Display Lists of Data with Collection 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: Add a Collection of Users to View Model

Open the **ViewModelClasses\UserViewModel.cs** file and add a new private property.

```
private ObservableCollection<User> _Users = new();
```

Add a new public property named **Users**.

```
public ObservableCollection<User> Users
{
  get { return _Users; }
  set
  {
    _Users = value;
    RaisePropertyChanged(nameof(Users));
  }
}
```

Modify the GetAsync() method to populate the **Users** property.

```
#region GetAsync Method
public async Task<ObservableCollection<User>> GetAsync()
{
  RowsAffected = 0;

  try {
    if (Repository == null) {
      LastErrorMessage = REPO_NOT_SET;
    }
    else {
      Users = new ObservableCollection<User>(await
Repository.GetAsync());
      RowsAffected = Users.Count;
      InfoMessage = $"Found {RowsAffected} Users";
    }
  }
  catch (Exception ex) {
    PublishException(ex);
  }

  return Users;
}
#endregion
```

Lab 2: Use a List View to Display a List of Users

Use the ListView control to display a larger list of data. Open the **Views\UserListView.xaml** file and add an XML namespace to the partial views' namespace, the view model namespace, and the entity layer namespace.

```
xmlns:partial="clr-
namespace:AdventureWorks.MAUI.ViewsPartial"
xmlns:vm="clr-
namespace:AdventureWorks.MAUI.CommandClasses"
xmlns:model="clr-
namespace:AdventureWorks.EntityLayer;assembly=AdventureW
orks.EntityLayer"
```

Add an **x:DataType** attribute to the <ContentPage…> element.

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

Replace the <VerticalStackLayout> element with the following code.

```
<Border Style="{StaticResource Border.Page}">
  <Grid Style="{StaticResource Grid.Page}">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <partial:HeaderView ViewTitle="User List"
                        ViewDescription="The list of
users in the system." />

    <ListView Grid.Row="1"
              ItemsSource="{Binding Users}">
      <ListView.ItemTemplate>
        <DataTemplate x:DataType="model:User">
          <ViewCell>
            <HorizontalStackLayout>
              <Label Text="{Binding LastNameFirstName}"
/>
            </HorizontalStackLayout>
          </ViewCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </Grid>
</Border>
```

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

```
using AdventureWorks.MAUI.CommandClasses;

namespace AdventureWorks.MAUI.Views;

public partial class UserListView : ContentPage
{
  public UserListView(UserViewModelCommands viewModel)
  {
    InitializeComponent();

    ViewModel = viewModel;
  }

  private readonly UserViewModelCommands ViewModel;

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

    BindingContext = ViewModel;

    await ViewModel.GetAsync();
  }
}
```

Open the **MauiProgram.cs** file and add a new service for DI.

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

NOTE: A ListView does not work well on Android devices. You should use a CollectionView instead.

Try It Out

Run the application and click on the **Users** menu to see the list of users.

NOTE: We have removed the ability to navigate to the user detail screen. That will be added back later.

Lab 3: Create a Better Looking DataTemplate

Open the **Resources\Styles\AppStyles.xaml** file and add a new **keyed** style.

```
<Style TargetType="BoxView"
        x:Key="Grid.Item.Separator">
   <Setter Property="VerticalOptions"
           Value="Fill" />
   <Setter Property="BackgroundColor"
           Value="Black" />
   <Setter Property="HeightRequest"
           Value="2" />
   <Setter Property="Margin"
           Value="0,0,0,10" />
 </Style>
```

Open the **Views\UserListView.xaml** file and replace the <ViewCell> element in the <DataTemplate> with the following XAML.

```
<ViewCell>
  <VerticalStackLayout Spacing="5">
    <HorizontalStackLayout>
      <Label FontAttributes="Bold"
             FontSize="Title"
             Text="{Binding LastNameFirstName}" />
    </HorizontalStackLayout>
    <HorizontalStackLayout>
      <Label Text="Email" />
      <Label Text="{Binding Email}" />
    </HorizontalStackLayout>
    <HorizontalStackLayout>
      <Button Text="Edit" />
      <Button Text="Delete" />
    </HorizontalStackLayout>
    <BoxView Style="{StaticResource
Grid.Item.Separator}" />
  </VerticalStackLayout>
</ViewCell>
```

Try It Out

Run the application and click on the **Users** menu to see the new user template.

Lab 4: Create a Product View Model

Right mouse-click on the **ViewModelClasses** in the **AdventureWorks.ViewModelLayer** project and add a new class named **ProductViewModel**. 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 ProductViewModel : ViewModelBase
{
  #region Constructors
  public ProductViewModel() : base()
  {
  }

  public ProductViewModel(IRepository<Product> repo) :
base()
  {
    Repository = repo;
  }
  #endregion

  #region Private Variables
  private readonly IRepository<Product>? Repository;

  private ObservableCollection<Product> _Products =
new();
  private Product? _CurrentEntity = new();
  #endregion

  #region Public Properties
  public ObservableCollection<Product> Products
  {
    get { return _Products; }
    set
    {
      _Products = value;
      RaisePropertyChanged(nameof(Products));
    }
  }

  public Product? CurrentEntity
  {
    get { return _CurrentEntity; }
    set
    {
      _CurrentEntity = value;
      RaisePropertyChanged(nameof(CurrentEntity));
    }
  }
  #endregion

