10

Especificação Login

Transcrição

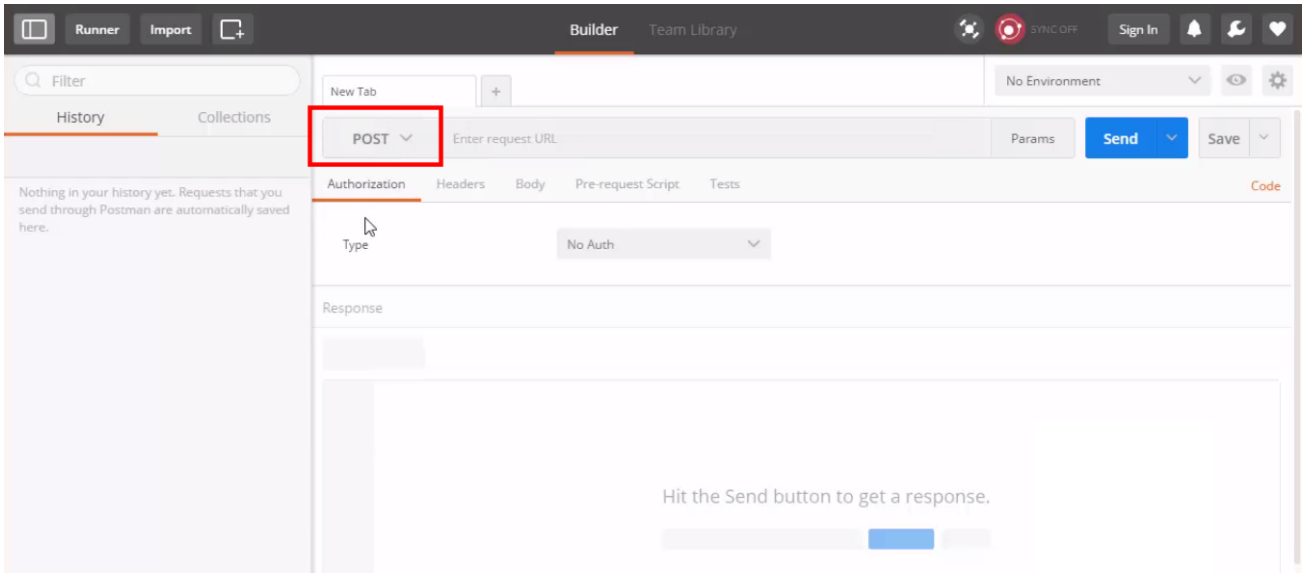
De acordo com as especificações recebidas pelo cliente, existem algumas instruções a serem seguidas para a autenticação dos dados do usuário no servidor da Aluracar:

- Endereço do serviço de login: https://aluracar.herokuapp.com/login (https://aluracar.herokuapp.com/login)
- Método http: POST
- Campos do conteúdo do POST para testes (application/x-www-form-urlencoded):
  - e-mail: joao@alura.com.br
  - senha: alura123

Poderíamos simplesmente abrir o projeto Xamarin codificando-o no próprio C#. Caso encontremos algum erro na aplicação durante a requisição, pode ser que este endereço de login que estamos utilizando não seja o correto, que o e-mail e a senha recebidos na especificação não estejam corretos, ou até que o serviço da Aluracar ainda não esteja em funcionamento.

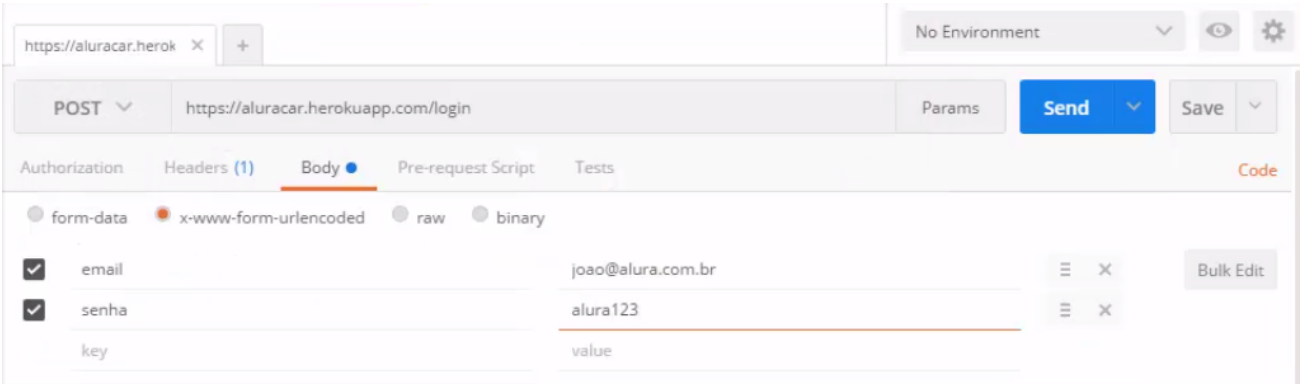
Há chances de acabarmos "quebrando a cabeça" ao tentar resolver um problema que não é nosso ou do aplicativo. Sendo assim, testaremos esta requisição inicialmente em uma ferramenta externa para verificarmos se estas especificações realmente fazem parte de um serviço em funcionamento. Existem muitas delas à disposição, muitas vezes gratuitamente. Neste caso, faremos os testes utilizando uma extensão do navegador Google Chrome chamada "postman". Buscaremos por "chrome extensions postman" no Google, encontramos o link (https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop) e a instalamos na nossa máquina.

