01

Frame

Transcrição

Começaremos a editar a página de perfil do usuário e, idealmente, seria melhor se pudéssemos reunir as informações "soltas" na tela dentro de um elemento visual. Nosso layout ficará parecido com [aquele recebido pela especificação do cliente (https://s3.amazonaws.com/caelum-online-public/Xamarin+-+parte+3/Video2.1_01_wireframes-perfil.png)](), Aluracar.

Felizmente, o Xamarin Forms possui um controle para tal (uma moldura, ou *frame*), o qual utilizaremos para envolver os campos com as informações relativas ao usuário logado.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TestDrive.Views.MasterView"
             Title="Perfil">
  <Frame>
      <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
          <Label Text="João da Silva"></Label>
          <Label Text="joao@alura.com.br"></Label>
      </StackLayout>
  </Frame>
</ContentPage>
```

Vamos rodar a aplicação para verificarmos a aparência do controle, que engloba os campos de perfil do usuário. A partir do login, clicaremos no ícone do menu lateral e vemos que não há nenhuma alteração aparente. Isto ocorre pois o `Frame`, sozinho, não exibe nenhuma modificação, sendo necessário definir sua aparência, como a cor da borda a ser utilizada (a partir de `OutlineColor`, que indica a cor da borda externa).

Verificaremos esta alteração rodando a aplicação, logando e abrindo a página de perfil do usuário. Nenhuma alteração é visível ainda. O que não estamos sabendo fazer é definir algumas propriedades de centralização da moldura. Pausaremos a aplicação e definiremos uma margem para este `Frame`:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TestDrive.Views.MasterView"
             Title="Perfil">
  <Frame OutlineColor="Silver"
         Margin="15">
      <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
          <Label Text="João da Silva"></Label>
          <Label Text="joao@alura.com.br"></Label>
      </StackLayout>
  </Frame>
</ContentPage>
```

Temos agora uma moldura ocupando praticamente a tela toda, com exceção da margem de `15` que acabamos de definir. No momento, queremos que a moldura tenha uma altura menor, variando de acordo com a altura dos componentes que estamos colocando no centro.

Modificaremos a propriedade `VerticalOptions`, para deixá-la centralizada e expandida, fazendo com que ele respeite e siga o tamanho dos elementos que estão no centro da tela:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TestDrive.Views.MasterView"
             Title="Perfil">
  <Frame OutlineColor="Silver"
         VerticalOptions="CenterAndExpand"
         Margin="15">
      <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
          <Label Text="João da Silva"></Label>
          <Label Text="joao@alura.com.br"></Label>
      </StackLayout>
  </Frame>
</ContentPage>
```

Rodando a aplicação, vemos a modificação que acabamos de fazer:

Agora, continuaremos implementando esta interface tomando por base as especificações enviadas pelo cliente, neste mesmo `Frame`.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TestDrive.Views.MasterView"
             Title="Perfil">
  <Frame OutlineColor="Silver"
         VerticalOptions="CenterAndExpand"
         Margin="15">
      <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
          <Label Text="João da Silva"></Label>
```

```
            <Label Text="joao@alura.com.br"></Label>
            <Button Text="Perfil"></Button>
        </StackLayout>
    </Frame>
</ContentPage>
```

É possível conferir a criação deste botão quando rodamos a aplicação, fazendo login e acessando a página de perfil do usuário. O botão que acrescentamos está aparecendo, porém, sem ter uma ação implementada ainda.

Para melhorar o layout visualmente, entre os campos do perfil do usuário, poderíamos acrescentar um separador. No entanto, abrindo `MasterView.xaml` e buscando por algum *separator* ("separador" em inglês), não encontramos este tipo de controle.

Sendo assim, acrescentaremos um controle chamado `BoxView`, um retângulo com uma cor fixa, sólida. Este será definido como o valor da altura de `1px`. Também determinaremos a extensão horizontal da tela, dentro de `StackLayout`, e colocaremos este separador entre o `Label` e o `Button`. Para definir a cor deste separador, definiremos como `Color Gray`, que significa "cinza", em inglês.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TestDrive.Views.MasterView"
             Title="Perfil">
    <Frame OutlineColor="Silver"
           VerticalOptions="CenterAndExpand"
           Margin="15">
        <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
            <Label Text="João da Silva"></Label>
            <BoxView Color="Gray" HeightRequest="1" HorizontalOptions="Fill"></BoxView>
            <Label Text="joao@alura.com.br"></Label>
            <BoxView Color="Gray" HeightRequest="1" HorizontalOptions="Fill"></BoxView>
            <Button Text="Perfil"></Button>
        </StackLayout>
    </Frame>
</ContentPage>
```

Vamos rodar a aplicação para vermos como está o layout após estas modificações. Faremos login, clicando depois em "Entrar" e abrindo a página de perfil de usuário pelo ícone no topo da app.

Com esta `MasterView`, seria bom termos uma `ViewModel` relativa a ela. Criaremos a classe `MasterViewModel`, a partir do clique com o lado direito do mouse sobre a pasta "*ViewModels*" no menu lateral direito do Visual Studio, e a seleção de "*Add* > *Class*". Esta classe terá as propriedades a serem amarradas aos componentes visuais da `MasterView` que, até o momento, são os campos de nome de usuário e e-mail.

Em `MasterViewModel.cs`, tornamos a classe pública, dentro da qual definiremos as propriedades digitando "`propfull`" e apertando a tecla "TAB" duas vezes, de forma que o programa cria automaticamente sua estrutura. Em seguida, definiremos alguns valores iniciais para estas duas propriedades que acabamos de criar. Em `nome`, deixaremos um valor fictício, `meu nome`, fazendo o mesmo com `email`, ou seja, `meu email`.

