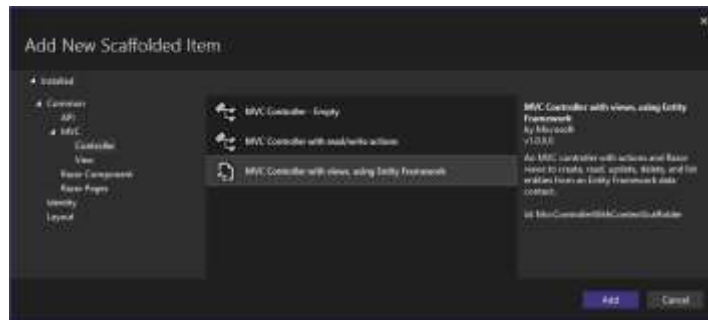


Cursos	
Id	
Nome	
Disponivel	
Categoria	
Descricao	
DescricaoResumida	
IdadeMinima	
Requisitos	

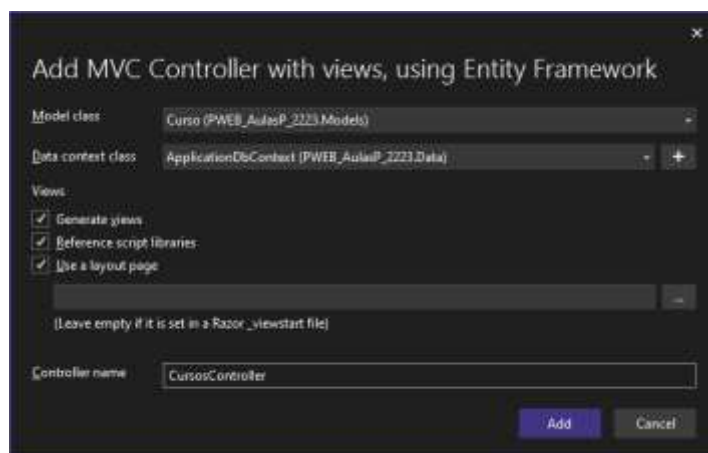
>> Entity Framework & Controllers & Views

11. Crie um Controller e respectivas vistas que permitam realizar as seguintes operações:
 - Listar Cursos
 - Ver detalhes de um Curso
 - Editar um Curso
 - Inserir um Curso
 - Apagar um Curso

Para isso, no explorador da solução, clique com o botão direito em cima do projeto ou da pasta Controllers e escolha a opção “Add new Scaffolded Item” e no ecrã seguinte escolha a opção **MVC » MVC Controller with views, using Entity Framework**, conforme a imagem seguinte:

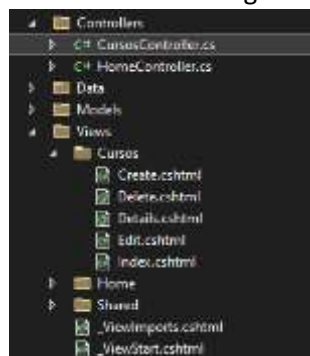


No ecrã seguinte escolhe o Modelo a partir do qual pretende gerar o controller e as views - no nosso caso é o Model Curso – bem como a classe de contexto a ser utilizada.



Altere o nome do controller para “**CursosController**” em vez de “CursosController”.

12. Analise os ficheiros gerados – Views e controller



13. Execute a aplicação e chame a View Index do controller Cursos.



14. Adicione uma entrada no menu superior da aplicação que chame a View do ponto anterior.

Teste o menu e garanta que o link está a funcionar.

15. O template gráfico criado na ficha laboratorial número 1 tem um ligeiro “problema” de layout fazendo com que o conteúdo das views geradas fiquem por baixo do menu superior. Para resolver esse problema:

- Faça download dos ficheiros “_Layout.cshtml” e “_Layout2.cshtml” que estão disponíveis no Moodle.
- Copie estes ficheiros para a pasta “Views\Shared” e substitua o ficheiro de _layout existente.
- Edite todas as vistas “do” Controller Cursos e adicione a seguinte linha de código:

Layout = "~/Views/Shared/_Layout2.cshtml";

```
@{  
    ViewData["Title"] = "Cursos";  
    Layout = "~/Views/Shared/_Layout2.cshtml";  
}
```

16. Analise o controller Cursos

```

namespace PWEB_AulasP_2223.Controllers
{
    // 1 reference
    public class CursosController : Controller
    {
        // 0 references
        private readonly ApplicationDbContext _context;

        // 0 references
        public CursosController(ApplicationDbContext context)
        {
            _context = context;
        }

        // GET: Cursos
        // 3 references
        public async Task<IActionResult> Index()
        {
            return View(await _context.Cursos.ToListAsync());
        }
    }
}

```

Analise com o docente:

- Varável de contexto
- Construtor c/ Injeção de dependência
- Métodos assíncronos
- Views e Models
- `_context.Update()`;
- `SaveChangesAsync()`;
- `ModelState.IsValid`

17. Altere a View **Index** por forma a mostrar uma tabela com a seguinte estrutura


Nome	Disponível	
Categoria A1 - Ciclomotores (idade mínima 16 anos)	Sim	Edit Details Delete
Categoria A1 - Motociclo - 11kw/125cc (idade mínima 16 anos)	Sim	Edit Details Delete
Categoria A2 - Motociclo - 35kw (idade mínima 18 anos)	Sim	Edit Details Delete
Categoria A - Motociclo (idade mínima 24 anos)	Sim	Edit Details Delete
Categoria B1 - Quadriciclo (idade mínima 16 anos)	Sim	Edit Details Delete
Categoria A1B1 - Motociclo + Quadriciclo (idade mínima 16 anos)	Sim	Edit Details Delete
Categoria A2B - Ligero + Motociclo - 35kw (idade mínima 18 anos)	Sim	Edit Details Delete
Categoria A6 - Ligero + Motociclo (idade mínima 24 anos)	Sim	Edit Details Delete
Categoria B - Ligero/Caixa Automática (idade mínima 18 anos)	Sim	Edit Details Delete
Categoria A1B - Motociclo 125cc + Ligero (idade mínima 18 anos)	Sim	Edit Details Delete

Para isso na tag table necessita de adicionar as seguintes classes de estilo
table table-bordered table-striped table-hover

18. Altere o texto dos links “Edite, Details, Delete, Create new” para “Editar, Detalhes, Apagar e Adicionar curso”.

19. Adicione as opções de consulta “Todos”, “Ativos”, “Inativos”, conforme a seguinte imagem:



Nome	Disponível	
Categoria A2 - Motociclo - 35kw (idade mínima 18 anos)	Sim	Edit Details Delete

Estas opções (links) devem ser criadas com a TAG <a> e com recurso aos tag-helpers ASP-ACTION e ASP-ROUTE*

TAG-HELPERS

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/MVC/VIEWS/TAG-HELPERS/INTRO?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro?view=aspnetcore-6.0)

Exemplo:

<a asp-action="Index" asp-route-disponivel="true">Activos

20. Faça as alterações necessárias no controller **Cursos** por forma a que quando o utilizador clica nestes links, só sejam mostrados os cursos que correspondam à escolha selecionada.
21. Faça as alterações necessárias no **controller Home** e na **View Index** deste por forma a substituir os três cursos existentes pela lista de cursos que existem na base de dados e que estão ativos (disponível=true).

Notas:

- Deve manter o mesmo formato de apresentação usado na ficha laboratorial número 1 (formato card).
 - A classe Curso ainda não tem preço pelo que nesta fase vamos construir o card sem o preço.
22. Altere o botão “**saber mais**” do card para redirecionar o utilizador para a View Detalhes do Controller Cursos, passando como parâmetro o Id do curso selecionado.
 23. Altere a classe **Curso** – adicione a seguinte propriedade:
 - Preço – **Decimal?**
 24. Crie uma nova migração e atualize a base de dados.
 25. Faça as alterações necessárias nas views Details, Insert, Edit por forma a contemplar esta nova propriedade.
 26. Faça as alterações necessárias no controller Cursos por forma a que seja possível modificar esta propriedade (inserir, editar).

No final deve ser capaz de inserir um novo curso com preço, editar o preço de um curso, bem como ver todos os detalhes do curso (preço incluído).

>> Exercícios extra

27. Altere a classe Curso – adicione a seguinte propriedade:

- EmDestaque – `bool`

28. Crie uma Migração e atualize a base de dados

29. Modifique o controller Cursos e as respectivas vistas por forma a contemplarem esta nova propriedade.

30. Modifique o comportamento da Página Inicial da aplicação para mostrar apenas os cursos que estejam activos (`disponível=true`) e em destaque (`EmDestaque=true`).

31. Criar um Model que represente uma Categoria de Carta (A, B, C, D, etc.)

- Id – `int`
- Nome – `string`
- Descricao – `string`
- Disponivel – `bool`

32. Altere o ficheiro de contexto e adicione este modelo

33. Crie uma Migração e atualize a base de dados

34. Crie uma propriedade de navegação nas classes Categoria e Curso que permita relacionar ambas (relação 1-N).

- Uma Categoria tem uma lista de cursos
- Um Curso pertence a uma Categoria

> Ficha Prática Nº 3

Objetivos:

- Consolidar conceitos sobre Entity Framework Core.
- Introdução ao LINQ
- Introdução aos Formulários, ViewModels e PartialViews

> Parte I – Conceitos

>> Entity Framework Core – Convenções e Relacionamentos entre entidades

[HTTPS://LEARN.MICROSOFT.COM/EN-US/EF/CORE/MODELING/RELATIONSHIPS](https://learn.microsoft.com/en-us/ef/core/modeling/relationships)

- Por omissão todos os objetos na base de dados são criados no schema dbo.
- A EF Core vai criar tabelas na base de dados para todas entidades que tenham definidos na classe de contexto como propriedade DbSet: “DbSet<TEntity>”.
- A EF Core também cria tabelas na base de dados para entidades que não estejam definidos como DbSet na classe de contexto, mas que sejam referenciadas por outras entidades DbSet.
- Relacionamentos – definições:

Dependent entity	Entidade que contém a chave estrangeira.
Principal entity	Entidade que contém a chave primária / chave alternativa.
Principal key	Propriedades que identificam exclusivamente a entidade principal. Pode ser a chave primária ou uma chave alternativa.
Foreign key	Propriedades na entidade dependente que são usadas para armazenar os valores de chave principal da entidade relacionada
Navigation property	Propriedade definida na entidade principal e/ou dependente que faz referência à entidade relacionada
	<ul style="list-style-type: none">• Collection navigation property: propriedade de navegação que contém referências a muitas entidades relacionadas.• Reference navigation property: propriedade de navegação que contém uma referência a uma única entidade relacionada.• Inverse navigation property: propriedade de navegação na outra extremidade do relacionamento.
Self-referencing relationship	Um relacionamento no qual os tipos de entidade dependente e principal são os mesmos.

