Apontamentos Teóricos

- Aula 18/10/2021
  - Namespaces
  - POO - Principios
    - Criação de um objeto
    - Construtor
  - Gets e Sets
  - Parâmetros *ref* vs *out*
  - Membros virtuais
  - Classes Abstratas e Sealed
  - Palavra chave *as* em Csharp
  - Palavra chave *is* em Csharp
- Aula 25/10/2021
  - Strings
    - Caracter @
    - Conversão de Strings
  - Algoritmo de validação do NIF
  - Operadores
  - Switch
  - Instrução Switch
- Aula T 08/11/2021
  - Collections
    - Inicialização
  - Delegates
    - Delegate como parâmetro
    - Instanciar um delegate como um método anónimo
    - Métodos anónimos
    - Func -> retorna sempre alguma coisa
    - Action -> não retorna nada
    - Predicate -> retorna um bool
    - Delegate com expressão lambda
  - Operador lambda
    - Delegate com Lambda

- Delegate com Lambda , exemplo com um argumento
- Delegate com Lambda, exemplo com vários argumentos
- Outras formas de utilização do Lambda

Aula T 15/11/2021

- Tipo Nullable
- Operador Null-Coalescing
- LINQ
  - Arquitetura - LINQ to Entities
  - Query Syntax
  - Extension Methods - mais moderno
  - Com LINQ vs Sem LINQ
  - Operadores LINQ
  - LINQ - case insensitive
  - LINQ - regex
  - LINQ - Exemplo
  - LINQ - Where
  - LINQ - Exercicios
  - LINQ - OrderBy
  - LINQ - ThenBy
  - LINQ - Projeção (Select() e SelectMany())
  - LINQ - GroupBy

Aula 29/11/2021

- LINQ Joins
  - Inner Join
  - Group Join
  - Cross Join
- Any
- All
- Skip
- Take
- TakeWhile
- SkipWhile
- First / Last
- FirstOrDefault / LastOrDefault
- Single
- SingleOrDefault
- ElementAt
- ElementAtOrDefault
- Agregação

- Count
- Sum
- Min / Max
- Average
- Conjuntos (Union, Intersect, Concat, Except, Distinct)
- Exercicio -> Selcionar os alunos cujo nome começa por "F"
- Exercicio -> Selcionar os alunos cujo o ultimo nome começa por "F"

Aula 06/12/2021

- MVC
  - Model
  - View
  - Controller
  - Routing
- Estrutura ASP.NET Core
  - Estrutura de diretórios
- Action Methods
  - Action Selectors
- GET vs POST vs PUT vs DELETE
- O que é uma action?
- Um pedido a /clientes/remover/5 por conveção é manipulado por?
- Personalziar o Routing de forma a aceitar o que se pretende
- Convention Based Routing

Aula 20/12/2021

- Passagem de dados (Controlador - Vista - Controlador)
  - Para dados fortemente tipados (viewmodel)
  - Para dados fracamente tipados (viewdata e viewbag)
  - Em suma
  - Passagem de dados: ViewData
  - Passagem de dados: ViewBag
- Views
- Razor View Engine
  - Algumas regras
- Layout com Razor
- Tag Helpers

Aula 03/01/2022

- Entity Framework (EF)
  - Tipos de carregamentos de objetos
  - Formas para especificar o modelo de dados
- Code First

Utilização da Entity Framework em Code First
Mas qual o motivo das migrações?
Como recuperar de um erro de uma migração?
Code-First Conventions por omissão
DataAnnotations vs Fluent API
Fluent API
Entity Framework - Database First
Identity Framework
Formas de autenticação
Arquitetura

Aula 18/10/2021

Namespaces

- O código dos programas C# estão organizados em *namespaces*
  - Organiza um conjunto de classes relacionadas entre si
  - As classes podem ter nomes iguais, desde que estejam em *namespaces* diferentes
  - Um *namespace* pode conter outros namespaces

POO - Principios

- Abstração -> utilizada para a definição de entidades do mundo rial
- Encapsulamento -> tem como objetivo a ocultação de pormenores internos da classe
- Herança -> como um filho pode herdar coisas dos pais, uma classe pode herder atributos e métodos de outra
- Polimorfismo -> permite ao programador definir métodos genéricos em classes ou interfaces, que podem ser implementados de diversas formas nas respetivas subclasses
- Todo **objeto** possui:
  - **Estado**: conjunto de propriedades de um objeto

- **Comportamento**: conjunto de ações possíveis sobre o objeto
- **Unicidade**: todo o objeto é único

- **Classes**
  - Definem os tipos de objetos
  - Definem quais os dados e o código dos objetos

```
public class Carro
{
    private string cor;
    private string modelo;
    private string anoFabrico;
    private string combustivel;
    public void Anda()
    {   ...
    }
    public void Para()
    {   ...
    }
    public void Acelera()
    {   ...
    }
}
```

Criação de um objeto

- **Definição da classe**
- **Criação do objeto**

- Sintaxe geral:

```
acesso class nome_da_classe
   {
       // Bloco de Instruções
   }
```

- acesso
  - nível de proteção da classe (*internal / public*)
  - Por omissão, assume o nível *internal*
  - Pode ser também *abstract* ou *sealed*

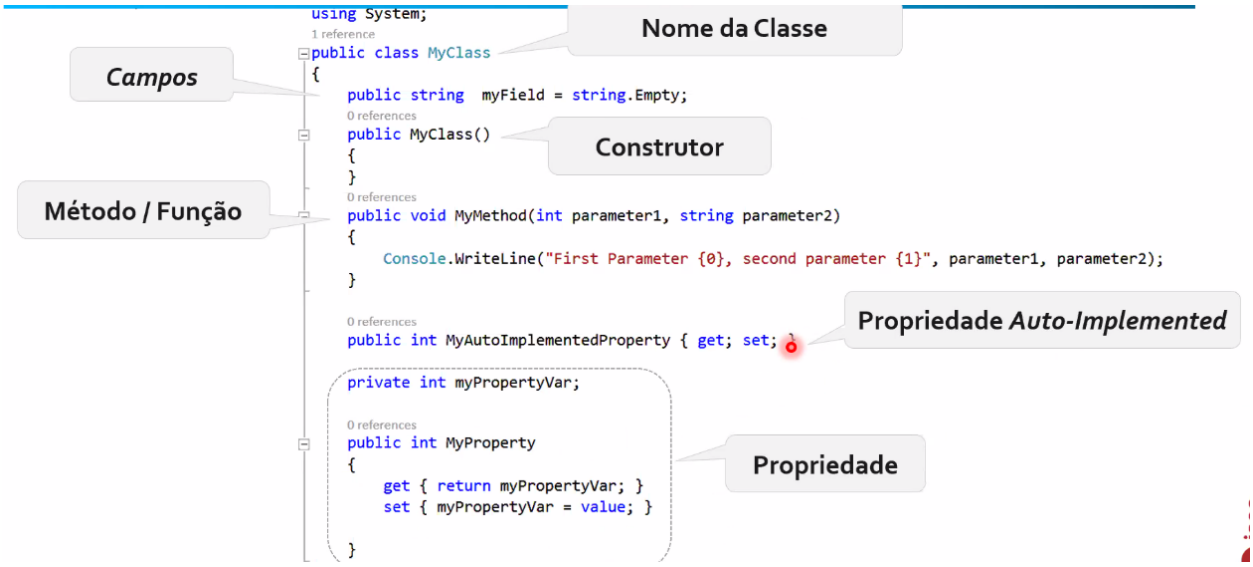
Construtor

- Se não for definido um construtor para uma classe, o C# define um por omissão

```
class MinhaClasse {
    public MinhaClasse()
    {
        // Construtor Default
    }

    public MinhaClasse(int meuInt)
    {
        // Construtor com parâmetro
    }
}
```

Gets e Sets

- Podemos meter **get** e **set** ou só 1 deles
- Tambem podemos ter código dentro do get e do set
- https://stackoverflow.com/a/5096967



Parâmetros *ref* vs *out*

```
using System;
namespace RefVSOut {
    class Program
    {
        static void Main(string[] args)
        {
            int val1 = 0;
            int val2;

            MethodRef(ref val1);
            Console.WriteLine(val1);

            MethodOut( out val2);
            Console.WriteLine(val2);
        }
    ...
```

```
static void MethodRef(ref int value)
 {
            value = 1;
 }
static void MethodOut(out int value)
  {
     value = 2;
  }
```

Basicamente usamos o ref e o out quando queremos

A diferença é que se usarmos a keyword ref temos de i

Membros virtuais

- É um membro na classe base que define uma implementação por omissão que pode ser alterada (overridden) pela classe derivada

Classes Abstratas e Sealed

- A palavra chave *abstract* permite criar classes e membros de classes incompletas que têm de ser implementados na classe derivada

- A palavra chave *sealed* evita que a classe oumembros virtuais sejam herdadados

Palavra chave *as* em Csharp

- *as* tenta converter um objeto para um tipo específico e retorna *null* se a conversão falhar

```
object meuObjecto = new Frango("FrangoObjecto");
Quadrado quadrado = meuObjecto as Quadrado;
if (quadrado==null)
{
    Console.WriteLine("O objecto não é um quadrado...");
}
```

Palavra chave *is* em Csharp

- *is* permite de forma rápida determinar se um determinado objecto é compatível com um tipo e devolve *false* caso não sejam

```
object meuObjecto = new Frango("FrangoObjecto");
Quadrado quadrado = meuObjecto as Quadrado;
if (quadrado is Frango)
{
    Console.WriteLine("O objecto é um Frango...");
}
```

Aula 25/10/2021

Strings

- **Interpolação de strings**

- Nas versões actuais do C#, podemos utilizar a seguinte alternativa, que é mais simples

```
var aluno = new Aluno();

var mensagem = $"{aluno.Curso} é o curso do aluno {aluno.Nome}";
```

- Para usarmos a interpolação de *strings*, é necessário preceder a *string* com o caractere $

Programação Web – 2021/2022 – LEI D+LE+PL – ISEC/IPC @JB

Caracter @

- Podemos usar o `@` para a contrução de strings mais facilmente sem recorrer tanto às sequências de *escape* usando antes *verbatim strings*

- Assim, em vez de se utilizar

```
string myFileName = "C:\\myfolder\\myfile.txt";
```

pode-se utilizar

```
string myFileName = @"C:\myfolder\myfile.txt";
```

Programação Web – 2021/2022

- O `@` ajuda tambem na formatação de espaços, mudanças de linhas ...

O @ ajuda então quando temos uma string com caracteres especiais e queremos fazer e

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            var mensagem = @"Caro aluno,

        A matéria tratada nesta secção respeita à construção de strings de um modo mais
simplificado.
As facilidades oferecidas permitirão a escrita de um código mais legível.

