XAML Lists Lab - WPF

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.WPF** project.

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

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

Delete the **Class1.cs** file.

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

Add a User Repository Class

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

Review this class with the students.

Add Product Classes

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

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

Review these classes with the students.

Add PhoneType Classes

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

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

Review these classes with the students.

Add Color Classes

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

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

Review these classes with the students.

Demo 2: Load a User

Open the **Views\UserDetailView.xaml** file and add a Loaded event procedure to the UserControl.

Locate the <Border> element with the DataContext attribute and add a Name property.

```
<Border x:Name="topBorder"
        Style="{StaticResource Screen.Border}"
        DataContext="{StaticResource viewModel}">
```

Open the **Views\UserDetailView.xaml.cs** file and remove the line from the constructor.

```
~~UserObject.LoginId = "asdfasdfasfd";~~
```

Add a GetUser() method.

```
private void GetUser(int id)
{
  // Get a specific User object and assign to
  // the top border's DataContext attribute
  topBorder.DataContext = new UserRepository().Get(id);
}
```

In the UserControl_Loaded() event procedure add code to call the GetUser method.

```
private void UserControl_Loaded(object sender,
System.Windows.RoutedEventArgs e)
{
  // Get a user
  GetUser(5);
}
```

Try It Out

Run the application and click on the **User** menu to see the user number 5 appear.

Try other user id's such as 6, 7, 8, etc.

Demo 3: Combo Box

Open the **Views\UserDetailView.xaml** file and **remove** the hard-coded array in the <UserControl.Resources> element for the phone types.

You can remove the **sys:** namespace if you wish.

Add a new XML namespace.

```
xmlns:data="clr-
namespace:AdventureWorks.DataLayer;assembly=AdventureWor
ks.DataLayer"
```

Create an ObjectDataProvider

In the <UserControl.Resources> element add an <ObjectDataProvider>.

```
<ObjectDataProvider x:Key="phoneTypes"
  ObjectType="{x:Type data:PhoneTypeRepository}"
  MethodName="Get" />
```

Locate the <ComboBox> of phone types and modify it to look like the following.

```
<ComboBox Grid.Column="1"
  ItemsSource="{Binding Source={StaticResource
phoneTypes}}"
  SelectedValuePath="TypeDescription"
  DisplayMemberPath="TypeDescription"
  SelectedValue="{Binding Path=PhoneType}" />
```

Try It Out

Run the application and click on the **User** menu to view the result.

Demo 4: Using the Data Grid Control

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

Add a new XML namespace.

```
xmlns:data="clr-
namespace:AdventureWorks.DataLayer;assembly=AdventureWor
ks.DataLayer"
```

Add a new XML namespace.

```
xmlns:PartialViews="clr-
namespace:AdventureWorks.WPF.PartialViews"
```

Add a <UserControl.Resources> element and add an object data provider.

```
<UserControl.Resources>
  <ObjectDataProvider x:Key="UserListView"
                      ObjectType="{x:Type
data:UserRepository}"
                      MethodName="Get" />
</UserControl.Resources>
```

Replace the <Grid> 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." />
    <DataGrid Grid.Row="1"
              MaxHeight="200"
              x:Name="userGrid"
              ItemsSource="{Binding
Source={StaticResource UserListView}}" />
  </Grid>
</Border>
```

Open the **Resources\Styles\StandardStyles.xaml** in the AdventureWorks.WPF.Resources project and add a new style.

```
<Style TargetType="DataGrid"
       BasedOn="{StaticResource BaseFrameworkElement}">
</Style>
```

Open the **MainWindow.xaml** file and change the <MenuItem> for the users. This is NOT necessary for this application, but in case later you want to use reflection to load the user controls, you can use the **Tag** attribute.

```
<MenuItem Header="Users"
          Tag="UserListView"
          Click="Menu_Click" />
```

Open the **MainWindow.xaml.cs** file and change the switch statement that displays the users to display the UserListView control.

