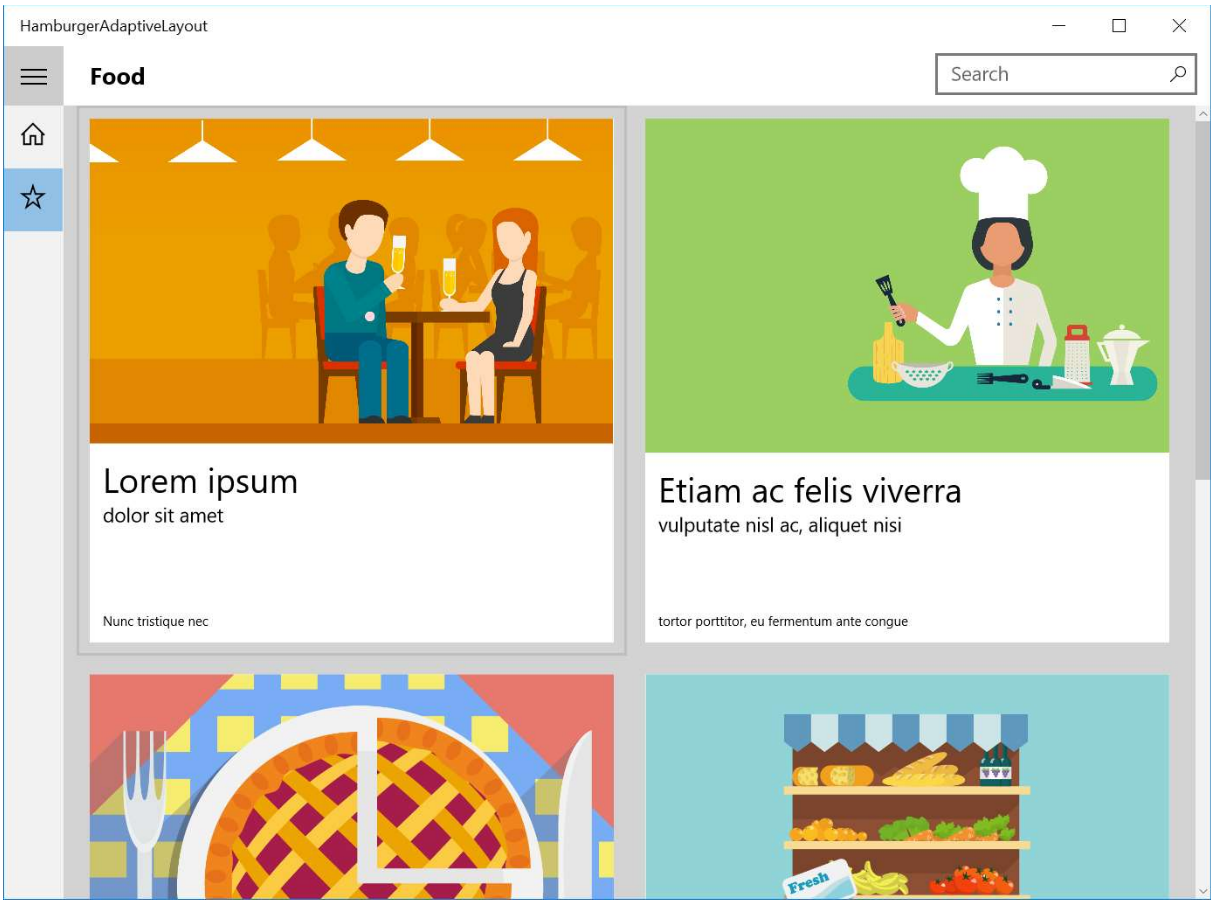
## UWP-044 - Adeptly Adaptive Challenge

The final solution I'm giving you is the Adeptly Adaptive Challenge. You will be asked to create an adaptive solution for a fake news app. The key to this app is that, as the window size is changed the size and layout of the individual news items, the visibility of the search textbox, and the font sizes used in the app are affected as well.

Here's an example of what the completed application should look like when the window is expanded to the largest form factor:

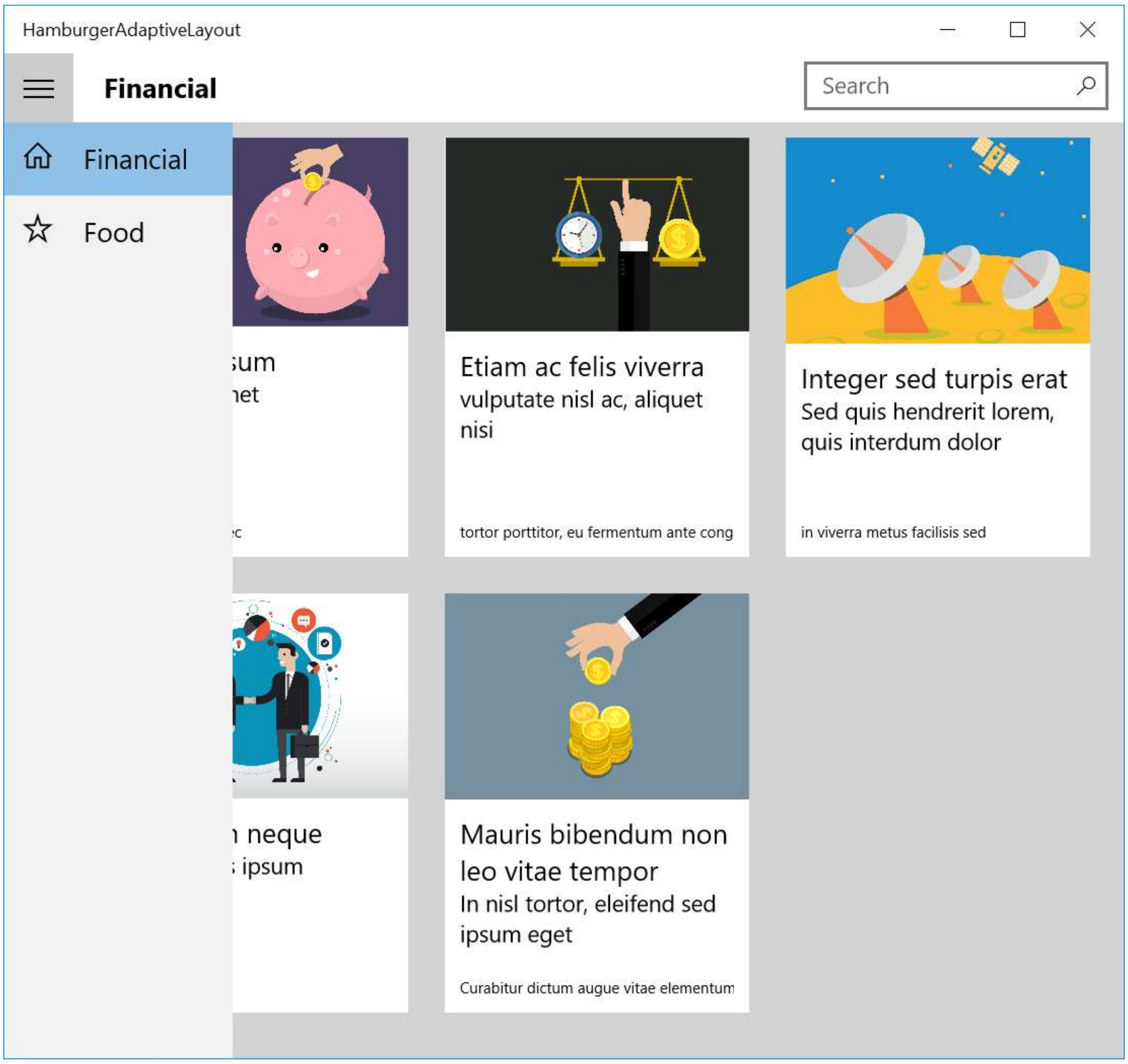


Lorem ipsum
dolor sit amet
Nunc tristique nec

Etiam ac felis viverra
vulputate nisl ac, aliquet nisi
tortor porttitor, eu fermentum ante congue