- O padrão mais comum para relacionamentos é ter propriedades de navegação definidas em ambas as extremidades do relacionamento e uma propriedade de chave estrangeira definida na classe da entidade dependente.
- No entanto não somos obrigados a definir totalmente as relações. Existem outros padrões/convenções.
- Por exemplo, para definir relacionamentos do tipo 1-N (Um para muitos) podemos usar uma das seguintes convenções:

1. **Convenção 1** - Incluir na entidade dependente uma propriedade de navegação (referência) para a entidade principal.

```
public class Curso{
    public int Id { get; set; }
    public string Nome { get; set; }

    public Categoria categoria { get; set; }
}

public class Categoria{
    public int Id { get; set; }
    public string Nome { get; set; }
}
```

2. **Convenção 2** - Incluir na entidade principal uma propriedade de navegação (coleção).

```
public class Curso{
    public int Id { get; set; }
    public string Nome { get; set; }
}

public class Categoria{
    public int Id { get; set; }
    public string Nome { get; set; }

    public ICollection<Curso> Cursos { get; set; }
}
```

3. **Convenção 3** - Incluir na entidade dependente uma propriedade de navegação (referência) para a entidade principal e incluir na entidade principal uma propriedade de navegação (coleção).

```
public class Curso{
    public int Id { get; set; }
    public string Nome { get; set; }

    public Categoria categoria { get; set; }
}

public class Categoria{
    public int Id { get; set; }
    public string Nome { get; set; }

    public ICollection<Curso> Cursos { get; set; }
}
```

4. **Convenção 4** - Definir a relação na sua totalidade em ambas as entidades, indicando, além as propriedades de navegação, a chave estrangeira na entidade dependente:

```
public class Curso
{
    public int Id { get; set; }
    public string Nome { get; set; }

    public int CategoriaId { get; set; }
    public Categoria categoria { get; set; }
}

public class Categoria
{
    public int Id { get; set; }
    public string Nome { get; set; }

    public ICollection<Curso> Cursos { get; set; }
}
```

Nota: Existem outros tipos de relacionamentos entre entidades que irão ser abordados mais tarde.

>> Entity Framework Core (EF Core) e LINQ

[HTTPS://LEARN.MICROSOFT.COM/EN-US/EF/CORE/QUERYING/](https://learn.microsoft.com/en-us/ef/core/querying/)

[HTTPS://LEARN.MICROSOFT.COM/EN-US/DOTNET/CSHARP/PROGRAMMING-GUIDE/CONCEPTS/LINQ/BASIC-LINQ-QUERY-OPERATIONS](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations)

- Entity Framework Core utiliza LINQ (Language-Integrated Query) para realizar consultas na base de dados.
- LINQ permite escrever consultas fortemente tipificadas em C# (ou outra linguagem .NET).
- Utiliza o objeto de contexto e as classes para referenciar os objetos na base de dados
- EF Core envia uma representação do LINQ para o provedor de base de dados, que por sua vez, o transforma em código específico para o motor de base de dados.
- As consultas são sempre executadas sobre a base de dados independentemente de as entidades em causa já existirem no contexto.

>> LINQ

- Sintaxe uniforme para obter dados de diferentes fontes e formatos. Elimina as diferenças entre linguagens de programação e bases de dados.
- Fornece uma única interface de consulta para diferentes tipos de fontes de dados.
- Existem duas formas de escrever consultas: Query Syntax & Method Syntax
 - **Query Syntax**
 - Similar a código SQL;
 - Inicia com o “From” e termina com “End”

- **Method Syntax**
 - Também conhecido como Fluent Syntax utiliza extension methods que estão incluídos nas classes estáticas *Enumerable* or *Queryable*.
 - São usados de forma semelhante a outro qualquer extension method de uma classe.
- O Compilador converte o código escrito em Query Syntax para Method Syntax no momento da compilação.
- Query syntax e method syntax são idênticos. A escolha entre utilizar um ou outro é pessoal e normalmente relacionada com o facto algumas pessoas acharem o Syntax Query mais fácil de ler/perceber.
- Algumas consultas têm obrigatoriamente de ser efetuadas com recurso a Extension Methods, como por exemplo uma consulta de devolve o número de elementos que correspondem a uma determinada condição.
- Exemplo prático:

```
// “fonte” de dados:
IList<string> disciplinas = new List<string>() {
    "Introdução à Programação",
    "Algoritmos",
    "Programação Orientada a Objectos",
    "Linguagens de Script",
    "Programação Web"
};

// Consulta LINQ com Query Syntax
var result = from s in disciplinas
              where s.Contains("Programação")
              select s;

// Consulta LINQ com Method Syntax
var result = disciplinas.Where(d => d.Contains("Programação"));
```

>> Formulários em ASP.NET Core

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/MVC/VIEWS/WORKING-WITH-FORMS?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-6.0)

- Os formulários são uma forma de enviar dados do cliente para o servidor.
- São construídos com a tag HTML “<form>”

```
<form action="/students/search" method="post">
  <label for="fname">Student name:</label>
  <input type="text" id="fname" name="fname" value="PWEB Student">
  <input type="submit" value="Search">
</form>
```

- Em ASP.NET Core podem ser construídos de duas formas:
 - Através da **Tag-Helper** “<form>”
 - Através do **Html Helper** “Html.BeginForm”
- Fornecem um mecanismo de prevenção de ataques [Cross Site Request Forgery](#) (CSRF) desde que se utilize o atributo [ValidateAntiForgeryToken] no método chamado no HTTP Post (atributo action da tag form).

- A criação do token de verificação é automática quando se usa o Tag-Helper <form>.
- Quando se usa o HTML Helper é necessário forçar a sua geração através do código:

```
Html.AntiForgeryToken();
```

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/SECURITY/ANTI-REQUEST-FORGERY?SOURCE=RECOMMENDATIONS&VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery?source=recommendations&view=aspnetcore-6.0)

- Em conjunto com um Modelo (quando a View define um @model) podemos tirar partido das tag-helpers para construir os formulários de uma forma simples e eficiente, pois:
 - Os atributos **id** e **name** são gerados automaticamente de acordo com a propriedade no modelo identificada em **asp-for**.
 - O atributo **type**, na tag input é gerado consoante o tipo de dados da propriedade, e os *data annotations* (se existirem), no modelo identificada em **asp-for**.
 - Gera atributos de validação HTML 5 consoante os atributos (*data annotations*) definidos no modelo.

TIPO DE DADOS // TIPO DE INPUT

Tipo .NET	Tipo de entrada
Bool	type="checkbox"
String	type="text"
Datetime	type=" datetime-local "

ATRIBUTO NO MODELO COM DATA ANNOTATIONS

Atributo	Tipo de entrada
[EmailAddress]	type="email"
[Url]	type="url"

Byte	type="number"
int	type="number"
Single e Double	type="number"

[HiddenInput]	type="hidden"
[Phone]	type="tel"
[DataType(DataType.Password)]	type="password"
[DataType(DataType.Date)]	type="date"
[DataType(DataType.Time)]	type="time"

Exemplo de um formulário em ASP.NET Core construído com as tag-helper “<form>”, “asp-action”, “asp-controller”, “asp-for” e um Modelo:

MODEL

```
using System.ComponentModel.DataAnnotations;
namespace PWEB.demos
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email Address")]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

VIEW (CSHTML)

```
@model RegisterViewModel

<form asp-controller="Demo" asp-action="RegisterInput" method="post">
    Email: <input asp-for="Email" /> <br />
    Password: <input asp-for="Password" /><br />
    <button type="submit">Register</button>
</form>
```

OUTPUT ENVIADO PARA O CLIENTE – CÓDIGO HTML GERADO

```
<form method="post" action="/demo/RegisterInput">
    Email:
    <input type="email" data-val="true"
        data-val-email="The Email Address field is not a valid email address."
        data-val-required="The Email Address field is required."
        id="Email" name="Email" value=""><br />
    Password:
    <input type="password" data-val="true"
        data-val-required="The Password field is required."
        id="Password" name="Password"><br />
    <button type="submit">Register</button>
    <input name="__RequestVerificationToken" type="hidden" value="*****">
</form>
```

Reparem que os atributos, id, name, type e data-val* foram gerados de acordo com o modelo de dados.

>> Passar (enviar) dados de um Controller para uma View

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/MVC/VIEWS/OVERVIEW?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-6.0)

- Existem duas formas de enviar dados de um controller para uma view:
 - Strongly typed data** - Através de um *ViewModel* (@model)
 - Este é o método mais robusto.
 - O Model é especificado na View através da utilização da diretiva @model.
 - Normalmente é referenciado como sendo um *viewModel*.
 - O método responsável por mostrar a View tem de passar para esta um objeto do tipo especificado.
 - Exemplo:*

CSHTML

```
@model AddressViewModel

<h1>@ViewBag.Mensagem</h1>

<address>
    @Model.Name<br>
    @Model.Street<br>
    @Model.PostalCode @Model.City
</address>
```

C#

```
public IActionResult Contact()
{
    var viewModel = new AddressViewModel ()
    {
        Name = "ISEC/DEIS",
        Street = "R. Pedro Nunes",
        City = "Coimbra",
        PostalCode = "3030-199"
    };
    return View(viewModel);
}
```

- Weakly typed data** - Utilizando uma das seguintes coleções de dados:
 - ViewData / ViewData Attribute**
 - ViewData é um objeto ViewDataDictionary acessível por meio de chaves do tipo string.
 - Os dados do tipo string podem ser armazenados e usados diretamente sem a necessidade de serem convertidos.
 - Outro tipo de dados/objetos necessitam de conversão para o tipo específico antes de serem utilizados.
 - ViewBag**
 - ViewBag é um objeto que fornece acesso dinâmico aos objetos armazenados no ViewData.
 - Pode ser mais conveniente para trabalhar, já que não requer casting.

▪ Exemplo:

CSHTML

```
<h1>@ViewBag.Mensagem</h1>

<address>
    @ViewBag.Address.Name<br>
    @ViewBag.Address.Street<br>
    @ViewBag.Address.PostalCode @ViewBag.Address.City
</address>
```

C#

