Navigating From Lists to Detail Views 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: Display User Detail View from List View

Open the **Views\UserDetailView.xaml.cs** file.

Add two **[QueryProperty]** attributes above the public partial class UserDetailView definition as shown in the bold code below.

```
[QueryProperty(nameof(UserId), "id")]
[QueryProperty(nameof(IsAdding), "isAdding")]
public partial class UserDetailView : ContentPage
{
    // REST OF THE CODE HERE
}
```

Add two public properties named **UserId** and **IsAdding**.

```
public int UserId { get; set; }
public bool IsAdding { get; set; }
```

Modify the OnAppearing() event procedure to use the two new properties.

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

  // Set the Page BindingContext
  BindingContext = ViewModel;

  // Get the Phone Types
  ViewModel.GetPhoneTypes();

  ViewModel.IsAdding = IsAdding;
  if (IsAdding) {
    // TODO: Create an empty entity
  }
  else {
    // Retrieve a User
    await ViewModel.GetAsync(UserId);
  }
}
```

Add Command for Editing

Open the **CommandClasses\UserViewModelCommands.cs** file and add a new method to navigate to the User Detail View.

```
#region EditAsync Method
protected async Task EditAsync(int id)
{
  await
Shell.Current.GoToAsync($"{nameof(Views.UserDetailView)}
?id={id}&isAdding={false}");
}
#endregion
```

Add a new ICommand.

```
public ICommand? EditCommand { get; private set; }
```

Create the new command in the Init() method.

```
EditCommand = new Command<int>(async (int id) => await
EditAsync(id), (id) => true);
```

Open the **Views\UserListView.xaml** file and add an **x:Name** attribute to the ContentPage.

```
x:Name="UserListPage"
```

Locate the <Button Text="Edit" /> and make it look like the following.

```
<Button Text="Edit"
        CommandParameter="{Binding UserId}"
        Command="{Binding Source={x:Reference
UserListPage}, Path=BindingContext.EditCommand}" />
```

Try It Out

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

Click on the **Edit** button on different users to see the User Detail view appear for each user.

NOTE: The cancel button will not work on the detail view yet.

Lab 2: Save User & Return to User List View

Open the **CommandClasses\UserViewModelCommanding.cs** file and add a **SaveAsync()** method. After calling the SaveAsync() method in the base class, navigate to the user list view.

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

  if (ret != null) {
    await Shell.Current.GoToAsync("..");
  }

  return ret;
}
#endregion
```

Open the **ViewModelClasses\UserViewModel.cs** file and **remove** the **System.Diagnostics.Debugger.Break();** line of code from the SaveAsync() method.

Try It Out

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

Click on the **Edit** button for a user to see the User Detail view appear.

Click on the **Save** button and see it **navigate** back to the User List view.

Lab 3: Cancel Changes & Return to User List View

Open the **CommandClasses\UserViewModelCommanding.cs** file and add a **CancelAsync()** method. This method simply navigates back to the user list view.

```
#region CancelAsync Method
protected async Task CancelAsync()
{
  await Shell.Current.GoToAsync("..");
}
#endregion
```

Add a new ICommand to invoke the CancelAsync() method.

```
public ICommand? CancelCommand { get; private set; }
```

Modify the Init() method and create an instance of new CancelCommand.

```
CancelCommand = new Command(async () => await
CancelAsync(), () => true);
```

Open the **Views\UserDetailView.xaml** file, locate the Cancel button and add a Command attribute.

```
<Button Text="Cancel"
        Command="{Binding CancelCommand}" />
```

Try It Out

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

Click on the **Edit** button for a user to see the User Detail view appear.

Click on the **Cancel** button and see it **navigate** back to the User List view.

Lab 4: Display Product Detail from Collection View

Replace the Product Detail View XAML

Open the **Views\ProductDetailView.xaml** file and add an XML namespace.

