02

Controlando Requisições e a Listagem de Produtos

Para aprendermos o ASP.NET MVC, desenvolveremos um sistema de controle de estoque. Esse sistema deve permitir o cadastro, listagem, visualização de detalhes e controle da quantidade de produtos. Cada produto deve ter um nome, um preço, uma categoria, uma descrição e a quantidade em estoque.

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

    public String Nome { get; set; }

    public float Preco { get; set; }

    public CategoriaDoProduto Categoria { get; set; }

    public int CategoriaId { get; set; }

    public String Descricao { get; set; }

    public int Quantidade { get; set; }
}
```

Categorias tem apenas um nome e uma descrição.

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

    public String Nome { get; set; }

    public String Descricao { get; set; }
}
```

Para armazenarmos as informações no banco de dados, utilizaremos o Entity Framework com o SQL Server Express Edition que é instalado junto com o Visual Studio 2013. Todo o código que utiliza o Entity Framework pode ser encontrado dentro das classes especializadas em acesso ao banco de dados conhecidas como `DAO` (Data Access Object).

O controller de produtos

Vamos inicialmente baixar o projeto inicial do curso no link http://s3.amazonaws.com/caelum-online-public/asp-net-mvc5/CaelumEstoque.zip (http://s3.amazonaws.com/caelum-online-public/asp-net-mvc5/CaelumEstoque.zip). Descompacte o zip e depois abra o arquivo `CaelumEstoque.sln` com o Visual Studio 2013 Express for Web. Esse arquivo irá importar um projeto que já fornece uma implementação para os `DAOs` de produtos (`ProdutosDAO`), de categorias (`CategoriasDAO`) e de usuários (`UsuariosDAO`).

Vamos agora desenvolver o controller para os produtos da aplicação, da mesma forma que fizemos no capítulo inicial do curso, chamado `ProdutoController` :

```
public class ProdutoController : Controller
{
    // Implementação
}
```

Dentro do método `Index` desse controller, executaremos a lógica para listar todos os produtos cadastrados no banco de dados utilizando uma tabela do html, mas no controller que acabamos de criar, a action `Index` simplesmente redireciona o usuário para a camada de visualização.

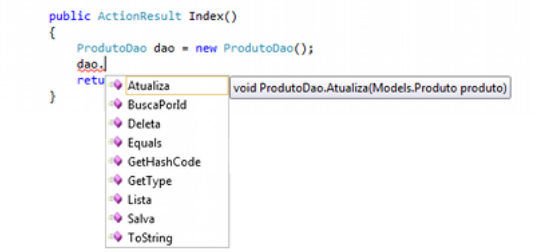
Como foi explicado na seção anterior, a camada de visualização do MVC deve conter apenas lógica de visualização. Logo, a lista de produtos deve ser enviada do controlador para a view.

Para conseguirmos a lista de produtos, devemos acessar os dados que estão gravados no banco de dados e, como mencionado anteriormente, o acesso ao banco é isolado dentro dos `DAOs`. Logo, precisamos instanciar um DAO que saiba acessar os produtos no banco, o `ProdutosDAO`.

```
public ActionResult Index()
{
    ProdutosDAO dao = new ProdutosDAO();

    return View();
}
```

Com o `DAO`, podemos utilizar o auto complete (`Ctrl + Espaço`) oferecido pela IDE para listar os métodos e propriedades do objeto.



Vemos que existe um método `Lista` que retorna um `IList<Produto>`. Utilizaremos esse método para recuperar a lista de produtos do banco de dados.

```
public ActionResult Index()
{
    ProdutosDAO dao = new ProdutosDAO();
    IList<Produto> produtos = dao.Lista();

    return View();
}
```

Obtivemos a lista, porém ela está visível apenas dentro do método do controlador. Para enviarmos a lista de produtos para a camada de visualização, precisamos utilizar uma propriedade que é herdada da classe `Controller` chamada `ViewBag`.

A `ViewBag` é um objeto em que podemos criar propriedades dinamicamente, para isso precisamos apenas fazer uma atribuição dentro do código do controller:

```
public ActionResult Index()
{
    ProdutosDAO dao = new ProdutosDAO();
    IList<Produto> produtos = dao.Lista();
    ViewBag.Produtos = produtos;
    return View();
}
```

Toda informação que colocamos dentro da `ViewBag` pode ser acessada pela camada de visualização, pois ela também tem acesso ao mesmo objeto `ViewBag` que utilizamos no controller. Com a `ViewBag` , podemos enviar qualquer número de objetos para a camada de visualização.

Com o método do controller pronto, precisamos somente criar nosso arquivo de view, esse arquivo será o `Views/Produto/Index.cshtml`

Esse arquivo será utilizado pelo ASP.NET MVC para apresentar o resultado da requisição. Queremos exibir o id, o nome e a quantidade dos produtos de nossa lista dentro de uma tabela, ou seja, fazer algo equivalente ao código abaixo:

```
<tr>
    <td>produto.Id</td>
    <td>produto.Nome</td>
    <td>produto.Quantidade</td>
</tr>
```

Não há como saber se o programador queria exibir o texto `"produto.Id"` ou recuperar o valor do `Id` da variável `produto` e, por esse motivo, o ASP.NET MVC nos obriga a colocar o prefixo `@` nas variáveis:

```
<tr>
    <td>@produto.Id</td>
    <td>@produto.Nome</td>
    <td>@produto.Quantidade</td>
</tr>
```

O código que criamos serve para exibir um produto, porém nosso controller enviou uma lista, logo precisamos executar o código acima para cada item da lista. Em C#, isso seria equivalente ao código:

```
foreach(var produto in LISTA DE PRODUTOS)
{
    <tr>
        <td>@produto.Id</td>
        <td>@produto.Nome</td>
        <td>@produto.Quantidade</td>
    </tr>
}
```

Na view, o `foreach` é quase igual ao código em C#, precisamos apenas adicionar um `@` antes do `foreach` :

```
@foreach(var produto in LISTA DE PRODUTOS)
{
```

```
        <tr>
            <td>@produto.Id</td>
            <td>@produto.Nome</td>
            <td>@produto.Quantidade</td>
        </tr>
    }
```

Agora podemos utilizar a `ViewBag` para acessar a lista de produtos que foi enviada pelo controller:

```
<table>
    <thead>
        <tr>
            <th>Id</th>
            <th>Nome</th>
            <th>Quantidade</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var produto in ViewBag.Produtos)
        {
            <tr>
                <td>@produto.Id</td>
                <td>@produto.Nome</td>
                <td>@produto.Quantidade</td>
            </tr>
        }
    </tbody>
</table>
```

O código completo da view para a tabela de produtos fica da seguinte forma:

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        <table>
            <thead>
                <tr>
                    <th>Id</th>
                    <th>Nome</th>
                    <th>Quantidade</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var produto in ViewBag.Produtos)
                {
                    <tr>
                        <td>@produto.Id</td>
```

```
                    <td>@produto.Nome</td>
                    <td>@produto.Quantidade</td>
                </tr>
            }
        </tbody>
    </table>
</div>
</body>
</html>
```

A lista está pronta!

Relembrando a primeira seção, a rota padrão do ASP.NET MVC faz com que requisições para `/Produto` sejam tratadas pelo método `Index` do `ProdutoController`, podemos acessar a action desenvolvida acessando a url `/Produto` após subir o servidor com o `F5`.