02

Sessões para autenticar usuários

Mantendo dados entre requisições

Sabemos como implementar uma aplicação web capaz de responder requisições, porém imagine que precisamos mostrar para um usuário quantas vezes ele acessou uma determinada página. Então devemos ter alguma forma de manter um contador que depende do usuário que está acessando a página.

O código que queremos deve ser da seguinte forma:

```
int contador = PegaContadorDoUsuarioAtual();
contador++;
GuardaOContadorDoUsuario(contador);
```

Para suportar esse tipo de estado, o ASP.NET MVC implementa o conceito de sessão.

No ASP.NET MVC, a sessão do usuário pode ser acessada de uma action através da variável `Session`, que foi herdada da classe `Controller`. A variável `Session` funciona como um dicionário de `String` para `object`, logo podemos armazenar diversos nomes com valores diferentes por usuário.

Em nosso exemplo, queremos armazenar um valor do tipo `int` dentro da `Session`.

```
Session["contador"] = contador;
```

Assim como em um dicionário do C#, podemos recuperar o valor associado a uma chave com `Session[nomeDaChave]`, porém como a sessão guarda o tipo object, precisamos converter o valor devolvido para o tipo correto. No caso do contador, precisamos converter o valor devolvido para o tipo `int`, essa converão pode ser feita através da classe `Convert`:

```
int contador = Convert.ToInt32(Session["contador"]);
```

Caso uma chave não tenha sido inserida na sessão, a busca `Session[chaveInexistente]` retorna `null`. Se passarmos o valor `null` para o método `Convert.ToInt32`, ele nos retorna o valor zero.

Vamos agora implementar nosso contador por usuário. Criaremos um novo controller chamado `ContadorController` e adicionaremos o método `Index`:

```
public ActionResult Index()
{
    return View();
}
```

Agora, precisamos fazer uma série de operações: recuperar o valor do contador da session, incrementá-lo, armazenar o valor atualizado na session e, por fim, enviá-lo para a view. Caso o contador não tenha sido definido, queremos definir o contador como 0.

```
public ActionResult Index()
{
    object valorNaSession = Session["contador"];
    int contador = Convert.ToInt32(valorNaSession);
    contador++;
    Session["contador"] = contador;
    return View(contador);
}
```

Agora só precisamos mostrar o valor do contador na camada de visualização da aplicação. Vamos então criar o arquivo `Views/Contador/Index.cshtml`

```
Contador atual: @Model
```

Tente acessar essa action de diversos navegadores e veja que o valor do contador é diferente para cada navegador.

O Sistema de autenticação

Queremos que apenas usuários logados no sistema tenham acesso às páginas de listagem e cadastro de produtos. Então precisamos de uma nova página da aplicação onde faremos o login dos usuários. A lógica dessa nova tela ficará dentro de um novo controller chamado `LoginController`. Dentro dessa classe, a action `Index` será a responsável por mostrar o formulário de login para o usuário:

```
public class LoginController : Controller
{
    public ActionResult Index()
    {
        return View();
    }
}
```

Agora precisamos implementar a view do formulário de login, arquivo `Views/Login/Index.cshtml`:

```
<form action="/Login/Autentica" method="post">
   <label for="login">Login:</label>
   <input name="login" id="login" class="form-control" />

   <label for="senha">Senha:</label>
   <input name="senha" id="senha" class="form-control" />

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

Mas com esse código os dados digitados no campo de senha do formulário ficarão expostos. Para esconder a senha digitada pelo usuário, podemos utilizar um novo tipo de `input` do html chamado `password`:

```
<form action="/Login/Autentica" method="post">
   <label for="login">Login:</label>
   <input name="login" id="login" class="form-control" />
```

```
    <label for="senha">Senha:</label>
    <input name="senha" id="senha" class="form-control" type="password" />

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

Os dados desse formulário serão enviados para a action `Autentica` do `LoginController`:

```
public ActionResult Autentica(String login, String senha)
{

}
```

Nessa action, precisamos verificar se as informações que foram enviadas pelo formulário realmente existem dentro do banco de dados, para isso utilizaremos o método `Busca` do `UsuariosDAO`. Esse método busca um usuário dado o login e a senha. Se as informações estiverem corretas, o método devolve o `Usuario` do banco de dados, senão ele devolve a referência nula:

```
public ActionResult Autentica(String login, String senha)
{
    UsuariosDAO dao = new UsuariosDAO();
    Usuario usuario = dao.Busca(login, senha);
}
```

Se o `usuario` tiver uma referência válida, vamos armazená-lo na sessão do servidor e depois redirecionar para a página inicial da aplicação (a lista de produtos), senão vamos enviar o usuário de volta para a página de login da aplicação.

```
public ActionResult Autentica(String login, String senha)
{
    UsuariosDAO dao = new UsuariosDAO();
    Usuario usuario = dao.Busca(login, senha);
    if(usuario != null)
    {
        Session["usuarioLogado"] = usuario;
        return RedirectToAction("Index", "Produto");
    }
    else
    {
        return RedirectToAction("Index");
    }
}
```

Agora podemos testar o login na aplicação entrando na url `/Login`. No banco de dados do projeto que foi importado no começo do curso temos um usuário chamado `caelum` com a senha `caelum` que pode ser utilizado para o teste dessa lógica de login.