03

Tópicos

Transcrição

Dentre os tópicos apresentados no curso, começaremos criando uma solução Xamarin Forms a partir do zero, escolhendo um projeto em branco, que seja *Portable*. Depois, navegaremos pela solução para vermos seus projetos e, como foi explicado anteriormente, trabalharemos somente com o emulador do Android, portanto selecionaremos o projeto relativo a isto (`TestDrive.Droid`).

Optaremos pelo emulador acelerado para a execução do projeto e veremos o que é o XAML, XML especializado para aplicações do Xamarin. O XAML é utilizado também em projetos WPF, aplicações equivalentes às do Windows Forms, com uma "filosofia de interface" um pouco diferente, sendo usado no antigo *Silver Light* e, agora, no Xamarin.

Partiremos do `StackLayout` , que é um container, dentro do qual conseguimos colocar controles e, como o nome indica (*stack* signifca "pilha" em inglês), ele permite o empilhamento de controles dentro deste layout.

Em seguida, veremos a "grade", o controle *grid* do Xamarin, que possibilita a definição do número de colunas e linhas. As "caixinhas" contidas neste layout podem ser posicionadas de acordo com a grade, sendo organizadas de maneira bem fácil.

Depois, veremos o controle `ListView` , o qual recebe uma coleção de dados, a partir do qual é possível repetir a quantidade de linhas para cada item contida no C#.

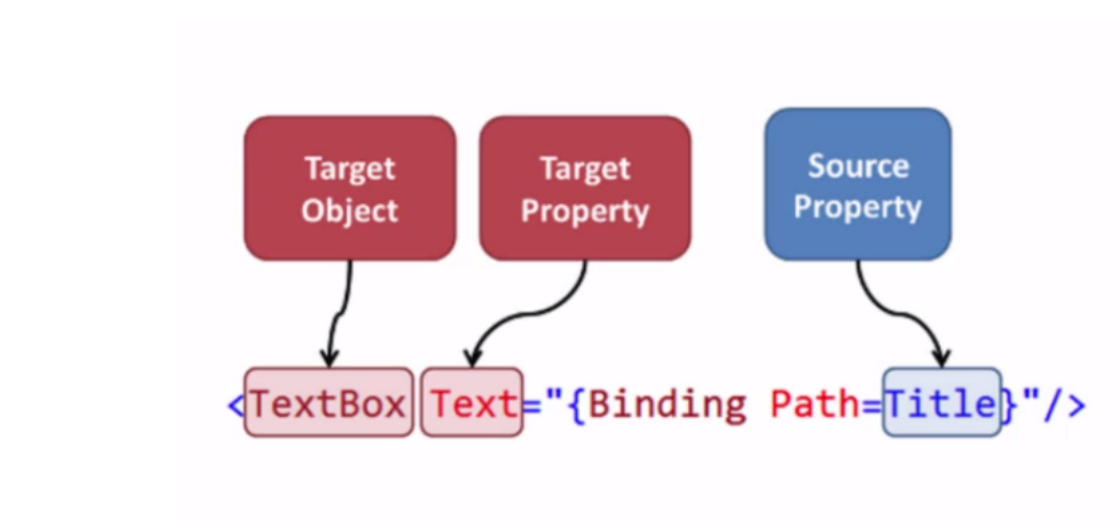
Veremos também o `Label` , que é um dos controles mais simples, e serve para exibir texto, podendo ter o formato mais simples e compacto, como em `<Label Text="{Binding nome}"></Label>` , ou outra sintaxe, cujo `Label` é "quebrado" em vários `spans` , controles menores. Para cada um deles, pode-se realizar alterações de fontes, tamanho, tipo, entre outros:

```
<Label>
    <Label.FormattedText>
        <FormattedString>
            <FormattedString.Spans>
                <Span Text="Fiesta 2.0"></Span>
                <Span Text=" - "></Span>
                <Span Text="R$ 52000"></Span>
            </FormattedString.Spans>
        </FormattedString>
    </Label.FormattedText>
</Label>
```

Veremos também o `StackLayout` com orientação horizontal, que empilhará não mais de cima para baixo, e sim da esquerda para a direita.