Abriremos o aplicativo Postman e vemos alguns campos a serem preenchidos. Precisamos informá-lo sobre os dados da requisição que queremos fazer. Este tipo de requisição não é GET, e sim POST, portanto modificaremos isto:

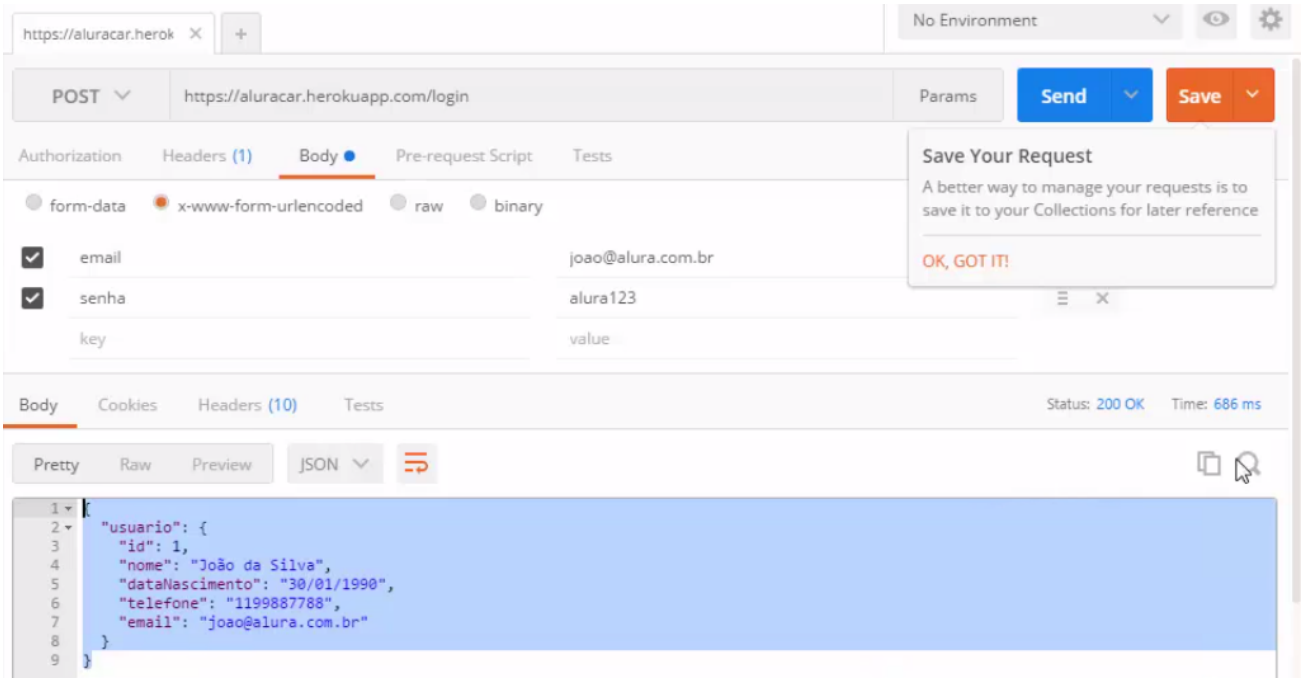


Preencheremos o campo ao lado com o endereço do serviço de login (https://aluracar.herokuapp.com/login) dado pelo cliente. Clicaremos na aba "*Body*" ("Corpo"), e selecionaremos o tipo de dado que estamos enviando ("x-www-form-urlencoded"). Agora, precisamos passar para a requisição a chave valor passada ao servidor da Aluracar. O primeiro

campo será "email", e ao lado digitaremos "joao@alura.com.br". O segundo campo, "senha", cujo valor será "alura123", assim:



Feito isto, basta apertarmos "*Send*" para envio da requisição. O resultado nos diz que o "*Status*" está " `200 OK` " (a requisição foi bem sucedida). Embaixo, temos os dados do usuário logado em formato JSON. Temos uma estrutura, um objeto, que é `"usuario"`, dentro do qual há as propriedades `"id"`, `"nome"`, `"dataNascimento"`, `"telefone"` e `"email"`.