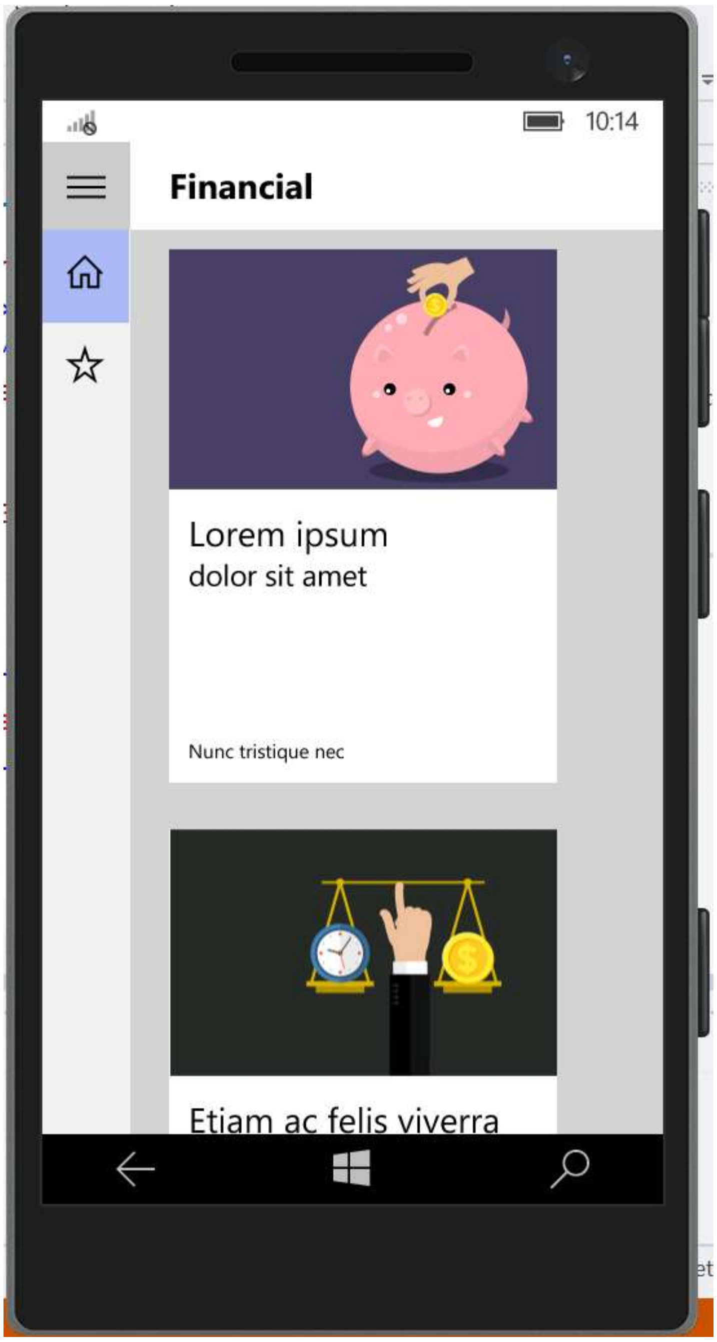
Notice that there's two columns and each of the "cards" representing fake news stories are large as well.

Here's what the completed application should look like when the window is sized a bit smaller:


HamburgerAdaptiveLayout
Financial
Search
Financial
Food
ꞩum
ıet
ꞩc
Etiam ac felis viverra
vulputate nisl ac, aliquet nisi
tortor porttitor, eu fermentum ante cong
Integer sed turpis erat
Sed quis hendrerit lorem, quis interdum dolor
in viverra metus facilisis sed
ı neque
ꞩ ipsum
Mauris bibendum non leo vitae tempor
In nisl tortor, eleifend sed ipsum eget
Curabitur dictum augue vitae elementum

Notice that there are multiple columns (three in this case) and that each "card" representing a fake news story is smaller.

Finally, here's what the application will look like when running on the smaller form factor of the Phone:



Notice that there's only one column of "cards" and that the search box usually at the top is not visible.

I'm giving you the zipped folder associated with this lesson contains the resources you'll need to complete the challenge. It also includes the instructions and screenshots that you can reference to correctly complete the challenge.

See the document UWP-044.Instructions.txt in the zipped resources folder for this lesson for the complete requirements / instructions for this challenge.

The individual fake news article images are also available in the zipped resource folder, Food1.png through Food5.png, and Financial1.png through Financial5.png.

As far as the implementation goes, each news item will contain an image, a headline, a subhead, and a dateline. So you'll create a NewsItem class, and it's going to have those properties, and you're going to then create a bindable collection of these items and then bind to them using a GridView.

If you look at the very bottom of the instructions txt file, I give you a helper method that will create instances of the NewsItem class (that you will need to build). This just saves you from so much typing and you will be using the exact “dummy data” I used in my solution video.

Use techniques that we discussed previously to create this. Having said that you're only going to need to create a single MainPage.xaml so we not going to utilize the device family-specific view. Instead we will do this all with the VisualStateManager and Adaptive Triggers.

You're going to use a GridView, you're also going to create a User Control that will display instances of the NewsItem class. The User Control techniques will allow you to re-size each of the individual data templates for the NewsItems in the DataTemplate for the GridView.

## UWP-045 - Adeptly Adaptive Challenge Solution - Part 1: Setup and MainPage Layout

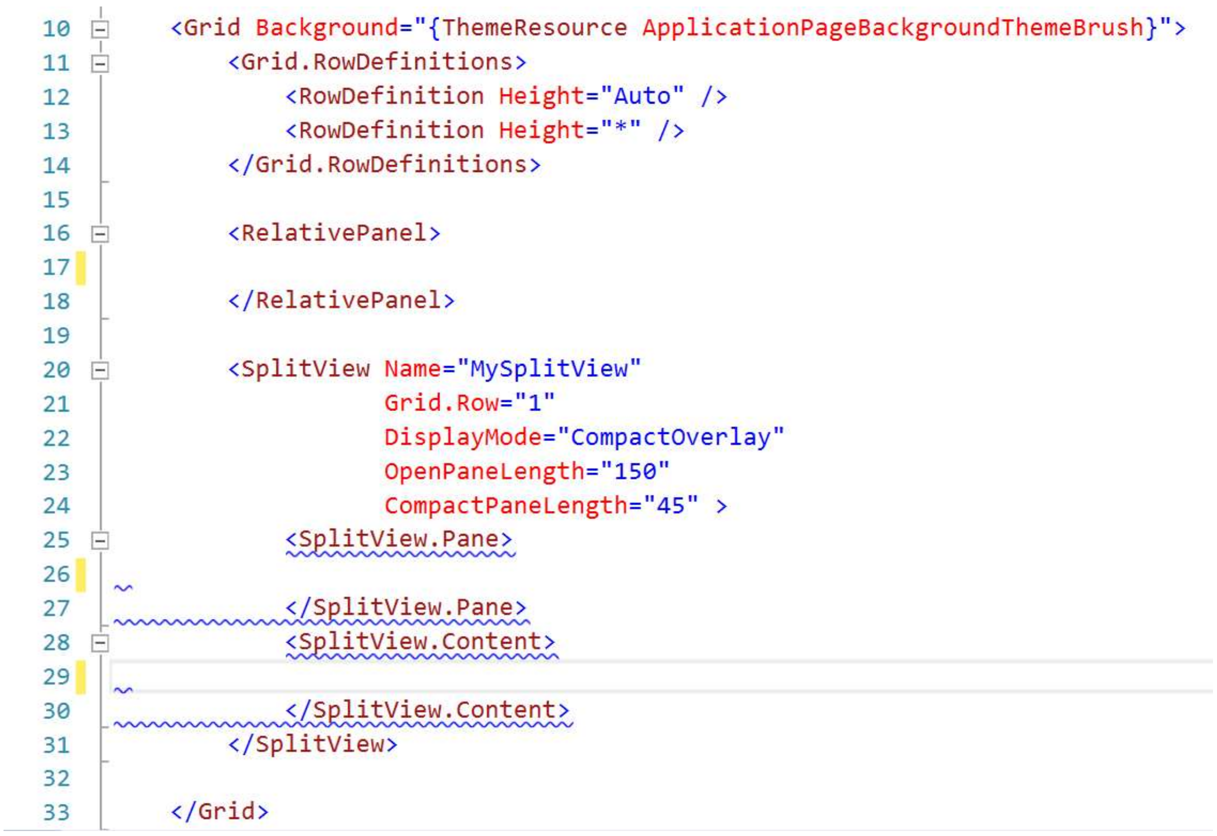
This is the first of several solution lessons for the Adeptly Adaptive Challenge.

Step 1: Create a new Universal Windows Blank App named “FakeNews”.

Step 2: In the Solution Explorer, create and Assets folder, then drag and drop the Food and Financial image assets from the zipped resource folder to the new Assets folder.

Step 3: In the App.xaml.cs remove the frame counter.