```
public IActionResult SomeAction()
{
    ViewBag.Mensagem = "Mensagem de teste";
    ViewBag.Address = new Address()
    {
        Name = "ISEC/DEIS",
        Street = "R. Pedro Nunes",
        City = "Coimbra",
        PostalCode = "3030-199"
    };

    return View();
}
```

> Parte II – Exercícios

- CRIAR FORMULÁRIOS
- USAR DATA ANNOTATIONS NO MODELO
- CRIAR UMA VISTA PARCIAL (PARTIAL VIEW)
- CRIAR A ENTIDADE CATEGORIA E RELACIONAR COM A ENTIDADE CURSO
- CRIAR CONTROLLER E VIEWS PARA GERIR A ENTIDADE CATEGORIA
- MODIFICAR AS VIEWS DO CONTROLLER CURSO PARA CONTEMPLAR A NOVA ENTIDADE (RELACIONAMENTO)
- PASSAR DADOS DO CONTROLLER PARA A VIEW ATRAVÉS DO VIEWBAG E VIEWDATA

>> Formulário de pesquisa de cursos

1. Na view **Index** do **Controller** Cursos crie um formulário, com recurso à Tag-Helper `<form>`, que permita ao utilizador pesquisar um curso (nome ou descrição) escrevendo um conjunto de caracteres numa caixa de texto.

2. Crie um método **Index** [*HttpPost*] que receba os dados do formulário e que devolva o resultado da pesquisa.
 - Faça a pesquisa com recurso ao *LINQ Query Syntax*.
3. Crie uma cópia da **View Index.cshtml** e renomeie para **Search.cshtml** (Cursos).
4. Altere o formulário na View **Search** para usar o método **GET** em vez do método POST.
5. Crie um método **Search** [*HttpPost*] no controller **Cursos** que devolva o resultado da pesquisa para a view **Search**.
 - Faça a pesquisa com recurso ao *LINQ Method Syntax*.
6. Veja a diferença entre um formulário GET e um formulário POST.
 - No browser, abra as **ferramentas de desenvolvimento**. Escolha o separador “**Rede**”.

- Efetue pesquisas num formulário e noutro (*View Index* e *View Search*) e veja as diferenças nos pedidos que são efetuados ao servidor.
7. Crie um *ViewModel* com o nome “*PesquisaCursoViewModel.cs*” com três propriedades:
- ListaDeCursos (**List<Curso>**).
 - NumResultados - n.º de cursos devolvidos pela pesquisa (**int**)
 - TextoAPesquisar - Texto introduzido pelo utilizador (**string**)
8. Faça as alterações necessárias na **View Search** e no **Controller** para utilizar o **viewModel** criado no ponto anterior:

- Altere o método do formulário para **POST**.
- Altere o formulário por forma a fazer uso de **tag-helpers** de acordo com o Modelo de dados (*ver exemplo apresentado nos conceitos*).
- Crie um método **search** [*HttpPost*] para tratar os dados enviados pelo formulário.
- Defina como argumento deste método um objeto do tipo do **ViewModel PesquisaCursoViewModel** e faça o **Bind** da propriedade **TextoAPesquisar**.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Search([Bind("textoAPesquisar")]
PesquisaCursoViewModel pesquisaCurso)
```

- Este método deve devolver para a View um objeto do tipo **PesquisaCursoViewModel** com: Lista de Cursos, Texto pesquisado e número de cursos encontrados pela pesquisa.

```
Return View(pesquisaCurso)

//as propriedade ListaDeCursos e NumResultados devem ser preenchidos
// antes de ser feito o return
```

9. Adicione o seguinte atributo à propriedade **TextoAPesquisar**:

```
[Display(Name = "Texto", Prompt = "introduza o texto a pesquisar")]
```

- Verifique as alterações provocadas na visualização do formulário, nomeadamente o conteúdo do **Label** e o **Placeholder** do input **TextoAPesquisar**.

>> Nova entidade **Categoria**

10. **Remova** a *propriedade* **Categoria (string)** da *entidade* **Curso**.
11. **Crie** uma *entidade* **Categoria** (*nova classe categoria.cs*) que represente uma Categoria de Carta (A, B, C, D, etc.) com as seguintes propriedades:
 - Id – int
 - Nome – string
 - Descricao – string
 - Disponivel – bool
12. **Crie** um relacionamento do tipo **1-N** entre a entidade **Categoria** e a entidade **Curso**.
13. **Adicione** a entidade **Categoria** à classe de contexto de acesso à base de dados.
14. **Adicione** uma **migração** e **atualize** a base de dados.

Nota:

É normal que o comando **update-database** dê erro por causa da chave estrangeira CategoriaId na entidade Curso.

Isto porque, já existem registos na tabela cursos e não podemos adicionar uma coluna nova (propriedade CategoriaId da classe curso) que é obrigatória e ao mesmo tempo uma chave estrangeira para a tabela Categorias (que ainda não tem registos).

Podem resolver este erro eliminando os registos da tabela cursos ou então adicionando o atributo nullable “?” na definição da propriedade CategoriaId na entidade Curso (para o caso de não quererem apagar os registos na tabela curso).

15. Crie o controller **CategoriasController.cs**, com Views, utilizando a **Entity Framework**, por forma a poder realizar as operações “CRUD” sobre esta entidade.
 - CRUD – Create, Read, Update, Delete – *Operações standard efetuadas sobre uma entidade numa base de dados.*
 - O *Visual Studio* fornece um mecanismo de criar automaticamente o código necessário (Controller e Views) para realizar estas operações.
16. Modifique as **Views** e os **métodos** do controller **CursosController** por forma a:
 - Mostrar a Categoria na lista de cursos.
 - Mostrar a Categoria nos detalhes do curso.
 - Escolher uma Categoria quando se cria um curso.
 - Escolher uma Categoria quando se edita um curso.
 - Nota:
 - Não vai ser possível criar/editar um curso devido à validação do Modelo.

- Para resolver a questão temporariamente, sem ser necessário criar um *ViewModel* específico, faça a seguinte alteração nos métodos **Edit** e **Create** [*HttpPost*]:

```
// remover a propriedade de navegação da validação do modelo
// adicionar a linha abaixo antes do If (ModelState.IsValid)
ModelState.Remove(nameof(curso.categoria));

// manter o restante código existente:
if (ModelState.IsValid)
{
    _context.Add(curso);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

17. Modifique a pesquisa (view search) por forma a ser possível pesquisar por categoria e por nome/descrição.

>> Vista Parcial

18. Crie um Partial View (**QuickSearchPartial.cshtml**) com um formulário de pesquisa para criar uma pesquisa rápida no site.
19. Adicione a Partial View ao ficheiro de `_layout` por forma a obter um resultado semelhante a:



> Ficha Prática Nº 4

Objetivos:

- Consolidar conceitos sobre Entity Framework Core, LINQ, Formulários, Models e ViewModels

> Exercícios

- CRIAR UMA PARTIAL VIEW COM FORMULÁRIO DE PESQUISA
- MODIFICAR VISTAS DE PESQUISA, DETALHES DO CURSO, EDITAR CURSO E CRIAR CURSO
- MODIFICAR VISTAS DETALHES DA CATEGORIA, EDITAR CATEGORIA E CRIAR CATEGORIA
- CRIAR FORMULÁRIO “TEMPORÁRIO” PARA AGENDAMENTO DE AULAS

>> Vista Parcial & Pesquisa – continuação da ficha 3

1. Crie um Partial View (**QuickSearchPartial.cshtml**) com um formulário de pesquisa para criar uma pesquisa rápida no site.
 - O formulário deve submeter os dados através do método GET para o *Controller Cursos*, método *Search*.
2. Adicione a Partial View aos dois ficheiros de _layout por forma a obter um resultado semelhante a:



- Para isso, procure no ficheiro de Layout o código: “<partial name=“_LoginPartial” />” e na linha anterior adicione o seguinte código:

```
<partial name="~/Views/Cursos/QuickSearchPartial.cshtml" />
```

Nota: se a *Partial View* for criada na diretoria “/Views/Shared/” não é necessário escrever o caminho completo, bastando escrever o nome.

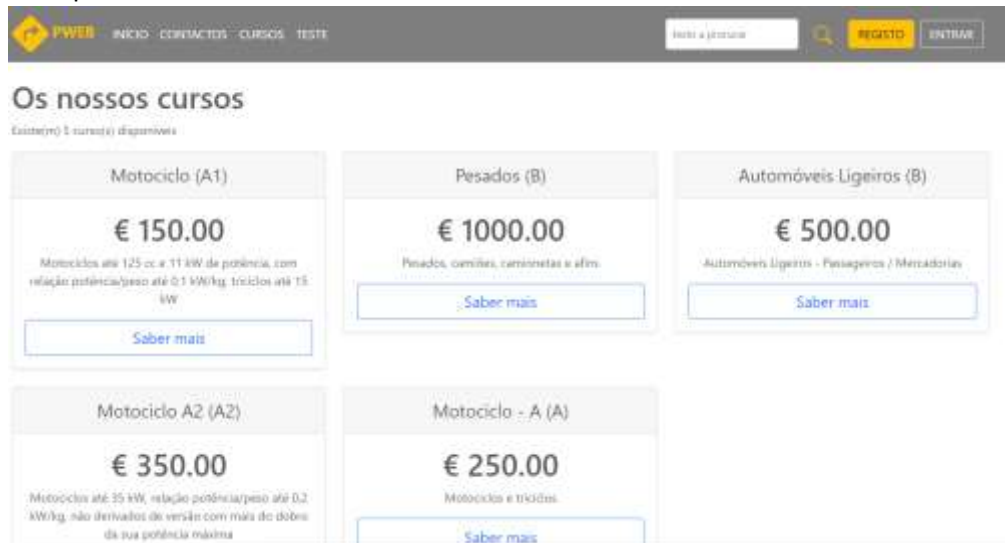
3. Faça as alterações necessárias para:

- No caso de a pesquisa não devolver resultados deve mostrar ao utilizador a seguinte mensagem:
 - “Não foi possível encontrar cursos com os dados introduzidos. Por favor, reveja a sua pesquisa”.



- Mostrar a lista de resultados em formato “card”, com os seguintes elementos:
 - Nome do Curso (Nome Categoria)
 - Descrição Resumida
 - Preço
 - Botão “saber mais” – redirecionar para os detalhes de um curso.

Exemplo:



4. Faça as alterações necessárias para:

- Se o nome do curso ou o resumo da descrição contiver o texto pesquisado, colocar esse pedaço de texto com fundo amarelo.

Exemplo:



>> Formulários

5. Altere as Views (*Edit*, *Create*) do Cursos por forma a utilizar a tag “<TextArea>” para as propriedades “Descricao”, “DescricaoResumida” e Requisitos em vez da tag “<input>”.
6. Altere a *view Details* do *controller* Cursos de forma a mostrar os detalhes de um curso de uma forma mais “apelativa”.

Nota: Veja exemplos em <https://getbootstrap.com/>.

7. Adicione *Data Annotations* às propriedades da classe “Curso.cs” de forma que o preenchimento dos formulários e a visualização das propriedades, seja melhorada através da geração da descrição da propriedade, do placeholder e da geração dos atributos de validação, se necessários.

Todas as propriedades devem, no mínimo, ter o atributo [Display] definido.

Exemplo:

```
[Display(Name = "Idade mínima", Prompt = "Introduza a idade Mínima", Description = "Idade mínima para poder frequentar esta formação")]
[Range(14, 100, ErrorMessage = "Mínimo: 14 anos, máximo: 100 anos")]
public int IdadeMinima { get; set; }
```

8. Faça o mesmo (exercícios 5, 6 e 7) para as Categorias.

9. Adicione um item no menu superior que “aponte” para a View “Index” do controller “Categorias”.
10. Criar um formulário de “agendamento” de aulas, que permita calcular o custo do agendamento.
 - Criar um Modelo novo chamado “TipoDeAula”, com as seguintes propriedades
 - Id - `int`
 - Nome - `String`
 - ValorHora - `Decimal`
 - Crie um Modelo novo chamado “Agendamento”, com as seguintes propriedades
 - Id - `int`
 - Cliente - `String`
 - DataInicio - `DateTime`
 - DataFim - `DateTime`
 - DuracaoEmHoras - `int`
 - DuracaoEmMinutos - `int`
 - Preco – `Decimal`
 - DataHoraDoPedido – `DateTime` – representa a data e Hora em que o Utilizador efetuou o cálculo.
 - Crie um formulário de agendamento, que permita ao utilizador escolher duas datas, e que consoante a escolha do utilizador, calcule o número de horas, bem como o custo desse agendamento.
 - Efetue as alterações necessárias para guardar na base de dados os pedidos de agendamento efetuados.

> Ficha Prática Nº 5

Objetivos:

- Introdução à segurança em aplicações web
- Introdução à segurança em aplicações web ASP.NET Core
- Introdução ao Microsoft Identity Core

> Parte I – Conceitos

>> Segurança em Sistemas de Informação

[HTTPS://PT.WIKIPEDIA.ORG/WIKI/SEGURAN%C3%A7A_DA_INFORMA%C3%A7%C3%A3o](https://pt.wikipedia.org/wiki/Seguran%C3%A7a_da_informa%C3%A7%C3%A3o)

A definição mais formal de segurança de sistemas de informação é o ato ou prática de proteger os sistemas de informação contra acesso, uso, modificação, destruição ou interrupção não autorizados.

Esta definição está, desde os anos 60, associada à confidencialidade, integridade e disponibilidade, também chamada como tríade da CIA (confidentiality, integrity, and availability - 3 pilares da segurança da informação). Mais recentemente e, devido à evolução e complexidade dos sistemas de informação/tecnologias/meios, associou-se um outro conceito, a autenticidade da informação.

Podemos então dizer que a segurança da informação tem como principais objetivos garantir os níveis adequados de integridade, autenticidade, disponibilidade e confidencialidade, por forma a mitigar o impacto de eventuais incidentes que possam comprometer o regular funcionamento dos sistemas de informação.

De uma forma resumida os 4 princípios da segurança em Sistemas de informação são:

- **Confidencialidade** - capacidade de proteger os dados daqueles que não estão autorizados a consultá-los.
- **Integridade** - capacidade de prevenir, recuperar e reverter alterações não autorizadas ou acidentais aos dados.
- **Disponibilidade** - Possibilidade de acesso aos dados de pessoas autorizadas e em tempo útil.
- **Autenticidade** - manutenção da fiabilidade da informação desde o momento da sua produção e ao longo de todo o seu ciclo de vida.

>> Segurança de aplicações web

[HTTPS://WWW.RFC-EDITOR.ORG/RF/RF9110.HTML#NAME-SECURITY-CONSIDERATIONS](https://www.rfc-editor.org/rfc/rfc9110.html#name-security-considerations)

[HTTPS://OWASP.ORG/](https://owasp.org/)

[HTTPS://DEVELOPER.MOZILLA.ORG/PT-BR/DOCS/LEARN/SERVER-SIDE/FIRST_STEPS/WEBSITE_SECURITY](https://developer.mozilla.org/pt-br/docs/learn/server-side/first_steps/website_security)

[HTTPS://WWW.CNCS.GOV.PT/PT/REGIME-JURIDICO/](https://www.cncs.gov.pt/pt/regime-juridico/)

Alguns dos tipos de ataque mais comuns:

- Cross site scripting (XSS) — vulnerabilidade que permite que um atacante injete scripts do lado do cliente numa página da internet para aceder a informações importantes diretamente, personificar o utilizador ou induzir o utilizador a revelar informações importantes.
- SQL injection (SQi) — injeção de SQL é um método pelo qual um atacante explora vulnerabilidades na maneira como a aplicação executa operações numa base de dados.
- Ataques de negação de serviço (DoS) e ataques de negação de serviço distribuída (DDoS) — por meio de uma variedade de vetores, os invasores conseguem sobrecarregar o servidor afetado ou sua infraestrutura circundante com diferentes tipos de tráfego de ataque. Quando um servidor perde a capacidade de processar com eficácia as solicitações recebidas, começa a comportar-se com lentidão e, eventualmente, passa a não conseguir responder / negar serviço às solicitações recebidas.
- Corrupção de memória — a corrupção da memória ocorre quando um local na memória é acidentalmente modificado, resultando em um possível comportamento inesperado do software. Os agentes mal-intencionados tentarão farejar e explorar a corrupção da memória por meio de ataques como injeções de código ou buffer overflow.
- Buffer overflow — o é uma anomalia que ocorre quando o software grava dados num espaço definido na memória conhecido como buffer. Se a capacidade do buffer for excedida, isso faz com que os dados em locais de memória adjacentes sejam substituídos. Esse comportamento pode ser explorado para injetar códigos maliciosos na memória, possivelmente criando uma vulnerabilidade na máquina afetada.
- Falsificação de solicitações entre sites (CSRF) — a falsificação de solicitações entre sites tem como objetivo induzir uma vítima a fazer uma solicitação que utiliza sua autenticação ou autorização. Ao tirar proveito dos privilégios de conta de um utilizador, um atacante é capaz de enviar uma solicitação como se fosse o utilizador. Após comprometer a conta de um utilizador, o invasor consegue extrair, destruir ou alterar informações importantes. As contas com muitos privilégios, como as de administradores ou executivos, são as mais frequentemente atacadas.
- Violação de dados — diferentemente dos vetores de ataque específicos, "violação de dados" é um termo generalizado que se refere à liberação de informações sensíveis ou confidenciais, que pode ocorrer por meio de ações mal-intencionadas ou por engano. O âmbito daquilo que

é considerado como uma violação de dados é bastante amplo e pode incluir desde alguns poucos registos, altamente valiosos, até milhões de contas de utilizadores expostas.

Top 10 Web Application Security Risks (2021 OWASP)

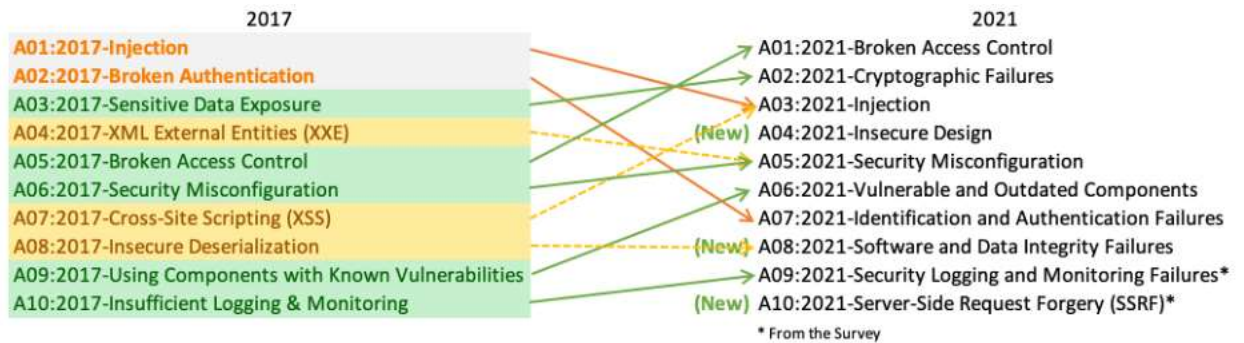


Figura 1 - Falhas de segurança mais comuns no ano 2021 – comparativo a 2017

Fonte: <https://owasp.org/www-project-top-ten/>

Como implementar os pilares da Segurança de SI em aplicações web?

- Confidencialidade
- Integridade
- Disponibilidade
- Autenticidade

>> Microsoft Identity Core

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/SECURITY/?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/security/?view=aspnetcore-6.0)

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/SECURITY/AUTHENTICATION/IDENTITY?VIEW=ASPNETCORE-6.0&TABS=VISUAL-STUDIO](https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio)

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/SECURITY/AUTHORIZATION/ROLES?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-6.0)

- É uma API que dá suporte à funcionalidade de logon da interface do utilizador.
- Permite gerir utilizadores, senhas, dados de perfil, funções, declarações, tokens, confirmação de email e muito mais.
- Normalmente é configurado utilizando uma base de dados SQL Server para armazenar nomes de utilizador, senhas e dados de perfil. Como alternativa, podem ser utilizados outros repositórios persistentes, como por exemplo, o Azure, MariaDB, etc..

> Exercícios

- IMPLEMENTAR MICROSOFT IDENTITY CORE NO PROJECTO
- CRIAR / MODIFICAR FORMULÁRIOS DE LOGIN E REGISTO DOS UTILIZADORES
- PERSONALIZAR A CLASSE IDENTITYUSER
- MARCAR PROPRIEDADES COMO PERSONALDATA
- ASSOCIAR UM AGENDAMENTO A UM UTILIZADOR

>> Microsoft ASP.Net Identity Core – Configuração Inicial

1. Analise com o docente as configurações do projeto por forma a verificar a configuração do Microsoft Identity Core.
2. Crie um utilizador através do formulário de registo de utilizadores disponibilizado.
3. Verifique na base de dados, nas tabelas relativas ao *Microsoft Identity - “aspnet*”*, os registos adicionados.
4. Faça login do site com os dados do utilizador que criou no ponto 2.
5. Verifique as alterações no menu superior.
6. Analise o conteúdo do ficheiro “*_loginPartial.cshtml*”.
7. Navegue pelas opções disponibilizadas (quando clica no nome de utilizador no menu superior).
8. Pesquise no projeto pelas “páginas” responsáveis pela geração dos formulários de **Login**, **Registo** e **MyAccount**.

>> Microsoft ASP.Net Identity Core – Personalização do Modelo IdentityUser

8. Crie uma classe chamada **ApplicationUser** que herda a classe **IdentityUser** e adicione as seguintes propriedades.
 - PrimeiroNome - *string*
 - UltimoNome - *string*
 - DataNascimento - *DateTime*
 - NIF – *int*

9. Faça as alterações necessárias ao projeto para utilizar esta classe em vez da classe ***IdentityUser***.

Nota:

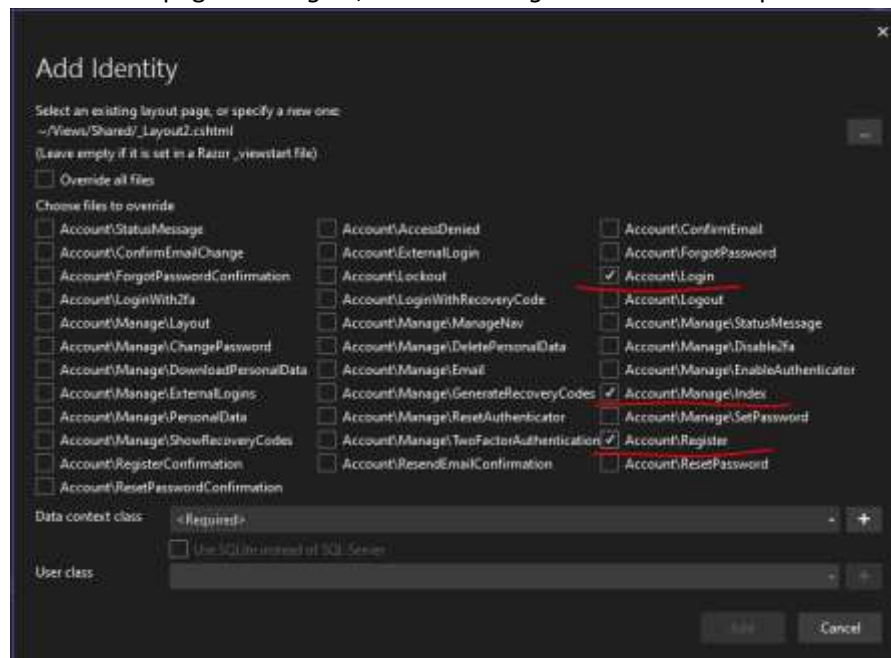
- Para isso necessita de modificar os ficheiros ***Program.cs*** e ***ApplicationDbContext.cs***

10. Verifique se tem instalado o seguinte pacote NuGet:

Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design

11. Gere as seguintes páginas *Razor*, relativas ao *Identity*, através da opção **“Add » New Scaffolded Item » Identity”**.

Selecione as páginas: *“Login”*, *“Index”* e *“Register”* conforme a próxima imagem.



[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/SECURITY/AUTHENTICATION/SCAFFOLD-IDENTITY?VIEW=ASPNETCORE-6.0&TABS=VISUAL-STUDIO](https://learn.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-6.0&tabs=visual-studio)

12. Modifique a página de Registo de Utilizadores por forma a utilizar o modelo personalizado. Ou seja, no registo devem ser solicitados ao utilizador as propriedades que constam na classe ***“ApplicationUser”***.

- Verifique se as propriedades *PrimeiroNome*, *UltimoNome*, *Data de Nascimento* e *NIF* são guardados na base de dados.

13. Modifique a página de perfil do utilizador por forma a utilizar o modelo personalizado.

- Verifique se as propriedades *PrimeiroNome*, *UltimoNome*, *Data de Nascimento* e *NIF* são guardados na base de dados, quando efectua alterações.

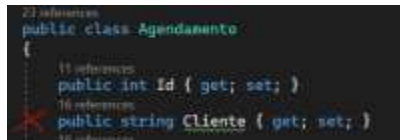
>> Microsoft ASP.Net Identity Core – Relacionamento com outras entidades

Nos próximos exercícios iremos fazer com os pedidos de agendamento apenas possam ser efetuados por utilizadores registados e autenticados.

Para isso iremos criar um relacionamento entre um agendamento e um utilizador. (*1 Utilizador – N Agendamentos*), bem como alterar os controllers e views que suportam esta funcionalidade.

14. Elimine todos os agendamentos existentes na base de dados.

15. Remova a propriedade **Cliente** da classe **Agendamento**, bem como todas as referências para esta propriedade (Views, ViewModels, Controller).