Utilize estas novas caracteristicas!
Bom estudo!";
            Console.WriteLine(mensagem);
        }
    }
}
```

Programação Web - 2021/2022 – LEI D+CE+PL – ISEC/IPC @JB

- O `@` tambem serve para formatar SQL querys por exemplo

```
var alunos = @"SELECT *
FROM Aluno
WHERE curso = 'EI'
```

Conversão de Strings

- A conversão de strings em arrays de bytes ou array de strings é algo habitualmente feito
- Por exemplo, vamos converter uma string para um array de bytes

- Para essa conversão utiliza-se o método estático *GetBytes*, presente na classe *Unicode* e na classe *UTF8* do namespace *System.Text.Encoding*
  - Exemplo de conversão de *string* para *byte[]*

```
string mensagem = "Programação Web com C# e ASP.Net Core";
byte[] conteudo =
System.Text.Encoding.Unicode.GetBytes(mensagem) ;
```

- Agora convertendo array de bytes em strings

- Para essa conversão utiliza-se o método estático *GetString*, presente na classe *Unicode* e na classe *UTF8* do namespace *System.Text.Encoding*

```
string strfinal = Encoding.Unicode.GetString(strconvertida, 0,
strconvertida.Length);
```

  - Para ter uma melhor percepção de como estas conversões são feitas, nomeadamente de *array* de *bytes* para *string,* analise e execute o código exemplo que se segue

- Exemplo concreto

```
using System;
using System.Text;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            string strinicial = "Programação Web com C# e ASP.Net Core";
            Console.Write("String Inicial:");
            Console.WriteLine(strinicial);
            byte[] strconvertida = System.Text.Encoding.Unicode.GetBytes(strinicial);
            Console.WriteLine("\nConcatenando todos os Bytes em uma string de bytes: ");
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < strconvertida.Length; i++)
            {
                builder.Append(strconvertida[i].ToString("x2"));
            }
            Console.WriteLine(builder.ToString());
            Console.WriteLine("\nUsando a classe BitConverter para colocar todos os bytes numa string:");
            string bitString = BitConverter.ToString(strconvertida);
            Console.WriteLine(bitString);
            Console.Write("\nObtendo a string inicial em Unicode a partir dos Bytes: ");
            string strfinal = Encoding.Unicode.GetString(strconvertida, 0, strconvertida.Length);
            Console.WriteLine(strfinal);

            Console.ReadKey();
        }
    }
}
```

Programação Web – 2021/2022 – LEID+CE+PL – ISEC/IPC

Algoritmo de validação do NIF

- O algoritmo para validação do NIF é bastante simples: é um número composto por 9 dígitos, sendo o último um digito de controlo, é este digito que iremos calcular para verificar se o NIF está correcto.
- Antes de procedermos ao cálculo do digito de controlo, temos de apurar algumas condições, uma bastante importante tem a ver com o 1º digito, este terá que ser válido, o primeiro dígito do NIF tem a ver com o tipo de entidade que o possui, que pode ser uma das seguintes:

- NIF e Número de Contribuinte – Número de Identificação Fiscal
  - 1 – pessoa singular
  - 2 – pessoa singular

Programação Web – 2021/2022 – LEID+CE+PL – ISEC/IPC

- NIPC – Número de Identificação de Pessoa Colectiva
  - 5 (pessoa colectiva),
  - 6 (pessoa colectiva pública),
  - 8 (empresário em nome individual),
  - 9 (pessoa colectiva irregular ou número provisório).
- Passado com sucesso esta condição, calcula-se o digito de controle
- Para calcular o digito de controlo iremos multiplicar individualmente e por ordem os restantes 8 algarismos, fazendo-o da seguinte forma:
  - O 1.º digito multiplicamos por 9, o 2.º dígito por 8, 3.º dígito por 7, 4.º dígito por 6, 5.º dígito por 5, 6.º dígito por 4, 7.º dígito por 3 e o 8.º dígito por 2.
- Iremos somar todos os resultados da operação anterior.
- Com o resultado da soma anterior iremos dividir por 11 e aproveitar o resto dessa divisão. Se o resto for 0 ou 1, o dígito de controle será 0, caso seja outro algarismo, o dígito de controle será o resultado de 11 – resto.

Programação Web – 2021/2022 – LEI D+CE+PL – ISEC/IPC @JB

- Ver a parte da validação do NIF com regex

Operadores

- **Para além dos Operadores mais comuns, o C# possui alguns operadores que pelas suas particularidades iremos referir mais em pormenor**
  - **Esses operadores são:**
    - Operador condicional nulo ?
    - Operador de coalescência nula ??
    - Operador ternário
    - Operador *is* e *is not*

Assim, em vez de

```
if (aluno != null && aluno.Endereco != null)
{
  Console.WriteLine($"{aluno.Nome}\n{aluno.Curso} - aluno.Idade}\n{aluno.Endereco.Rua} -
  {aluno.Endereco.CodigoPostal} - {aluno.Endereco.Pais}\nMail: {aluno.Mail}");
}
```

Pode-se ter

```
Console.WriteLine($"{aluno?.Nome}\n{aluno?.Curso} - {aluno?.Idade}\n{aluno?.Endereco?.Rua} -
{aluno?.Endereco?.CodigoPostal} - {aluno?.Endereco?.Pais}\nMail: {aluno?.Mail}");
```

Sem haver o risco de ocorrer alguma excepção!

Switch

- A estrutura condicional *switch* até a versão 6 do *C# tinha* algumas limitações e não era tão flexível como as estruturas equivalentes do Visual Basic e do Delphi
- Com a versão 7 o *switch* passou a suportar *pattern matching* e estas limitações deixaram de existir e o código gerado com a estrutura condicional *switch* ficou mais conciso e legível
  - Estas alterações ao *switch* permitem utilizações menos triviais
  - No exemplo seguinte mostra-se a utilização da cláusula ***when*** no ***switch***

•Exemplo 1

```
using System;
namespace TesteComStrings {
  class Program   {
    private static void IdentificaCaracter(char c)
    {
      switch (c)
      {
        case char l when (c >= 'a' && c <= 'z') || (c >='A' && c <= 'Z'):
          Console.WriteLine($"{c} é um caracter alfabético.");
        break;
        case char n when c >= '0' && c <= '9':
          Console.WriteLine($"{c} é um caracter numérico.");
        break;
```

```
            case char n when c >= '!' && c <= '/':
                Console.WriteLine($"{c} é um caracter simbolo.");
                break;
            default:
                break;
        }
    }
    static void Main(string[] args)
    {
        IdentificaCaracter('a');
        IdentificaCaracter('Z');
        IdentificaCaracter('9');
        IdentificaCaracter('*');
    }
}
}
```


Programação Web – 2021/2022 – LEI D+CE+PL – ISEC/IPC @JB

Instrução Switch

- Instruções Switch **não é o mesmo** do que Expressões switch
- Caracterizam-se por a variavel aparecer antes da palavra chave switch
- Permite uma maior compactação e é usado o operador *lambda*

- Alterações na sintaxe desta construção:
  - O posicionamento da variável antes da palavra-chave *switch* ajuda a distinguir visualmente a expressão *switch* da instrução *switch*
  - Os corpos são expressões, não instruções
  - Os elementos ***case*** e **:** são substituídos por **=>** (mais conciso e intuitivo)
  - O caso ***default*** é substituído por um **_**

Programação Web – 2021/2022 – LEI D+CE+PL – ISEC/IPC @JB

- No exemplo seguinte mostra-se a utilização de instruções ***switch*** em expressões ***switch***

- **Exemplo 4 – Neste utilizam-se instruções *switch* em expressões switch**
  Nota: Tem de se compilar com C# 9 para este exemplo funcionar

```
using System;
namespace TesteComStrings
{
    public enum Curso
    {
        Informatica,
        Civil,
        Mecanica,
        Electrotecnia
    }
    class Program
    {
        public static string Seleccionar(Curso curso) =>
```

```
class Program {
    public static string Seleccionar(Curso curso) =>
    curso switch
        {
            Curso.Informatica => "Programação e Manutenção de Sistemas Informáticos!",
            Curso.Civil => "Obras de Stª Engrácia!",
            Curso.Mecanica => "Carros? Isso é o quê?",
            Curso.Electrotecnia => "Não lhes peçam para arranjar um televisor!",
            _ => throw new ArgumentException(message: "Opção de Curso desconhecida!", paramName: nameof(curso))
        };
        static void Main(string[] args) {
            Console.WriteLine(Seleccionar(Curso.Informatica));
            Console.WriteLine(Seleccionar(Curso.Civil));
            Console.WriteLine(Seleccionar(Curso.Mecanica));
            Console.WriteLine(Seleccionar(Curso.Electrotecnia));
        }
    }
}
```

Microsoft Visual Studio Debug Console

```
Programação e Manutenção de Sistemas Informáticos!
Obras de Stª Engrácia!
Carros? Isso é o quê?
Não lhes peçam para arranjar um televisor!
```

Aula T 08/11/2021

Collections

- Existem dois tipos disponíveis:
  - ***Non-generic collections*** - Tipicamente desenvolvidas para trabalhar com tipos *System.Object* e portanto, são *containers* fracamente tipados:
    - ❖ `ArrayList, BitArray, Hashtable, Queue, SortedList, Stack HybridDictionary, ListDictionary, StringCollection, BitVector32`
    - ❖ Namespaces: `System.Collections,System.Collections.Specialized`
  - ***Generic collections*** – Mais seguras que as anteriores, sendo necessário especificar o "tipo" do tipo.
    O indicador de qualquer item genérico é o "type parameter" especificado com os <>, como por exemplo, `List<T>`
    - ❖ Namespace: `System.Collections.Generic`
- Todas as classes *collections* implementam a interface *IEnumerable*, portanto os valores da colecção podem ser acedidos usando o ***foreach***

Inicialização

- Inicialização com valores específicos:

```
var stringList = new List<string>
{
     "Blabla",
     "BliBli",
};
```

```
var temp = new List<string>();
temp.Add("Blabla");
temp.Add("Blibli");
var stringList = temp;
```

```
class Aluno  {
   public int BI { get; set;}
   public string Nome {
                    get; set;}
}
```

```
Aluno aluno= new Aluno {
        BI = 10,
       Nome = "Maria"
};
Console.WriteLine($"Aluno:{aluno.BI}-{aluno.Nome}");
```

```
List<Aluno> listaGenAlunos = new List<Aluno> {
    new Aluno { BI=102212232, Nome="Maria" },
    new Aluno { BI=21222334, Nome="Jose" },
    new Aluno { BI=102324532, Nome="Nuno" },
};
```

Delegates

- Tipos que permitem fazer referencia a métodos
- Usado para passar métodos como argumentos a outros metodos
- Um delegate é um tipo referencia
- Pode ser usado para encapsular uma método com nome ou anónimo

- **Declaração** de um *delegate* é semelhante à assinatura de um método
- Tem um retorno e qualquer número de parâmetros, de qualquer tipo
- Recorrer à *keyword* ***delegate***

```
public delegate void Exemplo(int valor);
```

- Todos os *delegates* derivam da class `System.Delegate`

- Quando se atribui um método a um *delegate*, o método deve ter o mesmo tipo de retorno bem como o mesmo tipo de parâmetros;
- Usar o Delegate:
  - Declarar
  - Instanciar
  - Invocar

```
public static void NomeMetodo(string msg) {
    Console.WriteLine(msg);
}

public delegate void Declarar(string msg);
Declarar xpto=NomeMetodo;
xpto("mensagem");
```

- O *delegate* é semelhante a um ponteiro de função, a vantagem é que é um ponteiro seguro

Delegate como parâmetro

- Tipo delegate como parâmetro

