O MVVM é um pattern que foi criado em 2005, por John Gossman, um dos arquitetos do WPF e Silverlight na Microsoft. O MVVM assemelha-se em alguns aspectos o MVC (Model View Controller) e ao MVP (Model View Presenter), podemos até dizer que o MVVM é uma especialização do MVP adaptado para a arquitetura do WPF E Silverlight. Conceitualmente, o MVVM e o MVP são idênticos, o que os diferencia é que o MVVM é específico para a arquitetura do WPF e Silverlight e o MVP é independente de plataforma. O MVVM, visa estabelecer uma clara separação de responsabilidades em uma aplicação WPF e Silverlight, mantendo uma espécie de façade entre o Modelo de objetos (entenda classes de negócio, serviços externos e até mesmo acesso a banco de dados) e a View que é a interface, com a qual o usuário interage. Para entendermos melhor, como se dá esta separação e visualizar como os componentes interagem dentro deste cenário, observe a figura abaixo:

Como ilustra a figura acima, há uma clara separação das camadas. A **camada Model(Modelo) não conhece a View(Camada de apresentação) e vice-versa, na verdade a View conhece a ViewModel e se comunica com ela através do mecanismo de binding**. E são os avançados mecanismos de binding, eventos roteados e comandos roteados, que fazem do MVVM um pattern poderoso para construção de aplicações WPF e Silverlight. Talvez você esteja se perguntado: tá bom a View está ligada a ViewModel através do mecanismo de binding mais como funciona esta comunicação? Observe abaixo:

Veja como é simples, a View atravéz do databinding interage com a ViewModel notificando a ocorrência de eventos e o disparo de comandos. A ViewModel por sua vez, responde a esta notificação realizando alguma ação no modelo; seja obtendo algum dado, atualizando ou inserindo informações no modelo.

Responsabilidades e características

View – A responsabilidade da View é definir a aparência ou estrutura que o usuário vê na tela. O ideal é que o codebehind da view, contenha apenas a chamada ao método InitializeComponent dentro do construtor, ou em alguns casos, código que manipule os controles visuais, ou crie animações; algo que é mais difícil de fazer em XAML. A View se liga ao ViewModel, através da propriedade DataContext que é setada para a classe ViewModel correspondente à aquela View. Veja no código de exemplo, que será disponibilizado para baixar, como é feita declarativamente a ligação da View com o ViewModel através da propriedade DataContext:

Características comuns

- A View é um elemento visual, como um objeto Window, Page, UserControl ou DataTemplate.
- A View referencia a ViewModel através da propriedade DataContext. Os controles da View são preenchidos com propriedades ou comando, expostos pela ViewModel.
- O codebehind da view, define comportamentos visuais (Behaviors) difíceis de expressar em XAM.

ViewModel – A responsabilidade da ViewModel no contexto do MVVM, é disponibilizar para a View uma lógica de apresentação. A View Model não tem nenhum conhecimento específico sobre a view, ou como ela implementada, nem o seu tipo. A ViewModel implementa propriedades e comandos, para que a View possa preencher seus controles e notifica a mesma, caso haja alteração

de estado; seja através de eventos ou notificação de alteração. A ViewModel é peça fundamental no MVVM, por que é ela quem vai coordenar as iterações da View com o Model, haja vista, ambos não terem conhecimento um do outro. E além de tudo isto, a ViewModel, também pode implementar a lógica de validação, para garantir a consistência dos dados.

