Slajd3

Dlaczego MVVM ?

Rozpoczynając wywód na temat MVVM należy najpierw zadań sobie kilka podstawowych pytań na temat MVVM jako wzorca projektowego :

Czy musisz dzielić projekt z projektantem (grafikiem) i mieć pewną elastyczność i niezależność od jego kodu, pisząc projekt w tym samym czasie ?

Czy chcesz pisać kod, który może zostać zastosowany wielokrotnie dla kilku projektów w Twojej firmie?

Czy chciałbyś żeby zmiana UI nie wiązała się ze zbytnią ingerencją w logikę aplikacji ?

Czy chcesz żeby podczas pisania projektu łatwiej można było zaimplementować testy jednostkowe?

Jeśli odpowiedziałeś twierdząco na którekolwiek z tych pytań wzorzec Model – ViewModel - View jest dla Ciebie.

Slajd 4

Obrazek MVVM

Slajd5

Model.

Model w tym wzorcu projektowym jest to nic innego jak pewna struktura, przechowująca dane na których pracujemy wykonując program. Prosty przykład modelu jest to Osoba, struktura ta będzie posiadać pola takie jak Imię , Nazwisko, Data urodzenia, Miejsce Zamieszkania,

Jeśli mówimy o modelu to najważniejszą rzeczą jaką należy zapamiętać w tym wzorcu to to, że model przechowuję informacje a nie zachowania albo metody odpowiedzialne za manipulację danymi.

Logika biznesowa powinna być trzymana jak najdalej od modelu. Model ma przechowywać tylko dane.

Slajd 6

Screen modelu(kod)

Z opisem

Slajd 7

Widok .

Widok w tym modelu to coś z czym użytkownik aplikacji ma do czynienia, z czym prowadzi interakcję. Jest to nic innego jak prezentacja danych. Widok korzystając z różnych bibliotek zdobi dane tak by były przedstawione

bardziej przestępnie dla użytkownika. Widok może również odpowiadać za wstępną walidację danych przedstawionych użytkownikowi. Widok zazwyczaj obsługuję zdarzenia dziejące się na urządzeniach wejścia (myszka , klawiatura), które mogą zmieniać dane które zawiera model.

W MVVM widok jest 'aktywny' w przeciwieństwie do widoku z wzorca MVC gdzie widok był całkowicie manipulowany przez kontroler. Teraz widok może jak już wcześniej powiedziałem prowadzić walidacje, posiadać pewne zachowania, które wymagają podstawowej wiedzy na temat Model i ViewModel. Dopóki te zachowania mogą być zmapowane na właściwości i metody, widok jest odpowiedzialny za swoje zdarzenia i nie przejmuje części odpowiedzialności od ViewModel.

Jedna ważna rzecz do zapamiętania :

Widok nie jest odpowiedzialny za utrzymanie swojego stanu. Synchronizuje to z ViewModel.

Slajd 8

Screen przykładowego widoku w XAML-u

Slajd 9

ViewModel.

ViewModel jest centrum całego modelu i jest chyba ze wszystkich części najbardziej istotny do omówienia, ponieważ mówi nam o koncepcji w której Widok jest całkowicie odseparowany od modelu. Weźmy na przykład datę urodzenia w modelu przechowywana jest informacja o tym jaka jest wartość daty. W widoku będzie przechowywana informacja na temat formatowania daty podczas wyświetlania. Kontroler będzie tutaj swoistym mostem pomiędzy tymi dwoma konstrukcjami. Może pobrać dane z widoku i umieścić je w modelu albo może pobrać dane z modelu przetłumaczyć właściwość i wysłać ją do wyświetlenia widokowi.

ViewModel uwidacznia metody i inne zdarzenia które pomagają zachować stan widoku, manipulować modelem ponieważ coś zostało wykonane w widoku i uruchomić pewne zdarzenia w widoku które będą miały wpływ na sam widok.

Slajd 10

Slajd 11

ViewModel- View

- Widok i Widok modelu komunikują się poprzez 'data-bindeing' , wywoływanie metody, właściwości, zdarzeń i wiadomości

- Widok modelu wysyła informacji nie tylko o modelu ale również o jego właściwościach i rozkazy

- Widok obsługuje swoje własne zdarzenia w UI, następnie mapuje je na metody w Widoku modelu

- model i właściwości w Widoku Modelu są przekazywane z widoku przez 'two-way data-binding'

Slajd 12

ViewModel-Model

-Widok modelu może przedstawić model albo jego właściwości dla ‘data-bing-u’

-Widok modelu może zawierać interfejs pozwalający manipulować modelem

---

Slajd 13

Jest kilka frameworków które bezpośrednio korzystają z wzorca MVVM, najpopularniejszym jest chyba MVVM light , ale są też Frameworki które są też dostępne dla ogólnego użytku, prism , caluburn , czy też właśnie MVVM cross.