```
static void DoOperacao(MetodoDel del,string txt)
{
    var ret = del(txt);
    Console.WriteLine(string.Format(" > delegate retornou {0}", ret));
}

static public string ConverteMaiusculas(string txt)
{
    return "Maiuscula = "+txt.ToUpper();
}
static public string ConverteMinusculas(string txt)
{
    return "Minuscula = " + txt.ToLower();
}
```

```
delegate string MetodoDel(string texto);
static void Main(string[] args)
{
    string txt = "Programacao Web";
    DoOperacao(ConverteMaiusculas, "Programacao Web");
    DoOperacao(ConverteMinusculas, "Programacao Web");
}
```

Instanciar um delegate como um método anónimo

```
delegate string MetodoDel(string texto);
 static void Main(string[] args)
{
    string txt = "Programacao Web";
     MetodoDel del2 = delegate (string texto)
     {
         return texto + "|" + texto;
     };
     Console.WriteLine("{0}", del2(txt));
    }
```

Métodos anónimos

- A ideia é permitir escrever métodos inline no codigo sem declarar um metodo formal com nome

- C# inclui tipos de **bluit-in delegates**!
- Existem 3 tipos de delegate pré-definidos:
  - *Func*

```
Func<int, int, int> soma = Soma;
```

  - *Action*

```
Action<int> imp = ImprimeConsola;
```

  - *Predicate*

```
Predicate<string> estaMai = EstaMaiuscula;
```

Func -> retorna sempre alguma coisa

- É um tipo genérico delegate
- Tem zero ou mais parâmetros de entrada e um parâmetro de saida

- O ultimo parâmetro é considerado o parâmetro de saída
- Deve incluir um parâmetro *out* para o resultado

```
public delegate int Operacao(int i, int j);
class Program
{
    static int Soma(int x, int y)
    {
        return x + y;
    }
    static void Main(string[] args)
    {
        Operacao soma = Soma;
        int resultado = soma(10, 10);
        Console.WriteLine(resultado);
    }
}
```

```
class Program
{
    static int Soma(int x, int y)
    {
        return x + y;
    }
    static void Main(string[] args)
    {
        Func<int, int, int> soma = Soma;
        int resultado = soma(10, 10);
        Console.WriteLine(resultado);
    }
}
```

- ***Func*** com método **Anónimo**

```
Func<int> geraAletatorio = delegate()
    {
        Random rnd = new Random();
        return rnd.Next(1, 100);
    };
```

Action -> não retorna nada

- É parecido ao *Func* no entanto aqui não retorna nada
- Pode ser usado quando o retorno é *void*

```
Action<int> imp = ImprimeConsola;
```

```
Action<int> imp =  new Action<int>(ImprimeConsola);
```

```
public delegate void Imprime(int val);

static void ImprimeConsola(int i)
{
    Console.WriteLine(i);
}

static void Main(string[] args)
{
    Imprime imp = ImprimeConsola;
    imp(10);
    imp(20);
    imp(30);

}
```

```
static void ImprimeConsola(int i)
{
    Console.WriteLine(i);
}

static void Main(string[] args)
{
    Action<int> imp = ImprimeConsola;
    imp(10);
    imp(20);
    imp(30);

}
```

```
static void Main(string[] args)
{
    Action<int> imp = delegate (int i)
    {
        Console.WriteLine(i);
    };
    imp(10);
    imp(20);
    imp(30);
}
```

Como método anonimo

Predicate -> retorna um bool

- Representa um método que contém um conjunto de critérios e verifica se os parâmetros passados cumprem esse critério ou não
- Deve ter um parâmetro de entrada e retorna um bool

```
static bool EstaMaiuscula(string str)
{
    return str.Equals(str.ToUpper());
}

static void Main(string[] args)
{
    Predicate<string> estaMai = EstaMaiuscula;

    bool result = estaMai("programacao");
    Console.WriteLine(result);
    result = estaMai("WEB");
    Console.WriteLine(result);
}
```

Delegate com expressão lambda

```
delegate string MetodoDel(string texto);
 static void Main(string[] args)
 {
    string txt = "Programacao Web";

    MetodoDel del3 = texto=>texto.ToUpper();

     Console.WriteLine("{0}", del3(txt));
    }
```

Operador lambda

- **OPERADOR LAMBDA =>**
  - Este operador pode ser usado de duas maneiras no C#:
    1. Como *operador lambda* numa *expressão lambda*: separa as variáveis de entrada do lado esquerdo do corpo lambda do lado direito
    2. Numa definição de corpo da expressão, em que separa um nome de membro da implementação do membro.
    - O operador => tem a mesma precedência que o operador de atribuição = e é associativo-à-direita

- As **expressões lambda** são expressões embutidas semelhantes aos métodos anónimos mas ainda mais flexiveis
- Método anónimo
  - sem nome
  - sem modificador de acesso
  - não retorna valor
  - usado como um atalho para a inicialização de um *delegate*

- Sintaxe:

```
argumentos => expressão
```

- Exemplo:

Expressão Lambda

```
int Quadrado(int numero)
{
    return numero * numero;
}
```

```
int Quadrado(int numero)=> numero * numero;
```

Delegate com Lambda

```
public delegate int OperacaoComInteiro(int input);

static void Main(string[] args)
{

    OperacaoComInteiro dobro = x => x * 2;
    OperacaoComInteiro quadrado = x => x*x;

    Console.WriteLine("  ->{0}", dobro(5));
    Console.WriteLine("  ->{0}", quadrado(5));
}
```

Delegate com Lambda , exemplo com um argumento

```
public delegate int DobroInteiro(int input);
DobroInteiro dobro = x => x * 2;
```

Com Lambda

Método anónimo

```
public delegate int DobroInteiro(int input);
DobroInteiro dobro = delegate (int x) {
    return x * 2;
};
```

Delegate com Lambda, exemplo com vários argumentos

```
public delegate int MultiplicaInteiros(int n1, int n2);

MultiplicaInteiros mult = (x,y) => x*y;

Console.WriteLine("->{0}", mult(2,7));
```

Outras formas de utilização do Lambda

- Passar uma expressão lambda como parâmetro de um método

```
List<int> lista1 = new List<int> { 1, 2, 6, 5, 4, 3, 2, 7, 8, 5, 24, 42 };
List<int> lista2 = new List<int> { 12,33,44,2,5,67,86,4,3,22,56,77 };

List<int> listaResult=lista1.FindAll(x => x >= 24);

List<int> listaResult2 = lista2.FindAll(x => x >= 24);
```

Aula T 15/11/2021

Tipo Nullable

```
Nullable<int> variavelInt = null;
int? variavelInt = null;

DateTime? variavelDate = null;

variavelDate.
```

```
int? varInt = null;
int varInt2;

Console.WriteLine(varInt.GetValueOrDefault());
Console.WriteLine(varInt.HasValue);
Console.WriteLine(varInt.Value);
```

```
C:\WINDOWS\system32\cmd.exe
0
False

Excepção não processada: System.InvalidOperationException: O objecto que
 pode ser nulo tem de ter um valor.
   em System.ThrowHelper.ThrowInvalidOperationException(ExceptionResourc
e resource)
   em System.Nullable`1.get_Value()
   em Exemplo.Program.Main(String[] args) em D:\Cristiana\_TRABALHO_\ISE
C\AULAS\_Programacao Web\Linq\lixo\Program.cs:line 20
Press any key to continue . . .
```

- Se quiser dar valor ao `varInt2` a partir do `varInt` tenho de fazer `varInt2 = varInt.GetValueOrDefault();`

- Se quiser dar valor ao `varInt` a partir do `varInt2` tenho de fazer `varInt = varInt2;`
- Ou seja, se quiser dar valor a uma variavel nao nula a partir de uma nula tenho de usar `GetValueOrDefault`
- Se quiser dar valor a uma variavel numa a partir de uma nao nula basta fazer como se faz normalmente

Operador Null-Coalescing

- Permite especificar um valor por omissão se o operando à esquerda for nulo

```
DateTime? data = null;
DateTime data1;
string str = null;
data1 = data ?? DateTime.Now;
Console.WriteLine("string = " + (str ?? " Sem valor!"));
```

```
int? varInt=null;
int varInt2;

varInt2 = varInt ?? 10;

int? varInt3 = varInt2;

Console.WriteLine(varInt2+" "+varInt3);
```

varInt2 = 10 e varInt3 = 10

```
int? varInt = null;
int? varInt2=null;

int varInt3 = varInt2 ?? varInt ?? 10;
```

varInt3 = 10

```
IEnumerable<Aluno> minhaLista = GetListaAlunos();
var al = minhaLista.SingleOrDefault(x => x.Id == 2) ??
                                        new Aluno { Id = 2 };
Console.WriteLine(al);
```

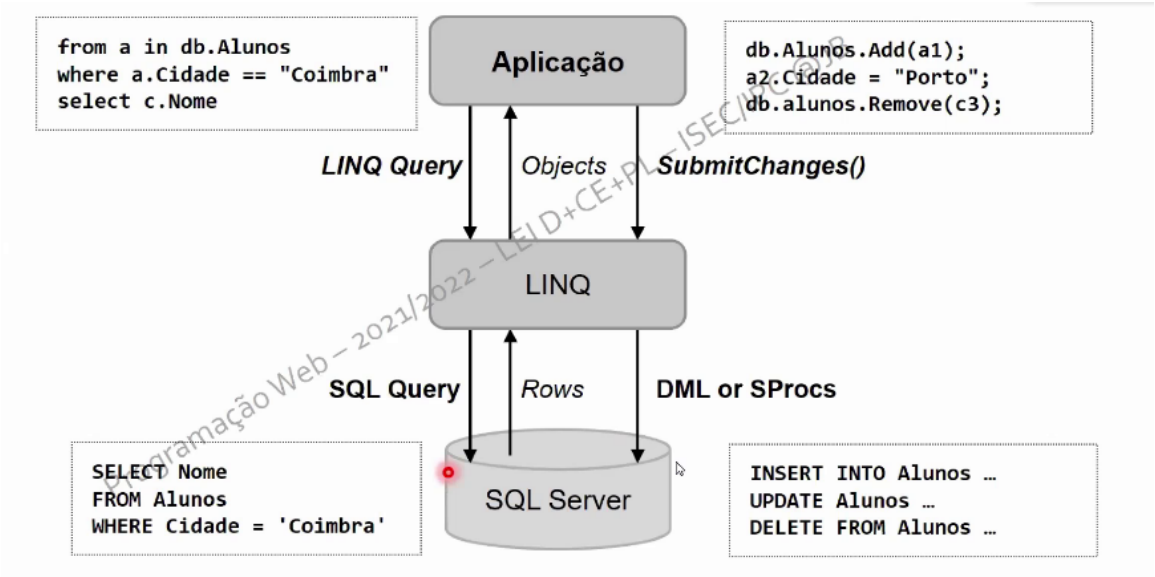
O que este código faz é ....

Se o aluno não tiver o ID 2 , então criamos um novo aluno com o ID 2

LINQ

- Permite efetuar pesquisar em C# ou VB a conjuntos de dados
- A diferença entre LINQ e Entity Framework é
  - A Entity Framework é para construir a base de dados
  - O LINQ é para fazer pesquisas nessas bases de dados

Arquitetura - LINQ to Entities



- O SQL Server não entende Query LINQ, só entende SQL Querys
- Então a Query LINQ tem de ser traduzida para SQL
  - A camada a meio (`LINQ`) faz a transformação

Query Syntax

- Semelhante À linguagem SQL
- Inicia com a palavra chave **from** para especificar de onde os dados provem e termina com a clausula *select* ou *group*

```
var alunosExcelentes = from a in db.Alunos
                       where a.Media >= 18
                       orderby a.Nome
                       select a.Nome
```

Extension Methods - mais moderno

- Usa uma extensão de métodos e expressões Lambda

```
var alunosExcelentes = db.Alunos.Where(a => a.Media >= 18)
                                .OrderBy(a => a.Nome)
                                .Select(a => a.Nome);
```

Com LINQ vs Sem LINQ

```
IList<string>  PWList = new List<string>() {
        "C#",
        "Entity Framework",
        "Aprender C#",
        "ASP.NET Core com C#" ,
        "Linq",
        "Lambda" ,
        "Identity" ,
};
```

```
var PWListCSharp = new List<string>();
foreach (string x in PWList)
{
    if (x.Contains("C#"))
        PWListCSharp.Add(x);
}
```

Sem LINQ

```
var listaCShap = from s in PWList
                 where s.Contains("C#")
                 select s;
```

Query Syntax

```
var listaCShap = PWList.Where(l=>l.Contains("C#"));
```

Extension Method

Operadores LINQ

Restriction	Where
Projection	Select, SelectMany
Ordering	OrderBy, ThenBy
Grouping	GroupBy
Joins	Join, GroupJoin
Quantifiers	Any, All
Partitioning	Take, Skip, TakeWhile, SkipWhile
Sets	Distinct, Union, Intersect, Except
Elements	First, Last, Single, ElementAt
Aggregation	Count, Sum, Min, Max, Average
Conversion	ToArray, ToList, ToDictionary
Casting	OfType<T>, Cast<T>

...

Programação Web – 2021/2022 – LEI+LCE+PL – ISEC/IPC @JB

LINQ - case insensitive

```
IList<string>  PWList = new List<string>() {
        "c#",
        "Entity Framework",
        "Aprender C#",
        "ASP.NET Core com C#" ,
        "Linq",
        "Lambda" ,
        "Identity" ,
};
```

```
var listaCSharp = from s in PWList
                  where s.Contains("C#", StringComparison.OrdinalIgnoreCase )
                  select s;
```

```
var listaCSharp = PWList.Where(l=>l.Contains("C#",StringComparison.OrdinalIgnoreCase));
```

LINQ - regex

```
string acentuados = @"[^a-zA-Z]";

Regex regex = new Regex(acentuados);

List<string> nomes = new List<string>() { "Cláudio", "Flávia", "Inácio", "Miguel", "Renato", "Ana", "Aurélio", "Francisco",
"Jorge", "Patrícia" };
var resultado = from n in nomes  where regex.IsMatch(n) select n;

Console.WriteLine("Nomes que possuem caracteres acentuados:");
    foreach (var item in resultado)
    {
        Console.WriteLine(item);
    }
```

A utilização de Expressões regulares em conjunto com o LINQ permite a selecção de padrões de caracteres que estejam de acordo com a expressão regular definida para padrão de comparação!

LINQ - Exemplo

```
public class Aluno
{
    public int Numero { get; set; }
    public string Nome { get; set; }
    public int Idade { get; set; }
}
```

Considere esta *class* e o *array de strings* e obtenha o nome dos alunos com 18<=idade<=23 usando *QS e EM*

```
Aluno[] alunos = {
        new Aluno() { Numero = 1, Nome= "Jose Antunes", Idade= 18 },
        new Aluno() { Numero = 2, Nome = "Filipa Moreira",  Idade = 21 },
        new Aluno() { Numero = 3, Nome = "Cristina Frias",  Idade = 25 },
        new Aluno() { Numero = 4, Nome = "Dinis Campos" , Idade = 20 },
        new Aluno() { Numero = 5, Nome = "Soraia Goncalves" , Idade = 31 },
        new Aluno() { Numero = 6, Nome = "Lena Rinheiro",  Idade = 17 },
        new Aluno() { Numero = 7, Nome = "Filipe Cruz",Idade = 19  },
 };
```

- **Selecionar todos os alunos com idade entre 18 e 23 com Query Syntax**

```
var alunosIdade18e23 = from a in alunos
                       where a.Idade > 18 &&
                             a.Idade < 23
                       select a;
```

- **Será possível especificar a restrição num bloco à parte?** (`SIM`)

- Seleccionar todos os alunos com idade entre 18 e 23 com QS

```
bool idadeEntre18e23(Aluno a) {
        return a.Idade > 18 && a.Idade < 23;
}
```

```
var alunosIdade18e23 = from a in alunos
                       where idadeEntre18e23(a)
                       select a;
```

- **Usando o delegate**
- **Selecionar todos os alunos com idade entre 18 e 23 com Extension Methods**

```
Aluno[] alunos = {
        new Aluno() { Numero = 1, Nome= "Jose Antunes", Idade= 18 },
        new Aluno() { Numero = 2, Nome = "Filipa Moreira",  Idade = 21 },
        new Aluno() { Numero = 3, Nome = "Cristina Fria",  Idade = 25 },
        new Aluno() { Numero = 4, Nome = "Dinis Campos" , Idade = 20 },
        new Aluno() { Numero = 5, Nome = "Soaraia Goncalves" , I
        new Aluno() { Numero = 6, Nome = "Lena Pinheiro",  Idade
        new Aluno() { Numero = 7, Nome = "Filipe Cruz",Idade = 1
var alunosIdade18e23 = alunos
                .Where(a=>a.Idade>18 && a.Idade<23);
```

Jose Antunes
Lena Pinheiro
Filipe Cruz

LINQ - Where

```
int[] numeros = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

var resultado = from n in numbers
                where n < 5
                select n;
```

```
string[] digits = { "zero", "one", "two", "three", "four",
                    "five", "six", "seven", "eight", "nine" };

var shortDigits = digits.Where((digit, index) =>
                               digit.Length < index));
```

five
six
seven
eight
nine

https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

LINQ - Exercicios

- Seleccionar os alunos cujo nome é "Filipa Moreira" usando LINQ EM

```
Aluno[] alunos = {
        new Aluno() { Numero = 1, Nome= "Jose Antunes", Idade= 18 },
        new Aluno() { Numero = 2, Nome = "Filipa Moreira",  Idade = 21 },
        new Aluno() { Numero = 3, Nome = "Cristina Fria",  Idade = 25 },
        new Aluno() { Numero = 4, Nome = "Dinis Campos" , Idade = 20 },
        new Aluno() { Numero = 5, Nome = "Soaraia Goncalves" , Idade = 31 },
        new Aluno() { Numero = 6, Nome = "Lena Pinheiro",  Idade = 17 },
        new Aluno() { Numero = 7, Nome = "Filipe Cruz",Idade = 19  },
 };
```

```
var filipa = alunos.Where(a => a.Nome == "Filipa Moreira");
```

- Seleccionar os alunos com número 5 usando LINQ EM

```
Aluno aluno5 = alunos.Where(a => a.Numero==5);
```

- Obter os números pares da lista

```
IList<int> listInts= new List<int>() {
                12,15,4,23,17,18,13
        };
```

```
var numPares = listInts.Where(n => n % 2 == 0);
```

- Obter apenas os números inteiros

```
object[] numeros =  {
      12,15.6,4.2,23,17.1,18,13
};
```

```
var apenasInteiros = numeros.OfType<int>();
```

LINQ - OrderBy

- Permite ordenar uma *collection* por um determinado valor
- O **OrderByDescending()** permite ordenar de forma descendente

```
var alunosOrdenados =alunos
                     .OrderBy(a => a.Nome);
```

▲ 1 of 2 ▼ (extension) IOrderedEnumerable<Aluno> IEnumerable<Aluno>.OrderBy<Aluno, TKey>(Func<Aluno, TKey> keySelector)
Sorts the elements of a sequence in ascending order according to a key.
keySelector: *A function to extract a key from an element.*

```
var alunosExcelentes = from a in Alunos
                       where a.Media >= 18
                       orderby a.Nome
                       select a;
```

```
var alunosExcelentes = db.Alunos.Where(a => a.Media >= 18)
                                .OrderBy(a => a.Nome);
```

```
var alunosExcelentes = from a in Alunos
                       where a.Naluno == 18
                       orderby a.Nome, a.Media, a.DataNascimento
                      descending
                       select a;
```

LINQ - ThenBy

- Permite ordenação de outros campos

```
var alunosExcelentes = alunos.Where(a => a.Media >= 18)
                             .OrderBy(a => a.Nome).ThenBy(a => a.Media);

var alunosExcelentes = alunos.Where(a => a.Media >= 18)
                             .OrderByDescending(a => a.Nome)
                             .ThenByDescending(a => a.Media);
```

Programação Web – 2021/2022 – LEI D+CE+PL

LINQ - Projeção (Select() e SelectMany())

- *Select()*
  - Permite aplicar uma transformação a todos os elementos em qualquer estrutura de dados que implemente a *IEnumerable*

LEI D+CE+PL – ISEC/IPC @JB

```
var alunos = from a in Alunos
             where a.NAluno == 1
             orderby a.Nome, a.DataNascimento descending
             select new {
                         Nome = a.Nome,
                         DataNasc = a.DataNascimento
             };
```

```
var alunos = alunos.Where(a => a.Media >= 18)
                   .OrderBy(a => a.Nome)
                   .ThenByDescending(a => a.Media)
                   .Select(a => new {
                                     NomeAluno = a.Nome,
                                     NomeCurso=a.Curso.Nome
                                    } );
```

```
using System.IO;
using System;
using System.Linq;
using System.Collections.Generic;

public class Aluno
{
  public int Numero {get; set;}
  public string Nome {get; set;}
  public int Idade {get; set;}
}

class Program
{
    static void Main()
    {
        Aluno[] alunos = {
            new Aluno() {Numero = 1, Nome="Jose",Idade=23},
            new Aluno() {Numero = 3, Nome="Bruno" ,Idade=25},
            new Aluno() {Numero = 4, Nome="Rafaela" ,Idade=11},
            new Aluno() {Numero = 2, Nome="Arlindo", Idade=12},
        };

        var alunosIdade = alunos.Where(a => a.Idade >= 18)
                                .OrderBy(a => a.Nome)
                                .Select(a => new {
                                    NomeAluno = a.Nome,
                                    IdadeaLuno = a.Idade
                                });

        foreach(var x in alunosIdade)
            Console.WriteLine(x);
    }
}
```

```
$mcs *.cs -out:main.exe

$mono main.exe
{ NomeAluno = Bruno, IdadeaLuno = 25 }
{ NomeAluno = Jose, IdadeaLuno = 23 }
```

- *SelectMany()*
  - Permite efetuar o *select* de uma colecção de colecções e "passa" de um IEnumerable<IEnumerable<T>> para IEnumerable<T>

```
var disAlunos = alunos.Where(a => a.Media >= 15)
                      .OrderBy(a => a.Nome)
                      .ThenByDescending(a => a.Media)
                      .SelectMany(a => a.Disciplinas);
```

(local variable) IEnumerable<Disciplina> disAlunos

