FIAP GRADUAÇÃO

FIAP

ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#04 – RAZOR E TAG HELPERS

 *profthiagoy@fiap.com.br*  *thiagoyama*

TRAJETÓRIA

- Plataforma .NET
- Linguagem C# e Orientação a Objetos
- ASP.NET Core – Rotas e Controller
- ASP.NET Core – Razor e Tag Helpers

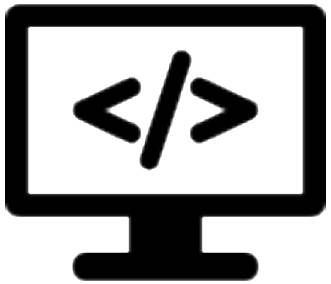
#04 - AGENDA

- Razor
- HTML Helpers
- Tag Helpers
  - Habilitando os Tag Helpers
  - Links e Formulários
  - Input e Selects
  - Label e Display
  - Criando Tag Helpers

RAZOR

RAZOR

- **View engine**, para construção de páginas **dinâmicas**:
- Lançada na versão MVC 3, em **janeiro 2011**;
- Sintaxe simplificada;
- **Fácil de aprender;**
- Possui extensão **.cshtml;**



@

BLOCO RAZOR

- Toda expressão **Razor** começa com o caracter **@**
  - Neste exemplo estamos declarando um bloco para inserir **código C#:**

```
@{
  //Código C#
}
```

```
@{
   for (int i = 0; i< 10; i++)
   {
      <p>@i</p>
   }
}
```

LOOPS E IF-ELSE

- Além do bloco, é possível inserir **comandos C#** de forma mais simples:

```
@if(true)
{
  <p>Olá Adminstrador!</p>
}
else
{
  <p>Olá Usuário</p>
}
```

```
@for(int i = 0; i< 10; i++)
{
  <p>@i</p>
}
```

Para acessar uma variável, utilize a sintaxe

@NomeDaVariavel

HTML Helpers

HTML HELPERS

- **HTML Helpers** é um método utilizado na view para **construir** algum **elemento HTML**, como links, formulários e seus campos;
- **Facilita a construção das páginas** e a interação com os **dados** e **controller**;

```
@Html.ActionLink("Texto do Link","Action","Controller")
```

Gera um **link** para uma determinada **Action** de um **Controller**;

Texto do Link

HTML HELPERS

- Existem vários **Helpers**, para construir diversos **elementos HTML**:

```
@Html.BeginForm("Cadastrar", "Cliente")
        @Html.LabelFor(c => c.Nome)
        @Html.TextBoxFor(c => c.Nome)
@Html.EndForm()
```

Gera um **formulário** para o controller **Cliente** e action **Cadastrar** com um **label** e um campo **input** do tipo texto para a propriedade **Nome** da classe modelo **Cliente**;

- É possível utilizar os **HTML Helpers** no **ASP.NET Core**, porém, existe uma outra forma mais amigável para trabalhar com os elementos HTML, os **Tag Helpers;**

TAG HELPERS

TAG HELPERS

- Permite que o **código** do lado do servidor participe da **criação e renderização de elementos HTML** em arquivos razor;
- Fornece uma **experiência de desenvolvimento** amigável com HTML, pois é muito similar a marcação **HTML padrão**;
- **Desenvolvedores** familiarizados com **HTML/CSS/JS** podem trabalhar com **Razor** sem dificuldades;
- A maioria das **Tag Helpers** são direcionadas a elementos HTML padrão, fornecendo **atributos do lado do servidor** para esse elemento, os atributos começam com **asp-...;**

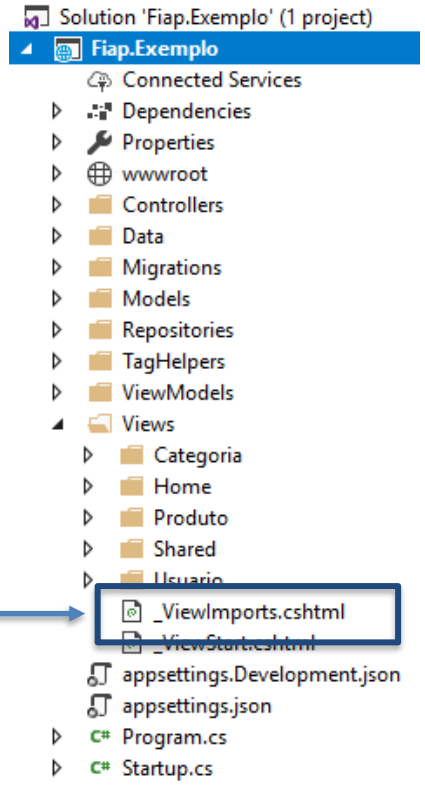
```
<a asp-controller="Home" asp-action="Index">Home</a>
```

