> 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

- 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 contem 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
	• **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/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

- 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

- 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

- 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 especifico antes de serem utilizados.
      -
    - **ViewBag**
      - ViewBag é um bjeto 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 querem 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: