07

DetalheViewModel

Transcrição

[00:00] Então, agora a gente está começando a implementar o padrão MVVM, que é o Model View ViewModels, a gente já começa a ver essa estrutura aparecendo na nossa solução. Aqui no nosso projeto, a gente já tem o Models aqui em cima, o View e o ViewModels, aqui é o M, aqui é o V e aqui é o VM.

[00:33] Então, o que a gente tem agora? No nosso xaml da Listagem View, não mais utilizando aqui em baixo o code behind como origem do binding, como o binding context, porque agora com o binding context, a gente utiliza o ListagemVieModel. Então, o que falta ainda é a gente trocar o binding das outras páginas, das outras views.

[01:02] Então, a gente tem aqui ainda o DetalheView apontando para o code behin, como binding context e o agendamento view a mesma coisa. Então, o que a gente vai fazer é mover a lógica de apresentação dessas outras duas páginas para cima, aqui para o ViewModels. Então, a gente vai fazer isso agora.

[01:25] Então, vamos dar uma olhada agora no DetalheView e ver o que a gente tem. Então, tem bastante propriedades que poderia ser movida para uma ViewModel. Eu vou selecionar todas as propriedades que a gente vai mover para o Viewmodel, só que primeiro, claro, eu tenho que criar um viewmodel.

[01:50] Eu vou criar um DetalheViewModel na pasta ViewModels, eu adiciono uma nova classe, chamada DetalheViewModel, vou deixar essa classe pública vamos dar uma olhada aqui no DetalheView, o que a gente pode mover para lá. Eu vou selecionar tudo o que me interessa, até aqui um pouco antes do construtor, vou recortar todas essas propriedades.

[02:23] E vou mover para o ViewModel. Agora, o que eu vou fazer? Eu vou ter que modificar a nossa classe do code behind, para ele poder apontar para o novo binding context. Então, o binding context, aqui em baixo, não vai mais ser o this, não vai mais ser o code behind, vai ser uma nova instância do ViewModel, que é o DetalheViewModel.

[02:52] Então, eu coloco aqui: new DetalheViewModel e vou resolver essa referência. E aqui no DetalheViewModel, eu vou ter que colocar um using TestDrive.Models. Vamos ver agora... O Visual Studio está reclamando aqui de alguns métodos que são o OnPropertyChanged, esse problema não aconteceu no nosso code behind, porque o code behind, ele herda.

[03:38] Ele faz uma herança de algumas classes, que já possuem o método OnPropertyChanged. Então, o que a gente vai precisar fazer é implementar esse método manualmente. Então, a primeira coisa que a gente faz é declarar (que é nosso) ViewModel, ele vai implementar, ele tem que implementar uma interface chamada: INotifyPropertyChanged.

[04:04] Então, eu vou colocar essa interface aqui: INotifyPropertyChanged. Então, eu vou resolver essa referência, resolvido. Agora, ele está reclamando que a nossa classe não implementa essa interface, então é isso que a gente vai fazer agora. Vou clicar com o botão direito, vou refaturar, implementa a interface.

[04:38] Com isso, ele criou método novo, aqui em baixo ele criou esse evento, que é o PropertyChanged. Então, agora, eu tenho que implementar esse método que está faltando, que é o OnPropertyChanged. Então, eu vou copiar esse nome, OnPropertyChanged. Eu vou criar aqui: public void e ele vai receber um nome.

[05:06] Ele vai receber o nome de uma propriedade, esse nome, eu vou chamar de name e o name, ele, por default, ele pode ter um valor, string vazia. Além disso, eu tenho que colocar um modificador, aliás, eu posso colocar um modificador chamado: CallerMemberName. Então, esse CallerMemberName, ele faz o seguinte.

[05:39] Se eu não passar nenhum parâmetro para o PropertyChanged, para o OnPropertyChanged, como aqui em cima nessa linha, a gente está vendo que aqui, ele está passando... Aliás, ele não está passando nenhum parâmetro para a chamada desse método.

[05:55] Então, nesse caso, o name, ele vai assumir que o name vai ser o CallerMemberName, que vai ser o nome dessa própria propriedade, que é o TemMP3Player. Então, se eu não passo nada na chamada desse método, ele vai assumir que o name, é o name da propriedade que está chamando esse método.