```
namespace TestDrive.ViewModels
{
    public class MasterViewModel
    {
      private string nome = "meu nome";

      public string Nome
      {
          get { return nome; }
          set { nome = value; }
      }

      private string email = "meu email@gmail.com";

      public string Email
      {
          get { return email; }
          set { email = value; }
      }
    }
}
```

Feito isto, iremos ao `MasterView.xaml` para fazer o *binding* da propriedade de texto relacionadas a esta `ViewModel`. No arquivo, no `Label` que contém o nome do usuário ("João da Silva"), adicionaremos `Binding` referente à propriedade `Nome`. Na parte de e-mail, deixaremos a propriedade `Email`.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TestDrive.Views.MasterView"
             Title="Perfil">
  <Frame OutlineColor="Silver"
         VerticalOptions="CenterAndExpand"
         Margin="15">
      <StackLayout VerticalOptions="Center" HorizontalOptions="Center">
          <Label Text="{Binding Nome}"></Label>
          <BoxView Color="Gray" HeightRequest="1" HorizontalOptions="Fill"></BoxView>
          <Label Text="{Binding Email}"></Label>
          <BoxView Color="Gray" HeightRequest="1" HorizontalOptions="Fill"></BoxView>
          <Button Text="Perfil"></Button>
      </StackLayout>
  </Frame>
</ContentPage>
```

Com isto, falta relacionarmos a `ViewModel` como um `BindingContext` desta *view*. Por meio do *code behind* do `MasterView`, criaremos uma propriedade chamada `ViewModel`, que setaremos como sendo uma nova instância de `MasterViewModel`, e então configuraremos o `BindingContext` desta *view* como sendo a `ViewModel`:

```
namespace TestDrive.Views
{
    public partial class MasterView : ContentPage
    {
        public MasterViewModel ViewModel { get; set; }

        public MasterView()
        {
          InitializeComponent();
          this.ViewModel = new MasterViewModel();
          this.BindingContext = this.ViewModel;
        }
    }
}
```

Rodaremos a aplicação mais uma vez verificando como será exibida a `MasterView` conforme a `ViewModel`. Na página de perfil de usuário, estão "meu nome" e "meuemail@gmail.com", propriedades que estão vindo por definição de `ViewModel`. Como faremos para exibir as informações reais do usuário logado? Precisaremos modificar a `MasterView` para receber no construtor uma instância do usuário. Em `MasterView.xaml.cs`, passamos no construtor `Usuario` o argumento `usuario`, o qual também precisa ser passado no `ViewModel`.

```
namespace TestDrive.Views
{
    public partial class MasterView : ContentPage
    {
        public MasterViewModel ViewModel { get; set; }
```

```
        public MasterView(Usuario usuario)
        {
          InitializeComponent();
          this.ViewModel = new MasterViewModel(usuario);
          this.BindingContext = this.ViewModel;
        }
    }
}
```

Como foi visto, o `MasterViewModel` não possui argumento no construtor, então precisaremos fazê-lo, com `Usuario` como parâmetro, no arquivo `MasterViewModel.cs`. Para armazenarmos localmente uma instância do usuário nesta classe, criaremos um campo privado chamado `Usuario`.

Quando o construtor for chamado, a instância de `usuario` terá que ser definida de acordo com o parâmetro passado. Além disto, em vez de passarmos os dados *fake* de nome e e-mail, pegaremos as informações diretamente do objeto `usuario`. No acessor do `set` e com o e-mail, também faremos o mesmo.

```
namespace TestDrive.ViewModels
{
    public class MasterViewModel
    {
      public string Nome
      {
          get { return this.usuario.nome; }
          set { this.usuario.nome = value; }
      }

      public string Email
      {
          get { return this.usuario.email; }
          set { this.usuario.email = value; }
      }

      private readonly Usuario usuario;

      public MasterViewModel(Usuario usuario)
      {
        this.usuario = usuario;
      }
    }
}
```

O Visual Studio aponta como erro o fato de `usuario` não possuir e-mail. Quando entramos na classe de `usuario`, temos `nome` e nada mais. Em `Usuario.cs`, criaremos outra propriedade chamada `email`, acrescentando-se a linha abaixo:

```
public string email { get; set; }
```

Agora estamos obtendo e setando diretamente a partir do objeto `usuario`. Como modificamos a `MasterView.xaml.cs` para a adição do parâmetro `Usuario` no construtor, teremos que alterar também a classe que consome a instância desta outra classe `MasterView`, que chamamos a partir de `MasterDetailView`.

Precisaremos passar este novo argumento do construtor que não tínhamos antes. Para isto, iremos à aba `MasterDetailView.xaml.cs` e incluiremos este mesmo argumento, o parâmetro do construtor, definindo-se uma propriedade, um campo, que será privado, setado apenas uma vez. Feito isto, definiremos a instância do `usuario` como sendo a mesma que está sendo recebida pelo construtor.

```
public partial class MasterDetailView : MasterDetailPage
{
    private readonly Usuario usuario;

    public MasterDetailView(Usuario, usuario)
    {
      InitializeComponent();
      this.usuario = usuario;
    }
}
```

Na classe que está consumindo `MasterDetailView`, ainda falta determinarmos o construtor que acabamos de criar, passando este parâmetro do construtor que não tínhamos antes. Dentro da classe `App`, que está instanciando a `MasterDetailView`, passaremos também o parâmetro `usuario`. O método anônimo `usuario`, que recebe este parâmetro, será copiado e colado como referência ou argumento de `MasterDetailView`.

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