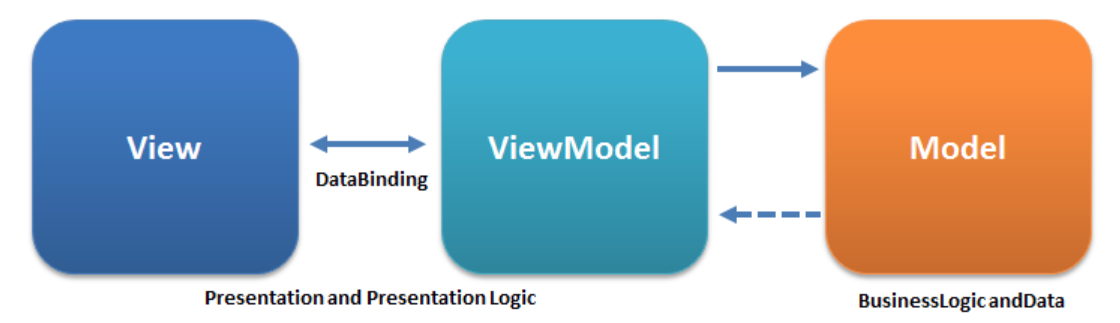
**Wzorzec MVVM**



Mamy trzy warstwy  
Warstwa View mamy wszystkie widoki, czyli pliki xaml.  
W warstwie Model mamy dane: nasze klasy domenowe, obiekty biznesowe i cała logika biznesowa.  
Warstwa ViewModel łączy warstwę widoku z warstwą modelu.   
Gwarantuje nam, że po wykonaniu jakieś akcji z widoku zostanie wywołana odpowiednia logika biznesowa. Warstwa ta komunikuje się z widokiem za pomocą wiązania danych.  
Model nie potrzebuje żadnych danych z warstwy View i ViewModel, równie dobrze mogła by być przeniesiona do innej aplikacji.  
Dzięki temu sam widok może projektować osoba, która nawet nie musi programować w C#.  
Mamy w aplikacji większy porządek. Możemy pisać testy jednostkowe.

**Zewnętrzne Frameworki Ułatwiające Pisanie Aplikacji w WPF:**

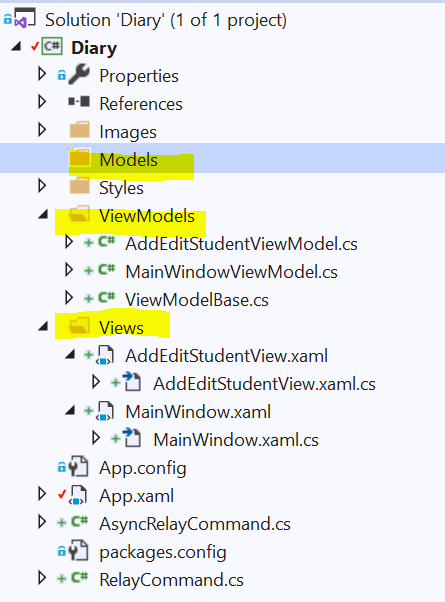
**Prism**Caliburn.Micro  
MVVM Light

Z powodzeniem możemy również pisać duże aplikacje, nie używając żadnego frameworka.

Frameworki te opakowują wiele mechanizmów, pozwalają pisać używając mniej kodu i poświęcając mniej czasu.

**Szkielet MVVM**

Tworzymy katalogi dla każdej z warstw:



Do katalogu Views wgrywamy wszystkie pliki xaml z widokami  
W katalogu ViewModel tworzymy pliki z klasami po jednym pliku dla każdego z widoków  
Nazwę klasy tworzymy dodając do nazwy widoku słowo ViewModel.  
Tworzymy jeden bazowy ViewModel np. ViewModelBase

**Wiązanie danych pomiędzy View a ViewModel**

1 krok wskazanie w View ViewModelu

Możemy to zrobić w pliku xaml, ale wygodniej jest to zrobić w code behind.  
Ten krok powtarzamy dla każdego widoku.

|  |
| --- |
| namespace Diary.Views  {  /// <summary>  /// Interaction logic for MainWindow.xaml  /// </summary>  public partial class MainWindow : MetroWindow  {  public MainWindow()  {  InitializeComponent();  DataContext = new MainWindowViewModel();  }  }  } |

**Krok 2: DataBinding**Bindujemy zdarzenie Click() przycisku odśwież  
W tym miejscu wskazujemy na właściwość **ICommand** czyli również **RelayCommand**, która musi się znaleźć w ViewModelu

|  |
| --- |
| <Button Command ="{Binding RefreshStudentsCommand}" Content="Odśwież" Height="30" Width="100" Margin="5"/> |

W **ViewModel** tworzymy właściwość wskazaną pod przyciskiem, czyli: RefreshStudentsCommand   
Następnie w konstruktorze inicjalizujemy tą właściwość, używamy do tego klasy RelayCommand  
i przekazujemy metodę, która ma się wykonać, czyli delegata wskazującego na tę metodę

|  |
| --- |
| Namespace Diary.ViewModels  {  class MainWindowViewModel : ViewModelBase  {  public MainWindowViewModel()  {  RefreshStudentsCommand = new RelayCommand(RefreshStudents,CanRefreshStudents);  }  public ICommand RefreshStudentsCommand { get; set; }  }  } |

Następnie klikamy na tych metodach i wywołujemy generate method  
generują się metody zgodne z sygnaturą konstruktora kalsy RelayCommand

|  |
| --- |
| public MainWindowViewModel()  {  RefreshStudentsCommand = new RelayCommand(RefreshStudents,CanRefreshStudents);  }  private bool CanRefreshStudents(object obj)  {  throw new NotImplementedException();  }  private void RefreshStudents(object obj)  {  throw new NotImplementedException();  } |

Pod właściwość ICommand RefreshStudentsCommand możemy podstawić obiekt klasy RelayCommand, ponieważ ten implementuje ten interfejs ICommand.