```
case "userlistview":
  ContentArea.Children.Add(new UserListView());
  break;
```

Try It Out

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

Demo 5: Synchronize User Detail with Data Grid

Open the **Views\UserDetailView.xaml** file and remove the **DataContext** attribute from the <Border>.

Remove the **x:Name** attribute on the <Border> also.

Remove the **<vm:User …>** element in the <UserControl.Resources> element.

Remove the **Loaded="UserControl_Loaded"** attribute from the <UserControl>.

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

Delete the UserControl_Loaded() event procedure.

Delete the GetUser() method.

Remove the **HeaderView** control and the <RowDefinition> for this row.

Modify User List View

Open the **Views\UserListView.xaml** file and add a third RowDefinition.

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

Add an additional attribute to the <DataGrid> to force it to select the first row.

```
SelectedIndex="0"
```

Drag the UserDetailView user control into the grid and make it look like the following:

```
<local:UserDetailView Grid.Row="2"
   DataContext="{Binding ElementName=userGrid,
Path=SelectedItem}" />
```

Try It Out

Run the application and click on the **User** menu and click on different users to see the User Detail update.

Demo 6: Data Grid with Custom Columns

Open the **Views\UserListView.xaml** file and either use the Properties window to add custom columns, or just type in the following XAML.

```
<DataGrid Grid.Row="1"
          MaxHeight="200"
          AutoGenerateColumns="False"
          x:Name="userGrid"
          SelectedIndex="0"
          ItemsSource="{Binding Source={StaticResource
UserListView}}">
  <DataGrid.Columns>
    <DataGridTextColumn Binding="{Binding Path=LoginId}"
                        Header="Login ID" />
    <DataGridTextColumn Binding="{Binding
Path=LastName}"
                        Header="Last Name" />
    <DataGridTextColumn Binding="{Binding
Path=FirstName}"
                        Header="First Name" />
    <DataGridTextColumn Binding="{Binding Path=Email}"
                        Header="Email Address" />
    <DataGridCheckBoxColumn Binding="{Binding
Path=IsActive}"
                        Header="Still Employed?" />
  </DataGrid.Columns>
</DataGrid>
```

Try It Out

Run the application and click on the **Users** menu to see the list with custom columns.

Demo 7: Using the List View Control

Right mouse-click on the **Views** folder and add a new User Control named **ProductListView**.

Add two XML namespaces:

```
xmlns:PartialViews="clr-
namespace:AdventureWorks.WPF.PartialViews"
xmlns:data="clr-
namespace:AdventureWorks.DataLayer;assembly=AdventureWor
ks.DataLayer"
```

Add a <UserControl.Resources> element and add an object data provider within.

```
<UserControl.Resources>
  <ObjectDataProvider x:Key="ProductListView"
                      ObjectType="{x:Type
data:ProductRepository}"
                      MethodName="Get" />
</UserControl.Resources>
```

Change the Grid to a <Border> and make it look like the following.

```
<Border Style="{StaticResource Screen.Border}">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
    <PartialViews:HeaderView ViewTitle="Product List"
                             ViewSubTitle="The list of
products in the system." />
    <ListView Grid.Row="1"
              MaxHeight="200"
              x:Name="products"
              SelectedIndex="0"
              ItemsSource="{Binding
Source={StaticResource ProductListView}}">
      <ListView.View>
        <GridView>
          <GridViewColumn Header="Product Name"
                          Width="200"
                          DisplayMemberBinding="{Binding
                      Path=Name}" />
          <GridViewColumn Header="Product Number"
                          Width="100"
                          DisplayMemberBinding="{Binding
                      Path=ProductNumber}" />
          <GridViewColumn Header="Color"
                          Width="100"
                          DisplayMemberBinding="{Binding
                      Path=Color}" />
          <GridViewColumn Header="Cost"
                          Width="100"
                          DisplayMemberBinding="{Binding
                      Path=StandardCost,
StringFormat={}{0:c}}" />
          <GridViewColumn Header="List Price"
                          Width="100"
                          DisplayMemberBinding="{Binding
                      Path=ListPrice,
StringFormat={}{0:c}}" />
        </GridView>
      </ListView.View>
    </ListView>
  </Grid>
</Border>
```

Open the **MainWindow.xaml** file and change the <MenuItem> for the users. This is NOT necessary for this application, but in case later you want to use reflection to load the user controls, you can use the **Tag** attribute.

