03

LoginService

Transcrição

Agora que estamos interceptando a mensagem `SucessoLogin` e passando o objeto desta mensagem (`usuario`) ao `MasterDetailView` , é necessário garantir que o usuário esteja vindo de onde a mensagem está sendo emitida. Ou seja, queremos que seus dados estejam sendo preenchidos de acordo com o servidor da Aluracar, no momento no qual o usuário é autenticado.

Vamos observar quando a mensagem `"SucessoLogin"` é lançada na aplicação, quando ela é enviada ao `MessagingCenter` , em `LoginService.cs` . Enviamos `Usuario` como uma instância vazia, sem nenhuma informação de usuário (`new Usuario()`). Portanto, precisaremos modificar este código para passar à mensagem os dados cadastrais deste usuário.

Utilizaremos os dados que estão chegando a partir de `resultado` no seguinte `if` :

```
if (resultado.IsSuccessStatusCode)
```

Depois, eles serão levados até a instância do `Usuario` e, aí sim, trabalharemos com as informações. O primeiro passo consiste em usarmos o resultado do `PostAsync` verificando-se seu conteúdo a partir da propriedade `Content` , de tipo HTTP Content, o que para nós não significa nada.

Não se trata de um usuário sendo autenticado, é simplesmente um conteúdo HTTP que converteremos em uma `string` , o que será realizado por meio de um método chamado `ReadAsStringAsync()` . A partir daí, transformaremos este conteúdo genérico HTTP em uma `string` .

```
if (resultado.IsSuccessStatusCode)
{
  resultado.Content.ReadAsStringAsync();
  MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
}
```

Ao fazermos isto, no entanto, estamos chamando o método é assíncrono (uma *task*), como o nome indica. Então precisaremos aguardar o resultado desta leitura pois, caso contrário, o programa continuará sendo rodado mesmo antes do método ser executado.

Acrescentaremos o operador `await` para determinar que o resultado foi aguardado, e posteriormente, acontecerá a execução.

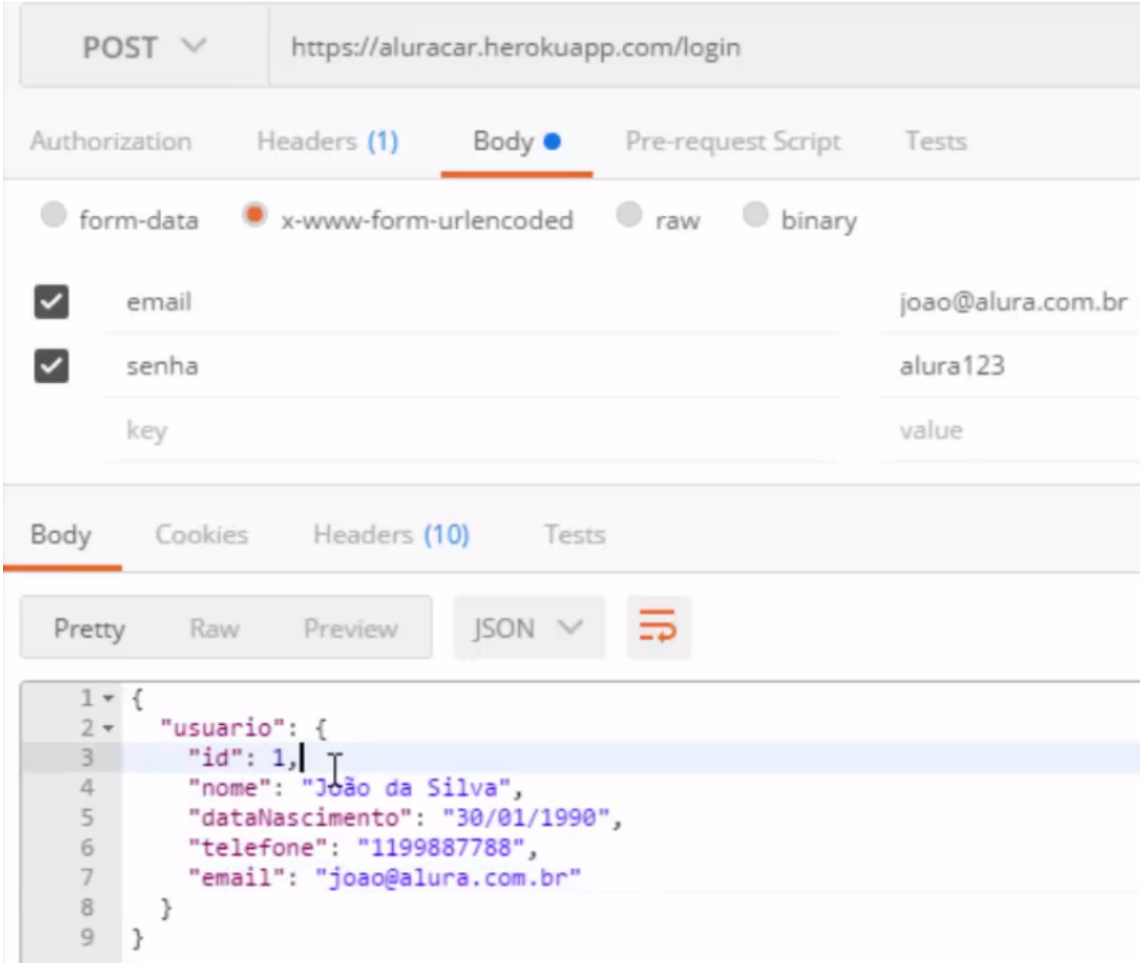
```
if (resultado.IsSuccessStatusCode)
{
  await resultado.Content.ReadAsStringAsync();
  MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
}
```

A `string` que vem do conteúdo precisa ser armazenada em algum lugar para depois ser convertida no tipo que quisermos. Colocaremos então uma variável local denominada `conteudoResultado` , que por sua vez terá uma `string`

que será o conteúdo da requisição HTTP utilizada para autenticar o usuário. Seu resultado virá em formato JSON, como vimos anteriormente.

```
if (resultado.IsSuccessStatusCode)
{
  var conteudoResultado = await resultado.Content.ReadAsStringAsync();
  MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
}
```

Vamos analisar no **Postman**, aquele programa que utilizamos para fazermos requisição do POST: na autenticação do usuário, obtemos uma `string` em formato JSON com propriedade `usuario`, dentro da qual temos seus dados, tais como nome, data de nascimento, telefone e outros.



