XAML Lists Lab - 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: Create a Data Layer

Open the **AdventureWorks.MAUI** project.

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

Be sure to set the .NET version to .NET 7.

Delete the **Class1.cs** file.

Set Dependencies

Right mouse-click on the **Dependencies** folder in the **AdventureWorks.DataLayer** project and add two project references to **AdventureWorks.EntityLayer** and **Common.Library**.

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

Add a User Classes

Open the **User.cs** file and add two additional read-only properties that you can use when building lists.

```
public string FullName
{
  get { return FirstName + " " + LastName; }
}

public string LastNameFirstName
{
  get { return LastName + ", " + FirstName; }
}
```

Add the **UserRepository.cs** file from the **Samples\RepositoryClasses** folder.

Add Product Classes

Add the **Product.cs** file from the **Samples\EntityLayer** folder.

Add the **ProductRepository.cs** file from the **Samples\RepositoryClasses** folder.

Add PhoneType Classes

Add the **PhoneType.cs** file from the **Samples\EntityLayer** folder.

Add the **PhoneTypeRepository.cs** file from the **Samples\RepositoryClasses** folder.

Add Color Classes

Add the **Color.cs** file from the **Samples\EntityLayer** folder.

Add the **ColorRepository.cs** file from the **Samples\RepositoryClasses** folder.

Lab 2: Load a User

Open the **UserViewModel.cs** file and add a using statement.

```
using AdventureWorks.DataLayer;
```

Add a new private variable.

```
private readonly UserRepository? Repository;
```

Add two constructors.

```
#region Constructors
public UserViewModel()
{
}

public UserViewModel(UserRepository repo)
{
  Repository = repo;
}
#endregion
```

REPLACE the Get(id) method to retrieve the data from the UserRepository class.

```
public User? Get(int id)
{
  try {
    if (Repository != null) {
      UserObject = Repository.Get(id);
    }
    else {
      // MOCK Data
      UserObject = new User {
        UserId = id,
        LoginId = "SallyJones",
        FirstName = "Sally",
        LastName = "Jones",
        Email = "Sallyj@jones.com",
        Phone = "615.987.3456",
        PhoneType = "Mobile",
        IsEnrolledIn401k = true,
        IsEnrolledInFlexTime = false,
        IsEnrolledInHealthCare = true,
        IsEnrolledInHSA = false,
        IsActive = true,
        BirthDate = Convert.ToDateTime("08-13-1989")
      };
    }
  }
  catch (Exception ex) {
    System.Diagnostics.Debug.WriteLine(ex.ToString());
  }

  return UserObject;
}
```

Open the **UserViewModelCommanding.cs** file and remove the constructor you created earlier. Add two constructors that look like the following.

```
#region Constructors
public UserViewModelCommanding()
{
}

public UserViewModelCommanding(UserRepository repo) :
base(repo)
{
}
#endregion
```

Add an override to the Init() method.

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

  SaveCommand = new Command(() => Save(), () => true);
}
#endregion
```

Add to Dependency Injection

To have the Repository class injected into the View Model class, you need to add it to the services. Open the **MauiProgram.cs** file and add an additional transient DI.

```
builder.Services.AddTransient<UserRepository>();
```

Try It Out

Run the application and click on the User menu to see the user associated with UserId 8 in the user repository appear.

Try other user id's such as 1, 2, 3, etc.

Lab 3: Load Phone Picker Using MVVM

Open the **UserViewModel.cs** file and add two new private variables.

```
private readonly PhoneTypeRepository?
_PhoneTypeRepository;
private ObservableCollection<string> _PhoneTypesList =
new();
```

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

```
public ObservableCollection<string> PhoneTypesList
{
  get { return _PhoneTypesList; }
  set
  {
    _PhoneTypesList = value;
    RaisePropertyChanged(nameof(PhoneTypesList));
  }
}
```

Add a new method to get all phone types.

