MVVM

1. VIew contains UI structure, and code behind is to be used as least as possible, it may contain code like for animation, i.e difficult in XAML.
2. View Model informs view about state changes , using Data Binding ,for commands and property.

- as we have extracted almost all UI logic in VM, we can unit test VM and M , without V.
- so UI can be changed without changing the VM

3. Model are Poco classes , independent from V and VM.

- so View Designer and VM Developer can work Parralel to eachother.

Data Binding in MVVM

1. Here VM is the Binding Source having one Public Property, and View is Binding Target with UI Property. between which data is passed.
2. Data Binding Mode, is the way to tell in which way data flows between two properties.

- 1. Default: i.e One Way , Sourse to Target
- 2. TwoWay : i.e S -> T and T -> S
- 3. OneWayToSource : T -> S
- 4. OneTIme -> S -> T, only when Binding Context Property changes.

1. Selective chaining

```
// if value is null return null, if not null return value
 PageTitle = value?.Name;
```

2. INotifyPropertyChanged

- used to Notify client that a property value has changed
- using a Bindable Object having OnPropertyChanged Method

- to update one property based on update in another property, we need INofityPropertyChanged Interface, as by default changes doesnt get reflected in UI if changes are made .

- its basically, like for a change to be updated in UI, we need to Instate the property once more, so we use OnPropertyChanged Method for that.

3. Binding ListView Context Action to ViewModel Command

- in Code Behind of page

```
              public ToDoPage()
             {
             BindingContext = new ToDoPageViewModel();
```

```
        InitializeComponent();
            }

```

- using xml only

```xml
<Content Page x:Name="pages">
<ContentPage.Content>
                            <ViewCell.ContextActions>
                               <MenuItem Text="Complete"
                                         Command="{Binding
Path=BindingContext.MarkAsCompletedCommand, Source={x:Reference pages}}"
                                         CommandParameter="{Binding .}"
                                         />
                           </ViewCell.ContextActions>
</ContentPage.Content>

```

4. Value Converter

- used when our Source Property and Target Property type are different
- so to make the conversion we need IValueConverter interface
- in Convert Method , value is Source property that we get from Model ,and we return property type for View.
- here we create a converter from bool to Color ,similarly bool to Text Decoration and other can be created.

```csharp
 public class BoolToColorConvertor : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
        {
            if((bool) value)
            {
                return Color.ForestGreen;
            }

            return Color.Default;
        }

        public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
        {
            return null;
        }
    }

```

```
  <ResourceDictionary>
            <convertor:BoolToColorConvertor x:Key="BoolToColor" />
        </ResourceDictionary>

 <Label Text="{Binding Name}"  TextColor="{Binding IsCompleted,Converter=
{StaticResource BoolToColor}}"/>
```

5. we can add header to List view to show items in List view its Count

- like to Calculate list item based on boolean criteria

```
 private void CalculateCompletedHeader()
        {
            CompletedHeader = $"Completed {ToDoItemCollection.Count(item =>
item.IsCompleted)} / {ToDoItemCollection.Count}";

            CompletedProgress = (double) ToDoItemCollection.Count(item =>
item.IsCompleted) / (double)ToDoItemCollection.Count;
        }
```

6. Add Animation to the View using EventHandler

- add animation to progress bar
- in viewModel

```
 public event EventHandler<double> UpdateProgressBar;
private void CalculateCompletedHeader()
        {

            // call the event on list item change
            UpdateProgressBar?.Invoke(this, CompletedProgress);
        }
```

- In Code Behind

```
// Subscribe to the event
 public ToDoPage()
        {
            ToDoPageViewModel toDoPageView = new ToDoPageViewModel();
            BindingContext = toDoPageView;

            InitializeComponent();
            toDoPageView.UpdateProgressBar += ToDoPageView_UpdateProgressBar;
        }
```

```
// Access Control specific Method to show animation
         private async void ToDoPageView_UpdateProgressBar(object sender, double e)
        {
            // listProgressBar is reference of Progress Bar xaml object
           await listProgressBar.ProgressTo(e, 300, Easing.Linear);
        }
```

ToDO

1. Xamarin.Forms Multi-Bindings

- with Multi-binding we can attach multiple properties(bindable objects) to a single target UI property.
- it evaluates all properties attached to it, and return single value through a IMultiValueConverter.
- it also reevaluates all of its Binding objects when any of the bound data changes.