Então, além da requisição com status OK, que informa que o usuário foi autenticado, precisamos do valor o qual está sendo passado no resultado da requisição POST. Neste momento, se houver qualquer erro na aplicação Xamarin Forms, durante o *TestDrive*, não se trata de um problema de autenticação, pois o serviço da Aluracar está funcionando corretamente. Caso encontremos algum erro na autenticação, teremos que verificar nosso código.

Voltando ao nosso projeto, da forma como implementamos o `EntrarCommand` (comando do botão "Entrar"), ele simplesmente envia a mensagem indicando que houve sucesso no login. Teremos que comentar a linha referente a `MessagingCenter`, substituindo-a por uma autenticação de verdade.

Como visto no curso anterior, é possível fazer requisições HTTP REST instanciando-se o **HttpClient**, dentro do qual implementaremos a autenticação, a chamada Http POST, que se comunicará com o servidor do cliente, obtendo-se o resultado. Para isto, passaremos a instância do cliente, bem como o método `PostAsync()`, em que colocaremos o endereço do POST recebido na especificação. Feito isto, basta passarmos os campos do formulário, declarando-se a variável `camposFormulario` algumas linhas acima:

```
EntrarCommand = new Command(() =>
{
    //MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
    using(var cliente = new HttpClient())
    {
        var camposFormulario = new FormUrlEncodedContent(new[]
          {
            new KeyValuePair<string, string>("email", "joao@alura.com.br"),
            new KeyValuePair<string, string>("senha", "alura123")
          });

          cliente.BaseAdress = new Uri("https://aluracar.herokuapp.com");
          cliente.PostAsync("/login", camposFormulario);
    }
  }, () =>
  {
    return !string.IsNullOrEmpty(usuario)
            && !string.IsNullOrEmpty(senha);
  });
```

Neste código, criamos o corpo a ser enviado à requisição POST. Definimos a base de endereço desta requisição, uma `string`, e uma nova `Uri`. Como o próprio nome indica, `PostAsync` é uma chamada assíncrona, de modo que quando o usuário clicar no botão, o código será executado e a chamada será feita no servidor da Aluracar. O POST será realizado, porém o resultado não será aguardado. Será que é isto que queremos? Fazer a chamada da requisição ignorando-se o resultado?

Ao fazermos esta chamada e autenticação, e o servidor recusá-las, não conseguiremos prosseguir com a aplicação, e o usuário fica impossibilitado de navegar para a tela seguinte. Assim, precisamos barrá-lo quando a requisição (ou autenticação) for negada.

Quando passamos o mouse por cima de `PostAsync`, vemos que ele retorna uma *task*, uma tarefa. Para aguardá-la, utilizaremos o operador `await`, que faz com que a execução pare nesta linha e aguarde o resultado desta tarefa. A linha `cliente.PostAsync("/login", camposFormulario);` passa a ser, portanto, `await cliente.PostAsync("/login", camposFormulario);`.

O Visual Studio aponta como erro o `await`, que só deve ser usado com `async`. Desta forma, iremos ao começo da expressão lambda definindo o método anônimo como assíncrono:

```
EntrarCommand = new Command(async () =>
```

O código será executado aguardando-se o POST, e somente após a resposta por parte do servidor, será dado prosseguimento. Agora, removeremos a linha `MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");` que havíamos comentado, passando-a para baixo, depois da chamada do método assíncrono. Feito isto, aguardaremos a chamada POST, e em seguida enviaremos a mensagem ao usuário, indicando que o login foi realizado com sucesso.

```
EntrarCommand = new Command(async () =>
{
    using(var cliente = new HttpClient())
    {
        var camposFormulario = new FormUrlEncodedContent(new[]
          {
```

```
            new KeyValuePair<string, string>("email", "joao@alura.com.br"),
            new KeyValuePair<string, string>("senha", "alura123")
          });

          cliente.BaseAdress = new Uri("https://aluracar.herokuapp.com");
          await cliente.PostAsync("/login", camposFormulario);
          MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
    }
  }, () =>
  {
    return !string.IsNullOrEmpty(usuario)
           && !string.IsNullOrEmpty(senha);
  });
```

Com estas alterações, rodaremos a app para verificarmos o resultado. Digitaremos "joao@alura.com.br" como usuário e "alura123" como senha, clicando em "Entrar" em seguida. Vemos que a aplicação pausou na linha da chamada POST, conforme esperado. Apertaremos `F5` para que a chamada seja feita e, quando o servidor a retorna, passa-se para baixo, no envio da mensagem, o que quer dizer que a mensagem que permite que a navegação à próxima tela se efetue será enviada.

Rodando-se novamente a aplicação, vemos que é exatamente isto que ocorre. Veremos em seguida como realmente fazemos a autenticação deste usuário no servidor.