Características comuns

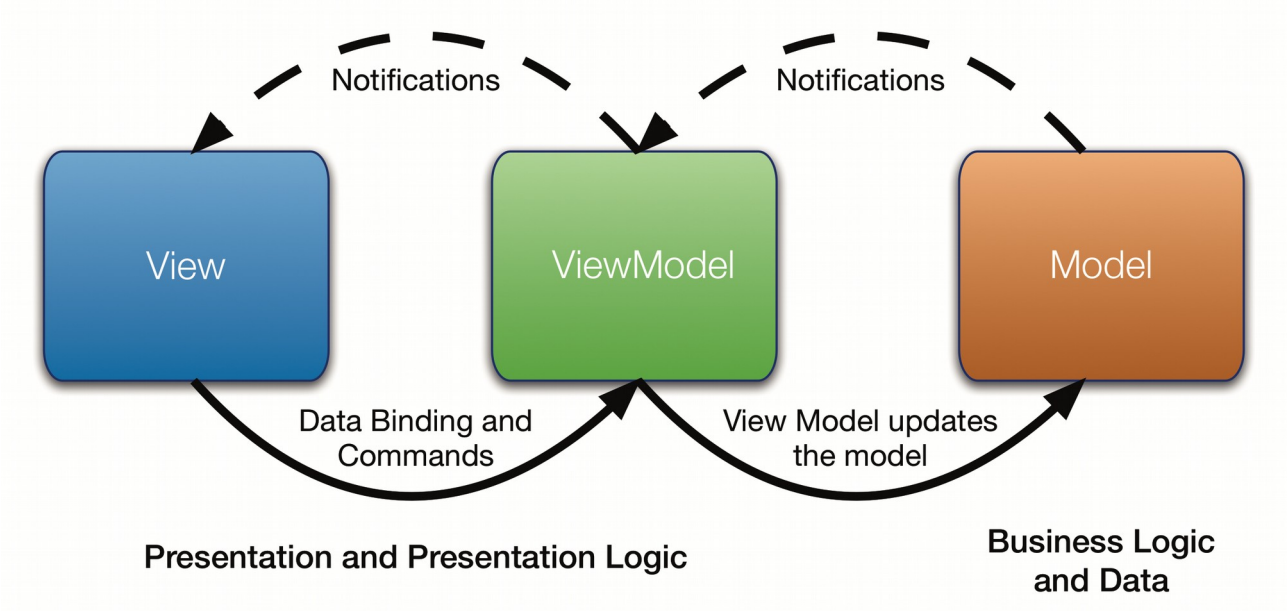
- A ViewModel é uma classe não visual, que expões para a View uma lógica de apresentação.
- A ViewModel é testável, independentemente da View ou Model.
- A ViewModel coordena as intenções entre a View e o Model.
- A ViewModel não referencia a View, na verdade não tem nenhum conhecimento sobre a mesma.
- A ViewModel implementa as interfaces INotifyPropertyChanged
- A ViewModel expões propriedade e comando, para que a View possa utilizar para preencher seus controles; e notifica a View quando o estado de uma determinada propriedade muda, via implementação da interface INotifyPropertyChanged ouINotifyCollectionChanged.
- A ViewModel pode conter a lógica de validação, através da implementação da intefaces IDataErrorInfo ou INotifyDataErrorInfo.

Model – o Model no MVVM, encapsula a lógica de negócios e os dados. O Modelo nada mais é do que o Modelo de domínio de uma aplicação, ou seja, as classes de negócio que serão utilizadas em uma determinada aplicação. O Modelo também contém os papéis e também a validação dos dados de acordo com o negócio, cuja aplicação em questão visa atender.

Características comuns

- O Modelo são classes que encapsulam a lógica de negócios e os dados.
- O Modelo não referencia diretamente a View ou ViewModel.
- O Modelo provê eventos de notificação de mudança de estado, através das interfaces INotifyPropertyChanged and INotifyCollectionChanged. Isto facilita o preenchimento de dados na View.
- O Modelo de dados contém validação de dados e reporta os erros através da interface INotifyDataErrorInfo.
- O Modelo de dados geralmente é utilizado, com um repositório (pode ser o Repository Pattern) ou serviço.

O MVVM permite a você ter uma visão, da clara separação da Interface com o usuário(View), sua lógica de apresentação (ViewModel) e os seus Dados(Model). E trabalhando desta forma, temos separação de responsabilidades, desacoplamento e conseguimos evoluir e manter melhor as nossas aplicações.



https://www.youtube.com/watch?v=ijXjCtCXcN4