```
#region GetPhoneTypes Method
public ObservableCollection<string> GetPhoneTypes()
{
  if (_PhoneTypeRepository != null) {
    var list = _PhoneTypeRepository.Get();

    PhoneTypesList = new
ObservableCollection<string>(list.Select(row =>
row.TypeDescription));
  }

  return PhoneTypesList;
}
#endregion
```

Modify the second constructor to accept a PhoneTypeRepository

```
public UserViewModel(UserRepository repo,
PhoneTypeRepository phoneRepo)
{
  Repository = repo;
  _PhoneTypeRepository = phoneRepo;
}
```

Open the UserViewModelCommanding.cs file and add the PhoneTypeRepository to the second constructor.

```
public UserViewModelCommanding(UserRepository repo,
PhoneTypeRepository phoneRepo) : base(repo, phoneRepo)
{
}
```

Add Phone Types to DI

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

```
builder.Services.AddTransient<PhoneTypeRepository>();
```

Modify the User Detail View

Open the **UserDetailView.xaml.cs** file and add a call to the GetPhoneTypes() method in the **OnAppearing** event procedure.

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

  BindingContext = ViewModel;

  ViewModel.GetPhoneTypes();
  ViewModel.Get(4);
}
```

Open the **UserDetailView.xaml** file and delete the <x:Array x:Key="phoneTypes" …> element from the <ContentPage.Resources> element.

Locate the <Picker> and modify it to look like the following.

```
<Picker Grid.Column="1"
  ItemsSource="{Binding Path=PhoneTypesList}"
  SelectedItem="{Binding Path=UserObject.PhoneType }" />
```

Try It Out

Run the application, click on the User menu and you should still see the same list of phone types, and the picker should be positioned on the value for the user read from the repository.

Lab 4: Using the ListView Control

Use the ListView control to display a larger list of data.

Modify the UserViewModel

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

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

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

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

Add a method to populate this UserList property.

```
#region Get Method
public ObservableCollection<User> Get()
{
  if (Repository != null) {
    UserList = new
ObservableCollection<User>(Repository.Get());
  }

  return UserList;
}
#endregion
```

Create a User List View

Right mouse-click on the **Views** folder and add a new ContentPage named **UserListView**.

Modify the **Title** attribute to "User List".

Add an XML namespace to the partial views namespace.

```
xmlns:partialViews="clr-
namespace:AdventureWorks.MAUI.PartialViews"
```

Add an XML namespace to the view model namespace.

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

Add an XML namespace to the entity layer library.

```
xmlns:model="clr-
namespace:AdventureWorks.EntityLayer;assembly=AdventureW
orks.EntityLayer"
```

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

```
x:DataType="vm:UserViewModelCommanding"
```

Replace the <VerticalStackLayout> element with the following code.

```
<Border Style="{StaticResource Screen.Border}">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <partialViews:HeaderView ViewTitle="User List"
                             ViewSubTitle="The list of
users in the system." />
    <ListView Grid.Row="1"
              x:Name="userList"
              ItemsSource="{Binding Path=UserList}">
      <ListView.ItemTemplate>
        <DataTemplate x:DataType="model:User">
          <ViewCell>
            <HorizontalStackLayout>
              <Label Text="{Binding
Path=LastNameFirstName}" />
            </HorizontalStackLayout>
          </ViewCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </Grid>
</Border>
```

Open the **Views\UserListView.xaml.cs** file and make it look like the following.

```
using AdventureWorks.MAUI.ViewModelsCommanding;

namespace AdventureWorks.MAUI.Views;

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

    ViewModel = viewModel;
  }

  private readonly UserViewModelCommanding ViewModel;

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

    BindingContext = ViewModel;

    ViewModel.Get();
  }
}
```

Open the **Resources\Styles\Styles.xaml** and add the BasedOn attribute to the ListView control.

```
<Style TargetType="ListView"
       BasedOn="{StaticResource BaseControl}">
  ... SETTERS HERE
</Style>
```

Open the **AppShell.xaml** file and change the <ShellContent> for the users.

```
<ShellContent Title="Users"
  ContentTemplate="{DataTemplate views:UserListView}"
  Route="UserListView" />
```

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

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

Try It Out

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

Lab 5: Display User Detail from List View

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

Add the **[QueryProperty]** attribute above the public partial class UserDetailView definition.