- it has following properites:

1. FallbackValue
2. Mode
3. StringFormat
4. TargetNullValue

```
public class AllTrueMultiConverter : IMultiValueConverter
{
   public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)
   {

   }

   public object[] ConvertBack(object value, Type[] targetTypes, object parameter, CultureInfo culture)
   {

   }
}
```

- Consume a Multibinding

```
<ContentPage.Resources>
       <local:AllTrueMultiConverter x:Key="AllTrueConverter" />
       <local:InverterConverter x:Key="InverterConverter" />
   </ContentPage.Resources>

   <CheckBox>
       <CheckBox.IsChecked>
```

```
            <MultiBinding Converter="{StaticResource AllTrueConverter}">
                <Binding Path="Employee.IsOver16" />
                <Binding Path="Employee.HasPassedTest" />
                <Binding Path="Employee.IsSuspended"
                         Converter="{StaticResource InverterConverter}" />
            </MultiBinding>
        </CheckBox.IsChecked>
    </CheckBox>
```

2. Create Context Action in Android as Same as iOS we have two options

Option 1

- Setup

1. Add ContextViewCell Package to All project
2. Call ContextMenuViewRenderer.Preserve() in AppDelagate for iOS and MainActivity for Android

```

<context:SwipeActionContextHolder MovedCommand="{Binding 
Path=BindingContext.MarkAsCompletedCommand, Source={x:Reference pages}}">
    <context:SwipeActionContextMenuView >
        <context:SwipeActionContextMenuView.View>
            <Frame CornerRadius="10"
                                      HasShadow="True"
                                      IsClippedToBounds="True"
                                      Margin="3"
                                      BackgroundColor="{Binding 
IsCompleted,Converter={StaticResource BoolToColor}}"

WidthRequest="{Binding Source={x:Reference Todolist}, Path=Width}"
                                      Padding="0" >
                <StackLayout Padding="10,5,0,0"
                                                Margin="8,5,8,5"
                                                MinimumHeightRequest="100">
                    <Label Text="{Binding Name}"  TextColor="WhiteSmoke"
                                          TextDecorations="{Binding 
IsCompleted,Converter={StaticResource BoolToTextDecor }}"/>



                </StackLayout>
            </Frame>
        </context:SwipeActionContextMenuView.View>
        <context:SwipeActionContextMenuView.ContextTemplate>
            <DataTemplate>
                <Label Text="MOVE TO DELETE" />

            </DataTemplate>
        </context:SwipeActionContextMenuView.ContextTemplate>
    </context:SwipeActionContextMenuView>
    </context:SwipeActionContextHolder>
```

Option 2

- SwipeView
- this is a better approach , using this we can creating slide gesture menu for almost all controls , from label to list

```
<ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <SwipeView>
                                <SwipeView.RightItems>
                                    <SwipeItem  Text="Delete"
BackgroundColor="Red"
                                                Command="{Binding
Path=BindingContext.DeleteCommand, Source={x:Reference history}}"
                                                CommandParameter="{Binding}"
                                                />
                                </SwipeView.RightItems>
                                <Grid Padding="0,15,0,15">
                                    <Label Text="{Binding .}" />
                                </Grid>
                            </SwipeView>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
```