Aprenderemos sobre a Navegação e como implementá-la dentro de uma aplicação Xamarin. Utilizaremos uma classe específica de pilha de navegação com as páginas a serem navegadas e, ao apertarmos o botão "Voltar" do dispositivo, no caso o Android, removeremos estas páginas adicionadas, conseguindo voltar à anterior. Desta forma, conseguimos fazer um fluxo de navegação do aplicativo.

Depois, veremos o `Binding`, conceito bastante utilizado no Xamarin, a partir do qual associamos a propriedade de um controle visual, por exemplo, o texto de um `Label` ou `Input`, algo do tipo, trocando seu valor. Em vez de forçar a mudança diretamente no controle que exibirá o dado, conseguiremos fazê-lo a partir de uma propriedade, de uma classe que se encontra amarrada a um controle visual, o qual automaticamente será notificado desta alteração, exibindo-se seu novo valor.



Em seguida veremos o `TableView`, utilizado para coleta de dados, formulários, cujas configurações podem ser habilitadas ou desabilitadas conforme queremos. Veremos também como utilizar o botão "Próximo" para a navegação para a página seguinte.

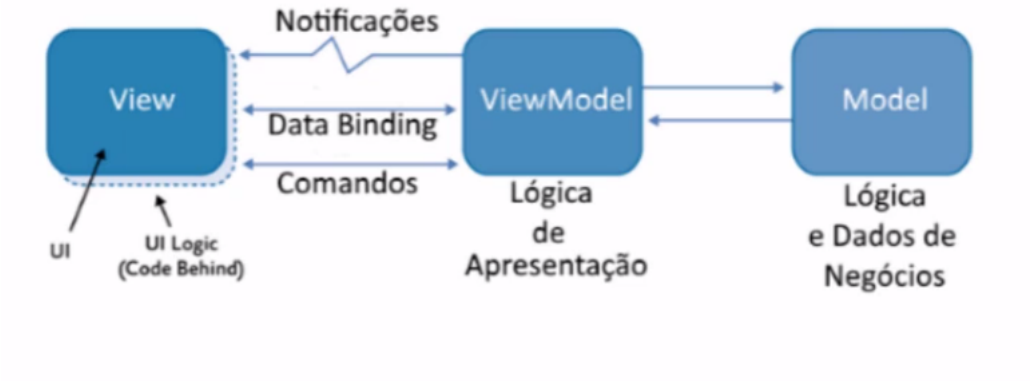
Por trás dos panos, no código C#, veremos também como faremos notificações de que houve mudanças em determinada propriedade à interface Xamarin Forms, para que os controles sejam atualizados com novo valor de uma classe amarrada ao `Binding` do Xamarin Forms, por meio de `OnPropertyChanged`.

Na última tela serão feitos os cadastros, também com controle do agendamento, formulário dentro do `TableView`, e controle de entrada de dados (utilizando o `EntryCell`) por meio de teclados diferentes; por exemplo, o campo "Nome" será preenchido com teclado normal, o "Fone" e "E-mail", com teclado especializado, para números e símbolos.