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

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

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

Modify the OnAppearing () event procedure to use the **UserId** property.

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

  BindingContext = ViewModel;

  ViewModel.GetPhoneTypes();
  ViewModel.Get(UserId);
}
```

Register the Route

Since you removed the UserDetailView from the <ShellContent> routes, you need to register this route with C#. Open the **AppShell.xaml.cs** file and modify the constructor.

```
public AppShell()
{
 InitializeComponent();

 // Add routes
 Routing.RegisterRoute(nameof(Views.UserDetailView),
typeof(Views.UserDetailView));
}
```

Open the **Views\UserListView.xaml** and add an **ItemSelected** attribute to the <ListView> element.

```
<ListView Grid.Row="1"
        x:Name="userList"
        ItemSelected="userList_ItemSelected"
        ItemsSource="{Binding Path=UserList}">
```

Open the **Views\UserListView.xaml.cs** file and write the **userList_ItemSelected**() event procedure.

```
private async void userList_ItemSelected(object sender,
SelectedItemChangedEventArgs e)
{
  ListView lst = sender as ListView;
  int id = 0;

  if (lst.SelectedItem != null) {
    id = ((User)lst.SelectedItem).UserId;
    await
Shell.Current.GoToAsync($"{nameof(Views.UserDetailView)}
?id={id}");
  }
}
```

Try It Out

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

Click on different users to see the User Detail view appear.

Also notice there is a back button that now appears on the shell.

Modify the Save Button

Open the **UserViewModelCommanding.cs** file and completely override the **Save()** method using the new keyword. After calling the Save() method in the base class, navigate to the user list view.

```
public new async Task<bool> Save()
{
  var ret = base.Save();

  if (true) {
    await
Shell.Current.GoToAsync($"///{nameof(Views.UserListView)
}");
  }

  return ret;
}
```

Modify the Init() method to use the asynchronous version of the Save() method.

```
public override void Init()
{
  base.Init();

  SaveCommand = new Command(async () => await Save(), ()
=> true);
}
```

Try It Out

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

Click on different users to see the User Detail view appear.

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

Lab 6: Create a Product View Model

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

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

namespace AdventureWorks.ViewModelLayer;

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

  public ProductViewModel(ProductRepository repo)
  {
    Repository = repo;
  }
  #endregion

  #region Private Variables
  private ProductRepository? Repository;
  private ObservableCollection<Product> _ProductList =
new();
  private Product? _ProductObject = new();
  #endregion

  #region Public Properties
  public Product? ProductObject
  {
    get { return _ProductObject; }
    set
    {
      _ProductObject = value;
      RaisePropertyChanged(nameof(ProductObject));
    }
  }

  public ObservableCollection<Product> ProductList
  {
    get { return _ProductList; }
    set
    {
      _ProductList = value;
      RaisePropertyChanged(nameof(ProductList));
    }
  }
  #endregion

  #region Get Method
  public ObservableCollection<Product> Get()
  {
    if (Repository != null) {
      ProductList = new
ObservableCollection<Product>(Repository.Get());
    }

    return ProductList;
  }
  #endregion

  #region Get(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 Product? Get(int id)
  {
    try {
      if (Repository != null) {
        // Get a Product from a data store
        ProductObject = Repository.Get(id);
      }
      else {
        // MOCK Data
        ProductObject = new Product {
          ProductID = id,
          Name = "A New Product",
          Color = "Black",
          StandardCost = 10,
          ListPrice = 20,
          SellStartDate =
Convert.ToDateTime("7/1/2023"),
          Size = "LG"
        };
      }
    }
    catch (Exception ex) {
      System.Diagnostics.Debug.WriteLine(ex.ToString());
    }

    return ProductObject;
  }
  #endregion
}
```

Right mouse-click on the **ViewModelsCommanding** folder and create a new class named **ProductViewModelCommanding**.

```
using AdventureWorks.DataLayer;
using AdventureWorks.ViewModelLayer;

namespace AdventureWorks.MAUI.ViewModelsCommanding;

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

  public ProductViewModelCommanding(ProductRepository
repo) : base(repo)
  {
  }
  #endregion
}
```

Lab 7: Using the Collection View Control

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 named **ProductListView**.