Unit Testing

1. The step involved in creating unit case are

- 1. Arrange the object
- 2. Act on Object
- 3. Assert the expected result

2. Naming Convention while writing test

> UnitOfWork_ExpectedBehaviour_ScenarioUnderTest

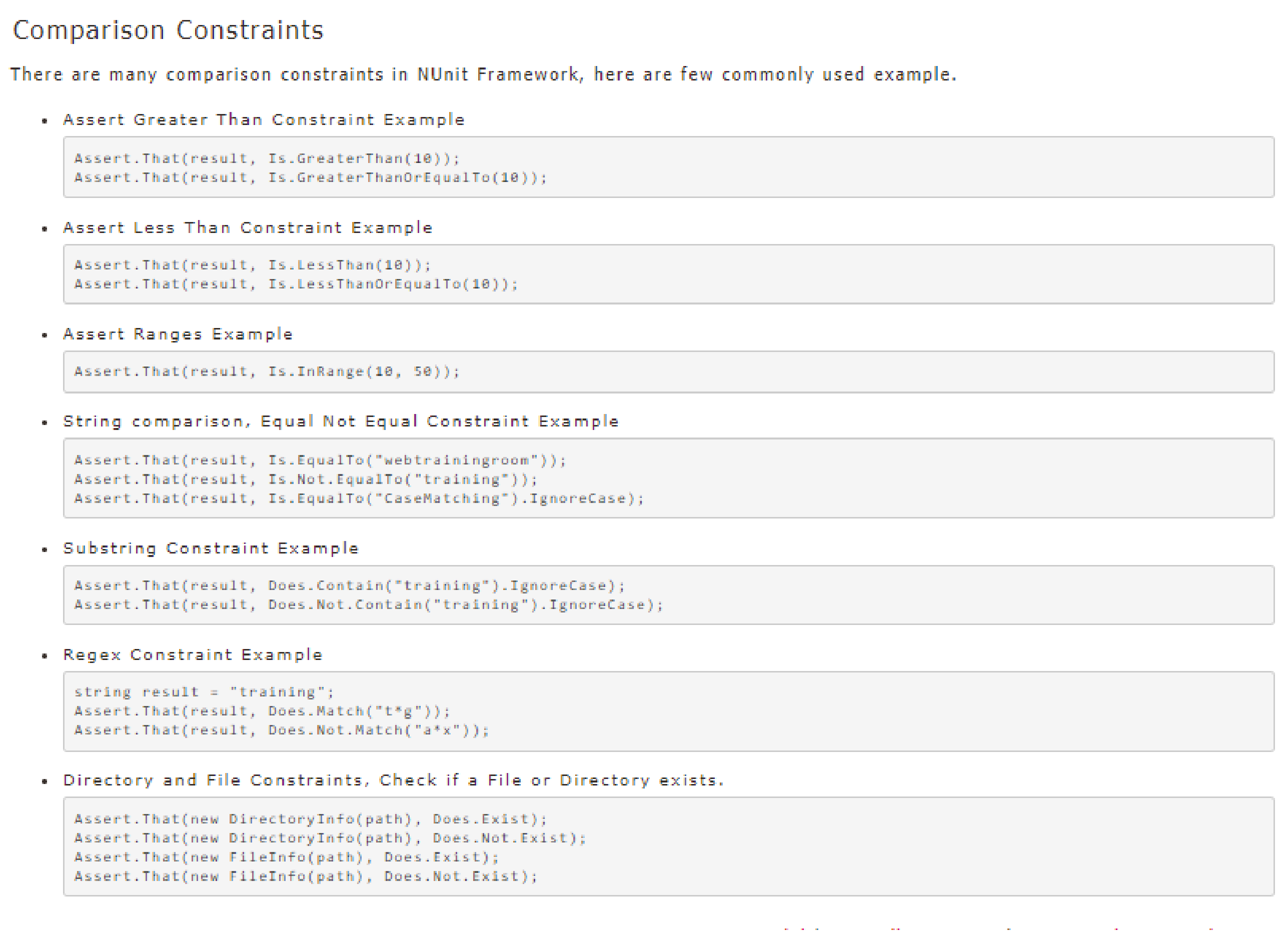
- UnitOfWork => Class or a method
- Expectedbehaviour => Outcome
- ScenarioUnderTest => When it Occurs

3. Attributes used in Test Project

- The [TestFixture] attribute denotes a class that contains unit tests.
- The [Test] attribute indicates a method is a test method.
- A [TestCase] attribute is used to create a suite of tests that execute the same code but have different input arguments

- [Ignore] attribute ignores a test case
- [Category] Categories the classes or test cases
- [Repeat] method executed multiple times while runing

4. Comparison Constraints



Comparison Constraints

There are many comparison constraints in NUnit Framework, here are few commonly used example.

- Assert Greater Than Constraint Example

```
Assert.That(result, Is.GreaterThan(10));
Assert.That(result, Is.GreaterThanOrEqualTo(10));
```

- Assert Less Than Constraint Example

```
Assert.That(result, Is.LessThan(10));
Assert.That(result, Is.LessThanOrEqualTo(10));
```

- Assert Ranges Example

```
Assert.That(result, Is.InRange(10, 50));
```

- String comparison, Equal Not Equal Constraint Example

```
Assert.That(result, Is.EqualTo("webtrainingroom"));
Assert.That(result, Is.Not.EqualTo("training"));
Assert.That(result, Is.EqualTo("CaseMatching").IgnoreCase);
```

- Substring Constraint Example

```
Assert.That(result, Does.Contain("training").IgnoreCase);
Assert.That(result, Does.Not.Contain("training").IgnoreCase);
```