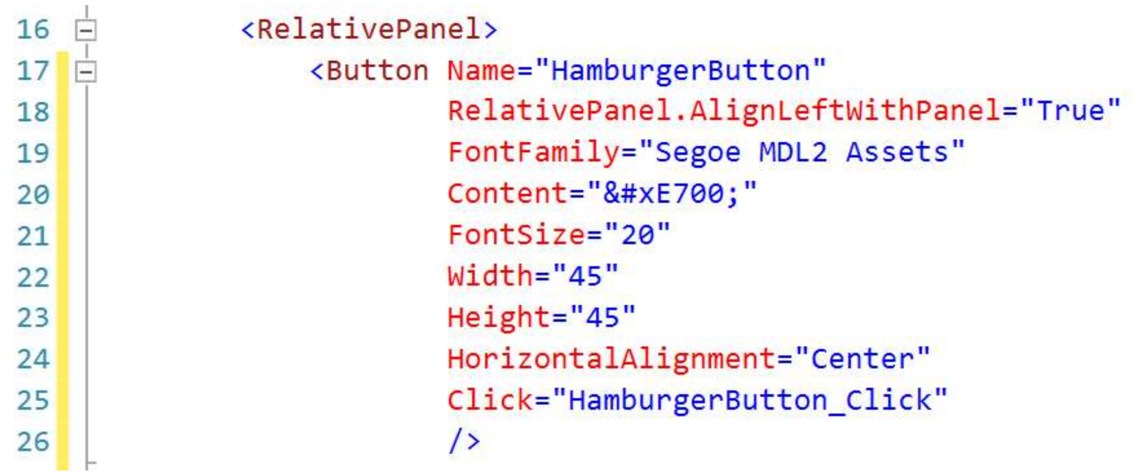
Step 4: In the MainPage.xaml create the overall “hamburger” style navigation layout by adding two row definitions.  Add a RelativePanel in the first row and a SplitView in the second row.  Name the SplitView “MySplitView”, set its DisplayMode to “CompactOverlay”, set its OpenPaneLength to 150 and its CompactPaneLength to 45.  Add the Pane and Content elements as well.



```
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <RelativePanel>

        </RelativePanel>

        <SplitView Name="MySplitView"
                   Grid.Row="1"
                   DisplayMode="CompactOverlay"
                   OpenPaneLength="150"
                   CompactPaneLength="45" >
            <SplitView.Pane>

            </SplitView.Pane>
            <SplitView.Content>

            </SplitView.Content>
        </SplitView>

    </Grid>
```

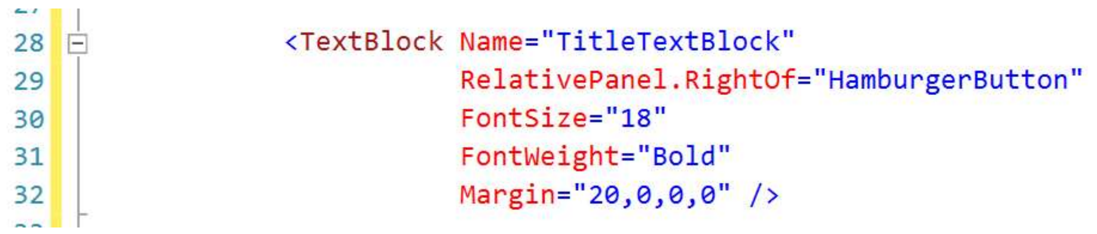
Step 6: Focus on adding the hamburger button, the title and the search textbox to the RelativePanel.

Name the hamburger button “HamburgerButton” and create a click event handler.  Furthermore, use the FontFamily and Content to create the hamburger icon and center it in the button.  Resize the button to 45 by 45.  Position the button inside of the RelativePanel to be aligned left with the panel.



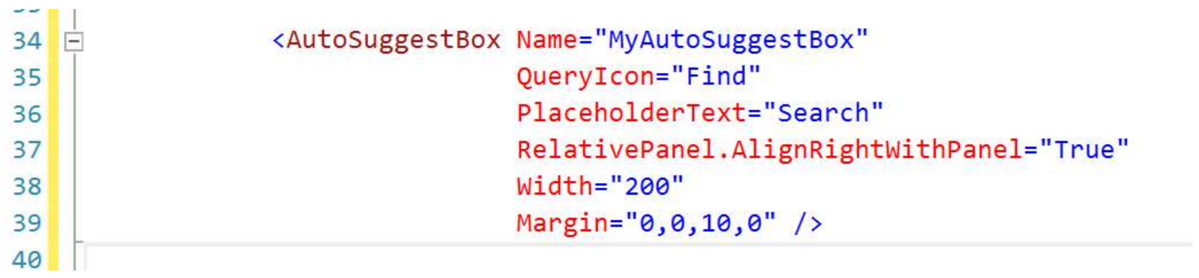
```
        <RelativePanel>
            <Button Name="HamburgerButton"
                    RelativePanel.AlignLeftWithPanel="True"
                    FontFamily="Segoe MDL2 Assets"
                    Content="&#xE700;"
                    FontSize="20"
                    Width="45"
                    Height="45"
                    HorizontalAlignment="Center"
                    Click="HamburgerButton_Click"
                    />
```

Create a TextBlock named “TitleTextBlock”.  Position it to the right of the HamburgerButton.  Give it a little margin on the left to separate it nicely from the button.



```
            <TextBlock Name="TitleTextBlock"
                       RelativePanel.RightOf="HamburgerButton"
                       FontSize="18"
                       FontWeight="Bold"
                       Margin="20,0,0,0" />
```

Add an AutoSuggestBox named “MyAutoSuggestBox” and position it to be aligned right with the RelativePanel.  Use the “Find” QueryIcon and set the Placeholder Text to “Search”.  The width should be 200 pixels wide.  Add a small 10 pixel margin to space it nicely from the right side of the Window.



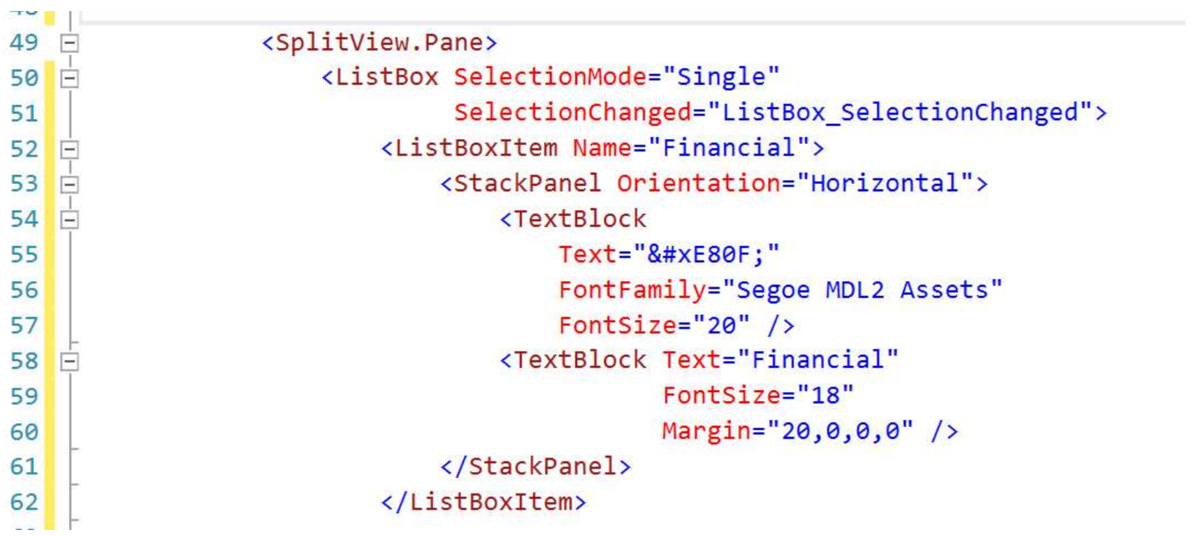
```
            <AutoSuggestBox Name="MyAutoSuggestBox"
                            QueryIcon="Find"
                            PlaceholderText="Search"
                            RelativePanel.AlignRightWithPanel="True"
                            Width="200"
                            Margin="0,0,10,0" />
```

Step 7: Focus on the SplitView.Pane by adding a ListBox and two ListBoxItems.

The ListBox should only allow one selection at a time.  We will also want to handle the SelectionChanged event.

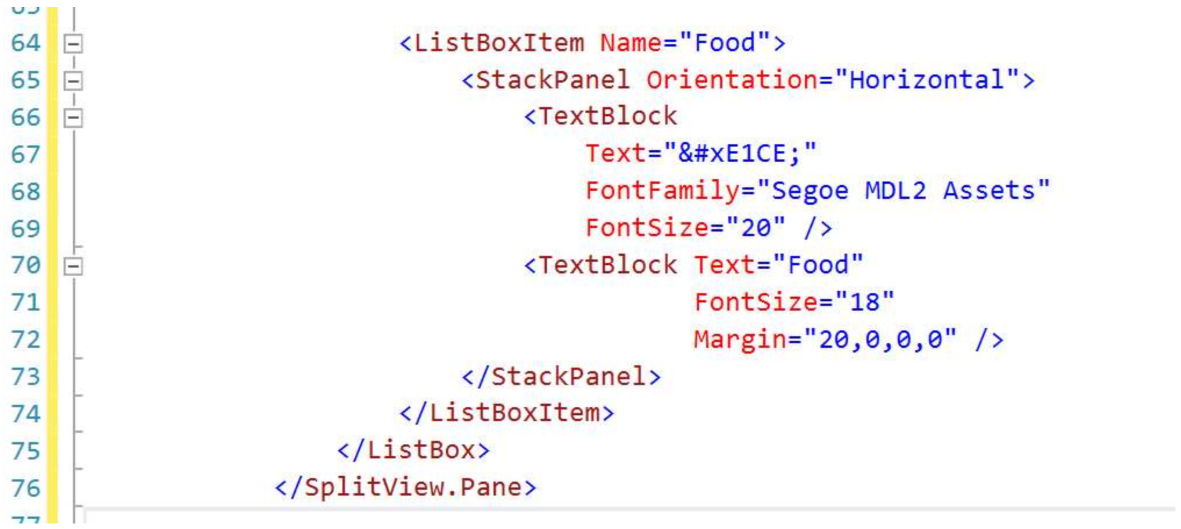
Begin by creating the first ListBoxItem, then copy, paste and edit the pertinent values for the second ListBoxItem.