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

Add two XML namespaces:

```
xmlns:partialViews="clr-
namespace:AdventureWorks.MAUI.PartialViews"
xmlns:vm="clr-
namespace:AdventureWorks.MAUI.ViewModelsCommanding"
```

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

```
x:DataType="vm:ProductViewModelCommanding"
```

Replace the <VerticalStackLayout> with the following.

```
<Border Style="{StaticResource Screen.Border}">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <partialViews:HeaderView ViewTitle="Product List"
                             ViewSubTitle="The list of
products in the system." />
    <CollectionView Grid.Row="1"
                    x:Name="productList"

SelectionChanged="productList_SelectionChanged"
                    SelectionMode="Single"
                    ItemsSource="{Binding
Path=ProductList}">
      <CollectionView.ItemTemplate>
        <DataTemplate x:DataType="model:Product">
          <Border Stroke="Black"
                  StrokeThickness="1"
                  Padding="2"
                  Margin="8,4">
            <Grid RowDefinitions="Auto,Auto,Auto"
                  ColumnDefinitions="Auto,*">
              <Label Grid.Row="0"
                     Grid.ColumnSpan="2"
                     FontSize="Large"
                     Text="{Binding Path=Name}" />
              <Label Grid.Row="1"
                     Grid.Column="0"
                     Text="Color" />
              <Label Grid.Row="1"
                     Grid.Column="1"
                     Text="{Binding Path=Color}" />
              <Label Grid.Row="2"
                     Grid.Column="0"
                     Text="Price" />
              <Label Grid.Row="2"
                     Grid.Column="1"
                     Text="{Binding Path=ListPrice ,
StringFormat='{0:c}'}" />
            </Grid>
          </Border>
        </DataTemplate>
      </CollectionView.ItemTemplate>
    </CollectionView>
  </Grid>
```

```
</Border>
```

Open the **ProductListView.xaml.cs** file and make it look like the following.

```
using AdventureWorks.MAUI.ViewModelsCommanding;

namespace AdventureWorks.MAUI.Views;

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

    ViewModel = viewModel;
  }

  private readonly ProductViewModelCommanding ViewModel;

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

    BindingContext = ViewModel;

    ViewModel.Get();
  }

  private async void productList_SelectionChanged(object
sender, SelectionChangedEventArgs e)
  {
    CollectionView lst = sender as CollectionView;
    int id = 0;

    if (lst.SelectedItem != null) {
      id = ((Product)lst.SelectedItem).ProductID;
      await
Shell.Current.GoToAsync($"{nameof(Views.ProductDetailVie
w)}?id={id}");
    }
  }
}
```

Open the **AppShell.xaml** file and change the <ShellContent> element for the Products.

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

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

```
builder.Services.AddTransient<ProductRepository>();
builder.Services.AddTransient<ProductViewModelCommanding
>();
builder.Services.AddTransient<ProductListView>();
```

Try It Out

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

DO NOT CLICK on any products yet.

Lab 8: Display Product Detail from Collection View

Open the **Views\ProductDetailView.xaml** file.

Add an XML namespace for the view model commanding.

