01

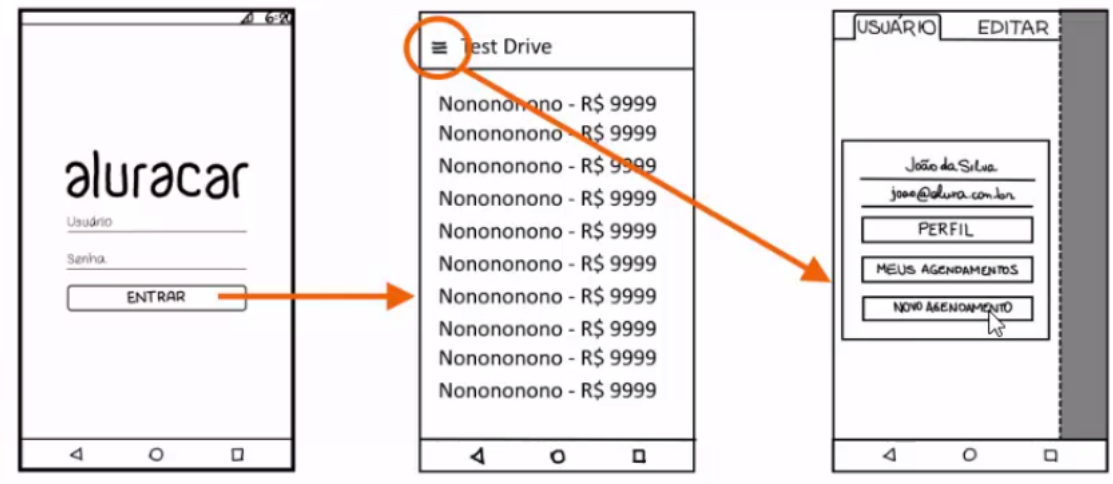
Introdução MasterDetailPage

Transcrição

No último vídeo, vimos como implementarmos a tela de login da aplicação, e quando o usuário digita o e-mail ou a senha incorretamente, sua autenticação é negada. Se houver algum problema de comunicação em relação aos dados do *device*, a aplicação também rejeita a entrada do usuário. Ao digitar seus dados corretamente, e havendo conexão, o usuário consegue acessar a aplicação normalmente, sendo redirecionado à página de listagem de veículos.

Recebemos uma nova especificação por parte do cliente (Aluracar), para mostrarmos ao usuário seu perfil. Implementaremos, então, uma tela lateral que possibilitará a visualização de seus dados cadastrais e pessoais. Esta tela precisa ser exibida em um formato diferente, pois da forma como criamos a navegação, conseguimos avançar ou recuar, porém, durante esta navegação, também poderemos acessar outra tela ou página, disponível neste menu lateral.

Nesta especificação recebida, temos o botão "Entrar", que como visto nos levará à listagem de veículos, e em cima há um menu que muitas vezes chamam de "menu hamburguer", padrão de alguns sistemas operacionais de *smartphones*. Outra opção é deslizar o dedo no sentido da esquerda para a direita da tela, conseguindo acessar a página nova que não chega a ocupar a tela toda, e contém dados do usuário tais quais nome, e-mail, telefone, data de nascimento, e informações afins. Há também os botões "Perfil", "Meus agendamentos", "Novo agendamento", comandos executáveis a partir dali.



Veremos como implementar isto em uma aplicação Xamarin, que possui algumas opções de *views*, de navegação:

Uma destas opções, denominada `ContentPage`, é a que vimos anteriormente, uma página simples que não leva automaticamente para nenhum lugar. Vimos também o `NavigationPage` que permite navegação a partir do botão " < " no topo, o qual indica "voltar para uma localização prévia".

Ainda não vimos o `TabbedPage`, o qual permite uma visualização com abas, muito comum em aplicações, e `CarouselPage` que, como o próprio nome indica, permite uma navegação "deslizável", para ambos os lados. E há o `MasterDetailPage` com uma "dupla visualização", uma tela principal ("mestra") e uma "filha", que recebe algumas de suas informações. Esta última opção é a que queremos utilizar, cuja tela mestra será a de exibição do perfil do usuário, e o detalhe, pra onde iremos automaticamente, a listagem de veículos, a ser exibida por padrão.

Para criarmos um `MasterDetailPage`, clicaremos com o lado direito do mouse na pasta `Views` do menu localizado à direita no Visual Studio, e depois em "*Add* > *New Item*" e, dentre as opções existentes, escolheremos `MasterDetailPage`. Porém, vimos que esta opção inexiste! Neste caso, podemos selecionar o template que já conhecemos, `Forms Xaml Page`, que nos fornecerá um `ContentPage`. Mais para a frente, modificaremos ele para que se transforme no `MasterDetailPage`. Chamaremos esta nova *view* de `MasterDetailView`.