Name the first ListBoxItem “Financial”.  Add a StackPanel oriented horizontally.  Inside the StackPanel, add two TextBlocks; one for the icon and one for the text.  You can use any symbol from the Segoe MDL2 Assets font family for the first TextBlock.  The Text of the second TextBlock should be set to “Financial”.  Set a small margin to the left to give it room to breathe between the two TextBlocks.



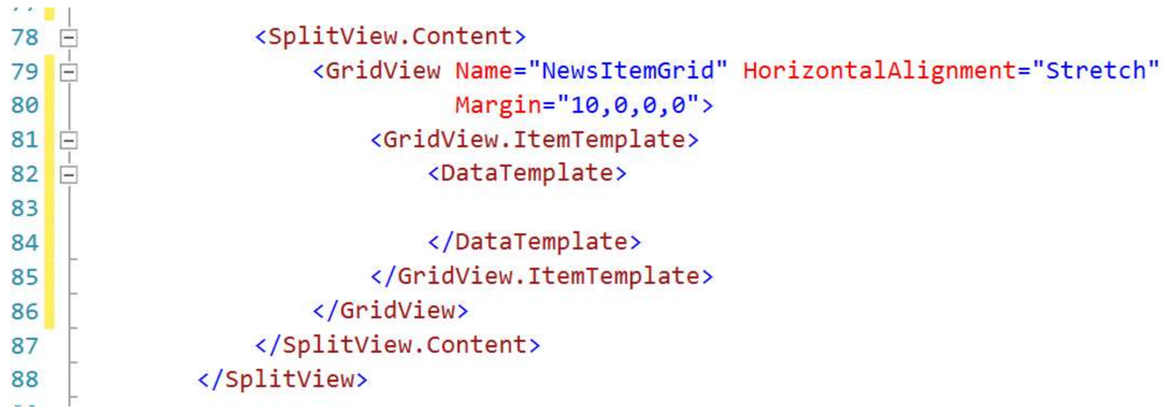
```
            <SplitView.Pane>
                <ListBox SelectionMode="Single"
                         SelectionChanged="ListBox_SelectionChanged">
                    <ListBoxItem Name="Financial">
                        <StackPanel Orientation="Horizontal">
                            <TextBlock
                                Text="&#xE80F;"
                                FontFamily="Segoe MDL2 Assets"
                                FontSize="20" />
                            <TextBlock Text="Financial"
                                       FontSize="18"
                                       Margin="20,0,0,0" />
                        </StackPanel>
                    </ListBoxItem>
```

Copy and paste the first ListBoxItem, change the ListBoxItem’s name (“Food”), the symbol in the first TextBlock and the Text property of the second TextBlock.



```
                    <ListBoxItem Name="Food">
                        <StackPanel Orientation="Horizontal">
                            <TextBlock
                                Text="&#xE1CE;"
                                FontFamily="Segoe MDL2 Assets"
                                FontSize="20" />
                            <TextBlock Text="Food"
                                       FontSize="18"
                                       Margin="20,0,0,0" />
                        </StackPanel>
                    </ListBoxItem>
                </ListBox>
            </SplitView.Pane>
```

Step 8: Add a GridView to the SplitView.Content area.  Name the GridView "NewsItemGrid", to make it take up the entire available area set the HorizontalAlignment to "stretch".  Go ahead and add the markup for the ItemTemplate / DataTemplate.

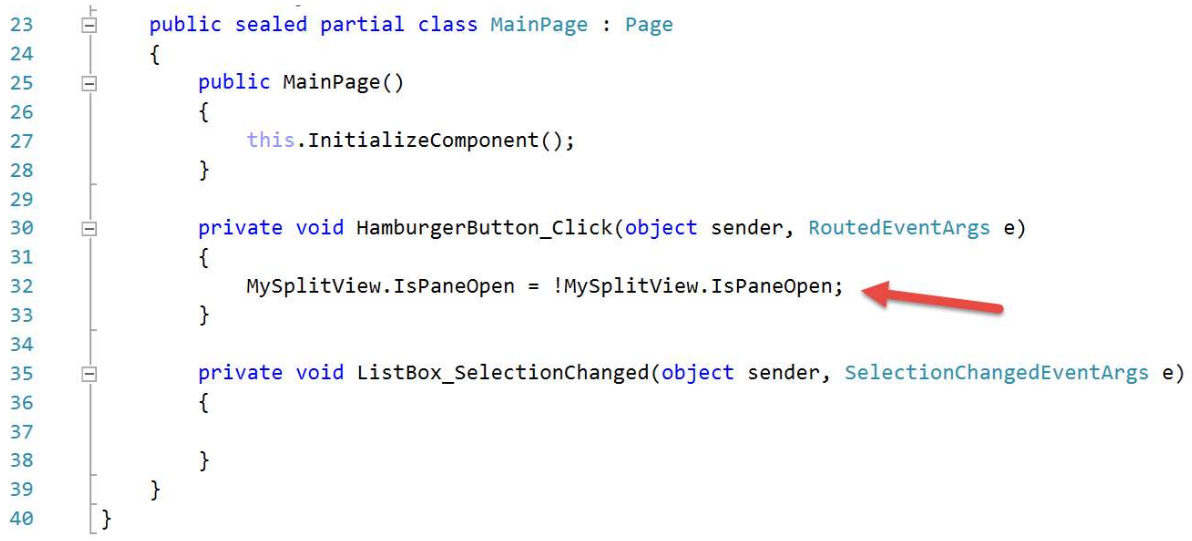


```
            <SplitView.Content>
                <GridView Name="NewsItemGrid" HorizontalAlignment="Stretch"
                          Margin="10,0,0,0">
                    <GridView.ItemTemplate>
                        <DataTemplate>

                        </DataTemplate>
                    </GridView.ItemTemplate>
                </GridView>
            </SplitView.Content>
        </SplitView>
```

And I know that create two areas, a top area that will contain the SearchBox and the Title of the current Page that we're on and then in the second Row of the Grid we will add a SplitView with a pane and a content area.

Step 9: Create the event handler method stubs for the HamburgerButton_Click and the ListBox_SelectionChanged using the F12 keyboard technique.

Step 10: Add the code to open / close the SplitView when the user clicks the HamburgerButton.

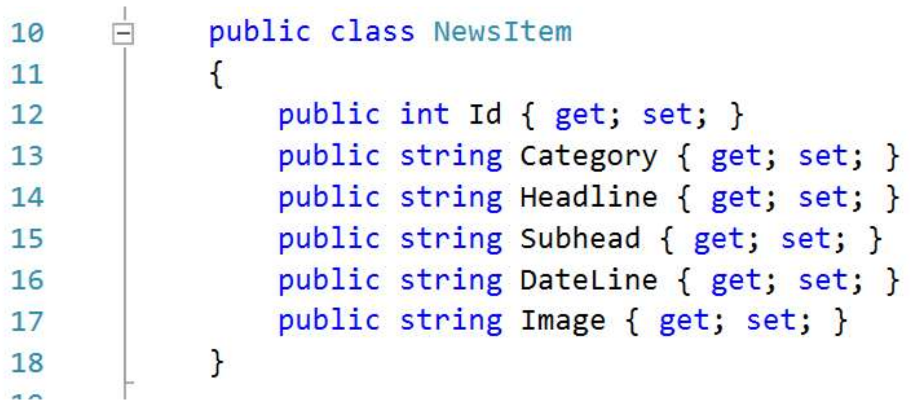


```
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }

        private void HamburgerButton_Click(object sender, RoutedEventArgs e)
        {
            MySplitView.IsPaneOpen = !MySplitView.IsPaneOpen;
        }

        private void ListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {

        }
    }
}
```

## UWP-046 - Adeptly Adaptive Challenge Solution - Part 2: Creating the Data Model and Data Binding

Step 1: In the Solution Explorer add a Model folder. Inside of the Model folder add a new class file named NewsItem.cs.

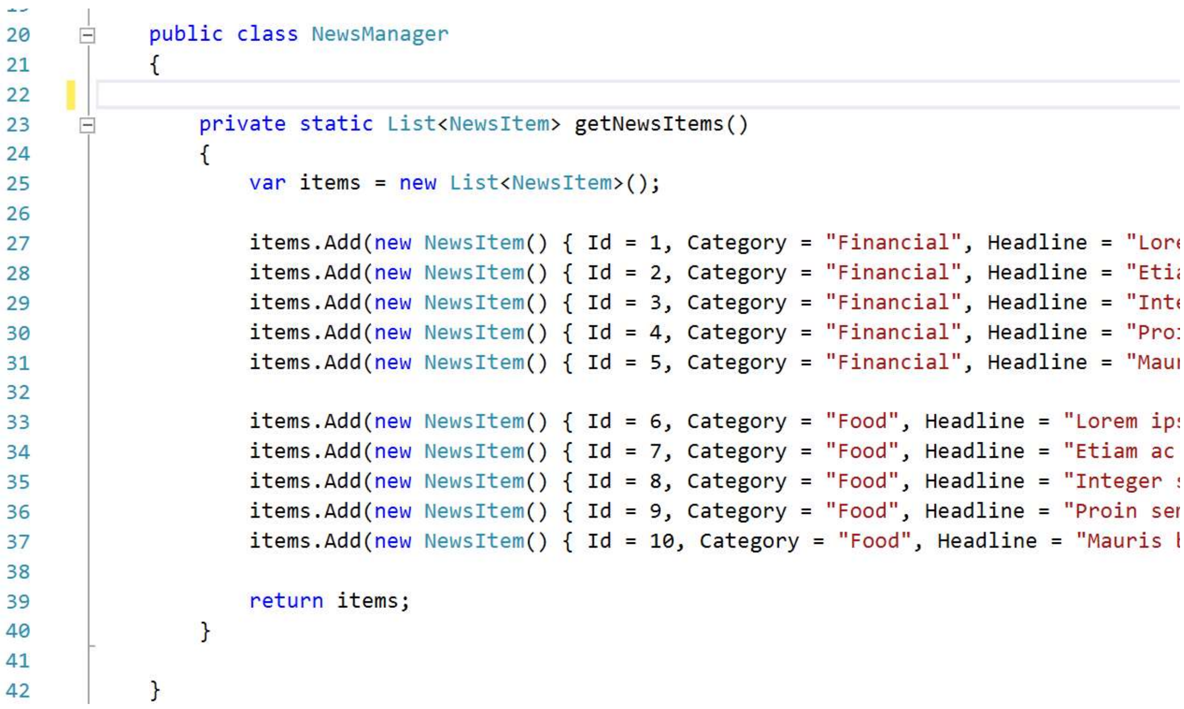
Step 2: Inside of the NewsItem.cs, add a NewsItem POCO containing the properties needed to display all the information about a given fake news story. Extrapolate the property names from the instructions txt file in the zipped resource folder. Look at the bottom of that instructions file for the getNewsItems() method.



```
public class NewsItem
{
    public int Id { get; set; }
    public string Category { get; set; }
    public string Headline { get; set; }
    public string Subhead { get; set; }
    public string DateLine { get; set; }
    public string Image { get; set; }
}
```

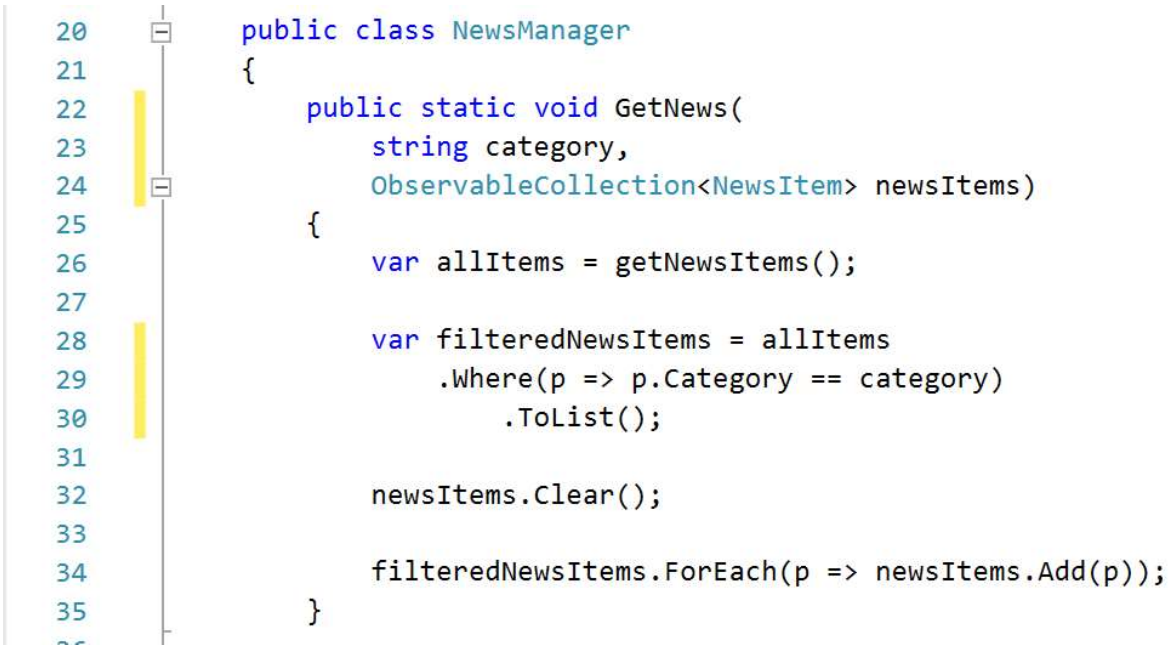
Step 3: In the same NewsItem.cs, add another class definition called NewsManager. This will be delegated the responsibility of populating an observable collection of NewsItems, filtering the news items by the Category property.

Copy the getNewsItems() helper method from the instructions file into the NewsManager class.



```
public class NewsManager
{

    private static List<NewsItem> getNewsItems()
    {
        var items = new List<NewsItem>();

        items.Add(new NewsItem() { Id = 1, Category = "Financial", Headline = "Lor
        items.Add(new NewsItem() { Id = 2, Category = "Financial", Headline = "Eti
        items.Add(new NewsItem() { Id = 3, Category = "Financial", Headline = "Int
        items.Add(new NewsItem() { Id = 4, Category = "Financial", Headline = "Pro
        items.Add(new NewsItem() { Id = 5, Category = "Financial", Headline = "Mau

        items.Add(new NewsItem() { Id = 6, Category = "Food", Headline = "Lorem ip
        items.Add(new NewsItem() { Id = 7, Category = "Food", Headline = "Etiam ac
        items.Add(new NewsItem() { Id = 8, Category = "Food", Headline = "Integer
        items.Add(new NewsItem() { Id = 9, Category = "Food", Headline = "Proin se
        items.Add(new NewsItem() { Id = 10, Category = "Food", Headline = "Mauris

        return items;
    }

}
```

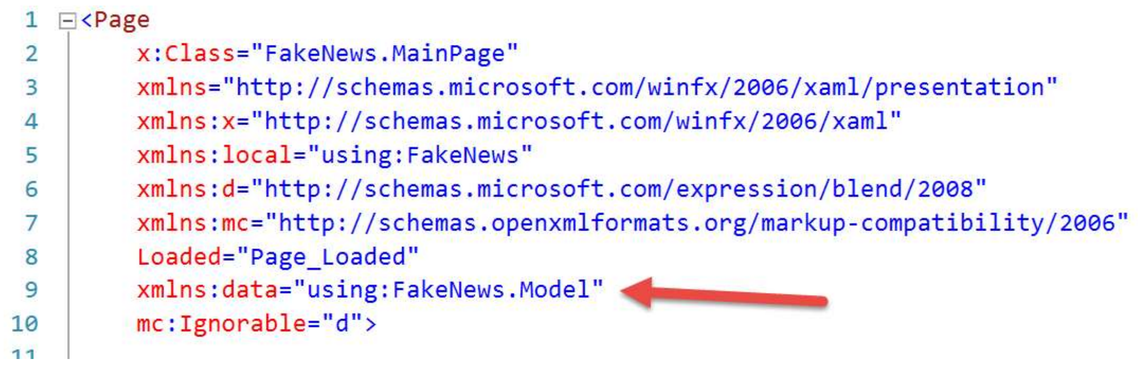
Step 4: Add a public static method called GetNews that will call the private helper method to get the list of news items, then filter that list based on the category. The method should accept the category to filter on as well as an instance of ObservableCollection<NewsItem>.



```
public class NewsManager
{
    public static void GetNews(
        string category,
        ObservableCollection<NewsItem> newsItems)
    {
        var allItems = getNewsItems();

        var filteredNewsItems = allItems
            .Where(p => p.Category == category)
                .ToList();

        newsItems.Clear();

        filteredNewsItems.ForEach(p => newsItems.Add(p));
    }
```

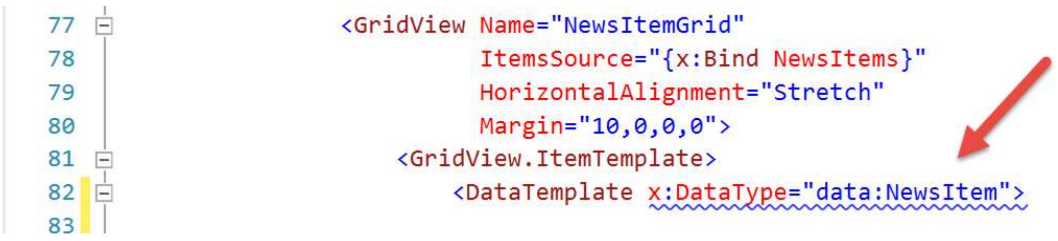
Note the two LINQ statements that greatly compress the amount of code necessary to filter the list of news items by category, then add each of those NewsItem instances to the ObservableCollection<NewsItem>.

Step 5: In the MainPage.xaml, we'll prepare for data binding to the ObservableList<NewsItem> in the GridView. First, add the Model namespace for the project to the Page's definition to make classes from this namespace available to our XAML.



```
<Page
    x:Class="FakeNews.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:FakeNews"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    Loaded="Page_Loaded"
    xmlns:data="using:FakeNews.Model"
    mc:Ignorable="d">
```

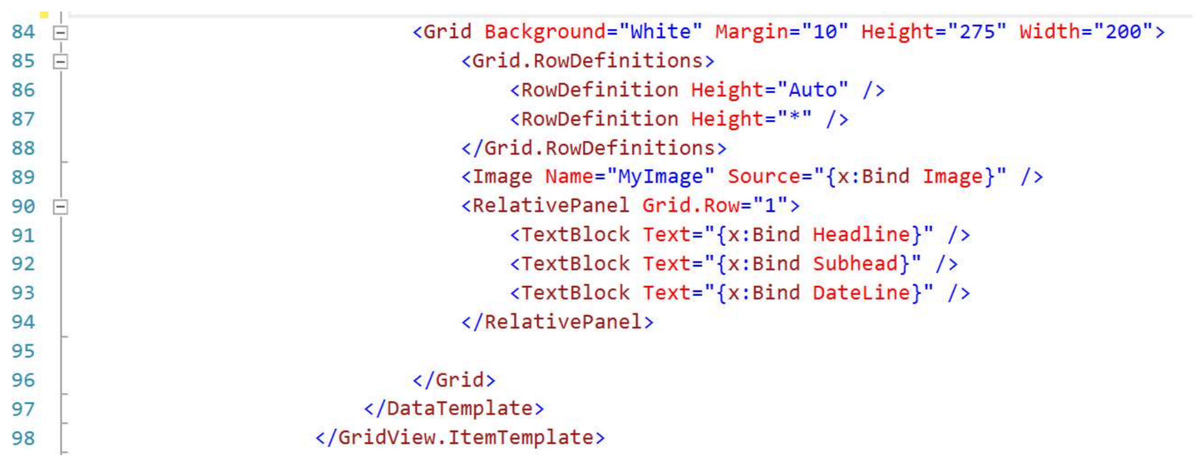
Next, in the GridView.ItemTemplate's DataTemplate, add the DataType that we'll bind to:



```
                <GridView Name="NewsItemGrid"
                          ItemsSource="{x:Bind NewsItems}"
                          HorizontalAlignment="Stretch"
                          Margin="10,0,0,0">
                    <GridView.ItemTemplate>
                        <DataTemplate x:DataType="data:NewsItem">

```

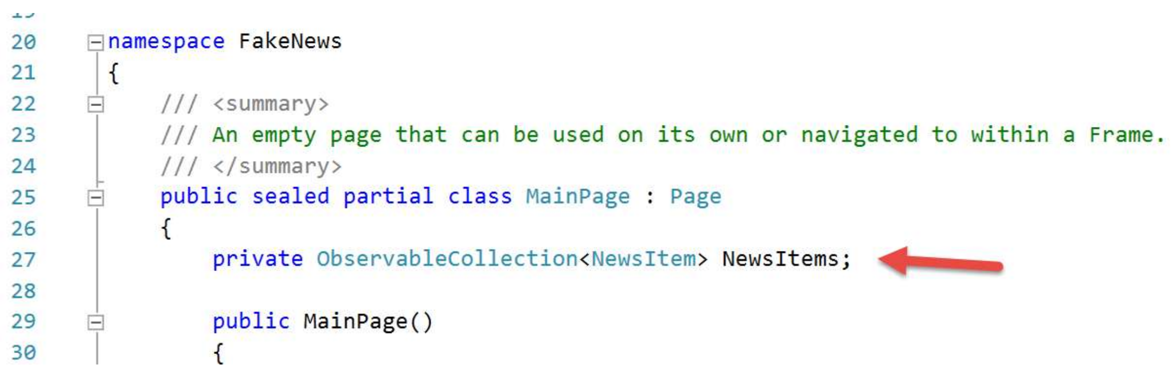
In this lesson, we'll create a temporary solution for displaying each NewsItem just so we can confirm that the data binding is working. Later, we'll remove this implementation and replace it with the UserControl that will allow us to adaptively resize each NewsItem "card" based on the size of the Window.

Add a Grid with two rows. Set its height to 275 and width to 200. Add a margin of 10 pixels around it and set its background to white. The grid should have two row definitions. In the top row add an Image control that binds to the Image property of the NewsItem class. In the second row add a RelativePanel with three TextBlocks … the TextBlocks should bind to the NewsItem's Headline, Subhead and DateLine properties, respectively.



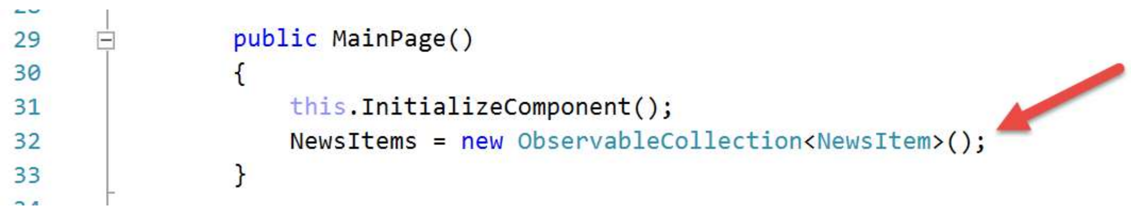
```
                                <Grid Background="White" Margin="10" Height="275" Width="200">
                                    <Grid.RowDefinitions>
                                        <RowDefinition Height="Auto" />
                                        <RowDefinition Height="*" />
                                    </Grid.RowDefinitions>
                                    <Image Name="MyImage" Source="{x:Bind Image}" />
                                    <RelativePanel Grid.Row="1">
                                        <TextBlock Text="{x:Bind Headline}" />
                                        <TextBlock Text="{x:Bind Subhead}" />
                                        <TextBlock Text="{x:Bind DateLine}" />
                                    </RelativePanel>

                                </Grid>
                            </DataTemplate>
                        </GridView.ItemTemplate>
```

Step 6: Wire the NewsManager to the MainPage.xaml inside of the MainPage.xaml.cs. First, create a private field to store an instance of the ObservableCollection<NewsItem> we will retrieve from the NewsManager. The data binding we set up in the previous step will look for this private field to bind to:



```
namespace FakeNews
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        private ObservableCollection<NewsItem> NewsItems;

        public MainPage()
        {
```

When the MainPage class is initialized, create a new instance of ObservableCollection<NewsItem>:



```
        public MainPage()
        {
            this.InitializeComponent();
            NewsItems = new ObservableCollection<NewsItem>();
        }
```

Based on the selection of the ListBox, whether the Food or Financial ListBoxItems, retrieve the filtered ObservableCollection<NewsItem> from the NewsManager for the given "category" that the user selected.

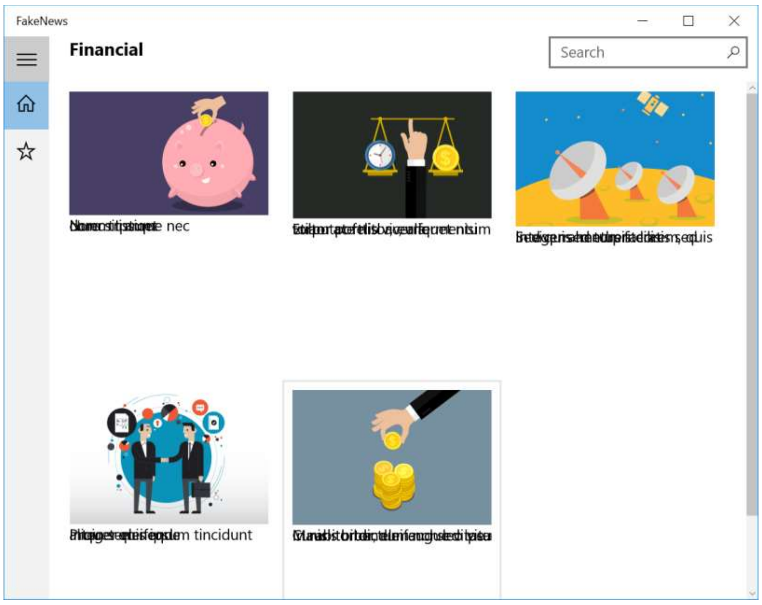
```
        private void ListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            if (Financial.IsSelected)
            {
                NewsManager.GetNews("Financial", NewsItems);
                TitleTextBlock.Text = "Financial";
            }
            else if (Food.IsSelected)
            {
                NewsManager.GetNews("Food", NewsItems);
                TitleTextBlock.Text = "Food";
            }
        }
```

Notice that I also changed the TitleTextBlock's Text property as well.