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

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

```
x:DataType="vm:ProductViewModelCommanding"
```

Add all the appopriate data bindings to each <Entry> element.

```
<Entry Grid.Column="1"
       Grid.Row="1"
       Text="{Binding Path=ProductObject.Name}" />

<Entry Grid.Row="2"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.ProductNumber}"
/>

<Entry Grid.Row="3"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.Color}" />

<Entry Grid.Row="4"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.StandardCost}"
/>

<Entry Grid.Row="5"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.ListPrice}" />

<Entry Grid.Row="6"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.Size}" />

<Entry Grid.Row="7"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.Weight}" />

<Entry Grid.Row="9"
       Grid.Column="1"
       Text="{Binding
Path=ProductObject.ProductCategoryID}" />

<Entry Grid.Row="9"
       Grid.Column="1"
       Text="{Binding
Path=ProductObject.ProductModelID}" />

<Entry Grid.Row="1"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.SellStartDate}"
/>

<Entry Grid.Row="2"
       Grid.Column="1"
       Text="{Binding Path=ProductObject.SellEndDate}"
/>

<Entry Grid.Row="3"
       Grid.Column="1"
       Text="{Binding
Path=ProductObject.DiscontinuedDate}" />
```

Modify the Code Behind

Open the **Views\ProductDetailView.xaml.cs** file and make it look like the following.

```
using AdventureWorks.ViewModelLayer;

namespace AdventureWorks.MAUI.Views;

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

    ViewModel =
(ProductViewModel)this.Resources["viewModel"];
   }

  public ProductViewModel ViewModel { get; set; }
  public int ProductId { get; set; }

  private void ContentPage_Loaded(object sender,
EventArgs e)
  {
    ViewModel.Get(ProductId);
  }

  private async void SaveButton_Click(object sender,
EventArgs e)
{
  System.Diagnostics.Debugger.Break();

  await
Shell.Current.GoToAsync($"///{nameof(Views.ProductListVi
ew)}");
}}
```

Register the Route

Open the **AppShell.xaml.cs** file and modify the constructor to look like the following.

```
public AppShell()
{
 InitializeComponent();

 // Add routes
 Routing.RegisterRoute(nameof(Views.UserDetailView),
typeof(Views.UserDetailView));
 Routing.RegisterRoute(nameof(Views.ProductDetailView),
typeof(Views.ProductDetailView));
}
```

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

```
builder.Services.AddTransient<ProductDetailView>();
```

Try It Out

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

Click on different products to see the Product Detail change.

Lab 9: Carousel View

By default, a CarouselView control provides looped access to its collection of items. Therefore, swiping backwards from the first item in the collection will display the last item in the collection. Similarly, swiping forwards from the last item in the collection will return to the first item in the collection.

CarouselView shares much of its implementation with CollectionView. However, the two controls have different use cases. CollectionView is typically used to present lists of data of any length, whereas CarouselView is typically used to highlight information in a list of limited length.

NOTE:	The CarouselView is NOT for use in Windows applications.

Create a Color View Model

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

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

namespace AdventureWorks.ViewModelLayer;

public class ColorViewModel : ViewModelBase
{
  #region Constructors
  public ColorViewModel()
  {
  }

  public ColorViewModel(ColorRepository repo)
  {
    Repository = repo;
  }
  #endregion

  #region Private Variables
  private ColorRepository? Repository;
  private ObservableCollection<Color> _ColorList =
new();
  #endregion

  #region Public Properties
  public ObservableCollection<Color> ColorList
  {
    get { return _ColorList; }
    set
    {
      _ColorList = value;
      RaisePropertyChanged(nameof(ColorList));
    }
  }
  #endregion

  #region Get Method
  public ObservableCollection<Color> Get()
  {
    if (Repository != null) {
      ColorList = new
ObservableCollection<Color>(Repository.Get());
    }

    return ColorList;
  }
  #endregion
}
```

Right mouse-click on the **ViewModelsCommanding** folder and add a new class named **ColorViewModelCommanding**.