```
23 agendamentos
public class Agendamento
{
    15 agendamentos
    public int Id { get; set; }
    16 agendamentos
    public string Cliente { get; set; }
    17 agendamentos
}
```

16. Modifique as classes **Agendamento** e **ApplicationUser** por forma a definir o relacionamento anteriormente referido (*1 Utilizador – N Agendamentos*). Defina o relacionamento na sua totalidade:

Convenção 4 - Definir a relação na sua totalidade em ambas as entidades, indicando, além as propriedades de navegação, a chave estrangeira na entidade dependente.

17. Crie uma Migração e aplique-a.

18. Altere a View **“Pedido”**, do controller **Agendamento**, por forma a mostrar o formulário apenas aos utilizadores autenticados.

19. Mostre a seguinte mensagem no caso de os utilizadores não estarem autenticados.

Só utilizadores autenticados é que podem efetuar agendamentos. Por favor, autentique-se.

20. Faça as alterações necessárias no controller **Agendamento** para guardar o “utilizador” quando cria um agendamento.

21. Crie uma funcionalidade que permita mostrar os agendamentos do utilizador (**Os meus agendamentos**).

22. Crie uma opção no menu superior que “aponte” para esta funcionalidade.

- Esta opção só deve estar visível se o utilizador estiver autenticado.
23. Faça as alterações necessárias no **controller *Agendamento*** e respetivas vistas de forma a mostrar os seguintes dados do utilizador (PrimeiroNome, UltimoNome, Username) na Lista de Agendamentos e nos detalhes de um agendamento.

> Ficha Prática Nº 6

Objetivos/Temas:

- Segurança em aplicações web
- Microsoft Identity Core

> Exercícios

- CRIAR REGRAS DE ACESSO
- CONFIGURAR ROLES
- CRIAR E MANIPULAR ROLES
- CRIAR E MANIPULAR UTILIZADORES
- ATRIBUIR ROLES

>> Microsoft ASP.Net Identity Core – Autorização

A autorização refere-se ao processo de determinar se um utilizador tem permissão para realizar determinada operação / aceder a um recurso (listar registos, editar registos, etc.).

Existem 3 tipos de autorização:

- Autorização Simples
 - Atributo **[Authorize]** - Só utilizadores autenticados é que podem aceder ao recurso (controller, método).
- Autorização com Roles
 - Atributo **[Authorize(Roles="Role1, Role2, ..., Role10")]** - Só utilizadores autenticados e que pertençam a uma das roles enumeradas é que podem aceder ao recurso (controller, método).
- Autorização com Políticas
 - Atributo **[Authorize(Policy = "Maiores18")]** – Só utilizadores que cumpram os requisitos definidos na politica de acesso “Maiores18” é que podem aceder ao recurso(controller, método).

1. Adicione o atributo **[Authorize]** na classe **AgendamentoController** - *controller Agendamento*.

```
[Authorize]  
public class AgendamentosController : Controller
```

2. Termine a sessão na aplicação web (caso tenha sessão iniciada).
3. Clique na opção “**Agendamentos**” no menu superior.
 - Analise o comportamento da aplicação.
4. Experimente aceder a qualquer vista do **controller Agendamentos** e analise o resultado obtido.
5. Remova o atributo **[Authorize]** adicionado no ponto 1.

```
[Authorize]
public class AgendamentosController : Controller
```

6. Adicione o atributo **[Authorize]** no método **Index** do **controller Agendamento**.

```
// GET: Agendamentos
[Authorize]
public async Task<IActionResult> Index(){
    var applicationDbContext = _context.Agendamentos.Include(a => a.tipoDeAula);
    return View(await applicationDbContext.ToListAsync());
}
```

7. Ainda com a sessão terminada, experimente aceder:

- À listagem de agendamentos – método *Index* – controller *Agendamentos*.
- Ao pedido de agendamento – método *Pedido* – controller *Agendamentos*.

Verifique o comportamento de aplicação. Veja as diferenças entre aplicar o atributo **[Authorize]** na classe que define o controller ou aplicar apenas num método específico.

>> Microsoft ASP.Net Identity Core – Roles (IdentityRole)

Roles servem para identificar as funções que um utilizador desempenha/pertence numa aplicação. Um utilizador pode pertencer a uma ou mais Roles. Por exemplo, o “*João*” pode ser **administrador** e **funcionário** enquanto o “*Paulo*” pode ser apenas **Cliente**.

Desta forma é possível restringir o acesso a determinados(as) recursos/funcionalidades da aplicação, através da criação de regras de acesso (autorização).

Para ativar/registar a funcionalidade **Roles** é necessário modificar o **Program.cs**:

```
builder.Services.AddDefaultIdentity<ApplicationUser>(
    options => options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```


Tipos de autorização com Roles:

- Simples – o utilizador tem de pertencer à role indicada

```
[Authorize(Roles = "Administrador")]
public class AgendamentosController : Controller
```

- Múltipla (ou) – o utilizador tem de pertencer **a uma** das roles indicadas

```
[Authorize(Roles = "Administrator,Funcionario,Gestor ")]
public class AgendamentosController : Controller
```

- Múltipla (e) – o utilizador tem de pertencer **a todas** as rolas indicadas

```
[Authorize(Roles = "Administrator")]
[Authorize(Roles = "Funcionario")]
public class AgendamentosController : Controller
```

Para criar Roles/Utilizadores Iniciais na aplicação faça o seguinte:

8. Crie uma classe, dentro da diretoria **Data**, com o nome **Inicializacao**, com o seguinte código (com a devida adaptação ao seu projecto):

```
using Microsoft.AspNetCore.Identity;
using PWEB_AulasP_2223.Models;
using System;

namespace PWEB_AulasP_2223.Data{
    public enum Roles{
        Admin,
        Formador,
        Cliente
    }

    public static class Inicializacao{
        public static async Task CriaDadosIniciais(UserManager<ApplicationUser>
userManager, RoleManager<IdentityRole> roleManager){
            //Adicionar default Roles
            await roleManager.CreateAsync(new IdentityRole(Roles.Admin.ToString()));
            await roleManager.CreateAsync(new IdentityRole(Roles.Formador.ToString()));
            await roleManager.CreateAsync(new IdentityRole(Roles.Cliente.ToString()));

            //Adicionar Default User - Admin
            var defaultUser = new ApplicationUser{
                UserName = "admin@localhost.com",
                Email = "admin@localhost.com",
                PrimeiroNome = "Administrador",
                UltimoNome = "Local",
                EmailConfirmed = true,
                PhoneNumberConfirmed = true
            };
            var user = await userManager.FindByEmailAsync(defaultUser.Email);
            if (user == null){
                await userManager.CreateAsync(defaultUser, "Is3C..00");
                await userManager.AddToRoleAsync(defaultUser,
Roles.Admin.ToString());
            }
        }
    }
}
```

9. Adicione as seguintes linhas ao ficheiro **Program.cs**, depois da linha `var app = builder.Build();`:

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    try
    {
        var userManager = services.GetRequiredService<userManager<ApplicationUser>>();
        var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
        await Inicializacao.CriaDadosIniciais(userManager, roleManager);
    }
    catch (Exception)
    {
        throw;
    }
}
```

10. Execute a aplicação e verifique na base de dados se foram criados os registos relativos às 3 Roles (**Admin, Formador, Cliente**) e ao utilizador **admin@localhost.com**.

dbo.AspNetRoles [Data] ApplicationDbContext.cs ScaffoldingRea

Max Rows: 1000

Id	Name	NormalizedNa...	ConcurrencySt...
06f0355a-66d6...	Admin	ADMIN	7eb10116-28d6...
81660b3a-4ce2...	Formador	FORMADOR	73054e29-7cf7...
ea858796-5f61...	Cliente	CLIENTE	064f02be-911b...
NULL	NULL	NULL	NULL

Figura 1 - tabelaAspNetRoles

	Id	UserName	NormalizedUserName	Email	NormalizedEmail
	140d57d3-813c-4...	tiago2@isec.pt	TIAGO2@ISEC.PT	tiago2@isec.pt	TIAGO2@ISEC.PT
	2a5e252f-14f3-4...	tiago3@isec.pt	TIAGO3@ISEC.PT	tiago3@isec.pt	TIAGO3@ISEC.PT
	32642186-8deb-4...	tiago4@isec.pt	TIAGO4@ISEC.PT	tiago4@isec.pt	TIAGO4@ISEC.PT
	417f625-af3c-4...	admin@localhost	ADMIN@LOCALHOST.COM	admin@localhost.com	ADMIN@LOCALHOST.COM
	73fac34c-cb91-4...	tiago@isec.pt	TIAGO@ISEC.PT	tiago@isec.pt	TIAGO@ISEC.PT
	NULL	NULL	NULL	NULL	NULL

Figura 2 tabela - AspNetUsers

11. Adicione as seguintes regras de autorização baseadas em Roles:

- *Controller Cursos*
 - Apenas os utilizadores com a Role *Admin* podem efetuar as operações de *Editar*, *Apagar* e *Criar*.
- *Controller Categorias*
 - Apenas os utilizadores com a Role *Admin* podem efetuar as operações de *Editar*, *Apagar* e *Criar*.

- *Controller Agendamentos*
 - Apenas os utilizadores com a Role *Admin* podem efetuar as operações de *Editar*, *Apagar*.
 - Apenas os utilizadores com a Role *Cliente* podem efetuar um agendamento (*Criar*).

12. Faça as alterações necessárias à página de Registo de utilizadores por forma a que os novos utilizadores sejam adicionados à Role Cliente.

>> Gestão Roles

13. Crie um *controller* “**RoleManager**”.

14. Crie uma *view* vazia com o nome “**Index**” (~/*Views/RoleManager/Index.cshtml*).

15. Substitua o código da Classe `RoleManagerController` por:

```
public class RoleManagerController : Controller
{
    private readonly RoleManager<IdentityRole> _roleManager;
    public RoleManagerController(RoleManager<IdentityRole> roleManager)
    {
        /* código a criar */
    }
    public async Task<IActionResult> Index()
    {
        /* código a criar */
        return View(roles);
    }
    [HttpPost]
    public async Task<IActionResult> AddRole(string roleName)
    {
        /* código a criar */
        return RedirectToAction("Index");
    }
}
```

16. Substitua o código da *view* *Index* que criou no ponto 15 por:

```

@model IEnumerable<Microsoft.AspNetCore.Identity.IdentityRole>
@{
    ViewData["Title"] = "Role Manager";
    Layout = "~/Views/Shared/_Layout2.cshtml";
}
<h1>Gestão de Roles</h1>
<div class="row">
    <div class="col-md-4">
        <form method="post" asp-action="AddRole" asp-controller="RoleManager">
            <div class="input-group">
                <input name="roleName" class="form-control">
                <span class="input-group-btn">
                    <button class="btn btn-warning">Add New Role</button>
                </span>
            </div>
        </form>
    </div>
</div>
<table class="table table-striped table-hover table-bordered mt-4">
    <thead>
        <tr>
            <th>Role</th>
            <th>Id</th>
        </tr>
    </thead>
    <tbody>
        /* código a criar */
    </tbody>
</table>

```

17. Faça as alterações necessárias ao código fornecido for forma a ser possível listar todas as Roles existentes bem como adicionar novas Roles.

>> Gestão de utilizadores / atribuição de Roles

18. Crie os seguintes ViewModel:

```

public class UserRolesViewModel
{
    public string UserId { get; set; }
    public string PrimeiroNome { get; set; }
    public string UltimoNome { get; set; }
    public string UserName { get; set; }
    public IEnumerable<string> Roles { get; set; }
}

public class ManageUserRolesViewModel
{
    public string RoleId { get; set; }
    public string RoleName { get; set; }
    public bool Selected { get; set; }
}

```

19. Crie um controller “**UserRolesManager**” e substitua o código da classe por:

```

public class UserRolesManagerController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;

    public UserRolesManagerController(UserManager<ApplicationUser> userManager,
    RoleManager<IdentityRole> roleManager)
    {
        /* código a criar */
    }
    public async Task<IActionResult> Index()
    {
        var users = await _userManager.Users.ToListAsync();
        /* código a criar */
        return View(userRolesViewModel);
    }
    private async Task<List<string>> GetUserRoles(ApplicationUser user)
    {
        return new List<string>(await _userManager.GetRolesAsync(user));
    }
    public async Task<IActionResult> Details(string userId)
    {
        /* código a criar */
        return View(model);
    }
    [HttpPost]
    public async Task<IActionResult> Details(List<ManageUserRolesViewModel> model,
    string userId)
    {
        /* código a criar */
        return RedirectToAction("Index");
    }
}

```

20. Crie duas views *Index* e *Details*.

21. Faça as alterações necessárias ao código fornecido e às views criadas de forma a:

- Listar todos os utilizadores e as suas roles.



- Ver em detalhe as Roles de um utilizador e modificar as mesmas



> Ficha Prática Nº 7

Objetivos/Temas:

- Upload de Ficheiros (Sistema de Ficheiros / Base de dados)

> Parte I – Conceitos

>> Upload de ficheiros (Base de dados / Sistema de ficheiros)

Aspetos que deve ter em conta quando se implementam funcionalidades que permitem aos utilizadores fazer o upload de ficheiros para o servidor:

- Potencial vetor de ataque:
 - Executar ataques do tipo negação de serviço;
 - Enviar ficheiros com vírus ou malware;
 - Comprometer servidores / redes;
- Melhores práticas para reduzir a eficácia deste tipo de ataques:
 - Área de upload dedicada - preferencialmente num disco que não seja do sistema operativo:
 - Criar restrições de segurança;
 - Remover as permissões de execução;
 - Não guardar os ficheiros na mesma diretoria da aplicação (nem sempre possível);
 - Utilizar nomes seguros:
 - Gerar o nome do ficheiro a ser guardado.
 - Não utilize o nome do ficheiro submetido pelo utilizador;
 - Restringir o tipo de ficheiros que o utilizador pode fazer upload:
 - Validar o tipo de ficheiros (pdf, jpg, png, docx, etc.);
 - Validar os dados do lado do cliente e do lado do servidor;
 - Definir um tamanho máximo para os ficheiros enviados, de forma a evitar uploads demasiado grandes;
 - Executar um antivírus/malware no ficheiro carregado antes de o guardar;

Dependendo dos requisitos da aplicação a ser desenvolvida e do tipo e quantidade de ficheiros a serem armazenados, podemos optar por armazenar os ficheiros em base de dados ou num sistema de ficheiros.

Existem vantagens e desvantagens em cada uma das abordagens. As mais importantes são:

- **Sistema de ficheiros**

Vantagens:

- Performance
 - A performance de acesso ao disco pode ser melhor do que o acesso à base de dados (escrita e leitura).
- Mais simples
 - Para guardar os ficheiros “apenas” é necessário chamar um método *Save*.
 - Para fazer o download “basta” aceder a uma URL.
- Migração de dados “simplificada”.
 - Copiar ficheiros de um disco para o outro.
 - Migrar para a cloud, como Amazon S3, etc..
- Custo mais baixo
 - É mais barato adicionar espaço de armazenamento físico do que pagar pelo aumento de espaço da base de dados (dependendo do tipo).

Desvantagens

- Não existe um mapeamento relacional.
 - Ficheiros apagados acidentalmente / intencionalmente
- Menor segurança
 - Falhas de permissões, etc.

- **Base de dados**

Vantagens:

- Consistência / mapeamento relacional;
- Maior segurança;

Desvantagens

- Necessidade de converter os ficheiros para guardar na base e dados;
- Backups maiores (base de dados);
- Menor performance;
- Custo;

>> Upload de ficheiros – HTML

[HTTPS://DEVELOPER.MOZILLA.ORG/EN-US/DOCS/WEB/HTML/ELEMENT/INPUT/FILE](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/file)

Para efetuar o upload de ficheiros através um formulário é necessário:

1. Especificar o atributo **enctype** no formulário:

```
<form id="profile-form" method="post" enctype="multipart/form-data">
...
</form>
```

2. Utilizar um **input** do tipo **file**:

```
<input type="file" id="avatar" name="avatar" accept="image/png, image/jpeg">
```

Exemplo:

```
<form id="profile-form" method="post" enctype="multipart/form-data">
  <label for="avatar">Choose a profile picture:</label>
  <input type="file" id="avatar" name="avatar" accept="image/png, image/jpeg">
  <button id="update-profile-button" type="submit">Upload</button>
</form>
```

Atributo **accept** é uma *string* que define os tipos de ficheiros que o input deve aceitar:

- É uma lista separada por vírgula;
- Como um determinado tipo de arquivo pode ser identificado de mais de uma maneira, é importante fornecer um conjunto completo de tipos;
- Exemplo: `.png, .jpg, image/png, image/jpeg`

É possível enviar mais do que um ficheiro em simultâneo para o servidor. Para isso é necessário adicionar o atributo **multiple** à tag `<input>`

```
<input type="file" multiple id="avatar" name="avatar" accept="image/png, image/jpeg">
```

>> Upload de ficheiros ASP.NET Core

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/MVC/MODELS/FILE-UPLOADS?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads?view=aspnetcore-6.0)

Existem duas abordagens:

- Buffering
 - `IFormFile` - representação C# de um ficheiro, é usado para transferir, processar e guardar um ficheiro no servidor web;
 - O ficheiro inteiro é lido para um objeto `IFormFile`, portanto, o requisito de disco e memória no servidor dependerá do número e tamanho do(s) ficheiro(s) submetido(s);

- Se o ficheiro for maior que 64 KB, será movido da memória para o armazenamento temporário em disco;
 - No caso de serem submetidos ficheiros grandes ou muitos ficheiros em simultâneo, deve-se considerar o uso da abordagem de *streaming*.
- Streaming
 - Ficheiro enviado numa solicitação multipart e processado / guardado diretamente pela aplicação;
 - Consome menos memória / espaço em disco em comparação com a abordagem de buffer;
 - O streaming deve ser a abordagem preferida se for permitida a submissão de ficheiros grandes ou se se prever uma submissão simultânea por parte de vários utilizadores;

> Exercícios

- ADICIONAR FOTOGRAFIA DO UTILIZADOR AO SEU PERFIL (BASE DE DADOS)
- MOSTRAR FOTOGRAFIAS DOS UTILIZADORES NA LISTA DE UTILIZADORES
- ADICIONAR IMAGENS À ENTIDADE CURSO (SISTEMA DE FICHEIROS)
- MOSTRAR AS IMAGENS NOS DETALHES DO CURSO

Adicionar fotografia de perfil ao utilizador:

1. Adicione a seguinte propriedade à classe `ApplicationUser` (*Model*):

```
[Display(Name = "O meu Avatar")]
public byte[]? Avatar { get; set; }
```

2. Adicione as seguintes propriedades à classe `InputModel` (página *razor* de perfil do utilizador):

```
[Display(Name = "O meu Avatar")]
public byte[]? Avatar { get; set; }

public IFormFile AvatarFile { get; set; }
```

3. Modifique a página *razor* relativa ao perfil do utilizador de forma que seja possível enviar para o servidor um ficheiro do tipo JPG/PNG.

Para isso, inclua no formulário da página Razor de perfil do utilizador o seguinte código:

```
<div class="form-floating">
  <div>
    <img id="MyAvatar" class="img-thumbnail" />
  </div>
  <div>
    <label asp-for="Input.AvatarFile ">Escolha uma imagem de perfil:</label>
    <input type="file" accept=".png,.jpg,.jpeg,image/png,image/jpeg"
      asp-for="Input.AvatarFile" class="form-control" />
    <span asp-validation-for="Input.AvatarFile " class="text-danger"></span>
  </div>
</div>
```

4. Faça as alterações necessárias ao método `OnPostAsync` e ao método `LoadAsync` por forma a guardar a imagem na base de dados e a obter a imagem da base de dados.

- `OnPostAsync`

```
if (Input.AvatarFile != null)
{
    if (Input.AvatarFile.Length > (200*1024))
    {
        StatusMessage = "Error: Ficheiro demasiado grande";
        return RedirectToPage();
    }
    // método a implementar - verifica se a extensão é .png,.jpg,.jpeg
    if (!IsValidFileType(Input.AvatarFile.FileName))
    {
        StatusMessage = "Error: Ficheiro não suportado";
        return RedirectToPage();
    }
    using (var dataStream = new MemoryStream())
    {
        await Input.AvatarFile.CopyToAsync(dataStream);
        user.Avatar = dataStream.ToArray();
    }
    await _userManager.UpdateAsync(user);
}
```

- Implemente o método `"bool IsValidFileType(string fileName){...}"`;

- `LoadAsync`

```
Input = new InputModel{
    PhoneNumber = phoneNumber,
    DataNascimento = user.DataNascimento,
    PrimeiroNome = user.PrimeiroNome,
    UltimoNome = user.UltimoNome,
    NIF = user.NIF,
    Avatar = user.Avatar,
};
```


Analise com o docente as alterações.

5. Faça as alterações necessárias no ficheiro cshtml da página *razor* anterior por forma a mostrar a imagem que foi guardada na base de dados.

Exemplo de como *renderizar* uma imagem através de um model:

```

```

6. Mostre uma imagem default caso o utilizador não tenha especificado o seu avatar.
 - Faça o download da seguinte imagem e guarde-a na directoria "**wwwroot\img**" com o nome "**user.png**":  <https://cdn-icons-png.flaticon.com/512/149/149071.png>;
 - Faça as alterações na vista por forma a mostrar a imagem de Avatar ou a imagem de Default;
7. Adicione uma pré-visualização da imagem, quando o utilizador seleciona uma nova imagem.
 - Para isso, no input (file) adicione o atributo **onChange** com o seguinte conteúdo:

```
document.getElementById('MyAvatar').src = window.URL.createObjectURL(this.files[0])
```

8. Altere o controller/views de gestão de utilizadores por forma a que seja possível:
 - Visualizar os avatares dos utilizadores na listagem de utilizadores;
 - Visualizar o avatar do utilizador quando se escolhe ver os detalhes;

Adicionar imagens a um curso:

Deve ter em atenção os seguintes aspetos:

- Os Ficheiros têm de ser guardados no sistema de ficheiros (diretório na aplicação).
- Um curso pode conter mais que uma fotografia.
- Quando se visualiza os detalhes do curso, devem ser mostradas todas as imagens existentes.

Métodos C# necessários/úteis:

- `Directory.GetCurrentDirectory()`

Devolve uma string com o caminho do diretório da aplicação sem a "/" no final.

- `Directory.Exists(string path)`

Devolve um bool indicando se um determinado diretório existe.

- `Path.Combine(string path1, string path2)`

Combina duas strings num caminho.

- `File.Exists(string path)`

Devolve um bool indicando se um determinado ficheiro existe.

- `File.Create(string path)`

Cria um ficheiro.

- `IFormFile.CopyToAsync(Stream target)`

Copia o conteúdo do ficheiro enviado para um stream.

9. Defina a estrutura de ficheiros / diretórios a utilizar.
10. Altere o formulário de edição de um curso por forma a ser possível adicionar uma ou mais imagens.
11. Altere a view de detalhes do curso por forma a ser possível visualizar as imagens existentes
 - Se não existir nenhuma imagem deve mostrar uma mensagem ao utilizador
12. Altere a view de edição de um curso por forma a ser possível eliminar imagens.

> Ficha Prática Nº 8

Objetivos/Temas:

- Gestão de estados em aplicações web
- Gestão de estados em aplicações ASP.NET Core
- Variáveis de Sessão
- TempData
- Gráficos

> Parte I – Conceitos

>> Gestão de estados em aplicações web

[HTTPS://LEARN.MICROSOFT.COM/PT-PT/ASPNET/CORE/FUNDAMENTALS/APP-STATE?VIEW=ASPNETCORE-6.0](https://learn.microsoft.com/pt-pt/aspnet/core/fundamentals/app-state?view=aspnetcore-6.0)

[HTTPS://LEARN.MICROSOFT.COM/EN-US/ASPNET/CORE/SECURITY/GDPR?VIEW=ASPNETCORE-6.0#TEMPDATA-PROVIDER-AND-SESSION-STATE-COOKIES-AREN'T-ESSENTIAL](https://learn.microsoft.com/en-us/aspnet/core/security/gdpr?view=aspnetcore-6.0#tempdata-provider-and-session-state-cookies-arent-essential)

O HTTP é um protocolo sem estado. Por padrão, as solicitações HTTP são mensagens independentes que não retêm valores do utilizador.

O estado pode ser armazenado utilizando diferentes abordagens:

Abordagem	
Cookies	HTTP cookies. Pode incluir dados utilizando o código da aplicação (servidor)
Session state	HTTP cookies + código da aplicação (servidor)
TempData	HTTP cookies ou estado de sessão
Query strings	HTTP Query strings
Campos ocultos	HTTP form fields
HttpContext.Items	Código da aplicação (servidor)
Cache	Código da aplicação (servidor)

Cookies

- Um cookie é um pequeno ficheiro de texto que é guardado no dispositivo que está a consultar o sítio web;
- Normalmente são usados para personalização do sítio web (guardar preferências do utilizador, idioma, moeda, tema gráfico, etc.)
- São enviados em cada solicitação HTTP;
- Podem ser manipulados pelo cliente, pelo que não devem ser usados para guardar informação sensível.;

	<ul style="list-style-type: none"> • Como tal, devem ser validados pela aplicação; • Dois tipos: <ul style="list-style-type: none"> ○ Cookies persistentes; ○ Cookies de sessão; • Deve ter-se especial atenção ao seu uso (ver RGPD https://commission.europa.eu/law/law-topic/data-protection_pt)
Session state	<ul style="list-style-type: none"> • Session State / estado da sessão é uma funcionalidade do ASP.NET Core para armazenamento de dados do utilizador enquanto o utilizador navega numa aplicação web; • Armazenamento mantido pela aplicação para persistir dados através de pedidos de um cliente; • Apoiados por uma cache e considerados dados efémeros; • A aplicação web deve continuar a funcionar sem os dados da sessão. • Os dados críticos da aplicação devem ser armazenados em base de dados e armazenados em sessão apenas como uma otimização do desempenho. • Não suportado pelo SignalR. • O ASP.NET Core mantém o estado da sessão fornecendo um cookie ao cliente que contém um ID de sessão. O ID da sessão cookie: <ul style="list-style-type: none"> ○ é enviado para a aplicação com cada pedido; ○ É utilizado pela aplicação para ir buscar os dados da sessão; • O estado da sessão apresenta os seguintes comportamentos (entre outros): <ul style="list-style-type: none"> ○ O cookie de sessão é específico para o browser; ○ As sessões não são partilhadas entre browsers; ○ Os cookies de sessão são apagados quando a sessão do programa de navegação termina; ○ Se um cookie é recebido para uma sessão expirada, é criada uma nova sessão que utiliza o mesmo cookie de sessão; ○ As sessões vazias não são retidas. A sessão deve ter pelo menos um valor definido para persistir a sessão através dos pedidos;
TempData	<ul style="list-style-type: none"> • Propriedade (Páginas Razor e/ou controllers) que armazena dados até que estes sejam lidos no próximo pedido;
Query strings	<ul style="list-style-type: none"> • Dados embebidos na URL; • As URL são públicas – não utilizar para armazenar dados sensíveis. • Limite de tamanho • Vetor de ataques do tipo CSRF;
Campos ocultos	<ul style="list-style-type: none"> • Campos de formulário escondidos/ocultos - <code><input type="hidden" /></code> • Podem ser manipulados pelo cliente e por este motivo devem ser sempre validados pelo código da aplicação (servidor);
HttpContext.Items	<ul style="list-style-type: none"> • Uma coleção usada para armazenar dados durante o processamento de um pedido; • O conteúdo da coleção é descartado após o pedido ser processado; • A coleção Items costuma ser usada para permitir que componentes ou middleware comuniquem entre si quando operam em diferentes momentos do pedido e não têm nenhuma maneira direta de passar parâmetros;

Cache	<ul style="list-style-type: none"> • O cache é uma forma eficiente de armazenar e obter dados. A aplicação pode controlar a vida útil dos itens armazenados em cache. • Os dados em cache não estão associados a um pedido específico, utilizador ou sessão; • Não se deve armazenar em cache dados específicos de utilizadores - que possam ser obtidos por outros pedidos de outros utilizadores;
--------------	--

> Parte II – Exercícios

>> Criar e Ler um Cookie que permita alterar a cor de fundo da página de perfil do utilizador

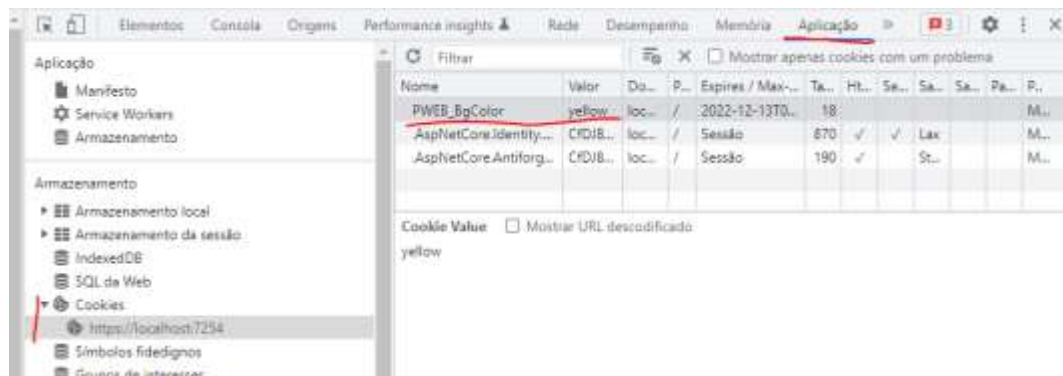
1. Adicione o seguinte código C# ao método OnGetAsync() na página Razor de perfil do utilizador:

```
var cookieOptions = new CookieOptions();
cookieOptions.Expires = DateTime.Now.AddDays(10);
cookieOptions.Path = "/";
var cookie = Request.Cookies["PWEB_BgColor"];
if (cookie == null)
    cookie = "yellow";

// adiciona / modifica o cookie com o nome PWEB_BgColor
Response.Cookies.Append("PWEB_BgColor", cookie, cookieOptions);