O objeto `Usuario` não contém todas as informações, temos apenas "nome" e "e-mail". Acrescentaremos as propriedades que estão faltando, abrindo `Usuario.cs`, e temos o `id`, que será um número inteiro, `dataNascimento` e `telefone`:

```
public class Usuario
{
  public int id { get; set; }
  public string nome { get; set; }
  public string email { get; set; }
  public string dataNascimento { get; set; }
  public string telefone { get; set; }
}
```

Feito isto, precisamos converter o `conteudoResultado` no objeto `Usuario`, a partir da criação de um objeto que irá conter exatamente o resultado da autenticação do POST, em JSON, um `usuario`, que é propriedade. Não temos as propriedades do usuário diretamente na raiz deste objeto, e sim apenas `usuario`. Só depois é que acessamos suas propriedades.

Precisamos criar um objeto com propriedade `usuario` na aplicação, ou seja, em um nível acima desta estrutura do retorno da autenticação. Criaremos uma classe no arquivo `Usuario.cs`, a ser denominado `ResultadoLogin`, com apenas uma propriedade (`Usuario`), mantendo-se o mesmo nome do resultado vindo do objeto JSON (`usuario`):

```
public class Usuario
{
  public int id { get; set; }
  public string nome { get; set; }
  public string email { get; set; }
  public string dataNascimento { get; set; }
  public string telefone { get; set; }
}

public class ResultadoLogin
{
  public Usuario usuario { get; set }
}
```

Abrindo-se `LoginService.cs`, transformaremos `conteudoResultado` em `ResultadoLogin`. Para usarmos esta `string` transformando-a em uma instância de uma classe do C#, necessitamos da biblioteca da Newtonsoft, o **JSON Convert**, capaz de converter uma `string` para um objeto, e vice-versa.

Quando temos um objeto e o transformamos em uma `string`, serializamos; o caminho inverso, ou seja, de `string` para objeto, é chamado de **desserialização**.

Assim, criamos a variável local chamada `resultadoLogin`, que será uma conversão. Pegaremos a classe `Json.Convert` e desserializaremos uma `string` em objeto. Como se trata de um método genérico, devemos passar o tipo, a classe do objeto que estamos convertendo, que será um `ResultadoLogin`, o qual armazenará a conversão desta `string conteudoResultado`.

```
if (resultado.IsSuccessStatusCode)
{
  var conteudoResultado = await resultado.Content.ReadAsStringAsync();

  //conteudoResultado ->ResultadoLogin
  var resultadoLogin =
      Json.Convert.DeserializeObject<ResultadoLogin>(conteudoResultado);

  MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
}
```

Feito isto, obteremos `resultadoLogin` com as informações da autenticação do usuário. Onde há uma nova instância de `Usuario` (`new Usuario()`), substituiremos por `resultadoLogin.usuario`, e este `usuario` será o resultado da conversão da `string` JSON em um objeto que pode ser utilizado no C#.

```
if (resultado.IsSuccessStatusCode)
{
  var conteudoResultado = await resultado.Content.ReadAsStringAsync();

  var resultadoLogin =
      Json.Convert.DeserializeObject<ResultadoLogin>(conteudoResultado);

  MessagingCenter.Send<Usuario>(resultadoLogin.usuario, "SucessoLogin");
}
```

Testaremos a aplicação para ver se a conversão funciona, logando e acessando a parte de perfil de usuário. É exibido um erro que nos diz que o construtor *default* não foi encontrado para o tipo `MasterView`. O que está ocorrendo é que estamos tentando instanciá-lo, porém não temos um construtor vazio, sem parâmetros.

Ou seja, no `MasterDetailView.xaml`, estamos tentando instanciar um `MasterView` sem nenhum parâmetro. Como já o alteramos, ele obrigatoriamente está recebendo `usuario` como parâmetro do construtor. Pode-se passar um construtor entre as tags `<view:MasterView>`, que é algo ruim de se fazer em XAML. Porém, pode-se simplesmente remover o trecho abaixo, passando o conteúdo de `Master` no código C#, no *code behind*:

```
<MasterDetailPage.Master>
  <view:MasterView>

  </view:MasterView>
</MasterDetailPage.Master>
```

Esta opção é muito mais fácil, portanto iremos à aba `MasterDetailView.xaml.cs` e definiremos o `Master` como sendo um novo `MasterView`, para o qual pode-se passar um parâmetro neste construtor, que será o `usuario` sendo recebido mais acima.

```
public MasterDetailView(Usuario usuario)
{
  InitializeComponent();
  this.usuario = usuario;
  this.Master = new MasterView(usuario);
}
```

Agora temos o necessário para a execução deste código, o qual rodaremos novamente, repetindo todo o procedimento feito até então em termos de login. Na página de perfil do usuário, vemos os dados "`João da Silva`" e "`joao@alura.com.br`" como usuário e senha, que são as informações provenientes da autenticação do servidor. Conseguimos exibir na página de perfil as informações referentes ao usuário logado com sucesso!