Slajd 14

Będąc developerem chcąc pisać aplikacje chcemy żeby każdy kto chce skorzystać mogł to uczynić , niestety jak pokazują statystyki nie wszyscy posiadają smart fony z Windows phone , udział w rynku jest mniej więcej taki że każdy system operacyjna na smart fony zabiera dużą część rynku. I co w takim razie? Jest kilka możliwości jak developer aplikacji mobilnych może poradzić sobie z tym problemem, możemy np. nauczyć się pisać w objctiveC dla iOS i nauczyć się pisać w Javie dla Androida . To jest dobry sposób żeby zacząć pisać uniwersalne aplikacje na wszy skit systemu operacyjne w ciągu 2-3 lat. Jest też drugi sposób można skorzystać z Frameworka MVVMcross który działa na podstawie wzorca MVVM . Jak mówiłem wcześniej mostem pomiędzy View a model View jest databinding korzystając z mvvm cross tworzymy najzwyczajniej w świecie oddzielny widok dla każdego systemu operacyjnego dzieki czemu może włożyć dużo mniej trudu w pisanie aplikacji dla wielu systemów Inny databinding i inny widok reszta czyli , całe zachowanie aplikacji i baza danych zostaje wspólna .

Slajd 15

O MVVM cross

Kiedy została stworzona , po co ? przez kogo ?

Mvvm cross jest open-surceowym frameworkiem który został stworzony przez Sturta Lodge’a. Bazuje na wzorcu MVVM i jest stworzony by ułatwiać pisanie Aplikacji mobilnych na wiele platform takich jak Xamarin.Android, Xamarin.iOS, Windows Phone, Windows Store, WPF, and Mac OS X.

Slajd 16,17,18 – mvvmCross Statement

Slajd 19

Xamarin – jest to zestaw narzędzi które dostarczają wysokiej jakości zestaw dzięki któ®emu jesteś w stanie pisać aplikacje dla różnych platform w c#. Do wykorzystywania xamarin możesz używać zarówno xamarin

studio jak i visual studio wzbogacone o odpowiednie dodatki. Od strony dewelopera xamarin doisotarcza trzy podstawowe narzędzia Xamarin.Mac, Xamarin.iOS (MonoTouch.dll) and Xamarin.Android (Mono.**A**ndroid.dll).

Mvvm i xamarin

Używanie mvvm cross dla androida i iOS

Używanie mvvmcross jest proste i przejrzyste ponieważ wystarczy dodać kilka NuGet pakietów. Po tym mamy praktycznie kilka podstawowych kroków które jeśli pokonamy to uruchomimy aplikację .

Slajd 20

Musimy podzielić aplikację zasadniczą naczęść CoreProject – czyli model i widok modelu która musi zawierać klasę App , oraz na części w zależności od platformy na któ®ej będziemy to chcieli wyświetlić.

-

Zacznijmy od klasy App

```
public class App : MvxApplication
{
  public override void Initialize()
  {
    this.CreatableTypes()
      .EndingWith("Service")
      .AsInterfaces()
      .RegisterAsLazySingleton();
    this.RegisterAppStart<HomeViewModel>();
  }
}
```

Slajd 21

W twojej drugiej części (albo dla iOS albo Androidowej) musisz stworzyć klase Setup klasa ta posiada referencje to Core przez co cześć druga będzie miała informacje o core.

```
public class Setup : MvxAndroidSetup
{
  public Setup(Context applicationContext) : base(applicationContext)
  {
  }
```

```
  protected override IMvxApplication CreateApp()
  {
    return new Core.App();
  }
}
```

Slajd 22

Czyli klasa setup tworzy aplikacje korzystając z app. Ładuje również model widoku podczas uruchomienia wykorzystując RegistarAppStart

Każdy widok (iOS android, win phone) jest częścią swojej aplikacji. Jest to jedyna części która będzie się zmieniać podczas tworzenia specyficznych aplikacji. Dla Androida klasa czerpie z MvxActivity dla iOS z MvxViewControlller

```
[Activity(ScreenOrientation = ScreenOrientation.Portrait)]
public class HomeView : MvxActivity
{
  protected override void OnViewModelSet()
  {
    SetContentView(Resource.Layout.HomeView);
  }
}
```

Slajd 23

Mvvm cross musi znać Widok Modelu z którym powiązany jest dany widok . można to zrobić za pomocą nadpisania właściwości ViewModel w widoku albo korzystając z atrybutu MvxViewForAttribute.

```
 [Activity(ScreenOrientation = ScreenOrientation.Portrait)]
[MvxViewFor(typeof(HomeViewModel))]
public class HomeView : MvxActivity
{ ... }
```

NA iOS i na Androidzie widoki zaprojektowane są wg trohche innych podejść . w iOS widok definiowany jest w c#. W adroidzie też możesz użyć c# . Możesz też użyć AXML (xml definiujący widok dla androida). Z tego względu data bindings będą się różnic w zależności od platformy.

W iOS tworzysz plik BindingDescripton który jest mostem miedzy widokiem i widokiem modelu.