Neste novo arquivo criado, substituiremos `ContentPage` por `MasterDetailPage`. A aplicação não chegará a esta página automaticamente, precisamos criar uma instrução para que isto ocorra. Faremos isto abrindo o arquivo `App.xaml.cs`, na pasta `App.xaml`, através da propriedade `MainPage`. Teremos que defini-la como nova instância do `MasterDetailView`. Comentaremos a linha referente a `MainPage` substituindo-a por uma declaração que determinará a nova página principal para que, assim que o usuário fizer login, ele tenha que ir à `MasterDetailPage`.

```
protected override void OnStart()
{
    MessagingCenter.Subscribe<Usuario>(this, "SucessoLogin",
    (usuario) =>
    {
      //MainPage = new NavigationPage(new ListagemView());
      MainPage = new MasterDetailView();
      })
}
```

A partir disto, o Visual Studio nos aponta um erro que indica que não é possível convertermos um tipo `TestDrive.Views.MasterDetailView` para um `Xamarin.Forms.Page`. O que quer dizer que, no `MasterDetailView.xaml`, o declaramos como um `ContentPage`. Alteramos o tipo de classe no arquivo `App.xaml`, mas não no *code behind*. Faremos isto abrindo `MasterDetailView.xaml.cs` trocando `ContentPage` por `MasterDetailPage`:

```
namespace TestDrive.Views
{
  public partial class MasterDetailView :  MasterDetailPage
  {
    public MasterDetailView()
    {
      InitializeComponent();
    }
  }
}
```

Corrigimos o erro, voltamos a `App.xaml.cs` e rodaremos a aplicação verificando como este `MasterDetailView` é exibido assim que realizamos login com sucesso. Os campos de "Usuário" e "Senha" serão preenchidos com "joao@alura.com.br" e "alura123" respectivamente. Apertaremos "Entrar", e... Temos um erro de comunicação com o servidor. Nossa conexão está funcionando bem, este erro não deveria estar ocorrendo.

Vamos investigar o porquê desta mensagem estar sendo exibida indo ao `LoginService.cs`, que é onde ela é lançada. O que encontramos aqui é um bloco *Try/Catch* envolvendo toda a parte do `using`, uma área muito grande de código para obtenção de erro de comunicação com o servidor.

Restringiremos esta declaração do bloco *Try/Catch* somente para a linha que faz o post, ou seja, àquela referente ao `PostAsync`, pois os erros que surgirem para além dela, "deixaremos estourar", acontecer, já que não podemos misturar uma exceção com outra. Vamos acrescentar um `try` apenas antes do `PostAsync`, sendo que o `catch` ficará depois do mesmo:

```
try
{

  var resultado = await cliente.PostAsync("/login", camposFormulario);
}
catch
{
    MessagingCenter.Send<LoginException>(new LoginException(@"Ocorreu um erro de comunicação cor
    Por favor verifique a sua conexão e tente novamente mais tarde."),
        "FalhaLogin");
}
```

Desta forma, restringimos a área em que o *Try/Catch* irá funcionar. Vamos pegar o erro de falha de comunicação se realmente houver um erro no `PostAsync`. Corrigiremos também o problema do resultado, pois há uma declaração sendo feita ali dentro, cujo resultado está sendo usado fora do bloco `try`. Teremos que declarar o resultado como um tipo `HttpResponseMessage`, portanto, antes de `try`, acrescentaremos este tipo, retirando o `var` antes de `resultado`, podendo utilizá-lo fora do escopo do `try`, também:

```
HttpResponseMessage resultado = null;

try
{
  resultado = await cliente.PostAsync("/login", camposFormulario);
}
```

Feito isto, vamos rodar a aplicação e verificar o que está realmente acontecendo quando tentamos ir ao `MasterDetailPage`. Preencheremos corretamente os dados relativos a "Usuário" e "Senha", apertando "Entrar" em seguida. Mais uma mensagem de erro é exibida, dizendo que `Master` e `Detail` precisam ser definidos antes da adição de `MasterDetailPage` a um contêiner.

Uma página `MasterDetailPage` precisa de um "mestre" e de "detalhe", e não os definimos na página `MasterDetailView.xaml`. Deste modo, recisaremos modificá-la para que estas duas informações sejam contidas (`Master` e `Detail`). Também removeremos o `Label`, uma vez que ele não faz parte da nossa `MasterDetailPage`.