```
<MenuItem Header="Products"
          Tag="ProductListView"
          Click="Menu_Click" />
```

Open the **MainWindow.xaml.cs** file and change the switch statement that displays the products to display the ProductListView control.

```
case "productlistview":
  ContentArea.Children.Add(new ProductListView());
  break;
```

Try It Out

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

Right Justify Columns

First within the <ListView> add a Resources element.

```
<ListView.Resources>
  <Style TargetType="ListViewItem">
    <Setter Property="HorizontalContentAlignment"
            Value="Stretch" />
  </Style>
</ListView.Resources>
```

Then use a <GridViewColumn> with a CellTemplate for both the Cost and Price fields.

```
<GridViewColumn Header="Cost">
  <GridViewColumn.CellTemplate>
    <DataTemplate>
      <TextBlock HorizontalAlignment="Right"
                 Text="{Binding Path=StandardCost,
StringFormat={}{0:c}}" />
    </DataTemplate>
  </GridViewColumn.CellTemplate>
</GridViewColumn>
<GridViewColumn Header="List Price">
  <GridViewColumn.CellTemplate>
    <DataTemplate>
      <TextBlock HorizontalAlignment="Right"
                 Text="{Binding Path=ListPrice,
StringFormat={}{0:c}}" />
    </DataTemplate>
  </GridViewColumn.CellTemplate>
</GridViewColumn>
```

Try It Out

Run the application and click on the **Products** menu to see the Cost and Price fields are now right justified.

Demo 8: Synchronize Product Detail with Grid

Open the **Views\ProductListView.xaml** file and drag the **ProductDetailView** user control into the grid and make it look like the following:

```
<local:ProductDetailView Grid.Row="2"
   DataContext="{Binding ElementName=products,
Path=SelectedItem}" />
```

Assign Data Bindings

Open the **Views\ProductDetailView.xaml** file and remove the HeaderView row.

Add a new XML namespace.

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

Add a <UserControl.Resources> element.

```
<UserControl.Resources>
  <vm:Product x:Key="viewModel" />
</UserControl.Resources>
```

Add a DataContext to the <Border>.

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

Go through each TextBox and do all data bindings.

When you are done, remove the **DataContext** attribute, the **<UserControl.Resources>** element and the **XML namespace** you just added.

Get Product Object from DataContext

Open the **Views\ProductDetailView.xaml** file and add a click event named SaveButton_Click to the **Save** button.

Open the **Views\ProductDetailView.xaml.cs** file and add a public property.

```
public Product? ProductObject { get; set; }
```

In the SaveButton_Click() event procedure add the following.

```
private void SaveButton_Click(object sender,
RoutedEventArgs e) {
  ProductObject = this.DataContext as Product;
  System.Diagnostics.Debugger.Break();
}
```

Try It Out

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

Click on different products to see the Product Detail change.

Click on the **Save** button to view the data in the **ProductObject**.

Demo 9: ComboBox with ItemTemplate

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

```
xmlns:data="clr-
namespace:AdventureWorks.DataLayer;assembly=AdventureWor
ks.DataLayer"
```

In the <UserControl.Resources> element add an <ObjectDataProvider>.

```
<UserControl.Resources>
  <ObjectDataProvider x:Key="colorList"
                      ObjectType="{x:Type
data:ColorRepository}"
                      MethodName="Get" />
</UserControl.Resources>
```

Locate the <TextBox> for Color and replace it with a <ComboBox>.

```
<ComboBox Grid.Row="2"
          Grid.Column="1"
          ItemsSource="{Binding Source={StaticResource
colorList}}"
          SelectedValuePath="ColorName"
          SelectedValue="{Binding Path=Color}">
  <ComboBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <Border BorderBrush="{Binding Path=ColorName}"
                BorderThickness="10" />
        <TextBlock Text="{Binding Path=ColorName}" />
      </StackPanel>
    </DataTemplate>
  </ComboBox.ItemTemplate>
</ComboBox>
```

Try It Out

Run the application and click on the Products menu.

Click on different products to see different colors appear in the combo box.

Drop down the combo box to view all the different colors.