02

Validando um produto com Model Validation

Um problema muito comum enfrentado por desenvolvedores web é a validação dos dados. Usuários sempre se esquecem de preencher determinados campos, preenchem e-mails inválidos, data de nascimento incorretas, e assim por diante. A tarefa da aplicação web é perceber esses pequenos problemas, e avisar o usuário sobre o mal entendido.

Mas o código de validação geralmente é feio e cheio de ifs. Veja o controller abaixo fazendo validações:

```
[HttpPost]
public ActionResult Adiciona(Produto produto)
{
    if(produto.Quantidade > 0 && produto.Preco > 0 && produto.Preco < 50000 && !String.IsNullOrEmpty
        ProdutosDAO dao = new ProdutosDAO();
        dao.Adiciona(produto);
        return RedirectToAction("Index");
    }
    else {
        return RedirectToAction("erros");
    }
```

Veja o `if` que escrevemos para garantir que a quantidade, preço e descrição estão dentro do esperado. Código feio e difícil de manter.

O ASP.NET MVC resolve o problema da validação através dos **Model Validators**. Quando definimos as classes de modelo, podemos anotar seus atributos com as regras de validação necessárias para que o modelo seja válido. Por exemplo, para fazer com que o campo Nome tenha no máximo 20 caracteres, devemos anotá-lo com a classe `StringLengthAttribute` :

```
class Produto
{
    [StringLengthAttribute(20)]
    String Nome { get; set; }

    // outros atributos do produto.
}
```

Mas como vimos, por convenção, no `C#` , todas as anotações são nomeadas utilizando-se o sufixo `Attribute` e quando utilizadas como anotação, o sufixo é desnecessário. Logo, podemos simplificar o código acima para:

```
class Produto
{
    [StringLength(20)]
    String Nome { get; set; }

    // outros atributos do produto.
}
```

Como vimos na seção anterior, quando enviamos os dados de um modelo para uma action, o ASP.NET MVC utilizará os dados da requisição para preencher os campos do modelo no processo de Model Binding. Além disso, o ASP.NET MVC também verifica as anotações presentes nos atributos da classe e executa as validações necessárias. O resultado da validação fica disponível em uma variável chamada `ModelState` que é herdada da classe `Controller`.

O `ModelState` guarda todas as regras de validação que foram violadas. Para verificar se todas as regras de validação foram obedecidas utiliza-se o atributo `IsValid` do `ModelState`.

```
if(ModelState.IsValid) {
    // O Modelo foi validado corretamente, então pode ser gravado no banco de dados.
}
else
{
    // O Modelo não foi validado corretamente.
}
```

Vamos modificar a action `Adiciona` do `ProdutoController` para que ela cadastre o novo produto apenas se não houverem erros de validação e, caso contrário, exiba o formulário de cadastro. Como queremos reutilizar a view da action `Form`, vamos utilizar uma segunda versão do método `View` que recebe o nome da action cuja view queremos utilizar.

```
[HttpPost]
public ActionResult Adiciona(Produto produto)
{
    if(ModelState.IsValid) {
        ProdutosDAO dao = new ProdutosDAO();
        dao.Adiciona(produto);
        return RedirectToAction("Index");
    }
    else
    {
        CategoriasDAO categoriasDAO = new CategoriasDAO();
        ViewBag.Categorias = categoriasDAO.Lista();
        return View("Form");
    }
}
```

Agora precisamos apenas mostrar os erros de validação na View.

Mensagens de Validação e o HtmlHelper

Agora que sabemos como verificar os erros de validação, vamos exibir as mensagens geradas. Para isso, utilizaremos a classe `HtmlHelper`.

O ASP.NET MVC disponibiliza em todas as views uma instância de `HtmlHelper` na variável `Html`. Essa variável é capaz de gerar pedaços de código HTML. Para exibir as mensagens de validação utilizaremos o método `ValidationMessage` da classe `HtmlHelper`.