Finally, as the Page loads, select the Financial ListBoxItem to begin.

```
        private void Page_Loaded(object sender, RoutedEventArgs e)
        {
            Financial.IsSelected = true;
        }
```

At this point, the app should run and bind to the news items. Even so, there are significant formatting issues that we'll address in the next lessons.



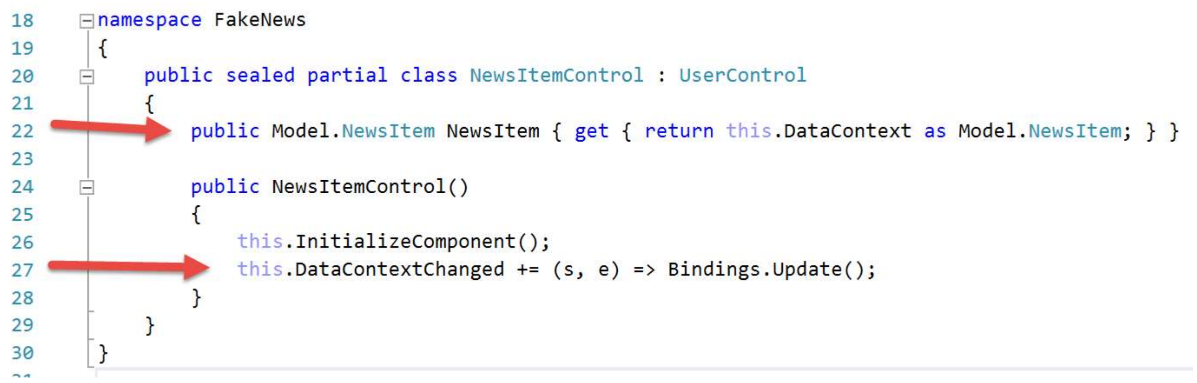
## UWP-047 - Adeptly Adaptive Challenge Solution - Part 3: Creating a User Control as the Data Template

In this lesson we'll create a UserControl as the DataTemplate so that each news item "card" can be resized adaptively.

Step 1: Add a new User Control to the project named "NewsItemControl.xaml". (Remember: Project menu > Add New Item … > select User Control in the middle).

Step 2: In the MainPage.xaml, cut out everything inside the opening and closing DataTemplate element and paste it into the NewsItemControl.

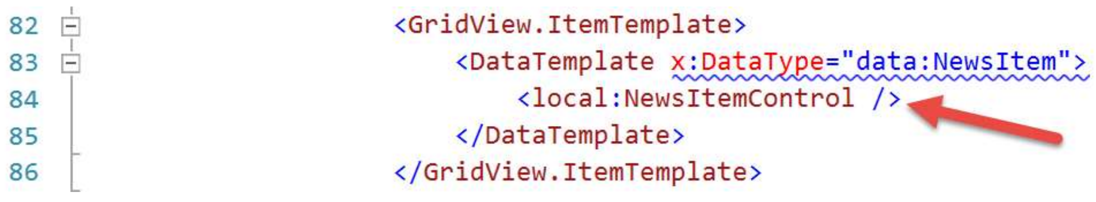
Step 3: In the NewsItemControl.xaml.cs code behind, add the necessary (templated) code to return the DataContext as an instance of Model.NewsItem, then add handle the DataContextChanged event by updating the bindings in the User Control.



```
namespace FakeNews
{
    public sealed partial class NewsItemControl : UserControl
    {
        public Model.NewsItem NewsItem { get { return this.DataContext as Model.NewsItem; } }

        public NewsItemControl()
        {
            this.InitializeComponent();
            this.DataContextChanged += (s, e) => Bindings.Update();
        }
    }
}
```

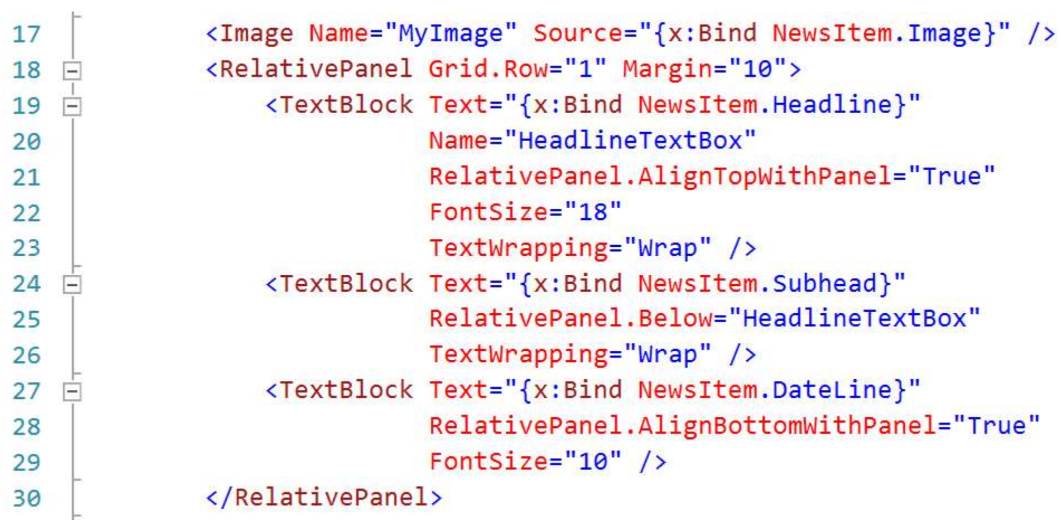
Remember: this code may be a bit complicated, but you can copy from the cheat sheet replacing the model's data type as necessary.

Step 4: Back in the MainPage.xaml, reference the NewsItemControl as the data template for the GridView.



```
<GridView.ItemTemplate>
    <DataTemplate x:DataType="data:NewsItem">
        <local:NewsItemControl />
    </DataTemplate>
</GridView.ItemTemplate>
```

Step 5: In the NewsItemControl.xaml, update the three TextBlocks in the RelativePanel so that they display properly. Align the first to the top, the second below the first, and align the third one to the bottom. Set the font sizes appropriately and use TextWrapping to ensure all the text is displayed.



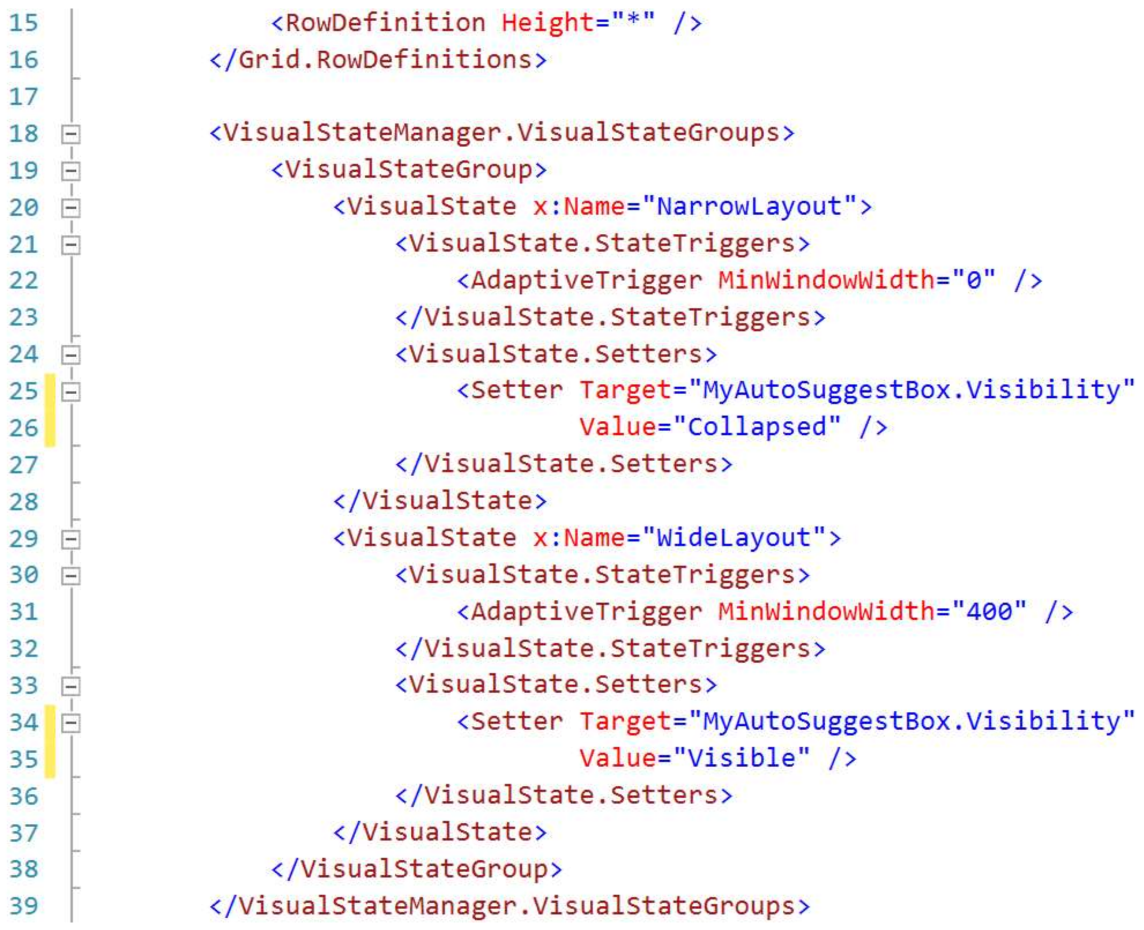
```xml
        <Image Name="MyImage" Source="{x:Bind NewsItem.Image}" />
        <RelativePanel Grid.Row="1" Margin="10">
            <TextBlock Text="{x:Bind NewsItem.Headline}"
                       Name="HeadlineTextBox"
                       RelativePanel.AlignTopWithPanel="True"
                       FontSize="18"
                       TextWrapping="Wrap" />
            <TextBlock Text="{x:Bind NewsItem.Subhead}"
                       RelativePanel.Below="HeadlineTextBox"
                       TextWrapping="Wrap" />
            <TextBlock Text="{x:Bind NewsItem.DateLine}"
                       RelativePanel.AlignBottomWithPanel="True"
                       FontSize="10" />
        </RelativePanel>
```