```
foreach (var a in disAlunos)
        Console.Write(a.Nome + ",");
```

```
var alunosExcelentes = db.Alunos.Where(a => a.Media >= 18)
                                .OrderBy(a => a.Nome)
                                .Select(a => a.Nome);

foreach (var aluno in alunosExcelentes)
    Console.WriteLine(aluno.Nome);
```

Isto está mal

Como usamos o Select lá em cima, estamos a transformar o Objeto para o mesmo apenas conter agora o Nome, logo temos de utilizar apenas a palavra "aluno" no Console.WriteLine

LINQ - GroupBy

- *GroupBy()*
  - Permite agrupar os dados em grupos diferentes

Com QS

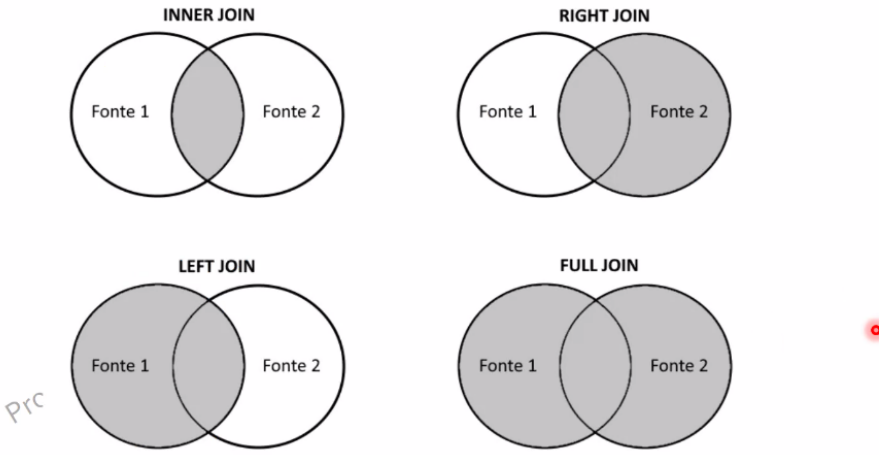
```
var alunos = from a in alunos
             group a by a.Media
             into g
             select g;

foreach (var grupo in alunos){
    Console.Write("{0} - {1}", grupo.Key, grupo.Count());
    foreach (var aluno in grupo)
        Console.Write("\t{0}", aluno.Nome);
}
```

Com EM

```
var grupos = alunos
             .GroupBy(a => a.Media);

foreach (var grupo in alunos)
{
    Console.Write(grupo.Key);
    foreach (var aluno in grupo)
        Console.Write("\t{0}", aluno.Nome);
}
```

Aula 29/11/2021

LINQ Joins



Inner Join

- **Exemplo**: Obter a lista comum entre as listas

```
var nomes1 = new List<string>() { "Jose", "Maria", "Filipa","Carlos" };
var nomes2 = new List<string>() { "Filipa", "Carlos", "Pedro","Joana","Nuno"};
```

```
var resultado = from n1 in nomes1
                join n2 in nomes2 on n1 equals n2
                select new { n1, n2 };
```

```
var  resultado = nomes1.Join(nomes2, n1 => n1, n2 => n2,
                                (n1, n2) => new { n1, n2 });
```

Group Join

- Query syntax

  - Total de alunos por curso

Curso	Alunos
Licenciatura em Engª Informática e de Sistemas	4
Mestrado em Engenharia Informática	0

```
var alunos = from c in db.Cursos
             join a in db.Alunos on c.CursoId equals a.CursoId into g
             select new { NomeCurso = c.Nome, TotalAlunos = g.Count()};
```

- Extension method

  - Total de alunos por curso

Curso	Alunos
Licenciatura em Engª Informática e de Sistemas	4
Mestrado em Engenharia Informática	0

```
var alunosPorCurso = cursos
            .GroupJoin(alunos, c => c.CursoId, a=>a.CursoId,
                       (curso, alunos) => new {
                          NomeCurso = curso.Nome,
                          TotalAlunos=alunos.Count()
                       });
```

Cross Join

- Query syntax
  - Lista de todos os cursos de todos os alunos:

```
var alunosQ = from c in cursos
              from a in alunos
              select new {
                NomeCurso = c.Nome,
                NomeAluno = a.Nome
              };
```

- Extension method

  - Lista de todos os cursos de todos os alunos

```
var alunosQ = cursos.SelectMany(c => alunos,
                                (curso, aluno) => new {
                                    NomeCurso = curso.Nome,
                                    NomeAluno = aluno.Nome
                                });
```

Any

- Retorna um boleano
- Existe algum aluno com media superior a 10???

All

- Retorna um boleano
- Todos os alunos têm média positiva????

```
var todosComMediaPositiva = alunos.All(a => a.Media > 10);
```

Skip

- Permite saltar um determinado numero de elementos e retorna os restantes

Take

- Retorna um numero especifico de elementos

```
var lAlunos = alunos.Skip(3).Take(2);
```

TakeWhile

- Retorna elementos que satisfaçam uma determinada condição. Quando a condição já for verdadeira, termina

SkipWhile

- Permite fazer o skip de elementos em sequência baseados numa condição. Quando deixar de ser verdadeira, retorna os restantes elementos

First / Last

- Retorna o primeiro/último elemento da sequência ou o primeiro/último elemento correspondente ao predicado especificado
- se não existirem elementos na sequência, é gerada uma excepção **Invalid Operation Exception** com a mensagem "Sequence contains no elements"

FirstOrDefault / LastOrDefault

- Retorna o primeiro/último elemento da sequência ou o primeiro/último elemento correspondente ao predicado especificado

Single

- Se a sequência contém exactamente um elemento, ouexactamente um elemento que satisfaça a condição especificada no predicado, então esse elemento é devolvido

SingleOrDefault

- Igual ao comportamento do Single(), no entanto se nãoexistirem elementos ou se não existirem elemetos que satisfaçam a condição do predicado, em vez de gerar excepção, é retornado o valor por omissão do tipo correspondente default(T)

ElementAt

- Retorna um elemento da sequência baseado na posição index especificada

ElementAtOrDefault

- Igual ao comportamento do Element(), no entanto se nãoexistir elemento no indice especificado, retorna o valor por omissão

Agregação

Count

- ***Count()*** - Contagem

```
var num = alunos.Count();
```

```
var quant = alunos.Where(a=>a.Media > 10)
                  .Count();
```

```
var quant = alunos.Count(a=>a.Media > 18);
```

Sum

- ***Sum()*** - Somatório

```
var quant = alunos.Where(a=>a.Media)
                  .Sum();
```

```
var quant = alunos.Sum(a=>a.Media);
```

Min / Max

- ***Min()*** / ***Max()***
  - Permite obter o valor mínimo/máximo de uma *list* ou *collection*

```
int[] num = {2,1,10,4,5,6,7,8,9};
int minimo = num.Min();
int maximo = num.Max();
```

```
var mediaMaior = alunos.Max(a => a.Media);
```

Average

- ***Average()***
  - Permite calcular a média

```
var quant = alunos.Select(a => a.Media).Average();
```

```
var quant = alunos.Average(a => a.Media);
```

Conjuntos (Union, Intersect, Concat, Except, Distinct)

- ***Union()***
  - Remove elementos repetidos
- ***Intersect()***
- ***Concat()***
  - Não remove os elementos repetidos
- ***Except()***
  - Implica distintos, portanto, remove elementos repetidos
- ***Distinct()***

Exercicio -> Selcionar os alunos cujo nome começa por "F"

- Seleccionar os alunos cujo nomes começam por “F”

```
Aluno[] alunos = {
        new Aluno() { Numero = 1, Nome= "Jose Antunes", Idade= 18 },
        new Aluno() { Numero = 2, Nome = "Filipa Moreira",  Idade = 21 },
        new Aluno() { Numero = 3, Nome = "Cristina Fria",  Idade = 25 },
        new Aluno() { Numero = 4, Nome = "Dinis Campos" , Idade = 20 },
        new Aluno() { Numero = 5, Nome = "Soaraia Goncalves" , Idade = 31 },
        new Aluno() { Numero = 6, Nome = "Lena Pinheiro",  Idade = 17 },
        new Aluno() { Numero = 7, Nome = "Filipe Cruz",Idade = 19  },
 };
```

```
var alunosF = alunos.Where( (a) => {
                return (a.Nome.StartsWith("F"));
            }  );
```

```
var alunosF = alunos.Where( a =>
                a.Nome.StartsWith("F"));
```

Programação Web – 2021/2022 – LEI D+CE+PL – ISEC/IPC @JB

Exercicio -> Selcionar os alunos cujo o ultimo nome começa por "F"

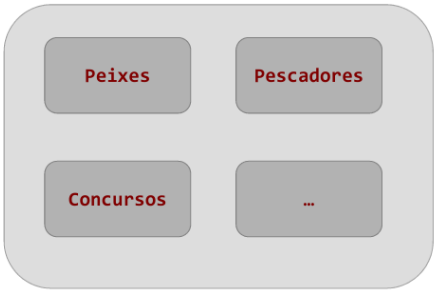
```
Aluno[] alunos = {
        new Aluno() { Numero = 1, Nome= "Jose Antunes", Idade= 18 },
        new Aluno() { Numero = 2, Nome = "Filipa Moreira",  Idade = 21 },
        new Aluno() { Numero = 3, Nome = "Cristina Fria",  Idade = 25 },
        new Aluno() { Numero = 4, Nome = "Dinis Campos" , Idade = 20 },
        new Aluno() { Numero = 5, Nome = "Soaraia Goncalves" , Idade = 31 },
        new Aluno() { Numero = 6, Nome = "Lena Pinheiro",  Idade = 17 },
        new Aluno() { Numero = 7, Nome = "Filipe Cruz",Idade = 19  },
 };
```

```
var alunosF = alunos.Where( a => a.Nome.Split(' ')
                            .Last()
                            .StartsWith("F"));
```

Aula 06/12/2021

MVC

Model