HABILITANDO OS TAG HELPERS

- Para utilizar os **Tag Helpers** nas views precisamos modificar o arquivo **_ViewImports.cshtml;**
- **A tag @addTagHelper** disponibiliza as Tag Helpers para as views:

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Arquivo para adicionar **diretivas** e **namespaces comuns** a todas as **views**;



LINKS

- Podemos definir a **action** e o **controller** para o destino de um **link**, caso o controller **não seja fornecido**, será utilizado o **controller atual**;

```
<a asp-controller="Home" asp-action="Index">Home</a>

<a asp-action="Index">Home</a>
```

```
<a asp-action="editar" asp-route-id="@produto.Codigo">Editar</a>
```

```
[HttpGet]
public IActionResult Editar(int id)
{
   //...
}
```

O **link** envia um **valor** para a action através da rota; Lembre-se de que os links enviam requisições **GET**;

FORMULÁRIO

- **Por padrão**, um **formulário HTML** envia requisição **GET** e envia as informações para a **URL atual**;
- Caso algum atributo **asp-..** for definido, o formulário será construído para enviar requisição **POST**;

```
<form asp-action="Cadastrar" asp-controller="Produto">

</form>
```

FORMULÁRIO - MODEL

- Para implementar os **campos do formulário**, vamos utilizar como exemplo o modelo **Produto** abaixo;
- **Categoria** é uma **classe** e o **Estado** é um **enum**;
- O caractere **?** define que a **CategoriaId** pode receber **null**;

```
public class Produto
{
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public string Descricao { get; set; }
    public DateTime DataLancamento { get; set; }
    public decimal Valor { get; set; }
    public Categoria Categoria { get; set; }
    public int? CategoriaId { get; set; }
    public Estado Estado { get; set; }
    public bool Importado { get; set; }
}
```

VIEW IMPORTS

- Nas **views** definimos na primeira linha a **classe Model** que será utilizada na view:

```
@model Fiap.Exemplo.Models.Produto
```

- Para facilitar, é possível modificar o arquivo **_ViewImports.cshtml** para adicionar o **namespace** para todas as views:

```
@using Fiap.Exemplo.Models

@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

- Dessa forma, **não é preciso** definir o **namespace** na página:

```
@model Produto
```

INPUT

- O atributo **asp-for** na tag **input** define os valores para os atributos **id** **name e type**;

```
@model Produto

<input asp-for="Nome"/>

<input asp-for="DataLancamento"/>

<input asp-for="Importado”/>
```

INPUT - TYPE

- O atributo **type** é definido com base no **tipo da propriedade** do model;
- Caso o **type** seja especificado no HTML, **não será sobrescrito** o seu valor;

Tipo .NET	Tipo do input
Bool	checkbox
String	text
DateTime	datetime
Byte	number
int	number
Single e Double	number

INPUT - ANNOTATION

- É possível utilizar **anotações** para auxiliar no **tipo de campo** gerado na view:

Atributo	Tipo do input
[EmailAddress]	email
[Url]	url
[HiddenInput]	hidden
[Phone]	tel
[DataType(DataType.Password)]	password
[DataType(DataType.Date)]	date
[DataType(DataType.Time)]	time

INPUT – ANNOTATION EXEMPLO

- O **ProdutoId** será um campo do **tipo hidden** e a **DataLancamento** do tipo **date**;

```
public class Produto
{
    [HiddenInput]
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public string Descricao { get; set; }

    [DataType(DataType.Date)]
    public DateTime DataLancamento { get; set; }
    public decimal Valor { get; set; }
    public Categoria Categoria { get; set; }
    public int? CategoriaId { get; set; }
    public Estado Estado { get; set; }
    public bool Importado { get; set; }
}
```

LABEL

- A **tag label** também possui o atributo **asp-for** para determinar a propriedade do model para o label:
- O **nome da propriedade** será **exibida** no label da página:

```
@model Produto

<label asp-for="Nome"></label>
<input asp-for="Nome"/>

<label asp-for="DataLancamento"></label>
<input asp-for="DataLancamento"/>

<input asp-for="Importado" />
<label asp-for="Importado"></label>
```

LABEL - DISPLAY

- Para **modificar o valor** que será exibido no label, utilizamos a anotação **[Display]:**

```
[Display(Name ="Data de Lançamento")]
public DateTime DataLancamento { get; set; }
```

SELECT - ENUM

- Para trabalhar com **enum**, podemos utilizar a **tag select** com o atributo **asp-items**, para definir os valores (options) do select;
- O **HTML Helper @Html.GetEnumSelectList()** recupera os **valores** definidos no **enum**;

```
<label asp-for="Estado"></label>
<select asp-for="Estado"
            asp-items="@Html.GetEnumSelectList<Estado>()">
    <option>Selecione</option>
</select>
```

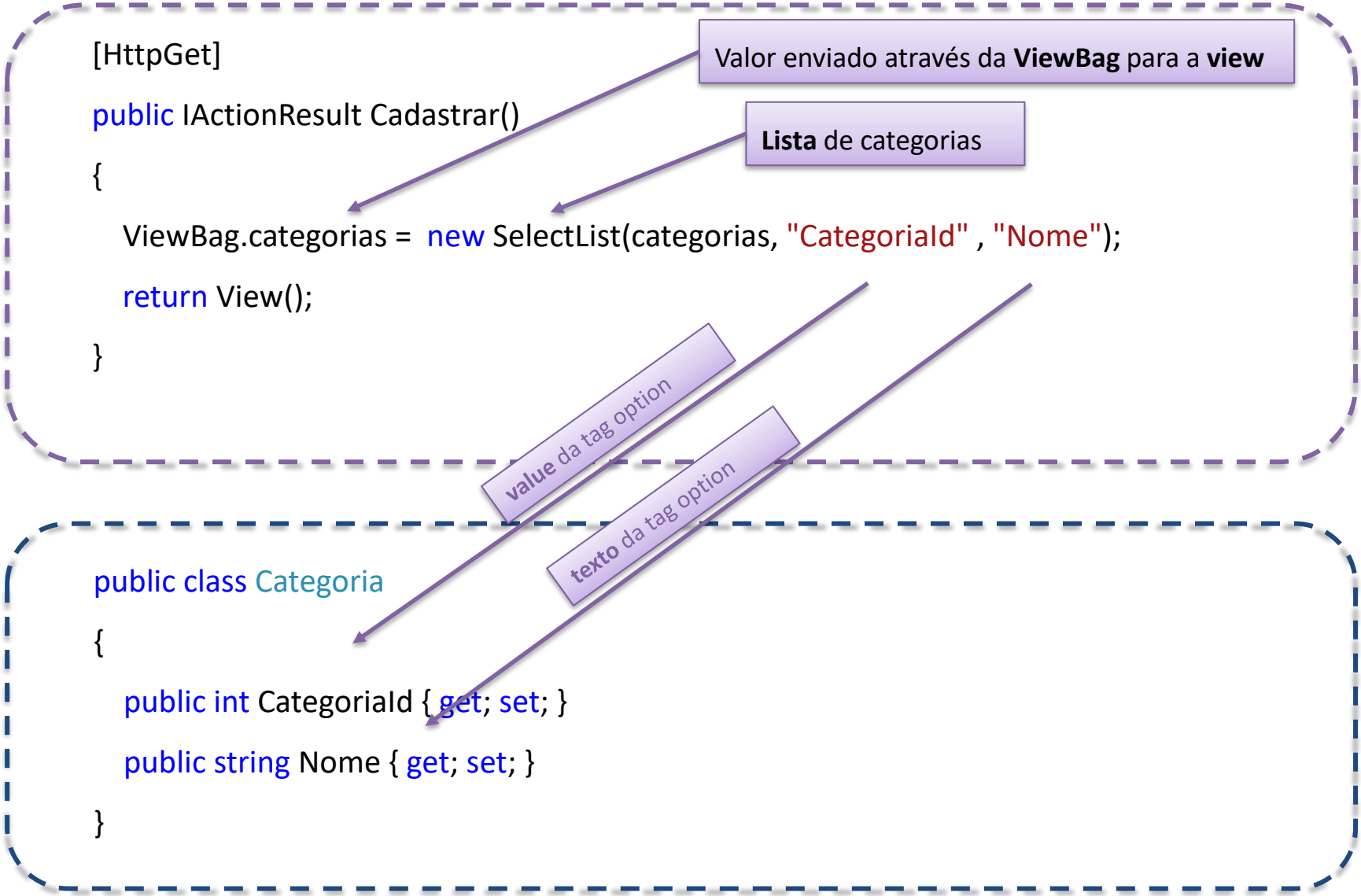
SELECT - RELACIONAMENTO

- Para implementar um **select** com **opções dinâmicas** (valores cadastrados no banco de dados, por exemplo) o valor do **asp-items** deve ser do tipo **SelectList**, enviado pelo **controller**;

