03

Controller e Model

Transcrição

Continuando o que vimos anteriormente, a seguir, veremos outras páginas da nossa aplicação. Na barra de menu superior, temos três links disponíveis: *home*, *about* e *contact*. O primeiro nos leva à página inicial, o segundo traz informações sobre a aplicação, enquanto o terceiro tem informações de contato, como endereço, telefone e CEP. Como o ASP.NET Core fornece estas páginas para o nosso browser?

Ele não vai simplesmente jogar as views no browser. Em vez disso, ele utilizará um intermediário, que é chamado de *controller* — ou seja, um controlador — que será responsável por administrar o acesso a cada uma das partes da nossa aplicação. Retornaremos ao Visual Studio e, no projeto, abriremos a pasta chamada "Controllers", na qual encontramos o `HomeController.cs`, arquivo que contém uma classe chamada `HomeController`.

Mas o que é o `HomeController`? Trata-se de uma classe C# que herda de uma classe base chamada `Controller` e contém métodos. Cada um destes fornece uma view para o browser, que tem o nome de cada um dos métodos. Por exemplo, a view `Index.cshtml` é retornada pelo método `Index()`, da mesma forma, o método `About.cshtml` é retornado pelo método `About()`, e assim por diante. Isto não é uma regra, mas é um padrão. É possível retornar views diferentes.

Como o ASP.NET Core encontra o método correto do *controller*? No endereço, ele tem as informações `localhost:58246/Home/Contact`, e identifica `Home` como o nome do *controller*. Sendo assim, ele une este nome ao `Controller`, formando `HomeController`, e encontra a classe de mesmo nome. Em seguida, ele utiliza o segundo nome, `Contact`, que se trata de uma ação, ou *action*, e é traduzida para um método do *controller* e então retornada pelo método `Contact()`.

Na classe `HomeController`, podemos observar o método `Contact()`:

```
namespace CasaDoCodigo.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }

        public IActionResult About()
        {
            ViewData["Message"] = "Your application description page.";

            return View();
        }

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

            return View();
        }

        public IActionResult Error()
```

```
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIc
        }
    }
}
```

Colocaremos um ponto de parada na linha `ViewData["Message"] = "Your contact page.";`, rodaremos novamente a aplicação, utilizando a tecla de atalho "F5". Vemos que o ASP.NET Core redirecionou a chamada para este endereço, `HomeContact`, encontrando `HomeController` e, em seguida, `Contact()`. Assim, o método fornece para a página a informação dentro de um dicionário, chamado de `ViewData`.

`ViewData` é um dicionário padrão ao qual podemos atribuir certas informações que são consumidas pela página, ou seja, pela *view*. No caso, definimos uma informação que tem como chave `["Message"]`, e que traz em seu conteúdo "*Your contact page.*" (ou "Sua página de contato."). Em seguida, o método `Contact()` retorna uma *view*, e esta é a `Contact.cshtml`, que receberá como informação adicional a `ViewData`. Por isso, ao atualizarmos, a mensagem é exibida na tela.

Mas por que isso foi montado desta forma? Abriremos o arquivo `Contact.cshtml`, que está dentro da pasta "Home". Nela, encontramos o `@ViewData["Message"]`, isto é um código C#. O ASP.NET Core monta uma página a partir de um arquivo `.cshtml`, e esta extensão compreende um arquivo C# HTML, que é convertida na página que é retornada para o browser.

Trocaremos a mensagem do `ViewData` para "Sua página de contato!". Executaremos novamente a aplicação, abriremos a view de contato e, abrindo a página, veremos a mensagem atualizada.

O ASP.NET Core injetou a informação do modelo do MVC, do `ViewData`, na *view* para poder formar a página de contato.