```
namespace MVC1.Models
{
    public class Pescador
    {
        public int Id { get; set; }
        public string Nome { get; set; }
    }
}
```

View

- Interface do utilizador para apresentação dos ados, isto é, representa a UI da aplicação

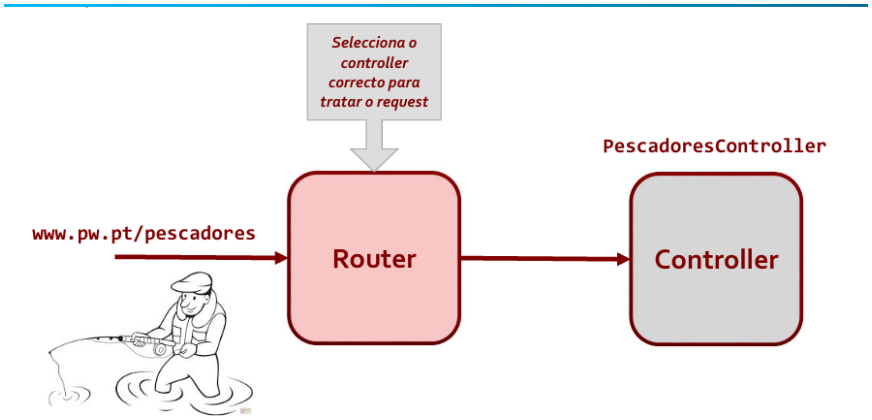
```
@{
    ViewBag.Title = "Vencedor";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Vencedor</h2>
```

Controller

- Traduz as ações do utilizador em operações apropriadas
- **Cérebro da operação**, responsável por tratar do pedido HTTP

Routing

- A framework ASP.NET Core recorre a URLs amigáveis que permitem descrever melhor a ação do user



Estrutura ASP.NET Core

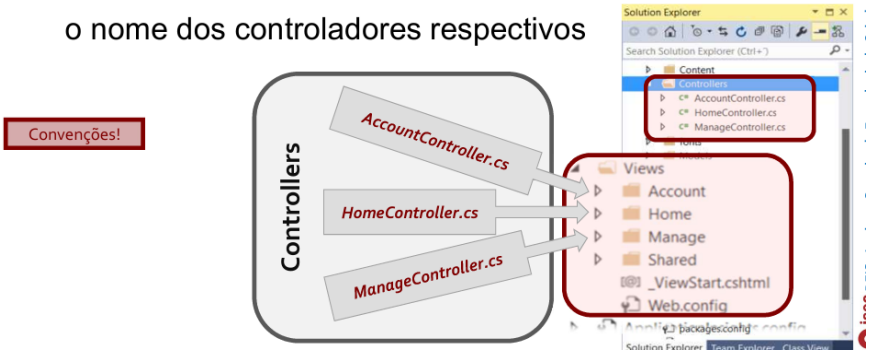
Estrutura de diretórios

- **App_Data**
  - Base de dados
- **App_Code**
  - Classes onde se pode incluir código
- **Content**

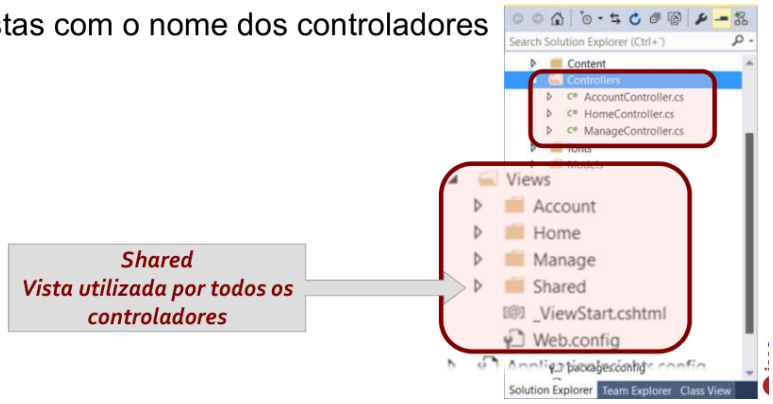
- Ficheiros CSS,imagens

- **Controllers**
  - Controladores
- **packages.config**
  - Utilizador pelo *NuGet package* e permite gerir as dependências da aplicação

    - Os ficheiros das Views são colocados em Pastas com o nome dos controladores respectivos



    - Pastas com o nome dos controladores



Action Methods

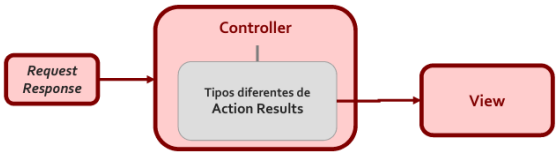
- Todos os métodos públicos da classe *Controller* são designados como **Action methods**

```
using System.Web.Mvc;
using MVC1.Models;
namespace MVC1.Controllers
{
  public class PescadoresController:Controller
    {
      public ActionResult Vencedor()
      {
          var pescador = new Pescador() {
                Nome = "Nuno Manuel Pereira"
              };
          return View();
      }
    }
}
```

- Devem ser públicos
- Não podem ser *overloaded*
- *Não podem ser estáticos*

```
using System.Web.Mvc;
using MVC1.Models;
namespace MVC1.Controllers
{
    public class PescadoresController : Controller
    {
        // GET: Pescadores/Vencedor
        public ActionResult Vencedor()
        {
            var pescador = new Pescador() { Nome = "Nuno Manuel Pereira" };
            return View();
```

Retorna uma ActionResult usando o método View() definido na class Controller



- Os controladores podem ter um método action por omissão, especificado no **RouteConfig**

```
routes.MapRoute(
   name: "Default",
   url: "{controller}/{action}/{id}",
   defaults: new {controller="Home",
                  action="Index",
                  id = UrlParameter.Optional } );
```

Action Selectors

- São atributos que podem ser aplicados aos *Action Methods*
- Permitem o mecanismo de routing identificar o action correto para manipular determinado pedido

- *ActionName*
  - Permite especificar outro nome para a action, que não o nome do método;
- *NonAction*
  - Indica que um método público de um controlador não deva ser considerado um *action method.*

```
using System.Web.Mvc;
using MVC1.Models;

namespace MVC1.Controllers
{
    public class PescadoresController : Controller
    {
        [ActionName("Winner")]
        public ActionResult Vencedor()
        {
          ...
        }
    }
}
```

- *ActionVerbs*
  - Utilizado quando se pretende controlar a selecção de um *action method* com base no metodo do Http request
    - [HttpGet] - Restringe um action method a pedidos HTTP GET.
    - [HttpPost] - Restringe um action method a pedidos HTTP POST.
    - [HttpPut] - Restringe um action method a pedidos HTTP Put.
    - [HttpDelete] - Restringe um action method a pedidos HTTP Delete.
    - `[AcceptVerbs(HttpVerbs.Get | HttpVerbs.Delete)]` – Combinação de vários.

GET vs POST vs PUT vs DELETE

- **GET**
  - Método genérico para qualquer pedido que **obtem** dados do servidor
- **POST**
  - Método genérico para qualquer pedido que **envia** dados para o servidor
- **PUT**
  - Método especifico para atualização de dados no servidor

```
@using (Html.BeginForm("Edit", "Exemplo", FormMethod.Post))
 {
    @Html.HttpMethodOverride(HttpVerbs.Put)
    ...
 }
```

- **DELETE**
  - Método especifico para remoção de dados no servidor

```
@using (Html.BeginForm("Remove", "Exemplo", FormMethod.Post))
 {
     @Html.HttpMethodOverride(HttpVerbs.Delete)
     ...
 }
```

O que é uma action?

- O que é uma *action*?
  a) Uma regra que determina qual o método do controlo que deva ser invocado
  b) Um método num controlo responsável por manipular um pedido
  c) Um método que é chamado durante o *startup*

Um pedido a /clientes/remover/5 por conveção é manipulado por?

- Um pedido a /clientes/remover/5 por convenção é manipulado por?
  a) Clientes.Index()
  b) Clientes.Remover()
  c) Clientes.Remover(int id)
  d) ClientesController.Remover(int id)

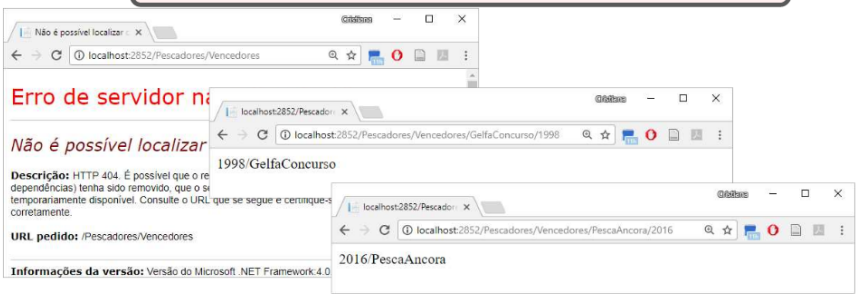
Personalziar o Routing de forma a aceitar o que se pretende

```
public static void RegisterRoutes(RouteCollection routes)
{
   routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

   routes.MapRoute(
     name: "Default",
     url: "{controller}/{action}/{id}",
     defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}
```

A ORDEM IMPORTA!

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute("PescadoresPorVencedoresConcursoAno",
            "Pescadores/Vencedores/{concurso}/{ano}",
            new { Controller="Pescadores", action="VencedoresConcursoAno" });

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
    );
}
```

```
public ActionResult VencedoresConcursoAno(string concurso, int ano)
{
    return Content(ano + "/" + concurso);
}
```



Convention Based Routing

- Problemas
  - Ficheiro *routing* irá aumentar…
  - Torna-se confuso…
  - Ao renomear as actions os routings não se atualizam

Criação de Routing "Limpas"

Attribute Routing

Aula 20/12/2021

Passagem de dados (Controlador - Vista - Controlador)

- A passagem de dados do **Controlador** à **Vista**, e ao próximo request, pode ser efetuada de outras formas para além da passagem de dados através de um

argumento da View

- **Podem ser usados dados fortemente tipados: Viewmodel**
- **Podem ser usados dados fracamente tipados: ViewData e ViewBag**

Para dados fortemente tipados (viewmodel)

- A abordagem mais robusta é especificar um tipo de modelo na view. Este modelo é comumente referido como um ***viewmodel***. É passada uma instância do tipo ***viewmodel*** para a ***view*** da ***action***
- Usar um ***viewmodel*** para passar dados para uma ***view*** permite que a ***view*** tire proveito da forte verificação de tipo. A tipagem forte (ou fortemente tipada) significa que cada variável e constante tem um tipo explicitamente definido (exemplo, *string*, *int, DateTime, etc*)
  - A validade dos tipos usados numa *view* é verificada em tempo de compilação.

Para dados fracamente tipados (viewdata e viewbag)

- Ao contrário dos tipos fortes, os tipos fracos (ou tipos soltos) significam que não se declara explicitamente o tipo de dados que irão ser passados
- Pode-se usar esta abordagem para passar pequenas quantidades de dados para dentro e para fora de controladores e de vistas.
- A ter em conta que:
  - **ViewData**
  - **ViewBag**

Utilizados para comunicar entre o controlador e a vista correspondente, mas apenas para chamadas ao servidor.
*Torna-se **null** se ocorrer redireccionamento.*

Mecanismo para **manter estado** entre o controlador e a vista correspondente

Para isto usa-se o TempData

- ***ViewData*** e ***ViewBag*** são resolvidos dinamicamente em tempo de execução.
- Uma vez que eles não oferecem verificação de tipo em tempo de compilação, ambos são geralmente mais sujeitos a erros do que usar um modelo de visualização.
  - Por esse motivo não é consensual a utilização de ***ViewData*** e ***ViewBag***

Em suma

- ***ViewData*** *- Objeto do tipo dicionário*
  - Acessível usando *strings* como chaves
  - Necessita do *type casting* para tipos complexos

- **ViewBag** – *Objecto do tipo dinâmico*
  - Não necessita de *type casting* nem verificações de *null*
  - *Propriedades adicionadas em tempo de execução*

- ***TempData*** *- Objeto do tipo dicionário*
  - Permanece durante o tempo de um pedido HTTP
  - *Pode ser usado para manter dados entre redireccionamentos*

Passagem de dados: ViewData

Controller

```
public ActionResult Vencedor()
        {
            var pescador = new Pescador() { Nome = "Nuno Manuel Pereira" };
            ViewData["PescadorData"] = pescador;
            return View();
        }
```