- Regex Constraint Example

```
string result = "training";
Assert.That(result, Does.Match("t*g"));
Assert.That(result, Does.Not.Match("a*x"));
```

- Directory and File Constraints, Check if a File or Directory exists.

```
Assert.That(new DirectoryInfo(path), Does.Exist);
Assert.That(new DirectoryInfo(path), Does.Not.Exist);
Assert.That(new FileInfo(path), Does.Exist);
Assert.That(new FileInfo(path), Does.Not.Exist);
```

5. Demo

```
[Test]
      public void ViewModel_populates_ToDoItemCollection_correctly()
      {
          //Arrange

          var ViewModel = new ToDoPageViewModel();
          // Action

          Assert.AreEqual(5, ViewModel.ToDoItemCollection.Count);

          //Assert

      }
```

6. mocking library

- moq are used to automatically fake some interface

- it save time as we dont write mock implementation for interface
- for using this, install Moq Package in Test project
- demo

```
  [Test]
        public void
DispalyAlertCommand_displays_alert_on_button_pressed_using_moq()
        {
            // using moq
            var mockDialogMessage = new Mock<IDialogMessage>();

            mockDialogMessage.Setup(x => x.DisplayAlert("Hello", "Have a Great
Day", "OK")).Returns(Task.CompletedTask);

            var viewModel = new DIDemoPageViewModel(mockDialogMessage.Object);

            viewModel.DisplayAlertCommand.Execute(null);

            mockDialogMessage.Verify(x => x.DisplayAlert("Hello", "Have a Great
Day", "OK"), Times.Once);



        }
```

7. We use Interface to create services in Xamarin Forms, so that while

- Unit Testing We can creat Mock services, to fake our implementation
- rather than creating a new implementation, so its saves time.

8. AutoFac

- it s a a Inversion of Control container
- it automatically resolves dependencies
- set up for it is done in App class

```
           // using AutoFac

            //class used for registration
            var builder = new ContainerBuilder();
            //scan and register all classes in the assembly
            var dataAccess = Assembly.GetExecutingAssembly();
            builder.RegisterAssemblyTypes(dataAccess)
                   .AsImplementedInterfaces()
                   .AsSelf();

            NavigationPage navigationPage = null;
            Func<INavigation> navigationFunc = () => {
                return navigationPage.Navigation;
            };

            builder.RegisterType<NavigationService>().As<INavigationService>()
                .WithParameter("navigation", navigationFunc);

            //get container
            var container = builder.Build();


            navigationPage = new NavigationPage(container.Resolve<MainPage>());
            MainPage = navigationPage;
```

Dependency Injection

1. to decouple Ui from business logic
2. Benifit :

- Decoupling UI from ViewModel
- Testability Class with DI is independent and it is easier to test using mock up

3. 3 types of DI

- Constructor injection
- Property injection
- Method injection

4. here we created dependency service for DisplayAlert using Interface and Serviice class
5. to resolve dependency we use IOC container AutoFac, and its Setup Done in App class
6. he has given a new way to structure app, as Module containing

- Module Folder having View and VM,
- Common Folder containing Services
- Application Folder containing App related classes.

Creating Custom Navigation required for MVVM framework like Prism

1. here we created a navigation Service , from INavigation Interace Contianing PushAsync and PopAsync() methods

2. In Navigation service class, we are using DI to get Navigation related dependency ,and creating a base class for VM to handle initializing parameters passed during navigation .

3. All VM must inherit from this common BaseViewModel to be part of Navigation and pass Parameter at same time, same as in Prism

4. Now for Passing data back to the Source VM

- we can use MessagingCenter

- in current class, we can Send the event with parameter we want to notify about

```
MessagingCenter.Send(this, "Items", new List<string>(Items));
```

- in Source class we need to Subscribe to the event

```
 MessagingCenter.Subscribe<HistoryPageViewModel, List<string>>(this, "Items", (vm, 
list) =>
               {
                   _calculatorHistory = list;

               });
```