In this lesson we added no new functionality but merely prepared the project for adaptive resizing.

## UWP-048 - Adeptly Adaptive Challenge Solution - Part 4: Adaptively Resizing

In this final solution lesson we will add adaptive resizing to both the MainPage.xaml and the NewsItemControl.xaml.

Step 1: Add two VisualStates in the MainPage.xaml. This should be added immediately after the row definitions. The first should be named “NarrowLayout” and should have a minimum window width of 0. The second should be named “WideLayout” and should have a minimum window width of 400. When the NarrowLayout is triggered, set the MyAutoSuggestBox’s Visibility to Collapsed. When the WideLayout is triggered, set the MyAutoSuggestBox’s Visibility to Visible.



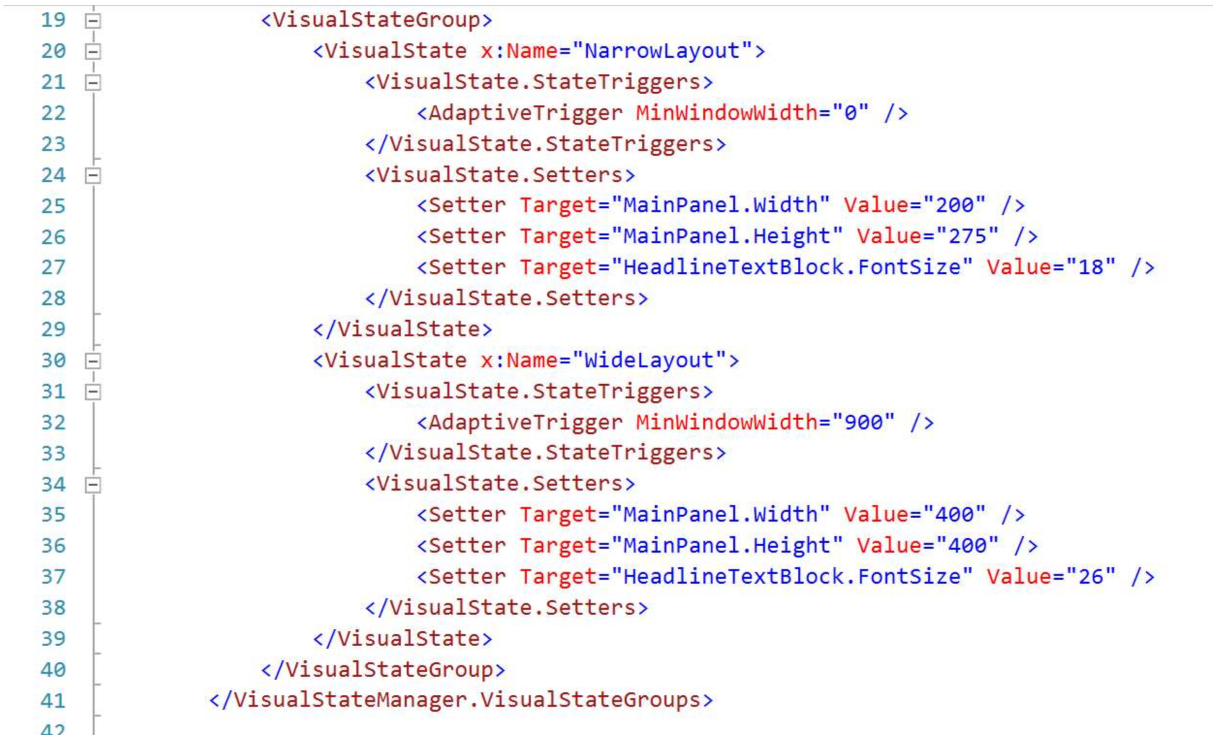
```
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup>
                <VisualState x:Name="NarrowLayout">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="0" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <Setter Target="MyAutoSuggestBox.Visibility"
                                Value="Collapsed" />
                    </VisualState.Setters>
                </VisualState>
                <VisualState x:Name="WideLayout">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="400" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <Setter Target="MyAutoSuggestBox.Visibility"
                                Value="Visible" />
                    </VisualState.Setters>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
```

Step 2: In NewsItemControl.xaml, give the Grid the name “MainPanel”. It will need a name since we’ll be resizing it adaptively.

Step 3: Add two VisualStates in the NewsItemControl.xaml beneath the Grid's row definitions. The first VisualState should be named "NarrowLayout" and should have a minimum window width of 0. The second should be named "WideLayout" and should have a minimum window width of 900.

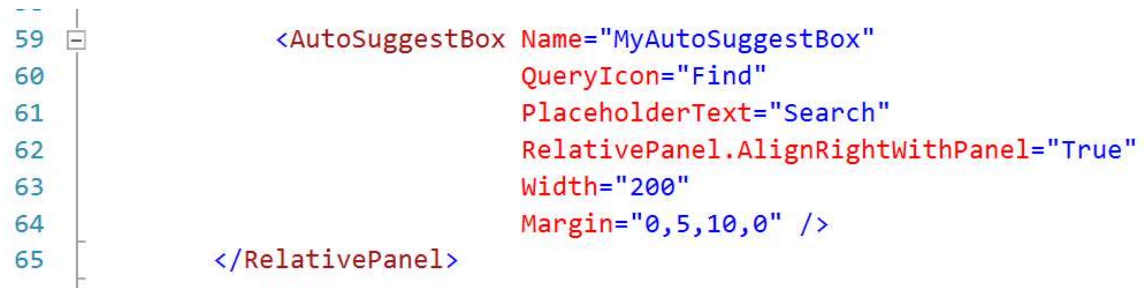
When the NarrowLayout is triggered, the MainPanel's width should be 200 and height should be 275. Furthermore, the HeadlineTextBlock's FontSize should be set to 18.

When the WideLayout is triggered, the MainPanel's width should be 400 and the height should be 400. Furthermore, the HeadlineTextBLock's FontSize should be set to 26.



```
            <VisualStateGroup>
                <VisualState x:Name="NarrowLayout">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="0" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <Setter Target="MainPanel.Width" Value="200" />
                        <Setter Target="MainPanel.Height" Value="275" />
                        <Setter Target="HeadlineTextBlock.FontSize" Value="18" />
                    </VisualState.Setters>
                </VisualState>
                <VisualState x:Name="WideLayout">
                    <VisualState.StateTriggers>
                        <AdaptiveTrigger MinWindowWidth="900" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <Setter Target="MainPanel.Width" Value="400" />
                        <Setter Target="MainPanel.Height" Value="400" />
                        <Setter Target="HeadlineTextBlock.FontSize" Value="26" />
                    </VisualState.Setters>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
```

Step 4: One final tweak to the MainPage.xaml would be to add a 5 pixel margin to the top of the AutoSuggestBox to push it down from the top a tiny bit.



```
            <AutoSuggestBox Name="MyAutoSuggestBox"
                            QueryIcon="Find"
                            PlaceholderText="Search"
                            RelativePanel.AlignRightWithPanel="True"
                            Width="200"
                            Margin="0,5,10,0" />
        </RelativePanel>
```