```
<label asp-for="Categoria"></label>
<select asp-for="CategoriaId" asp-items="@ViewBag.categorias">
    <option>Selecione</option>
</select>
```

Valor enviado através da **ViewBag** pelo **controller**

SELECT - RELACIONAMENTO

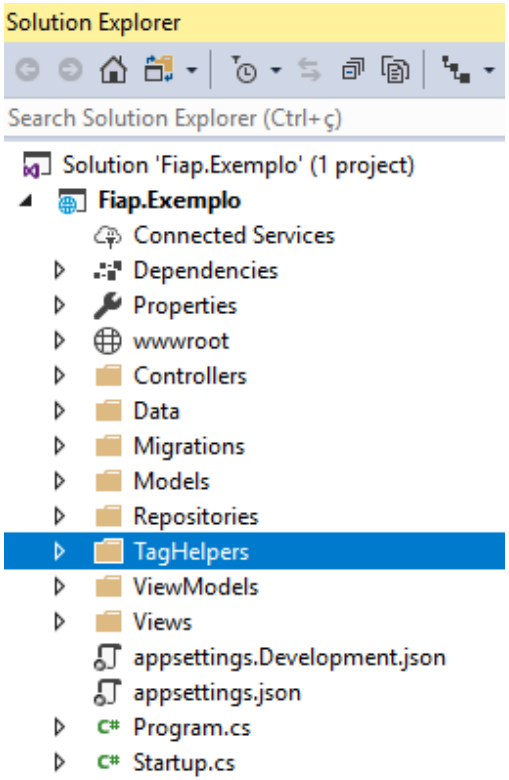


CRIANDO TAG HELPER

CRIANDO TAG HELPERS

- Podemos criar os nossos próprios **Tag Helpers;**
- Crie uma pasta chamada **TagHelpers** e depois uma **classe** que termine com **TagHelper**;

Crie uma pasta chamada **TagHelpers**



CLASSE TAG HELPER

- A classe deve **herdar** de **TagHelper**;
- As **propriedades** da classe serão **atributos** da tag HTML;
- A classe deve **sobrescrever** o método **Process();**

```
public class MessageTagHelper : TagHelper
{
    public string Mensagem { get; set; }

    public override void Process(
            TagHelperContext context, TagHelperOutput output)
    {
       output.TagName = "div";
       output.Attributes.SetAttribute("class", "alert alert-danger");
       output.Content.SetContent(Mensagem);
    }
}
```

VIEW IMPORTS

- Da mesma forma que habilitamos o uso dos **Tag Helpers**, precisamos **adicionar** os **nossos tag helpers** para as views;
- Modifique o arquivo **_ViewImports.cshtml**

```
@using Fiap.Exemplo.Models

@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper Fiap.Exemplo.TagHelpers.*, Fiap.Exemplo
```

UTILIZANDO O TAG HELPER

- Agora podemos utilizar o **nosso tag helper** nas views:

```
<message mensagem='Texto da mensagem'></message>
```

VOCÊ APRENDEU..

- Sobre **Razor** e HTML Helpers;
- Implementar Views utilizando **Tag Helpers**;
- Trabalhar com **links** e **formulários**;
- Utilizar **tag helpers** para gerar **inputs** com as propriedades de um model;
- Desenvolver **selects** com **enums** e **relacionamentos**;
- Criar um **Tag Helper** customizado;

FIAP

"O sucesso normalmente vem para quem está ocupado demais para pensar nele"