

Hand­s-on lab

Lab 4: Navigation and State

May 2015

Contents

[Overview 3](#_Toc420069875)

[Exercise 1: Implementing Navigation 4](#_Toc420069876)

[Task 1 – Opening the starter project 4](#_Toc420069877)

[Task 2 – Add a second page to the project 6](#_Toc420069878)

[Task 3 – Add a back button 13](#_Toc420069879)

[Exercise 2: Saving State when Suspending 16](#_Toc420069880)

[Task 1 – Opening the starter project 16](#_Toc420069881)

[Task 2 – Save state when suspending 17](#_Toc420069882)

[Summary 19](#_Toc420069883)

Overview

In this lab we will cover the basics of navigation and how to implement it using the capabilities that Template 10 provides. We will explore how we can pass data between pages of the application as we navigate and also how we can save data as we navigate to ensure that if our application is suspended, we can resume the application where the user was last active in the app, and restore the data they had entered. More information concerning application lifecycle such as suspend, resume and termination can be found here : <https://msdn.microsoft.com/en-us/library/windows/apps/hh464925.aspx>.

# Objectives

* 1. This lab will show you how to:
  + Navigate between pages
  + Pass parameters
  + Handle back navigation
  + Save state

# System requirements

* 1. You must have the following to complete this lab:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015

# Setup

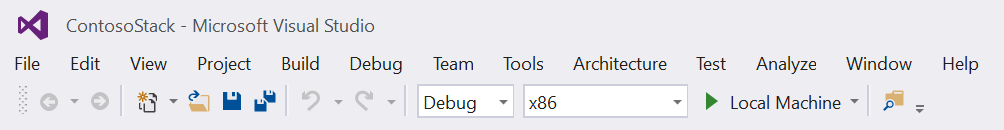
* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 10.
  3. Install Microsoft Visual Studio 2015.
  4. Install the Universal Windows Platform SDK.

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Implementing Navigation
  3. Saving State when Suspending
  4. Estimated time to complete this lab:  **45 to 60 minutes**.

Exercise 1: Implementing Navigation

1. In this exercise, you will implement simple navigation. You will add a page to the application, learn how to navigate to the page, pass in parameters and access those parameters. You will also implement back navigation.
2. Task 1 – Opening the starter project
   1. The first task is to open the starter project we have created for you.
3. Open the file location where you installed the hands-on labs. Navigate to the **\Lab 4\Solution\SimpleNav** folder and open **SimpleNav.sln** in Visual Studio 2015.
4. To prepare to build and run your app, use the Solution Configurations dropdown to choose the Debug configuration and use the Solution Platforms dropdown to target x86 (Figure 8). To run the app on the local machine, select Local Machine from the drop-down list next to the Start Debugging button on the debugger Standard toolbar.
   1. Note:  is the Start Debugging button.



* + 1. Figure 1
    2. Select options for building and running the app.

1. Build and run the app.
2. Return to Visual Studio and stop debugging.
3. Select the Local Machine dropdown and change the value to one of the device emulators – we chose **Emulator 10.0.1.0 720p 5 Inch 1GB.**