W tym przypadku mówisz co chcesz porezsalc do widoku .

```
var label = new UILabel(new RectangleF(10, 10, 300, 40));
Add(label);
var textField = new UITextField(new RectangleF(10, 50, 300, 40));
Add(textField);
var set = this.CreateBindingSet<HomeView,
  Core.ViewModels.HomeViewModel>();
set.Bind(label).To(vm => vm.Hello);
set.Bind(textField).To(vm => vm.Hello);
set.Apply();
```

---

slajd 24

W AXML możesz użyć atrybutu MvxBind by dokonać databinding.

```
<TextView xmlns:local="http://schemas.android.com/apk/res-auto"
        android:text="Text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tripitem_title"
        local:MvxBind="Text Name"
        android:gravity="center_vertical"
        android:textSize="17dp" />
```

atrubut MvXBind bierze taki parametr żeby ……

---

slajd 25

AMXL używa data Banndingu domyślnie w dwie strony. Więc tutaj trzeba uważać, ponieważ w XAML-u domyślnie bindowanie dywan się w jedną stronę,

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <declare-stylable name="MvxBinding">
    <attr name="MvxBind" format="string"/>
    <attr name="MvxLang" format="string"/>
  </declare-styleable>
```

```
  <declare-stylable name="MvxControl">
    <attr name="MvxTemplate" format="string"/>
  </declare-styleable>
  <declare-styleable name="MvxListView">
    <attr name="MvxItemTemplate" format= "string"/>
    <attr name="MvxDropDownItemTemplate" format="string"/>
  </declare-stylable>
  <item type="id" name="MvxBindingTagUnique">
  <declare-styleable name="MvxImageView">
    <attr name="MvxSource" format="string"/>
  </declare-stylable>
</resources>
```

Slajd 26

Zwartość tego pliku jest prosta ale bardzo istotna. File zawwieraatrybuty które można użyć w pliku widoku AXML . Widać tutaj jasno że w operacji data bidingu można użyć MvxBind albo MvxLang.

Możesz ponadto użyć nowych kontrolek MvxImageView, MvxListView , każda z nich jest dedykowana jakimś atrybutom . jak widać ponizej

```
<LinearLayout
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:layout_weight="2">
  <Mvx.MvxListView
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    local:MvxBind="ItemsSource Trips;ItemClick SelectTripCommand"
    local:MvxItemTemplate="@layout/tripitemtemplate" />
</LinearLayout>
```

Slajd 27

To powinno być znane osobom korzystającym wcześniej z XAML-a . właściwości The ItemsSource and ItemClick wiążą dane z widoku modelu . MvxItemTemplate - definiuje interfejs dla każdego przedmiotu w ListView.

Dla iOS sprawa wygląda dość podobnie. Nie jest to dokładnie to samo lecz, lecz można to ‘przepisać’ w logiczny sposób d o c#. Wywołanie użyte w poprzednim przykładzie do wybierania przedmiotu z listy jest ICommand.

```
 public ICommand<Trip> SelectTripCommand { get; set; }
```
implementacja tego interfejsu jest zapewniona przrez Mvvmcross i używa klase MvxCommand

```
private void InitializeCommands()
{
  this.SelectTripCommand = new MvxCommand<Trip>(
    trip => this.ShowViewModel<TripDetailsViewModel>(trip),
    trip => this.Trips != null && this.Trips.Any() && trip != null);
}
```

Slajd 28

Częstym problemem kiedy używa się wzorca MVVM jest konwersja typów. Dzieję się tak kiedy definiujesz właściwość typem który nie do końca jest ‘zrozumiały’ dla UI . W XAMLu możesz rozwiązać ten prooblemm za pomocą IValueConverter który mapuje Ci wartość między widokiem i widookiem modelu.
Proces jest nalogiczny w przypadku Mvvm Cross dzięki interfejsowi IMvxValueConverter i jego dwóm metodom Convert and ConvertBack. Ten interfejs pozwala konwertować dane podobnie jak w XAMLu .
Możesz również skorzystać z gotowej klasy

```
public class ByteArrayToImageConverter :
  MvxValueConverter<byte[], Bitmap>
{
  protected override Bitmap Convert(byte[] value, Type targetType,
    object parameter, CultureInfo culture)
  {
    if (value == null)
       return null;
    var options = new BitmapFactory.Options { InPurgeable = true };
    return BitmapFactory.DecodeByteArray(value,
     0, value.Length, options);
  }
}
```

Slajd 29

Która bierze parametr i zwraca Typ .

Teraz samo konwertowanie. Dla iOS używasz metody WithConversion

```
var set = this.CreateBindingSet<HomeView,
  Core.ViewModels.HomeViewModel>();
set.Bind(label).To(vm => vm.Trips).WithConversion("ByteArrayToImage");
set.Apply();
```

---

slajd 30

dla Androida robisz to od razu AXMLu

```
<ImageView
  local:MvxBind="Bitmap Image,Converter=ByteArrayToImage"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content" />
```

slajd 31

Mvvm zapewnia DLContainer . Możesz rejestrować tam klasy I interfejsy używając singleton registration i

dynamic registration. I wielu innych

```
Mvx.RegisterType<ISQLiteConnectionFactory, SQLiteConnectionFactory>();
Mvx.RegisterSingletong<ISQLiteConnectionFactory, SQLiteConnectionFactory>();
```