Commanding Labs - MAUI

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 a View Model for Commands

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

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

```
using AdventureWorks.EntityLayer;
using AdventureWorks.ViewModelLayer;
using Common.Library;
using System.Windows.Input;

namespace AdventureWorks.MAUI.CommandClasses;

public class UserViewModelCommands : UserViewModel
{
  #region Constructors
  public UserViewModelCommands() : base()
  {
  }

  public UserViewModelCommands(IRepository<User> repo) :
base(repo)
  {
  }

  public UserViewModelCommands(IRepository<User> repo,
IRepository<PhoneType> phoneRepo) : base(repo,
phoneRepo)
  {
  }
  #endregion

  #region Commands
  public ICommand? SaveCommand { get; private set; }
  #endregion

  #region Init Method
  public override void Init()
  {
    base.Init();

    // Create commands for this view
    SaveCommand = new Command(
      execute: async () => await SaveAsync(),
      canExecute: () => true);
  }
  #endregion
}
```

Open the **Views\UserDetailView.xaml** file and **CHANGE** the "**xmlns:vm**" to reference the new namespace.

```
xmlns:vm="clr-
namespace:AdventureWorks.MAUI.CommandClasses"
```

Change the x:DataType to the UserViewModelCommands class.

```
x:DataType="vm:UserViewModelCommands"
```

Modify the **Save** button and remove the **Clicked** event call. Add a **Command** attribute instead.

```
<Button Text="Save"
        Command="{Binding SaveCommand}" />
```

Change the View Model Reference

Open the **MauiProgram.cs** file and change the *using AdventureWorks.ViewModelLayer;* to the following.

```
using AdventureWorks.MAUI.CommandClasses;
```

Change the dependency injection that uses the UserViewModel to use the commanding view model.

```
builder.Services.AddScoped<UserViewModelCommands>();
```

Modify the UserDetailView Code

Open the **Views\UserDetailView.xaml.cs** file and change the *using AdventureWorks.ViewModelLayer;* to the following.

```
using AdventureWorks.MAUI.CommandClasses;
```

Modify the constructor to look like the following.

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

  ViewModel = viewModel;
}
```

Modify the readonly **ViewModel** variable to use the UserViewModelCommands class.

```
private readonly UserViewModelCommands ViewModel;
```

Remove the **SaveButton_Clicked**() event procedure.

```
private void SaveButton_Clicked(object sender, EventArgs
e)
{
  System.Diagnostics.Debugger.Break();
}
```

Try It Out

Run the application, click on the **Users | Navigate to Detail** menu, click on the **Save** button and you should end up on the break in the SaveAsync() method in the UserViewModel class.

Lab 2: Control Button IsEnabled Property (Optional)

Open the **CommandClasses\UserViewModelCommands** file and add a private variable and a public property.

```
#region Private Variables
private bool _IsSaveCommandEnabled = false;
#endregion

#region Public Properties
public bool IsSaveCommandEnabled
{
  get { return _IsSaveCommandEnabled; }
  set
  {
    _IsSaveCommandEnabled = value;
    RaisePropertyChanged(nameof(IsSaveCommandEnabled));
  }
}
#endregion
```

Modify the Init() method and change the hard-coded **true** value in the **canExecute** method to use the new property you just created.

```
SaveCommand = new Command(
      execute: async () => await SaveAsync(),
      canExecute: () => IsSaveCommandEnabled);
```

Try It Out

Run the application, click on **Users | Navigate to Detail** and notice that the **Save** button is disabled. This is because you initialized the IsSaveCommandEnabled property to a **false** value.

Open the **UserViewModelCommands.cs** file set the private variable **_IsSaveCommandEnabled** equal to **true**.

Lab 3: Change CanExecute (Optional)

The second parameter to the Command() constructor is not called everytime the IsSaveCommandEnabled is changed. Instead you must invoke the ChangeCanExecute() method on the Command object.

```
(SaveCommand as Command)?.ChangeCanExecute();
```

Because you are interacting with a Command object, you should always call a method in the ViewModelCommand class. This method calls a method in the

ViewModel class, then at the end reevaluate whether any of the Command objects' **CanExecute** methods need to be modified.

Add the following method to the **UserViewModelCommands** class.

```
#region SaveAsync Method
protected new async Task<User?> SaveAsync()
{
  Task<User?> ret = await base.SaveAsync();

  IsSaveCommandEnabled = false;

  // Have the SaveCommand reevaluate the CanExecute
method
  (SaveCommand as Command)?.ChangeCanExecute();

  return ret;
}
#endregion
```

Change the instantiation of the SaveCommand in the Init() method to the following.

```
SaveCommand = new Command(async () => await SaveAsync(),
() => IsSaveCommandEnabled);
```

Try It Out

Run the application and click on the **Users** menu.

Click on the **Save** button and you should see the Save button become disabled.

Lab 4: Simplify the User View Model Commands Class (Optional)

I don't typically use the **canExecute** method of the Command class, instead I control the enabling and disabling of buttons through the ViewModel classes as this is more transferable to other technologies. Thus, I suggest you just put back the UserViewModelCommands class to the following code.

```
using AdventureWorks.EntityLayer;
using AdventureWorks.ViewModelLayer;
using Common.Library;
using System.Windows.Input;

namespace AdventureWorks.MAUI.CommandClasses;

public class UserViewModelCommands : UserViewModel
{
  #region Constructors
  public UserViewModelCommands() : base()
  {
  }

  public UserViewModelCommands(IRepository<User> repo) :
base(repo)
  {
  }

  public UserViewModelCommands(IRepository<User> repo,
IRepository<PhoneType> phoneRepo) : base(repo,
phoneRepo)
  {
  }
  #endregion

  #region Commands
  public ICommand? SaveCommand { get; private set; }
  #endregion

  #region Init Method
  public override void Init()
  {
    base.Init();

    // Create commands for this view
    SaveCommand = new Command(
      execute: async () => await SaveAsync(),
      canExecute: () => true);
  }
  #endregion
}
```

Try It Out

Run the application, click on the **Users | Navigate to Detail** menu, click on the **Save** button and you should end up on the break in the SaveAsync() method in the UserViewModel class.