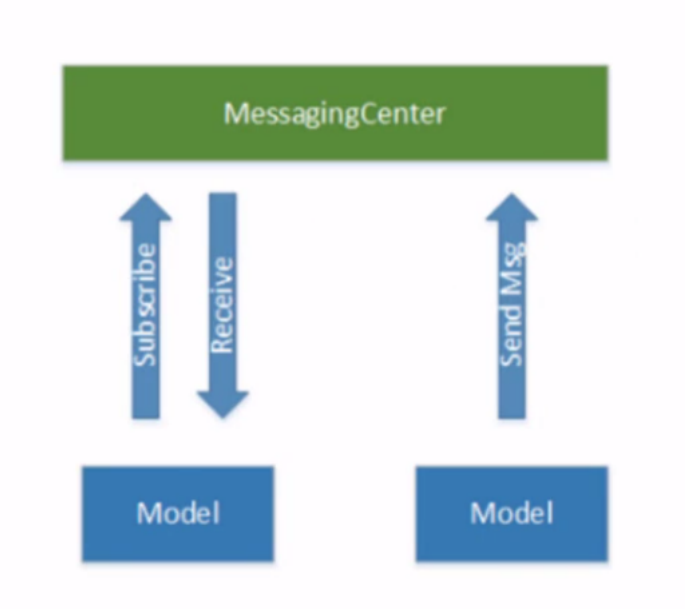
Teremos controles especializados para a coleta de dados de data e hora. Em vez de preenchermos estes campos manualmente, veremos como fazê-lo de acordo com o dispositivo que está sendo usado.

Para a exibição de mensagens ao usuário, utilizaremos uma tela de aviso com título e mensagem, e um botão de confirmação da mesma, com o `DisplayAlert`.

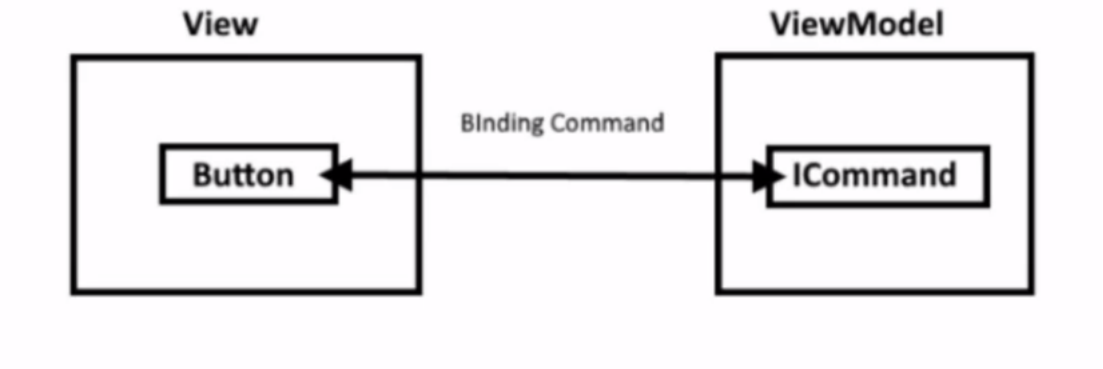
Veremos a fundo como desenvolver o padrão MVVM (*Model-view-viewmodel*), também muito utilizado em aplicações Xamarin. É fundamental adotá-lo para realizarmos o desacoplamento dos dados e dependências entre os componentes Xamarin, fazendo-se por exemplo o *Data Binding*, usando comandos e notificações, entre outros.



Com isto, conseguimos tirar da nossa *view*, do código que exibirá os dados, a lógica de negócios, de apresentação, movendo para componentes mais especializados nestas tarefas. Para prosseguirmos neste desacoplamento, usaremos a mensageria do Xamarin Forms, um componente chamado `MessagingCenter`, que fará este "meio de campo" na troca de mensagens entre os componentes, já que não ficarão se referenciando mutuamente, e sim enviando e recebendo mensagens, sem que se saiba quem os faz.



Utilizaremos um comando para a ação de um botão que, em vez de aparecer no *code behind* da página, traremos ao *view model*, dentro do qual faremos a execução em uma classe especializada chamada `ViewModel`. Assim, conseguiremos utilizar o conceito de validação do botão, quando ele é executado ou não.



Por fim, trocaremos a lista fixa, inicialmente preenchida via código C# e posteriormente alterada pelo acesso a um serviço HTTP da rede, populando-a no início da aplicação. Ao fim do agendamento do *Test Drive* pelo usuário, ele clicará em "Agendar", e consumiremos um serviço para salvar o agendamento no servidor.

Assim, completamos o ciclo deixando a aplicação se comunicar com o mundo, sem deixá-la isolada. Espero que gostem, muito obrigado, e boa aula!