View

```
@using MVC1.Models
@model MVC1.Models.Pescador
@{
    ViewBag.Title = "Vencedor";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@(((Pescador)ViewData["PescadorData"]).Nome)</h2>
```

Passagem de dados: ViewBag

Controller

```
public ActionResult Vencedor()
        {
            var pescador = new Pescador() { Nome = "Nuno Manuel Pereira" };
            ViewBag.PescadorProp = pescador;
            return View();
        }
```

View

```
@using MVC1.Models
@model MVC1.Models.Pescador
@{
    ViewBag.Title = "Vencedor";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>@ViewBag.PescadorProp.Nome</h2>
```

Views

- **A view não deve executar qualquer lógica de negócio, ou interagir com a base de dados diretamente**
- **Deve trabalhar somente com os dados que lhe são fornecidos pelo controlador**

```
public ActionResult Index()
{
    return View();
}
```

→ Renderiza a vista Index.cshtml

```
public ActionResult Index()
{
    return View("OutraView");
}
```

→ Renderiza a vista OutraView.cshtml

Razor View Engine

- Introduzido com o ASP.NET MVC 3
- Sintaxe simplificada
- Minimiza a quantidade de sintaxe e caracteres extra
- Codifica automaticamente qualquer *output* enviado através do @ de forma a prevenir ataques *cross site scripting*
- Razor suporta expressões e blocos de código
  - Razor View Engine

Algum erro?

```
@{
    var nome = "Cristiana";
    var Apelido ="Areias";
    <div>
        Nome: @nome
        Apelido: @apelido
    </div>
}
```

Algum erro?

```
@{
    var txt = "";
    if (DateTime.Now.Hour > 12)
        txt = "Boa tarde!";
    else
        txt = "Bom dia!";
}
```

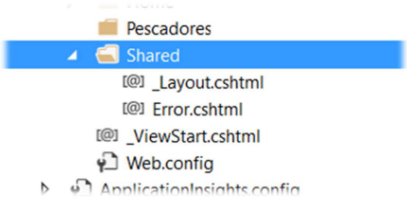
Sim, faltam {} no if e no else

Algumas regras

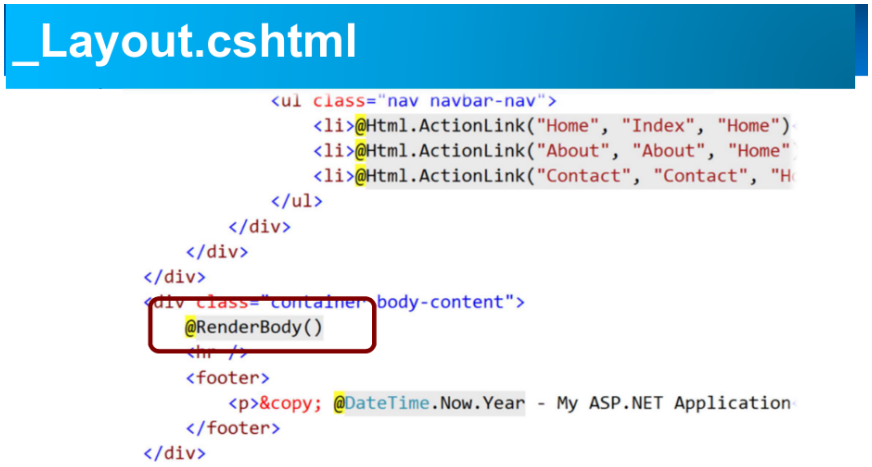
- Blocos de código estão delimitados com @{ ... }
- Expressões *Inline* (variáveis e funções) iniciam com @
- Declarações de código terminam com ;
- Variáveis são declaradas com a palavra chave *var*
- Strings devem estar entre aspas
- Código C# é *case sensitive*
- Ficheiros Razor tem extensão .cshtml

Layout com Razor

- _Layout.cshtml
  - Define o que se quer apresentar em todas as páginas da aplicação
  - Recorre a métodos herdados para especificar áreas de conteúdo
    - **RenderBody**
    - **RenderSection**



- **RenderBody**
  - Permite especificar onde será apresentado o conteúdo individual das views
  - É obrigatorio a sua chamada numa vista layout



- **RenderSection**
  - Chamada opcional numa *Layout View*
  - Fornece uma *content view* como a view Index, permitindo aplicar conteúdo em outras secções da página
  - A secção necessita de ter um nome e estar definida no ficheiro content view

RenderSection

```
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
                </ul>
            </div>
        </div>
    </div>
    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @RenderSection("Footer")</p>
        </footer>
    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
```

Alteração da *View*: About

```
@{
    ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>

<p>Use this area to provide additional information.</p>

@section Footer {
    Isto <em>foi criado para demonstração</em> nas aulas de PW!!...
    }
```

- **Partial Views**
  - Permite especificar código HTML e C# num ficheiro para que depois possa ser reutilizado em várias vistas
  - Poder ser usado simplesmente para simplificar uma vista
  - Por convenção deve-se especificar no prefixo do nome, o caracter _
  - Caso seja utilizada por várias vistas, deverá ser colocada na pasta *shared*, caso contrário coloca-se na pasta onde irá ser utilizada

- Para *renderizar* uma vista parcial, deve-se recorrer ao *html helper* Html.Partial
  - Deve ser passado o nome da vista parcial e o modelo que necessita

```
@foreach (var item in Model)
{
    @Html.Partial("_Rever",item)
}
```

Tag Helpers

- **HTML Helpers e URL Helpers**
  - Nas views e a partir do ASP.Net MVC utilizavam-se os **HTML Helpers** e os **URL Helpers** com o objetivo de facilitar a criação de views.
    - Os HTML Helpers podem ser invocados a partir da propriedade **Html** de uma view
    - URL Helpers podem ser invocados via propriedade Url. *Os URL Helpers também estão disponíveis nos controladores.*
    - *Também se podem utilizar os* Ajax Helpers via *propriedade Ajax.*

- **Tag Helpers**
  - São um novo recurso disponivel no ASP.Net Core no entanto continua a ser possível a utilização de HTML.Helpers
  - A sintaxe é semelhante à sintaxe do HTML mas é processado pelo Razor no servidor
  - Estes são uma alternativa aos **Html Helpers**

- A renderização pelo ***RAZOR*** de uma ***TextBox*** utilizando o ***Tag Helper*** <input … > resultaria em:

<input type="text" asp-for="Nome" />	**<input id="nome" name="nome" type="text" value="" />**

```
<form asp-action="Create" asp-controller="Aluno">
    //Os elementos do formulário
</form>
```

- A renderização deste ***Tag Helper*** geraria o seguinte ***HTML***:

```
<form action="Create" method="post">
    //Os elementos do formulário
   <input name="__RequestVerificationToken" type="hidden"
value="ABCDEFabcd34456 .....”</n>>
</form>
```

Tag Helpers

HTML Helpers	Tag Helpers

@using (Html.BeginForm("Criar"))
{
@Html.AntiForgeryToken()
//conteúdo
}
	<form asp-anti-forgery="false" asp-action="Criar">
	<!--conteúdo-->
	</form>

@Html.LabelFor(model => model.Nome, new { @class = "control-label col-md-2" })	<label asp-for="Nome" class="control-label col-md-2" />
@Html.EditorFor(model => model.Nome)	<input type="text" asp-for="Nome" />
@Html.ValidationMessageFor(model => model.Nome)	
@Html.DropDownListFor(model=>model.Tags)	<select asp-for="Tags" asp-items="(IEnumerable<SelectListItem>)Model.Tags" size="20" class="form-control"> <option value="">-- Selecione --</option> </select>

Aula 03/01/2022

Entity Framework (EF)

- Aumentar a produtividade do programador, reduzindo tarefas redundantes para persistência de dados

- Basicamente a **EF Core** é uma versão leve, extensivel e multiplataforma do **EF**
- O **EF Core** introduz muitas melhorias e novos recursos quando comparado com as versões anteriores
- Então o **EF Core** serve como uma espécie de *object relational mapper* para conseguirmos usar modelos de .NET e esses modelos representarem uma tabela de uma base de dados

Tipos de carregamentos de objetos

- **Eagerly Loading**
- **Lazy Loading**
- **Explicity Loading**

Formas para especificar o modelo de dados

- **Code First**
  - Parte-se das classes do domínio e a **EF** gera a base de dados
- **Database First**
  - Parte-se da base de dados para obter as classes de domínio

Code First

- Conjunto de regras que permite configurar de forma automática um modelo conceptual, a partir do modelo de classes

Utilização da Entity Framework em Code First

1. Instalação da ***Entity Framework***
2. *Definição do* ***Modelo de Domínio***
3. Criação do ***Contexto*** (classe que herda da ***DbContext*** e que é uma abstração da base de dados usada no acesso aos dados
4. Registar no método ***ConfigureServices*** de ***Startup.cs*** o ***DBContext*** como um serviço
5. Especificação da ***Connection String*** no ficheiro ***appsettings.json***

- **Então basicamente o que fazemos é**
  - Criamos uma classe que representa o nosso modelo(tabela)
  - Depois colocamos uma definição desse modelo no nosso DBCcontext

```
using FuncionariosWeb.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using System.Linq;

namespace FuncionariosWeb.Context
{
    11 references
    public class AppDbContext : IdentityDbContext<ApplicationUser>
    {
        0 references
        public AppDbContext(DbContextOptions<AppDbContext> options)
      : base(options)
        { }
        7 references
        public DbSet<Funcionario> Funcionarios { get; set; }
        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            foreach (var foreignKey in modelBuilder.Model.GetEntityTypes().SelectMany(e => e.GetForeignKeys()))
            {
                foreignKey.DeleteBehavior = DeleteBehavior.Restrict;
            }
        }
    }
}
```

- Registamos o DBContext como um serviço no *ConfigureServices* do **Startup.cs**

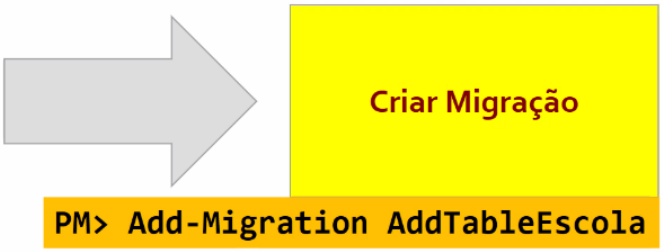
```
Startup.cs
FuncionariosWeb.Startup                    Startup(IConfiguration configuration)

2 references
public IConfiguration Configuration { get; }

// This method gets called by the runtime. Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDbContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
    services.AddIdentity<ApplicationUser, IdentityRole>()
                .AddEntityFrameworkStores<AppDbContext>();
    services.AddControllersWithViews();

    //Bloqueio global - só permite users logados
//    services.AddControllersWithViews(config =>
//    {
//        var policy = new AuthorizationPolicyBuilder()
//                .RequireAuthenticatedUser()
//                .Build();
//        config.Filters.Add(new AuthorizeFilter(policy));
//    });
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
issues found                                                                    Ln: 1
```

- Especificamos a *connection string* no ficheiro **appsettings.json**

```
ution Explorer    appsettings.json*
ema: https://json.schemastore.org/appsettings.json
1  {
2    "ConnectionStrings": {
3      "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=FuncionariosWeb;Trusted_Connection=True;MultipleActiveResultSets=true"
4    },
5    "Logging": {
6      "LogLevel": {
7        "Default": "Information",
8        "Microsoft": "Warning",
9        "Microsoft.Hosting.Lifetime": "Information"
10     }
11   },
12   "AllowedHosts": "*"
13 }
```

- Depois aplicamos a migração e atualizamos a base de dados
- Se quisermos fazer uma grande alteração num site, o melhor seria usar o database first porque no fundo não queremos alterar o que ja contem na base de dados, queremos manter esses dados
- Antes de fazermos as Migrations temos de fazer Build

```
namespace CodeFirstFromDatabase.Models
{
    public class Escola
    {
        public int EscolaId { get; set; }
         [Display(Name="Descrição")]
        public string Descricao { get; set; }
    }
}
```



Migrattions – Passos Principais

Usar Migrações Code First

2 Passos Principais:

1. **Adicionar as alterações**

   PM> Add-Migration *darumnomequalqueraadicao*

   Se falhar, fazer o Clean da Solution e a seguir o Rebuild

3. **Actualizar a Base de Dados**

   PM> Update-Database

Mas qual o motivo das migrações?