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

Add a DataType

```
x:DataType="vm:ProductViewModelCommands"
```

Replace the XAML between the <partial:HeaderView …> and the </Grid> element with the following.

```
<Label Text="Product Name"
       Grid.Row="1" />
<Entry Grid.Column="1"
       Grid.Row="1"
       Text="{Binding CurrentEntity.Name}" />
<Label Text="Product Number"
       Grid.Row="2" />
<Entry Grid.Row="2"
       Grid.Column="1"
       Text="{Binding CurrentEntity.ProductNumber}" />
<Label Text="Color"
       Grid.Row="3" />
<Entry Grid.Row="3"
       Grid.Column="1"
       Text="{Binding CurrentEntity.Color}" />
<Label Text="Cost"
       Grid.Row="4" />
<HorizontalStackLayout Grid.Row="4"
                       Grid.Column="1">
  <Entry Text="{Binding CurrentEntity.StandardCost}" />
  <Stepper Value="{Binding CurrentEntity.StandardCost}"
           Minimum="1"
           Maximum="{Binding CurrentEntity.ListPrice}"
           Increment="1" />
</HorizontalStackLayout>
<Label Text="Price"
       Grid.Row="5" />
<HorizontalStackLayout Grid.Row="5"
                       Grid.Column="1">
  <Entry Text="{Binding CurrentEntity.ListPrice}" />
  <Stepper Value="{Binding CurrentEntity.ListPrice}"
           Minimum="{Binding
CurrentEntity.StandardCost}"
           Maximum="9999"
           Increment="1" />
</HorizontalStackLayout>
<Label Text="Size"
       Grid.Row="6" />
<Entry Grid.Row="6"
       Grid.Column="1"
       Text="{Binding CurrentEntity.Size}" />
<Label Text="Weight"
       Grid.Row="7" />
<Entry Grid.Row="7"
       Grid.Column="1"
       Text="{Binding CurrentEntity.Weight}" />
<Label Text="Category"
```

```
       Grid.Row="8" />
<Entry Grid.Row="8"
       Grid.Column="1"
       Text="{Binding CurrentEntity.ProductCategoryID}"
/>
<Label Text="Model"
       Grid.Row="9" />
<Entry Grid.Row="9"
       Grid.Column="1"
       Text="{Binding CurrentEntity.ProductModelID}" />
<Label Text="Selling Start Date"
       Grid.Row="10" />
<DatePicker Grid.Row="10"
            Grid.Column="1"
            Date="{Binding CurrentEntity.SellStartDate}"
/>
<Label Text="Selling End Date"
       Grid.Row="11" />
<DatePicker Grid.Row="11"
            Grid.Column="1"
            Date="{Binding CurrentEntity.SellEndDate}"
/>
<Label Text="Discontinued Date"
       Grid.Row="12" />
<DatePicker Grid.Row="12"
            Grid.Column="1"
            Date="{Binding
CurrentEntity.DiscontinuedDate}" />
<HorizontalStackLayout Grid.Row="13"
                       Grid.Column="1">
  <Button Text="Save"
          Command="{Binding SaveCommand}" />
  <Button Text="Cancel"
          Command="{Binding CancelCommand}" />
</HorizontalStackLayout>
```

Lab 5: Replace the Product Detail View Code Behind

Open the **Views\ProductDetailView.xaml.cs** file and replace the entire contents of this file with the following code.

```
using AdventureWorks.MAUI.CommandClasses;

namespace AdventureWorks.MAUI.Views;

[QueryProperty(nameof(ProductId), "id")]
[QueryProperty(nameof(IsAdding), "isAdding")]
public partial class ProductDetailView : ContentPage
{
  public ProductDetailView(ProductViewModelCommands
viewModel)
  {
    InitializeComponent();

    ViewModel = viewModel;
  }

  public ProductViewModelCommands ViewModel { get; set;
}
  public int ProductId { get; set; }
  public bool IsAdding { get; set; }

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

    // Set the Page BindingContext
    BindingContext = ViewModel;

    ViewModel.IsAdding = IsAdding;
    if (IsAdding) {
      // TODO: Create a new empty entity
    }
    else {
      // Retrieve a Product
      await ViewModel.GetAsync(ProductId);
    }
  }
}
```

Lab 6: Modify Product View Model Commands

Open the **CommandClasses\ProductViewModelCommands.cs** file and add the following code after the constructors.

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

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

  // Create commands for this view
  SaveCommand = new Command(async () => await
SaveAsync(), () => true);
  EditCommand = new Command<int>(async (int id) => await
EditAsync(id), (id) => true);
  CancelCommand = new Command(async () => await
CancelAsync(), () => true);
}
#endregion

#region EditAsync Method
protected async Task EditAsync(int id)
{
  await
Shell.Current.GoToAsync($"{nameof(Views.ProductDetailVie
w)}?id={id}&isAdding={false}");
}
#endregion

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

  if (ret != null) {
    await Shell.Current.GoToAsync("..");
  }

  return ret;
}
#endregion

#region CancelAsync Method
protected async Task CancelAsync()
{
  await Shell.Current.GoToAsync("..");
```

```
}
#endregion
```

Lab 7: Modify Product List View XAML

Open the **Views\ProductViewList.xaml** file and add an **x:Name** attribute to the ContentPage.

```
x:Name="ProductListPage"
```

Locate the <Button Text="Edit" /> and make it look like the following.

```
<Button Text="Edit"
        CommandParameter="{Binding ProductID}"
        Command="{Binding Source={x:Reference
ProductListPage}, Path=BindingContext.EditCommand}" />
```

Open the **AppShell.xaml.cs** file and register the route for the ProductDetailView in the constructor.

```
Routing.RegisterRoute(nameof(Views.ProductDetailView),
typeof(Views.ProductDetailView));
```

Try It Out

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

Click on the **Edit** button for different Products to see the Product Detail view appear.

Click on the **Save** or **Cancel** buttons and see it **navigate** back to the Product List view.