// elimina o cookie com o nome PWEB_BgColor
// Response.Cookies.Delete("PWEB_BgColor");
```

2. Execute a aplicação, faça login e navegue até à página de perfil do utilizador.
3. Abra as ferramentas de programação do browser e verifique os cookies existentes.



4. Navegue por outras “páginas” da aplicação e veja se o cookie se mantém.

5. Adicione o seguinte código HTML ao formulário da página Razor de perfil do utilizador:

```
<div class="form-floating">
  <label for="pageBgColor">Cor de fundo da página</label>
  <select name="pageBgColor" id="pageBgColor"
    onchange="setCookie('PWEB_BgColor',this.value,1)"
    class="form-control form-select form-select-sm">
    <option value="white">White</option>
    <option value="red">Red</option>
    <option value="orange">Orange</option>
    <option value="blue">Blue</option>
  </select>
</div>
```

6. Adicione o seguinte código Javascript à página Razor de perfil do Utilizador (dentro da secção Scripts "@section Scripts {...}"):

```
<script>
  checkCookie();
  function setCookie(cname, cvalue, exdays) {
    console.log(cvalue);
    const d = new Date();
    d.setTime(d.getTime() + (exdays * 24 * 60 * 60 * 1000));
    let expires = "expires=" + d.toUTCString();
    document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
    document.body.style.setProperty("background-color", cvalue, "important");
  }

  function getCookie(cname) {
    let name = cname + "=";
    let ca = document.cookie.split(';');
    for (let i = 0; i < ca.length; i++) {
      let c = ca[i];
      while (c.charAt(0) == ' ') {
        c = c.substring(1);
      }
      if (c.indexOf(name) == 0) {
        return c.substring(name.length, c.length);
      }
    }
    return "";
  }

  function checkCookie() {
    let bgColor = getCookie("PWEB_BgColor");
    let select = document.getElementById('pageBgColor');
    if (bgColor != "") {
      document.body.style.setProperty("background-color", bgColor,
"important");
      select.value = bgColor;
    } else {
      alert("Please choose a Background color:");
    }
  }
</script>
```

7. Teste o funcionamento da aplicação – verifique se é possível mudar a cor de fundo da página.
8. Faça debug à aplicação e verifique se no método OnGetAsync() é possível ler o valor do cookie em cada pedido que é efetuado. Analise o código com o docente.

>> Criar um carrinho de compras (comprar cursos)

9. Configurar Sessões - Adicione o seguinte código (fundo a amarelo) C# ao **Program.cs**:

```

builder.Services.AddControllersWithViews();

// session state
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
    options.Cookie.Name = ".PWEbApp.Session";
    options.IdleTimeout = TimeSpan.FromSeconds(10);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

var app = builder.Build();
/*
...
...
*/
app.UseAuthentication();
app.UseAuthorization();

// session state
app.UseSession();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

```

10. Uma vez que por omissão apenas podemos guardar nas variáveis de sessão *Strings* e *Int32* temos de criar uma extensão para podermos guardar e obter objetos complexos (no formato JSON).

Para isso crie uma classe nova chamada de **SessionExtensions** com o seguinte código:

```

public static class SessionExtensions
{
    public static void SetJson<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonSerializer.Serialize(value));
    }

    public static T? GetJson<T>(this ISession session, string key)
    {
        var value = session.GetString(key);
        return value == null ? default : JsonSerializer.Deserialize<T>(value);
    }
}

```

*Sugestão: crie uma pasta com o nome **Helpers** e crie a classe dentro desta pasta*

11. De seguida crie as seguintes classes que vão auxiliar na gestão do carrinho de compras.

CarrinhoItem

```
public class CarrinhoItem
{
    public int CursoId { get; set; }
    public string CursoNome { get; set; }
    public int Quantidade { get; set; }
    public decimal PrecoUnit { get; set; }
}
```

Carrinho

```
public class Carrinho
{
    public List<CarrinhoItem> items { get; set; } = new List<CarrinhoItem>();

    public void AddItem(CarrinhoItem curso, int qtd){}
    public void RemoveItem(CarrinhoItem curso) {}
    public decimal Total(){}
    public void Clear() => items.Clear();
}
```

12. Implemente os métodos ***AddItem()***, ***RemoveItem()*** e ***Total()***
13. Crie uma *view* vazia com o nome ***_carrinho.cshtml***, dentro da pasta **Shared**.
14. Copie o seguinte código para a *view* que criou no ponto anterior:

```
@using PWEB_AulasP_2223.Helpers
@{
    var CarrinhoDeCompras = Context.Session.GetJson<Carrinho>("CarrinhoDeCompras") ??
    new Carrinho();
    int qtd = CarrinhoDeCompras == null ? 0 : CarrinhoDeCompras.items.Count();
}
<div class="form-inline d-flex ">
    <span class="badge badge-dark fs-6 ">@qtd</span>
    <strong>
        <svg bootstrap-icon="Cart"
            class="text-white me-2" width="24" height="24"
            aria-label="Search"></svg>
    </strong>
</div>
```

15. Inclua esta *view* parcial nos ficheiros de layout por forma a ficar com

```
<partial name="_carrinho" />
<partial name="~/Views/Cursos/QuickSearchPartial.cshtml" />
<partial name="_LoginPartial" />
```

16. Adicione um “botão” “comprar curso” na *view* ***search*** do controller ***Cursos*** e na *view* ***Index*** do controller **Home**, conforme imagem seguinte:

Os nossos cursos

Existe(m) 2 curso(s) disponíveis

NOME DO CURSO (A)	Curso 2 (A)
<p>€ 1312.00</p> <p>resumo resumo</p> <p>Saber mais comprar</p>	<p>€ 100.00</p> <p>descrição do curso2</p> <p>Saber mais comprar</p>

Este botão deve enviar um pedido HTTP GET para o método **comprar**, passando como parâmetro o Id do curso a comprar.

Exemplo do URL de um pedido: <https://localhost:7254/Cursos/Comprar/1>

17. Implemente o método **Comprar** no *controller Cursos*.

- O método deve:
 - Adicionar o curso escolhido (recebido como parâmetro) ao carrinho de compras.
 - Redirecionar o utilizador para a acção “Carrinho” – a criar nos pontos seguintes

18. Crie uma view vazia com o nome **Carrinho** e copie o seguinte código para a View:

```
@model PWEB_AulasP_2223.Models.Carrinho
<h1>Carrinho de compras</h1>
Items no carrinho: <span class="badge bg-secondary">@Model.items.Count</span>
<table class="table table-hover table-striped">
  <thead>
    <tr>
      <th>Item</th>
      <th>Preço Unitário</th>
      <th>Quantidade</th>
      <th>Sub-total</th>
    </tr>
  </thead>
  <tbody>
  </tbody>
</table>
<div class="row">
  <div class="col-12 text-end">
    <a href="#" class="btn btn-primary">continuar a comprar</a>
    <a href="#" class="btn btn-success">checkout</a>
  </div>
</div>
```

19. Implemente o método Carrinho e faça as alterações necessárias à vista por forma a obter um resultado semelhante a:

Item	Preço Unitário	Quantidade	Sub-total
NOME DO CURSO	1312.00	2	2624.00
Curso 2	100.00	4	400.00
Total			3624.00

20. Faça as alterações necessárias na **View** e no **Controller** por forma a poder alterar as quantidades de um item do carrinho, bem como poder remover um item do carrinho e/ou limpar o carrinho.

>> Criar gráficos com recurso a uma biblioteca JavaScript

Existem várias bibliotecas JavaScript para criar gráficos em aplicações web.

Neste exercício / exemplo, vamos utilizar a biblioteca **ChartJS** (<https://www.chartjs.org/docs/latest/>).

O objetivo deste exercício é criar um gráfico que nos permita ver um resumo de vendas por mês e por produto.

Nota: Como ainda não temos as vendas na base de dados, vamos usar um repositório local como fonte de dados e não a base de dados. Mais tarde, quando existirem vendas na base de dados podem e devem fazer as alterações necessárias para ler os dados da base de dados.

21. Crie uma view vazia, com o nome **GraficoVendas**, no controller cursos e copie o seguinte código:

```

@{
    ViewBag.Title = "Index";
    Layout = "~/Views/Shared/_layout2.cshtml";
}
<h1>Vendas mensais por curso</h1>
<h2>Gráfico com Chart.js</h2>
<div>
    <canvas id="chartGraficoVendas"></canvas>
</div>
@section Scripts {
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script>
        $.ajax({
            type: "POST",
            url: "/Cursos/GetDadosVendas",
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: function (data) {
                let Labels = data[0];
                let Datasets1 = data[1];
                let dataT = {
                    labels: Labels,
                    datasets: [{
                        label: "Cursos",
                        data: Datasets1,
                        fill: false,
                        borderWidth: 1,
                        backgroundColor: ["red", "green", "blue", "cyan", "yellow"]
                    }]
                };
                let ctx = $("#chartGraficoVendas").get(0).getContext("2d");
                let myNewChart = new Chart(ctx, {
                    type: 'bar',
                    data: dataT,
                    options: {
                        responsive: true,
                        title: { display: true, text: 'Vendas de cursos ' },
                        legend: { position: 'bottom' },
                    }
                });
            }
        });
    </script>
}

```

22. Crie o método **GraficoVendas** e o método **GetDadosVendas**, no controller cursos e copie o seguinte código:

```

// GET: Cursos/GraficoVendas/5
public async Task<IActionResult> GraficoVendas()
{
    return View();
}
[HttpPost]
// POST: Cursos/GraficoVendas/5
public async Task<IActionResult> GetDadosVendas()
{
    //dados de exemplo
    List<object> dados = new List<object>();

    DataTable dt = new DataTable();
    dt.Columns.Add("Cursos", System.Type.GetType("System.String"));
    dt.Columns.Add("Quantidade", System.Type.GetType("System.Int32"));
    DataRow dr = dt.NewRow();
    dr["Cursos"] = "CATEGORIA AM (Ciclomotor)";
    dr["Quantidade"] = 12;
    dt.Rows.Add(dr);
    dr = dt.NewRow();
    dr["Cursos"] = "CATEGORIA A1 (Motociclo - 11kw/125cc)";
    dr["Quantidade"] = 96;
    dt.Rows.Add(dr);
    dr = dt.NewRow();
    dr["Cursos"] = "CATEGORIA A2 (Motociclo - 35kw)";
    dr["Quantidade"] = 87;
    dt.Rows.Add(dr);
    dr = dt.NewRow();
    dr["Cursos"] = "CATEGORIA B1 (Quadriciclo)";
    dr["Quantidade"] = 67;
    dt.Rows.Add(dr);
    dr = dt.NewRow();
    dr["Cursos"] = "CATEGORIA B (Ligeiro Caixa Automática)";
    dr["Quantidade"] = 63;
    dt.Rows.Add(dr);

    foreach (DataColumn dc in dt.Columns)
    {
        List<object> x = new List<object>();
        x = (from DataRow dr in dt.Rows select dr[dc.ColumnName]).ToList();
        dados.Add(x);
    }
    return Json(dados);
}

```