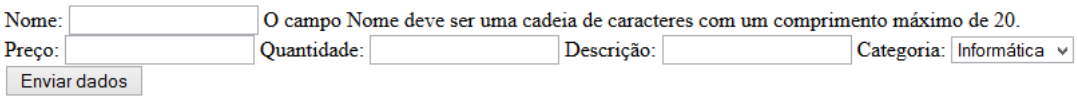
O `ValidationMessage` recebe como argumento o nome da mensagem de validação que deve ser gerada. As mensagens de validação geradas durante o processo de model binding tem nome da forma `NomeDoArgumentoDaAction.NomeDoAtributo`, logo devemos utilizar `produto.Nome` para recuperar as mensagens referentes ao campo Nome do produto.

```
@Html.ValidationMessage("produto.Nome")
```

Vamos modificar o formulário de cadastro de produtos para exibir a mensagem de validação:

```
<form action="/Produto/Adiciona" method="post">
  <label for="nome">Nome:</label>
  <input id="nome" name="produto.Nome" />
  @Html.ValidationMessage("produto.Nome")

  <label for="preco">Preço:</label>
  <input id="preco" name="produto.Preco" />

  <label for="quantidade">Quantidade:</label>
  <input id="quantidade" name="produto.Quantidade" />

  <label for="descricao">Descrição:</label>
  <input id="descricao" name="produto.Descricao" />

  <label for="categoria">Categoria:</label>
  <select id="categoria" name="produto.CategoriaId">
    @foreach(var categoria in ViewBag.Categorias)
    {
      <option value="@categoria.Id">@categoria.Nome</option>
    }
  </select>

  <input type="submit" />
</form>
```

Tente cadastrar um produto sem preencher o campo nome e veja as mensagens de validação geradas.



Implementando regras de validação complexas

Suponha que produtos da categoria Informática (categoria de id 1) devem custar pelo menos 100 reais, nesse caso as anotações definidas pelo ASP.NET MVC não conseguem resolver o problema. Temos que realizar a validação manualmente.

Em casos em que as anotações não resolvem o problema, devemos utilizar o método `AddModelError` do `ModelState`. Esse método recebe como argumento duas strings, a primeira indica o nome do erro de validação e a segunda a mensagem de erro que deve ser exibida.

```
ModelState.AddModelError(String nomeDoErro, String mensagem)
```

O nome do erro pode ser utilizado na view para se recuperar a mensagem de erro através do método `ValidationMessage` do `HtmlHelper`.

```
@Html.ValidationMessage(nomeDoErro)
```

Então a implementação de nossa regra de validação complexa fica da seguinte forma:

```
int idDaInformatica = 1;
if(produto.CategoriaId.Equals(idDaInformatica) && produto.Preco < 100)
{
    ModelState.AddModelError("produto.InformaticaComPrecoInvalido", "Produtos da categoria informát
}
```

Logo nossa action `Adiciona` com a regra complexa fica da seguinte forma:

```
[HttpPost]
public ActionResult Adiciona(Produto produto)
{
    int idDaInformatica = 1;
    if(produto.CategoriaId.Equals(idDaInformatica) && produto.Preco < 100)
    {
        ModelState.AddModelError("produto.InformaticaComPrecoInvalido", "Produtos da categoria infor
    }
    if(ModelState.IsValid) {
        ProdutosDAO dao = new ProdutosDAO();
        dao.Adiciona(produto);
        return RedirectToAction("Index");
    }
    else
    {
        CategoriasDAO categoriasDAO = new CategoriasDAO();
        ViewBag.Categorias = categoriasDAO.Lista();
        return View("Form");
    }
}
```

Agora precisamos apenas modificar o código do formulário de cadastro:

```
<form action="/Produto/Adiciona" method="post">
  @Html.ValidationMessage("produto.InformaticaComPrecoInvalido")

  <label for="nome">Nome:</label>
  <input id="nome" name="produto.Nome" />
  @Html.ValidationMessage("produto.Nome")

  <label for="preco">Preço:</label>
  <input id="preco" name="produto.Preco" />

  <label for="quantidade">Quantidade:</label>
  <input id="quantidade" name="produto.Quantidade" />

  <label for="descricao">Descrição:</label>
  <input id="descricao" name="produto.Descricao" />

  <label for="categoria">Categoria:</label>
  <select id="categoria" name="produto.CategoriaId">
```

```
    @foreach(var categoria in ViewBag.Categorias)
    {
      <option value="@categoria.Id">@categoria.Nome</option>
    }
  </select>

  <input type="submit" />
</form>
```

Acesse novamente o formulário de cadastro de produtos e teste a nova validação.

Mantendo os valores preenchidos em caso de erro de validação

No exemplo, exibimos as mensagens de erro de validação na view, porém deixamos o formulário de cadastro em branco novamente. Para fazer com que o formulário exiba os dados que foram enviados para a action de cadastro, precisamos fazer com que esses dados sejam reenviados para a view `Form.cshtml`. Para isso utilizaremos novamente a `ViewBag`.

