Apply the MVVM Design Pattern in .NET MAUI Labs

Perform these labs on your own computer using Visual Studio 2022 or later to ensure you understand the lessons presented in the corresponding videos and lectures.

Lab 1: Add Properties and Methods to Common Base Class

Let's now create a view model class to retrieve data and feed it to the view, but first, let's create a base class for all view models.

Open the **BaseClasses\CommonBase.cs** file and add a few more properties to help display information and error messages.

```
#region Private/Protected Variables
private string _InfoMessage = string.Empty;
private string _LastErrorMessage = string.Empty;
private Exception? _LastException = null;

protected const string REPO_NOT_SET = "The Repository
Object is not Set.";
#endregion

#region Public Properties
/// <summary>
/// Get/Set the last informational message to display
/// </summary>
[NotMapped]
[JsonIgnore]
public string InfoMessage
{
  get { return _InfoMessage; }
  set
  {
    _InfoMessage = value;
    RaisePropertyChanged(nameof(InfoMessage));
  }
}

/// <summary>
/// Get/Set the last error message from the last
operation
/// </summary>
[NotMapped]
[JsonIgnore]
public string LastErrorMessage
{
  get { return _LastErrorMessage; }
  set
  {
    _LastErrorMessage = value;
    RaisePropertyChanged(nameof(LastErrorMessage));
  }
}

/// <summary>
/// Get/Set the last exception object from the last
operation
/// Sets the ErrorMessage to LastException.Message if
ErrorMessage is blank
/// </summary>
```

```
[NotMapped]
[JsonIgnore]
public Exception? LastException
{
  get { return _LastException; }
  set
  {
    _LastException = value;
    if (_LastException != null) {
      if (string.IsNullOrEmpty(LastErrorMessage)) {
        LastErrorMessage = _LastException.Message;
      }
    }
    RaisePropertyChanged(nameof(LastException));
  }
}
#endregion
```

Lab 2: Create a View Model Base Class

Right mouse-click on the **Common.Library\BaseClasses** folder and add a new class named **ViewModelBase** and replace the entire contents of the new file with the following code.

```
namespace Common.Library;

public class ViewModelBase : CommonBase
{
  #region Private Variables
  private bool _IsAdding;
  private int _RowsAffected;
  #endregion

  #region Public Properties
  /// <summary>
  /// Get/Set whether or not the page is in add mode
  /// </summary>
  public bool IsAdding
  {
    get { return _IsAdding; }
    set
    {
      _IsAdding = value;
      RaisePropertyChanged(nameof(IsAdding));
    }
  }

  /// <summary>
  /// Get/Set the Numbers of Rows Affected by the last
operation
  /// </summary>
  public int RowsAffected
  {
    get { return _RowsAffected; }
    set
    {
      _RowsAffected = value;
      RaisePropertyChanged(nameof(RowsAffected));
    }
  }
  #endregion

  #region PublishException Method
  protected virtual void PublishException(Exception ex)
  {
    LastException = ex;

    System.Diagnostics.Debug.WriteLine(ex.ToString());
  }
  #endregion
```

```
}
```

Lab 3: Create View Model Layer Class Library

Right mouse-click on the **Solution** and add a new Class Library project named **AdventureWorks.ViewModelLayer**.

Delete the **Class1.cs** file.

Add Dependencies to View Model Layer

Right mouse-click on the **Dependencies** folder in this new **AdventureWorks.ViewModelLayer** project and add a project reference to the **Common.Library** project and the **AdventureWorks.EntityLayer** project.

Add a User View Model Class

Right mouse-click on the **AdventureWorks.ViewModelLayer** project and add a new folder named **ViewModelClasses**.