- A EF Core constroi o modelo EF Core a partir das classes de dominio e as migrações EF Core criarão ou atualizarão o esquema da base de dados com base no modelo EF Core
- Sempre que as classes de domínio são alteradas, é preciso executar uma migração para manter o esquema da base de dados atualizado

Como recuperar de um erro de uma migração?

- **Não convem remover a migração, alteramos o modelo e criamos uma nova migração**

Code-First Conventions por omissão

Convenções	Descrição
Nome das Tabelas	<Nome da classe da Entidade> + 's' EF cria a tabela da DB com o nome da class da entidade, e sufixo 's'
Nome da Chave Primária	1) Id 2) <Nome da classe da Entidade> + "Id" (Id não é sensível a minusculas) EF cria a coluna da chave primária par a propredade com nome Id ou então <Nome da Classe da Entidade> + "Id"
Nome da Chave Estrangeiras (Foreign key)	Por omissão, EF procura porpropriedades das chaves estrangeiras cujo nome seja igual ao nome da chave da entidade principal. Se a chave estrangeira náo existir, então a EF cria uma coluna FK na tabela com o nome <Nome da Propriedadede Navegação Dependente>+_+<Nome da chave da propriedade da entidade principal>
Colunas Nulas	EF especifica colunas nulas para todo o tipo de propriedades especificadas com tipos por referencia e propriedades com tipos nullable.
Colunas não Nulas	EF cria colunas não nulas para propriedades que sejam chaves primárias e para tipos que não sejam nullable
Ordem das Colunas na BD	EF cria a ordem das colunas com a mesma ordem que as propriedades estão especificadas na classe da entidade. No entanto, as colunas de chave primaria podem ser movidas para a 1ª posição
Mapeamento das Propriedades na BD	Por omissão, todas as propriedades são mapeadas para a base de dados ac não ser que se use o atributo [NotMapped] na propriedade ou classe.
Remoção em Cascata	Ativado por omissão para todos os tipos de relacionamento.

DataAnnotations vs Fluent API

- **DataAnnotations**
  - Mais simples de especificar
  - Menos fléxivel
  - Útil para anotações simples como obrigatórios, tamanhos maximos, etc

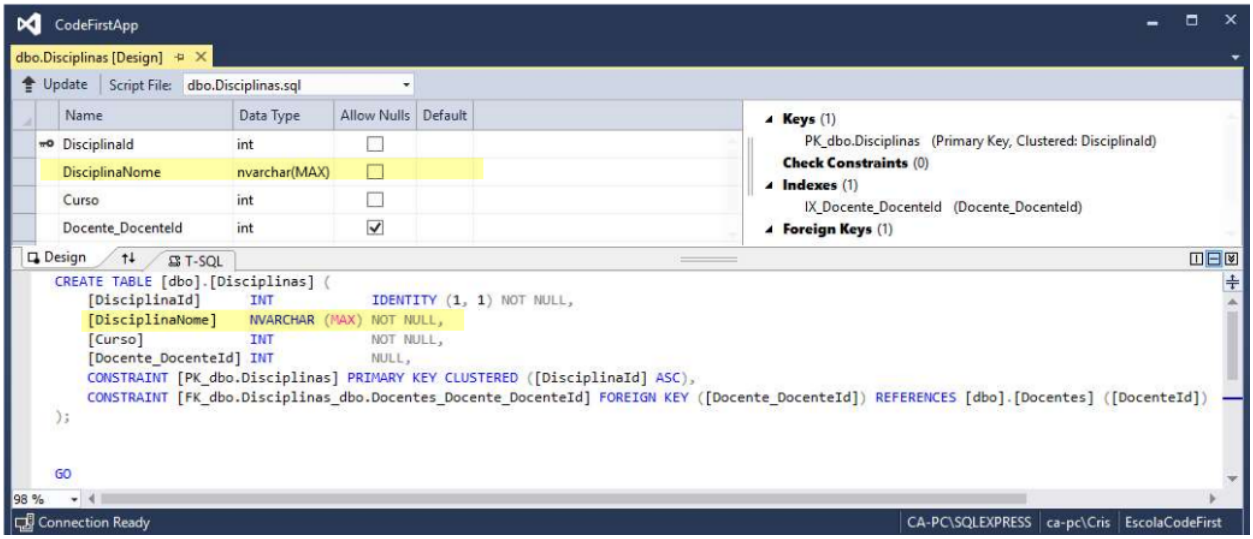
```
public partial class disciplinaDataAnnotation : DbMigration
    {
        public override void Up()
        {
          AlterColumn("dbo.Alunoes", "AlunoNome", c => c.String(nullable: false, maxLength: 150));
        }

        public override void Down()
        {
            AlterColumn("dbo.Alunoes", "AlunoNome", c => c.String());
        }
    }
```

Passamos a ter o AlunoNome como not null e um máximo de 150 chars

- **FluentAPI**
  - Permite a configuração de relacionamentos entre classes mais complexas
  - Permite a separação de concerns

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Disciplina>()
                .Property(t => t.DisciplinaNome)
                .IsRequired();

            ...
```

Fluent API



- O entity framework é utilizado quer para o database first quer para o code first

Fluent API

- O código deve ser aplicado no método **OnModelCreating(DbModelBuilder modelBuilder)** da class que deriva do DbContext
- **Por exemplo para alterar o nome de uma tabela**

```
protected override void OnModelCreating(DbModelBuilder
modelBuilder)
{
        modelBuilder.Entity<Aluno>()
                .ToTable("Alunos");
}
```

- **Por exemplo para criar uma PK**

- Chave Primária

```
modelBuilder.Entity<Aluno>()
                . HasKey(t => t.NAluno);
```

- **Relacionamentos**

- Métodos de acordo com o tipo de relacionamento:
  - **Has**Many() – Se A tem muitos objetos to B
  - **Has**Required – Se A tem apenas um objeto de B
  - **Has**Optional – Se A tem 0 ou 1 objeto de B

- Métodos de acordo com o tipo de relacionamento:
  - **With**Many() – Se B tem muitos objetos to A
  - **With**Required – Se B tem apenas um objeto de A
  - **With**Optional – Se B tem 0 ou 1 objeto de A

- *Professor* leciona várias *Disciplinas*
- *Disciplina* é lecionada apenas por 1 *Professor*

```
modelBuilder.Entity<Professor>()
            .HasMany(p => p.Disciplinas)
            .WithRequired(d => d.Professor)
            .HasForeignKey(d => d.ProfessorId);
```

Entity Framework - Database First

O ***Entity Framework Database First*** é uma abordagem alternativa no desenvolvimento de sites para a *Web* utilizando-se o ***ASP.Net Core Entity Framework***.

- Pode ser utilizada quer se disponha já de uma base de dados, por exemplo a utilizada num site que se quer actualizar ou quer se tenha desenvolvido especifica e previamente uma Base de Dados para um novo site
  - No entanto é comum que no desenvolvimento de raíz de um site, se utilize a abordagem *Code First*

Nesta situação de se dispor já de uma Base de Dados, se se adoptasse a abordagem *Code First*, poderiam ocorrer alguns problemas:

- Ter-se-ia de se definir manualmente cada uma das classes C# correspondentes a cada uma das tabelas e as respectivas relações
- Não se poderia usar migrações *Code First* porque a base de dados existente provavelmente estará a ser gerida de uma maneira diferente.
  - Embora isto não fosse um problema, perder-se-ia uma das grandes vantagens de se usar o *Code First*

Deste modo, nestas situações é preferível usar a abordagem ***Data Base First***:

- O que se faria criando manualmente o código das classes ***C#***, será feita pela ***EF*** através da inspecção da base de dados existente.
- O resultado dessa acção de análise da *BD* pela ***EF*** será então a criação das classes de dados ***C#***

- **Para usarmos o Database First temos de ter previamente uma Base de Dados, e para isso podemos usar o SQL Server do Visual Studio**

- Basicamente aqui temos uma base de dados e depois fazemos o Scaffold para gerar os modelos

Identity Framework

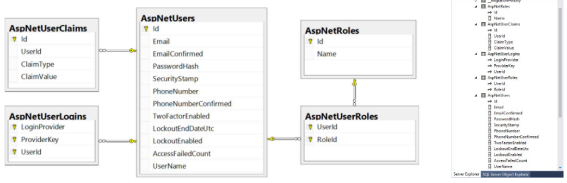
- **Autenticação**
  - Processo de verificação de identidade de um utilizador ou computador
- **Autorização**
  - Processo para determinar o que é que o utilizador pode aceder num determinado computador, rede, ...
- Então o Identity Framework é uma ferramente de **autenticação** e **autorização** para aplicações ASP.NET Core
- O mesmo facilita a escrita de testes unitários
- Permite login externo (**OAuth 2.0**)
- **Ao fazer login, a informação fica armazenada num cookie**

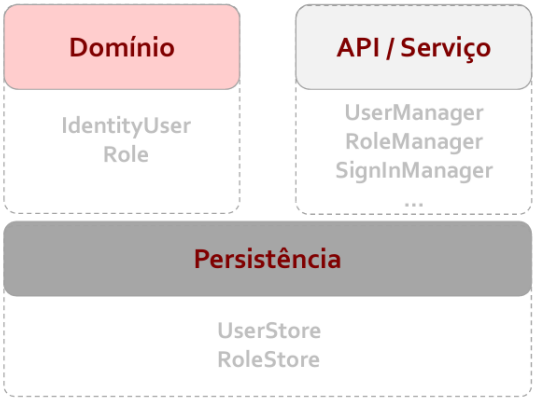
Formas de autenticação

- **No authentication**
  - Como o nome indica, sem autorização
- **Individual User Accounts**
  - Tipicamente utilizado para websites
  - Permite a criação de utilizadores locais

Arquitetura

- Geralmente, os dados de identidade do ***ASP.NET Core*** (utilizadores, *passwords*, perfis) são armazenados numa base de dados relacional por meio da ***EF Code First***




Domínio
IdentityUser
Role
API / Serviço
UserManager
RoleManager
SignInManager
...
Persistência
UserStore
RoleStore