```
using AdventureWorks.DataLayer;
using AdventureWorks.ViewModelLayer;

namespace AdventureWorks.MAUI.ViewModelsCommanding;

public class ColorViewModelCommanding : ColorViewModel
{
  #region Constructors
  public ColorViewModelCommanding()
  {
  }

  public ColorViewModelCommanding(ColorRepository repo)
: base(repo)
  {
  }
  #endregion
}
```

Create a Color List View

Right mouse-click on the **Views** folder and add a new ContentPage named **ColorListView**.

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

Add XML namespaces for the commanding view models, entity layer, and partial views.

```
xmlns:partialViews="clr-
namespace:AdventureWorks.MAUI.PartialViews"
xmlns:vm="clr-
namespace:AdventureWorks.MAUI.ViewModelsCommanding"
xmlns:model="clr-
namespace:AdventureWorks.EntityLayer;assembly=AdventureW
orks.EntityLayer"
```

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

```
x:DataType="vm:ColorViewModelCommanding"
```

Change the <VerticalStackLayout> to the following.

```
<Border Style="{StaticResource Screen.Border}">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <partialViews:HeaderView ViewTitle="Color List"
                              ViewSubTitle="The list of
colors in the system." />
    <CarouselView Grid.Row="1"
                  MinimumHeightRequest="200"
                  Margin="10"
                  x:Name="colorList"
                  ItemsSource="{Binding
Path=ColorList}">
      <CarouselView.ItemTemplate>
        <DataTemplate x:DataType="model:Color">
          <StackLayout>
            <Border Stroke="DarkGray"
                    StrokeThickness="1">
              <StackLayout>
                <Label FontSize="Large"
                       HorizontalOptions="Center"
                       VerticalOptions="Center"
                       Text="{Binding Path=ColorName}"
/>
              </StackLayout>
            </Border>
          </StackLayout>
        </DataTemplate>
      </CarouselView.ItemTemplate>
    </CarouselView>
  </Grid>
</Border>
```

Open the **ColorListView.xaml.cs** file and make it look like the following.

```
using AdventureWorks.MAUI.ViewModelsCommanding;

namespace AdventureWorks.MAUI.Views;

public partial class ColorListView : ContentPage
{
  public ColorListView(ColorViewModelCommanding
viewModel)
  {
    InitializeComponent();

    ViewModel = viewModel;
  }

  private readonly ColorViewModelCommanding ViewModel;

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

    BindingContext = ViewModel;

    ViewModel.Get();
  }
}
```

Open the **AppShell.xaml** file and add a new <ShellContent> element at the bottom of the file.

```
<ShellContent Title="Colors"
  ContentTemplate="{DataTemplate views:ColorListView}"
  Route="ColorListView" />
```

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

```
builder.Services.AddTransient<ColorRepository>();
builder.Services.AddTransient<ColorViewModelCommanding>(
);
builder.Services.AddTransient<ColorListView>();
```

Try It Out

Switch to the **Android Emulator**.

Run the application and click on the Colors menu.

Lab 10: Add Indicators

Open the **Views\ColorListView.xaml** and add a new <RowDefinition> at the end of the row definitions for the grid.

```
<RowDefinition Height="Auto" />
```

Add the following below the closing </CarouselView> element.

```
<IndicatorView Grid.Row="2"
               x:Name="colorIndicators"
               Margin="5"
               IndicatorSize="20"
               IndicatorColor="LightGray"
               SelectedIndicatorColor="DarkGray"
               HorizontalOptions="Center" />
```

Add a **PositionChanged** event attribute to the <**CarouselView**> element.

```
PositionChanged="colorList_PositionChanged"
```

Open the **ColorListView.xaml.cs** file and add the following line of code in the OnAppearing event after calling the Get() method.

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

  BindingContext = ViewModel;

  ViewModel.Get();
  colorIndicators.Count = ViewModel.ColorList.Count;
}
```

Add the following line of code to the colorList_PositionChanged() event procedure.

```
private void colorList_PositionChanged(object sender,
PositionChangedEventArgs e)
{
  colorIndicators.Position = e.CurrentPosition;
}
```

Try It Out

Switch to the **Android Emulator** and run the application. Swipe through the colors to see the IndicatorView control update.