5. Passing parameter to a child page of Tab page

- done by creating a variable of VM in tab VM class, as
- we navigate to tab page , we are getting parameters , we can now assign them to child VM
- and pass this child VM instance as BindingContext, for child page, while Registering Child page to Tab.

```
// in InfoPageVM
 public InfoPageViewModel(HistoryPageViewModel historyVM)
        {
            HistoryVM = historyVM;
        }

        public HistoryPageViewModel HistoryVM { get; set; }

        public override Task InitializeAsync(object parameter)
        {
            return HistoryVM.InitializeAsync(parameter);
        }


// in tab page i,e InfoPage Code Behind file
 public InfoPage(InfoPageViewModel viewModel,HistoryPage 
historyPage,AppInformationPage appInformationPage)
        {
            InitializeComponent();


            BindingContext = viewModel;

            historyPage.BindingContext = viewModel.HistoryVM;
            historyPage.IconImageSource = "history.png";

            appInformationPage.IconImageSource = "info.png";
            Children.Add(historyPage);
            Children.Add(appInformationPage);
        }
```

NetWork Service [REST APi]

Movie API

- here we are using OMDB Api to get Movies related info
- and for large number of images that will come we are usign

- FFIMageLoading Package , to get a custom Image class,
- FFImageLoading for Forms provides CachedImage - a direct Image class replacement.
- It's used just the same but it has some additional properties
- for this we needed to write platform specific boilerplate code

```
<ffimageloading:CachedImage LoadingPlaceholder="placeholder.png"
Aspect="AspectFill"
 HeightRequest="300"
 Source="{Binding ImageUrl}"/>
```

- here we populated one Collection as ItemSource for our Collection view using movie api

- and set one EmptyView , when no data is there in Itemsource.

  ## Custom Transition on Navigation

  1. here we are using SharedTransition Nuget Package in all our project
  2. using which , we get custom transition, like image to image,
  3. i.e while navigating to and from a image in animated for transition

4. done by adding code in

- page performing navigation

```
<ContentPage
xmlns:sharedTransitions="clr-
namespace:Plugin.SharedTransitions;assembly=Plugin.SharedTransitions"

sharedTransitions:SharedTransitionNavigationPage.TransitionSelectedGroup="{Binding
SelectedMovieId}" >

<DataTemplate>

 <ffimageloading:CachedImage LoadingPlaceholder="placeholder.png"
Source="{Binding ImageUrl}"

<!-- here we set image for transition, also SelectedMovieId should be equal to
imdbID, for transition-->
```

```
 sharedTransitions:Transition.Name="MovieImage"
  sharedTransitions:Transition.Group="{Binding imdbID}"
                                                 />
                                                 </ContentPage>
```

- page navigated to

```


<!-- set the same Transition name in navigated page image, for animation start and
stop on image-->
<ffimageloading:CachedImage Source="{Binding MoviesData.ImageUrl}"
Aspect="AspectFill" HeightRequest="500"

sharedTransitions:Transition.Name="MovieImage"/>
```

Database Connection

1. connect to database to safe movieInformation we marked as favorite
2. use Generic Repository Pattern for Saving to database
3. Repository Pattern is Simply one class that Separates Business logic from database

- It contains all database related operation like add,remove,update,select .

```

public  interface IDatabaseItem
    {

        int Id { get; set; }

    }

  public interface IRepository<T> where T: IDatabaseItem,new()
    {

        Task<T> GetById(int id);
        Task<int> DeleteAsync(T item);

        Task<List<T>> GetAllAsync();

        Task<int> SaveAsync(T item);


    }
// Generic Repository
    public class Repository<T> : IRepository<T> where T : IDatabaseItem, new()
    {

        readonly Lazy<SQLiteAsyncConnection> lazyInitializer = new
Lazy<SQLiteAsyncConnection>( () => {
```

```
            return new SQLiteAsyncConnection(DatabaseConstants.DatabasePath,
DatabaseConstants.Flags);
        });


        private SQLiteAsyncConnection Database => lazyInitializer.Value;

        public Repository()
        {

          InitializeAsync();
        }

         async Task InitializeAsync()
        {
            if (!Database.TableMappings.Any(m => m.MappedType.Name ==
typeof(T).Name))
            {
                await Database.CreateTableAsync(typeof(T)).ConfigureAwait(false);
            }
        }

    }

```

4 .Creating database using .Net standard library

```


public static class DatabaseConstants
    {
        public const string DatabaseFileName = "MoviesSQLite.db3";

        public const SQLite.SQLiteOpenFlags Flags =
            SQLite.SQLiteOpenFlags.ReadWrite |
            SQLite.SQLiteOpenFlags.Create |
            SQLite.SQLiteOpenFlags.SharedCache;


        public static string DatabasePath
        {
            get
            {
                var basePath =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);

                return Path.Combine(basePath, DatabaseFileName);
            }
        }

    }
}
```