[06:19] Então, agora, dentro do corpo desse método, eu tenho que invocar o evento que está declarado aqui em cima, que é o PropertyChanged. Então, eu vou fazer o invoke e vou passar aqui Invoke... Perdão, invoque, eu vou passar aqui a instância de quem está chamando, no caso, essa própria ViewModel, então é a referência this, que é essa ViewModel.

[06:50] E aí, eu vou colocar os novos argumentos da chamada desse evento. Então, eu coloco o new e aqui dentro, eu vou colocar o nome da propriedade e o nome da propriedade é a própria propriedade name, que está sendo chamada nesse método. Só que tem um porém, se o PropertyChanged, se ele for nulo, eu não posso fazer a execução desse linha.

[07:20] Então, nesse caso, eu posso utilizar o operador de nulo condicional. Então, o nulo condicional, o que seria? Se o PropertyChanged for nulo, então, ele não faz nada ou melhor, se ele for diferente de nulo, ele vai invocar esse método. Então, para eu não precisar fazer esse condicional, eu posso simplesmente utilizar o nulo condicional, que é o operador "?." (interrogação, ponto).

[07:50] Então, isso aqui é o nulo condicional, que vai ter a mesma função. Então, agora, um ponto muito importante, para poder fazer funcionar o MVVM corretamente, é fundamental que o seu ViewModel, ele tenha esse método OnPropertyChanged funcionando, porque sem isso, não é possível, por exemplo, notificar que uma propriedade foi alterada.

[08:25] E com isso, você vai ter problemas na hora de notificar outras propriedades que estão relacionadas, como por exemplo, quando a gente muda a chave de um acessório do nosso veículo, a gente não vai ter a propriedade total, o preço total, não vai ser atualizado, porque ele não está sendo notificado, o sistema

[08:46] O Xamarin, ele não está sendo notificado de que existe uma mudança em alguma propriedade, que vai afetar outras propriedades. Então, agora, o que a gente vai fazer é ir lá no nosso code behind e no code behind, que a gente já definiu qual é o contexto do binding, a gente vai ter que modificar aqui, para remover esse this.veiculo.

[09:15] A gente vai ter que mover isso lá para o nosso ViewModel. Aqui no ViewModel, eu vou colocar no nosso construtor, eu vou ter que criar um construtor para o ViewModel e eu vou ter que passar uma referência do veículo para a propriedade veículo dele. Só que o veículo no code behind, ele vinha na assinatura do método, na assinatura do construtor.

[09:47] Agora, eu tenho que fazer a mesma coisa aqui, eu tenho que pegar esse construtor e mover ele para o nosso ViewModel. Então, estou movendo para o ViewModel. Então, aqui ficou bastante enxuto o nosso code behind e agora, ele está reclamando... Opa, eu removi a mais aqui, eu removi o construtor que não deveria.

[10:13] Então, coloquei de volta e aí, o veículo que está sendo recebido pela view, ele vai ser passado também para o ViewModel. Então, aqui no DetalheViewModel, na chamada para o construtor, eu vou passar o veículo também. Então, agora que eu já movi tudo o que eu precisava para o DetalheViewModel, eu vou rodar a aplicação e ver se a segunda página, que é a página de detalhes, está funcionando corretamente.

[10:42] Então... opa, o Visual Studio acusou aqui um erro, vamos ver o que aconteceu. Bom, ele está tentando acessar uma propriedade veículo do nosso code behind, porque claro, eu removi essa propriedade, não deveria, vou colocar de volta, propriedade veículo, que vai ser (do tipo do veículo).

[11:04] E aqui no construtor desse code behind, eu vou colocar: this.veiculo = veiculo. Agora, resolvi o problema, vamos rodar de novo, ver como aparece na nossa tela. Rodando o emulador, vou selecionar um veículo. Legal, apareceu a segunda tela, eu consigo trocar os acessórios, ligar e desligar.

[11:35] Então, o valor total está sendo alterado conforme eu mexo, conforme eu seleciono ou tiro os acessórios, está modificando o valor total. Muito bem. Então, agora a gente tem a página de detalhe do veículo funcionando de acordo com o padrão MVVM. Então, agora a gente vai partir para a última página e colocar também um ViewModel para o agendamento.