

FIA/P GRADUAÇÃO

ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#09 - VALIDAÇÕES



thiagoyama



thiagoyama@gmail.com

#09 - VALIDAÇÕES

- Validações
- Validação no Controller
- View – Exibir mensagens de erro
- Validação com Data Annotation
- Validação no lado do cliente

VALIDAÇÕES

- Validações nos formulários são importantes para o sistema e o usuário;
- Sempre que um valor inserido pelo usuário for inválido (tipo de dados, valores, etc..) é necessário informar o usuário o motivo do erro e como corrigi-lo (Usabilidade);
- Existem várias técnicas para validação de formulários;

- Podemos adicionar as validações nas Actions dos Controllers:

```
[HttpPost]
public ActionResult Create(Cliente cliente)
{
    if (string.IsNullOrEmpty(cliente.Nome))
    {
        ModelState.AddModelError("Nome", "Nome Obrigatório");
    }
    //...
}
```



Propriedade do Model

Mensagem de Erro

- Os erros são armazenados no **ModelState**, através do método **AddModelError()**

VALIDAÇÃO NO CONTROLLER

- O **ModelState** possui a propriedade **IsValid**, que indica se existe erros de validação.
- Podemos utilizar essa propriedade para determinar se devemos redirecionar o usuário para uma página de erro ou processar a sua requisição.

```
if (ModelState.IsValid)
{
    //.. Cadastra o usuário
    return RedirectToAction("List");
}
else
{
    return View(cliente);
}
```

Se o model for válido

Cadastra o cliente e redireciona para a listagem

Se não, retorna para a mesma página de cadastro para exibir os erros.

- Código da **Action** completo:

```
[HttpPost]
public ActionResult Create(Cliente cliente)
{
    if (string.IsNullOrEmpty(cliente.Nome))
    {
        ModelState.AddModelError("Nome", "Nome Obrigatório");
    }

    if (ModelState.IsValid)
    {
        //.. Cadastra o cliente
        return RedirectToAction("List");
    }
    else
    {
        return View(cliente);
    }
}
```


MENSAGENS DE ERRO

- Para exibir as mensagens de erro de validação podemos utilizar
 - **@Html.ValidationSummary()**
 - Esse Html Helper adiciona todas as mensagens de erros;
- Para exibir as mensagens de erro para cada campo, podemos utilizar o **@Html.ValidationMessageFor()**
 - Exibe a mensagem de erro para um campo específico;

```
@using (@Html.BeginForm())  
{
```

```
    @Html.ValidationSummary()
```

Todas as mensagens de erro

```
    @Html.LabelFor(c => c.Nome)
```

```
    @Html.EditorFor(c => c.Nome)
```

```
    @Html.ValidationMessageFor(c => c.Nome)
```

Mensagem de erro para
o campo nome

```
    @Html.LabelFor(c => c.DataNascimento)
```

```
    @Html.EditorFor(c => c.DataNascimento)
```

```
    @Html.ValidationMessageFor(c => c.DataNascimento)
```

```
    <br />
```

```
    <input type="submit" value="Salvar" />
```

Mensagem de erro para
o campo data
nascimento

```
}
```

VALIDAÇÕES COM DATA ANNOTATIONS

- Validações de forma manual no controller é uma opção de validação, mas possui alguns pontos negativos, como ser trabalhoso, não pode ser reutilizado, ser sujeito a erros, etc..
- Outra forma de validação é utilizar **Data Annotations**:
 - Simples
 - Produtivo
 - Reutilizável
 - Menos erros de validações

- Campo obrigatório;

```
public class Cliente
{
    public int ClienteId { get; set; }

    [Required(ErrorMessage="Preencha o Nome!!")]
    public string Nome { get; set; }
}
```

- As anotações de validação possuem mensagem padrão de erro que podem ser alteradas através do atributo **ErrorMessage**.

- Determina que o valor deve estar entre outros dois valores configurados:

```
public class Cliente
{
    [Range(0,120)]
    public int Idade { get; set; }

    [Range(0.1, 10000, ErrorMessage="Valor errado!")]
    public decimal Salario { get; set; }
}
```

- Determina o máximo de caracteres que uma string pode ter, opcionalmente podemos definir o mínimo de caracteres também.

```
public class Cliente
{
    [StringLength(100)]
    public string Nome { get; set; }

    [StringLength(100, MinimumLength=4, ErrorMessage="Erro!")]
    [Required]
    public string Username { get; set; }
}
```


- Determina se é um formato de email válido.

```
public class Cliente
{
    [EmailAddress]
    public string Email { get; set; }
}
```

- Podemos utilizar expressão regular para realizar a validação:

```
public class Cliente
{
    [RegularExpression(@"^[a-zA-Z' '-'\s]{1,40}$", ErrorMessage=
    "Não é permitido números e caracteres especiais")]
    public string Username { get; set; }
}
```

<http://regexlib.com/>

- Podemos definir um tipo de dados para a propriedade do model:
 - DateTime
 - Date
 - Time
 - Currency
 - Email
 - Credicard

O tipo de dado configurado é utilizado para converter o valor inserido pelo usuário, assim se houver erro de conversão uma mensagem de erro é exibida.

```
public class Cliente
{
    [DataType(DataType.DateTime, ErrorMessage="Data inválida")]
    public DateTime DataNascimento { get; set; }
}
```

- As validações podem ser realizadas também no lado do cliente (navegadores);
- Melhora a experiência do usuário.
- Utiliza JQuery para as validações, assim basta adicionar os scripts nas páginas!

```
<script src="~/Scripts/jquery.validate.min.js"></script>  
<script src="~/Scripts/jquery.validate.unobtrusive.min.js">  
</script>
```

Copyright © 2013 - 2017 - Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

“Se você não traçou um plano para você mesmo, é possível que você caia no plano de outra pessoa. E adivinha o que ele planejou para você? Não muito”