  #region GetAsync Method
  public async Task<ObservableCollection<Product>>
GetAsync()
  {
    RowsAffected = 0;

    try {
      if (Repository == null) {
        LastErrorMessage = REPO_NOT_SET;
      }
      else {
        Products = new
ObservableCollection<Product>(await
Repository.GetAsync());
        RowsAffected = Products.Count;
        InfoMessage = $"Found {RowsAffected} Products";
      }
    }
    catch (Exception ex) {
      PublishException(ex);
    }

    return Products;
  }
  #endregion

  #region GetAsync(id) Method
  /// <summary>
  /// Get a single Product object
  /// </summary>
  /// <param name="id">The ProductId to locate</param>
  /// <returns>An instance of a Product object</returns>
  public async Task<Product?> GetAsync(int id)
  {
    try {
      // Get a Product from a data store
      if (Repository != null) {
        CurrentEntity = await Repository.GetAsync(id);
      }
      else {
        LastErrorMessage = REPO_NOT_SET;

        // MOCK Data
        CurrentEntity = await Task.FromResult(new
Product {
          ProductID = id,
          Name = "A New Product",
          Color = "Black",
          StandardCost = 10,
          ListPrice = 20,
          SellStartDate =
Convert.ToDateTime("7/1/2023"),
          Size = "LG"
        });
      }

      InfoMessage = "Found the Product";
      RowsAffected = 1;
    }
    catch (Exception ex) {
      RowsAffected = 0;
      PublishException(ex);
    }

    return CurrentEntity;
  }
  #endregion

  #region SaveAsync Method
  public async virtual Task<Product?> SaveAsync()
  {
    // TODO: Write code to save data


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

In the **AdventureWorks.MAUI** project, right mouse-click on the **CommandClasses** folder and create a new class named **ProductViewModelCommands**. Replace the entire contents of this new file with the following code.

```
using AdventureWorks.EntityLayer;
using AdventureWorks.ViewModelLayer;
using Common.Library;

namespace AdventureWorks.MAUI.CommandClasses;

public class ProductViewModelCommands : ProductViewModel
{
  #region Constructors
  public ProductViewModelCommands()
  {
  }

  public ProductViewModelCommands(IRepository<Product>
repo) : base(repo)
  {
  }
  #endregion
}
```

Lab 5: Use a Collection View to Display a List of Products

The CollectionView is for presenting lists of data using different layout specifications. It is a more flexible, and performant alternative to ListView.

Right mouse-click on the **Views** folder and add a new **Content Page (XAML)** named **ProductListView**.

Change the **Title** attribute to "Product List".

Add three XML namespaces:

```
xmlns:partial="clr-
namespace:AdventureWorks.MAUI.ViewsPartial"
xmlns:vm="clr-
namespace:AdventureWorks.MAUI.CommandClasses"
xmlns:model="clr-
namespace:AdventureWorks.EntityLayer;assembly=AdventureW
orks.EntityLayer"
```

Add a **x:DataType** attribute to the <ContentPage> element.

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

Replace the <VerticalStackLayout> with the following.

```
<Border Style="{StaticResource Border.Page}">
  <Grid Style="{StaticResource Grid.Page}">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <partial:HeaderView ViewTitle="Product List"
                        ViewDescription="The list of
products in the system." />

    <CollectionView Grid.Row="1"
                    SelectionMode="Single"
                    ItemsSource="{Binding Products}">
      <CollectionView.ItemTemplate>
        <DataTemplate x:DataType="model:Product">
          <VerticalStackLayout Spacing="5">
            <HorizontalStackLayout>
              <Label FontAttributes="Bold"
                     FontSize="Title"
                     Text="{Binding Name}" />
            </HorizontalStackLayout>
            <HorizontalStackLayout>
              <Label Text="Color:" />
              <Label Text="{Binding Color}" />
            </HorizontalStackLayout>
            <HorizontalStackLayout>
              <Label Text="Price:" />
              <Label Text="{Binding ListPrice ,
StringFormat='{0:c}'}" />
            </HorizontalStackLayout>
            <HorizontalStackLayout>
              <Button Text="Edit" />
      <Button Text="Delete" />
            </HorizontalStackLayout>
            <BoxView Style="{StaticResource
Grid.Item.Separator}" />
          </VerticalStackLayout>
        </DataTemplate>
      </CollectionView.ItemTemplate>
    </CollectionView>
  </Grid>
</Border>
```

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

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

namespace AdventureWorks.MAUI.Views;

public partial class ProductListView : ContentPage
{
  public ProductListView(ProductViewModelCommands
viewModel)
  {
    InitializeComponent();

    ViewModel = viewModel;
  }

  private readonly ProductViewModelCommands ViewModel;

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

    BindingContext = ViewModel;

    await ViewModel.GetAsync();
  }
}
```

Open the **AppShell.xaml** file and change the **<ShellContent>** element for the **Products** to point to **ProductListView** instead of **ProductDetailView**.

```
<ShellContent Title="Products"
  ContentTemplate="{DataTemplate views:ProductListView}"
/>
```

Open the **MauiProgram.cs** file and add a few new services for DI.

```
builder.Services.AddScoped<IRepository<Product>,
ProductRepository>();
builder.Services.AddScoped<ProductViewModelCommands>();
builder.Services.AddScoped<ProductDetailView>();
builder.Services.AddScoped<ProductListView>();
```

Try It Out

Run the application and click on the **Products** menu to see the list of products.