Right mouse-click on the **ViewModelClasses** folder and add a new class named **UserViewModel**. Replace the entire contents of this new file with the following code.

```
using AdventureWorks.EntityLayer;
using Common.Library;
using System.Collections.ObjectModel;

namespace AdventureWorks.ViewModelLayer;

public class UserViewModel : ViewModelBase
{
  #region Private Variables
  private User? _CurrentEntity = new();
  #endregion

  #region Public Properties
  public User? CurrentEntity
  {
    get { return _CurrentEntity; }
    set
    {
      _CurrentEntity = value;
      RaisePropertyChanged(nameof(CurrentEntity));
    }
  }
  #endregion

  #region GetAsync Method
  public async Task<ObservableCollection<User>>
GetAsync()
  {
    return await Task.FromResult(new
ObservableCollection<User>());
  }
  #endregion

  #region GetAsync(id) Method
  /// <summary>
  /// Get a single user object
  /// </summary>
  /// <param name="id">The UserId to locate</param>
  /// <returns>An instance of a User object</returns>
  public async Task<User?> GetAsync(int id)
  {
    try {
      // TODO: Get a User from a data store

      // MOCK Data
      CurrentEntity = await Task.FromResult(new User {
        UserId = id,
```

```
        LoginId = "SallyJones615",
        FirstName = "Sally",
        LastName = "Jones",
        Email = "Sallyj@jones.com",
        Phone = "615.987.3456",
        PhoneType = "Mobile",
        IsFullTime = true,
        IsEnrolledIn401k = true,
        IsEnrolledInFlexTime = false,
        IsEnrolledInHealthCare = true,
        IsEnrolledInHSA = false,
        IsActive = true,
        BirthDate = Convert.ToDateTime("08-13-1989"),
        StartTime = new TimeSpan(7, 30, 0)
      });

      RowsAffected = 1;
    }
    catch (Exception ex) {
      RowsAffected = 0;
      PublishException(ex);
    }

    return CurrentEntity;
  }
  #endregion

  #region SaveAsync Method
  public async virtual Task<User?> SaveAsync()
  {
    // TODO: Write code to save data
    System.Diagnostics.Debugger.Break();

    return await Task.FromResult(new User());
  }
  #endregion
}
```

Lab 4: Use the User View Model on XAML

Right mouse-click on the **Dependencies** folder in this new **AdventureWorks.MAUI** project and add a project reference to the **AdventureWorks.ViewModelLayer** project.

Open the **Views\UserDetailView.xaml** file and **change** the XML namespace "vm" to point to the ViewModelLayer assembly.

```
xmlns:vm="clr-
namespace:AdventureWorks.ViewModelLayer;assembly=Adventu
reWorks.ViewModelLayer"
```

Change the **x:DataType** attribute to use the **UserViewModel**.

```
x:DataType="vm:UserViewModel"
```

Remove the **<vm:User x:Key="viewModel" …>** from the <ContentPage.Resources> element.

Remove the BindingContext from the <Border> element.

```
<Border Style="{StaticResource Screen.Border}"
        BindingContext="{StaticResource viewModel}">
```

Change all Bindings

Change all {Binding PROPERTY_NAME} references to use {Binding **CurrentEntity.**PROPERTY_NAME}.

Update Source Event Name

Locate the <RadioButton> for the IsFullTime property and add the **UpdateSourceEventName** within the {Binding …}.

```
<RadioButton IsChecked="{Binding
CurrentEntity.IsFullTime,
UpdateSourceEventName=CheckChanged}"
                GroupName="EmployeeType" />
```

Modify the Code Behind

Open the **Views\UserDetailView.xaml.cs** file and **change** using statement that points to the EntityLayer to point to the **ViewModelLayer**.

```
using AdventureWorks.ViewModelLayer;
```

REMOVE the line of code from the **Constructor** that was used to set the ViewModel.

```
public UserDetailView()
{
 InitializeComponent();

  UserObject =
(UserViewModel)this.Resources["viewModel"];
}
```

Change the ViewModel property to the type of **UserViewModel** and initialize the property to a new instance.

```
public UserViewModel ViewModel { get; set; } = new();
```

Modify the OnAppearing() event procedure to look like the following.

```
protected async override void OnAppearing()
{
  base.OnAppearing();

  // Create a new instance of UserViewModel
  ViewModel = new();

  // Set the Page BindingContext
  BindingContext = ViewModel;

  // Retrieve a User
  await ViewModel.GetAsync(1);
}
```

Try It Out

Run the application and click on **Users | Navigate to Detail** to see the data coming from the View Model class.