

Passando Dados para a View - ViewBag

- É um tipo dinâmico que pode ser criado sem declaração
- Não é necessário se preocupar com o tipo dos dados
- Permite fácil checagem de valores nulos
- É necessário saber o tipo dinâmico criado para acessar pela View

```
public IActionResult SomeAction()
{
    ViewBag.Greeting = "Hello";
    ViewBag.Address = new Address()
    {
        Name = "Steve",
        Street = "123 Main St",
    }
}
```

```
@ViewBag.Greeting World!
```

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

Passando Dados para a View - ViewData

- Armazena um tipo de ViewDataDictionary
- Como implementa o IDictionary, precisa salvar a Chave e Valor
- Permite o uso dos métodos *ContainsKey*, *Add*, *Remove*, *Clear*
- Permite que a lista de itens seja acessada pela View

```
public IActionResult About()  
{  
    Title = "About Us";  
    ViewData["Message"] = "Your application description";  
  
    return View();  
}
```

```
@{  
    ViewData["Title"] = "About";  
}  
<h2>@ViewData["Title"]</h2>  
<h3>@ViewData["Message"]</h3>  
  
<p>Use this area to provide additional information.</p>
```

Passando Dados para a View - Typed

- Permite o envio do objeto para a View
- Os tipos do objeto referenciado e da View devem ser idênticos
- Valida a estrutura em tempo de compilação e não na execução
- O objeto deve ser incluído no parâmetro do método View();

```
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    var viewModel = new Address()
    {
        Name = "Microsoft",
        Street = "One Microsoft Way",
        City = "Redmond",
        State = "WA",
        PostalCode = "98052-6399"
    };

    return View(viewModel);
}
```

```
@model WebApplication1.ViewModels.Address

<h2>Contact</h2>
<address>
    @Model.Street<br>
    @Model.City, @Model.State @Model.PostalCode<br>
    <abbr title="Phone">P:</abbr> 425.555.0100
</address>
```

Typed View - List

- A View pode ser tipada para apenas um objeto
- Porém podemos vincular a uma lista de objetos do mesmo tipo
- Devemos usar a interface `IEnumerable<T>`

```
@model IEnumerable<MultipleModelInOneView.Teacher>
```

```
<p><b>Teacher List</b></p>
```

```
<table>
```

```
<tr>
```

```
<th>Id</th>
```

```
@foreach (Teacher teacher in Model)
```

```
{
```

```
<tr>
```

```
<td>@teacher.TeacherId</td>
```

```
<td>@teacher.Code</td>
```

```
<td>@teacher.Name</td>
```

```
</tr>
```

```
}
```

Model Binding

Model Binding é o recurso do ASP.NET para transformar os valores enviados dentro do “corpo” da requisição HTTP, em formato texto, para as propriedades e seus tipos correspondentes da classe na aplicação.

- Também é possível converter os valores da rota (url) e querystring
- Atualizar todas as propriedades de um objeto
- Permite a customização do processo de conversão

```
[HttpGet("{id}")]  
public ActionResult<Pet> GetById(int id, bool dogsOnly)
```

```
https://contoso.com/api/pets/2?DogsOnly=true
```

Model Binding

- Model Binding também funciona com tipo complexos (classes)
- É possível misturar tipos complexos com tipos simples

```
public class Instructor
{
    public int ID { get; set; }
    public string LastName { get; set; }
    public string FirstName { get; set; }
}
```

```
public IActionResult OnPost(int? id, Instructor instructorToUpdate)
```

Model Binding

- Permite a inclusão de atributos de *filtro* de origem
 - [FromQuery] – Apenas valores da query string.
 - [FromRoute] - Apenas valores da rota.
 - [FromForm] - Apenas valores de um formulário.
 - [FromBody] - Apenas valores do body da requisição.
 - [FromHeader] - Apenas valores do header HTTP.

```
public ActionResult<Pet> Create([FromBody] Pet pet)
```

Exercício 08

Passando e Recebendo Dados da View



Validações

Para representar objetos da vida real o Model precisa de uma ferramenta que identifica se ele é válido ou não.

- Foco na estrutura para evitar repetição de código
- Validação baseadas em atributos das classes do modelo
- Essa validação pode ser verificada diretamente na View
- O Razor possui os Tag Helpers para facilitar a validação
- Ao receber o objeto o Controller também verifica se é válido

Validações

```
public class Movie
{
    public int Id { get; set; }

    [StringLength(60, MinimumLength = 3)]
    [Required]
    public string? Title { get; set; }

    [DisplayName = "Release Date"]
    [DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z\s]*$")]
    [Required]
    [StringLength(30)]
    public string? Genre { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z0-9""'\s-]*$")]
    [StringLength(5)]
    [Required]
    public string? Rating { get; set; }
}
```

Validando na View

```
<h4>Movie</h4>
<hr />
<div class="row">
  <div class="col-md-4">
    <form asp-action="Create">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      <div class="form-group">
        <label asp-for="Title" class="control-label"></label>
        <input asp-for="Title" class="form-control" />
        <span asp-validation-for="Title" class="text-danger"></span>
      </div>
```

Create

Movie

Title

The field Title must be a string with a minimum length of 3 and a maximum length of 60.

Release Date

mm/dd/yyyy

Genre

The Genre field is required.

Price

z

The field Price must be a number.

Validando - jQuery

Para funcionar a validação no cliente é necessário incluir as seguintes bibliotes do jQuery:

- Biblioteca base do jQuery
- jQuery Validation
- jQuery Unobtrusive Validation

```
<script src="/scripts/jquery.js"></script>  
<script src="/scripts/jquery.validate.js"></script>  
<script src="/scripts/jquery.validate.unobtrusive.js"></script>
```

Modelo é Válido?

Também é possível validar o Modelo no lado do servidor, para isso utilizamos a variável global do ASP.NET chamada **ModelState**. Também é possível acessar os erros no lado do Controller.

```
if (ModelState.IsValid)
{
    _context.Add(movie);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
return View(movie);
```

Exercício 09

Validando o Modelo