Se os dados do formulário forem inválidos, queremos armazenar o produto recebido na `ViewBag`

```
if(ModelState.IsValid)
{
    // produto válido
}
else
{
    // produto inválido então vamos guardá-lo na ViewBag
    ViewBag.Produto = produto;
    // redireciona para o Form.cshtml
}
```

Com essa modificação, a action `Adiciona` fica da seguinte forma:

```
[HttpPost]
public ActionResult Adiciona(Produto produto)
{
    int idDaInformatica = 1;
    if(produto.CategoriaId.Equals(idDaInformatica) && produto.Preco < 100)
    {
        ModelState.AddModelError("produto.InformaticaComPrecoInvalido", "Produtos da categoria infor
    }
    if(ModelState.IsValid) {
        ProdutosDAO dao = new ProdutosDAO();
        dao.Adiciona(produto);
        return RedirectToAction("Index");
    }
    else
    {
        ViewBag.Produto = produto;
        CategoriasDAO categoriasDAO = new CategoriasDAO();
        ViewBag.Categorias = categoriasDAO.Lista();
        return View("Form");
    }
}
```

Falta apenas modificar a view `Form.cshtml` para exibir os dados do produto. Para isso, acessaremos os dados armazenados na `ViewBag`. Por exemplo, para preencher o nome do produto, preencheremos o atributo `value` da tag `input`:

```
<input name="produto.Nome" value="@ViewBag.Produto.Nome" />
```

Para o caso da tag `select`, precisamos indicar qual `option` deve ser inicialmente selecionado. Para decidir qual categoria deve ser selecionada, iremos, dentro do `@foreach`, comparar cada categoria com a categoria do produto da `ViewBag`. Se as categorias forem iguais, marcaremos a opção como selecionada.

```
@foreach(var categoria in ViewBag.Categorias) {
    if(categoria.Id.Equals(ViewBag.Produto.CategoriaId))
    {
        <option value="@categoria.Id" selected="selected">@categoria.Nome</option>
    }
    else
    {
        <option value="@categoria.Id">@categoria.Nome</option>
    }
}
```

Veja que o atributo `selected` da tag `option` funciona como um valor booleano indicando se a opção está ou não selecionada. Quando temos um atributo no Html que funciona como um valor booleano, no Razor, podemos atribuir diretamente uma expressão booleana (condição) para esse atributo. Se a expressão for `true`, o Razor inclui o atributo no Html gerado, senão o atributo é ignorado. Vamos então utilizar essa característica do Razor para marcar a categoria que foi selecionada:

```
<option value="@categoria.Id" selected="@categoria.Id.Equals(ViewBag.Produto.CategoriaId)">
  @categoria.Nome
</option>
```

O formulário final fica da seguinte forma:

```
<form action="/Produto/Adiciona" method="post">
  @Html.ValidationMessage("produto.InformaticaComPrecoInvalido")

  <label for="nome">Nome:</label>
  <input id="nome" name="produto.Nome" value="@ViewBag.Produto.Nome" />
  @Html.ValidationMessage("produto.Nome")

  <label for="preco">Preco:</label>
  <input id="preco" name="produto.Preco" value="@ViewBag.Produto.Preco" />
  @Html.ValidationMessage("produto.Preco")

  <label for="quantidade">Quantidade:</label>
  <input id="quantidade" name="produto.Quantidade" value="@ViewBag.Produto.Quantidade" />

  <label for="descricao">Descricao:</label>
  <input id="descricao" name="produto.Descricao" value="@ViewBag.Produto.Descricao" />
```

```
    <label for="categoria">Categoria:</label>
    <select id="categoria" name="produto.CategoriaId">
      @foreach(var categoria in ViewBag.Categorias)
      {
        <option value="@categoria.Id" selected="@categoria.Id.Equals(ViewBag.Produto.CategoriaId)">
          @categoria.Nome
        </option>
      }
    </select>

    <input type="submit" />
  </form>
```

Com isso o formulário será preenchido automaticamente caso ocorra um erro de validação, porém não conseguimos mais acessar a url `/Produto/Form` ! Esse erro ocorre pois a action `Form` não atribui um valor ao atributo `Produto` da `ViewBag` . Vamos então corrigir a action `Form` .

```
public ActionResult Form()
{
    CategoriasDAO dao = new CategoriasDAO();
    ViewBag.Produto = new Produto();
    ViewBag.Categorias = dao.Lista();
    return View();
}
```

Pronto! Validamos nossos dados!