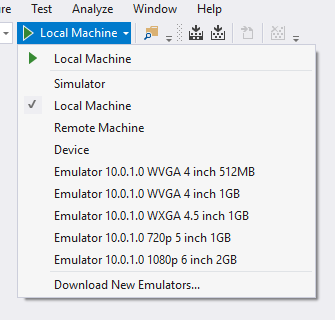
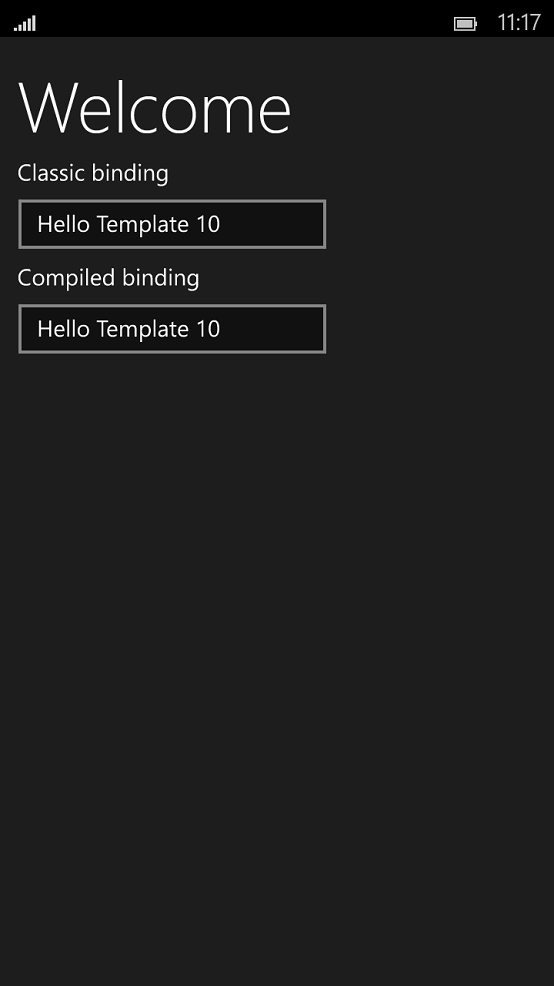


Figure 2

The device selection drop down

* 1. Note: If you do not see a list of emulators, you will need to install the emulators first before you can perform any emulator-based steps. **It is not essential to install the emulators to complete the lab – just skip those steps.**

1. Build and run the app and you will see the same code running on a phone emulator:
   * 1. 
     2. **Figure 3**
     3. The app running in an emulator
2. Close the running app and return to Visual Studio 2015. Select the Local Machine dropdown and return the value to **Local Machine**.

Task 2 – Add a second page to the project

* 1. This task will show how to add a new page to your application and create a view model.

1. Expand the project in **Solution Explorer** and expand the **Views** folder. Right-click on the **Views** folder and select **Add > New Item**. Select the **Blank Page** item and name it **Page2.xaml**.

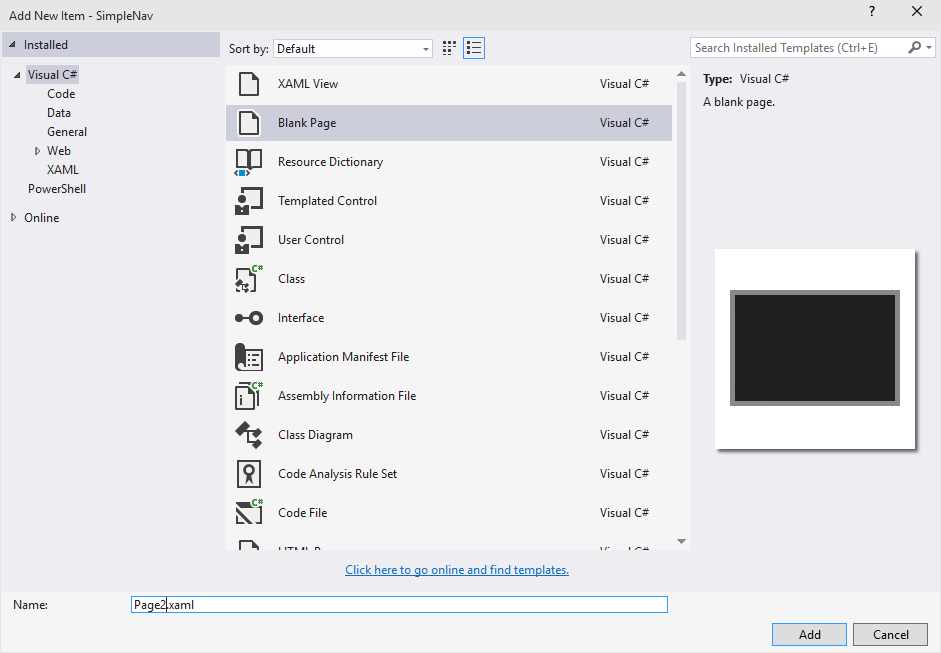
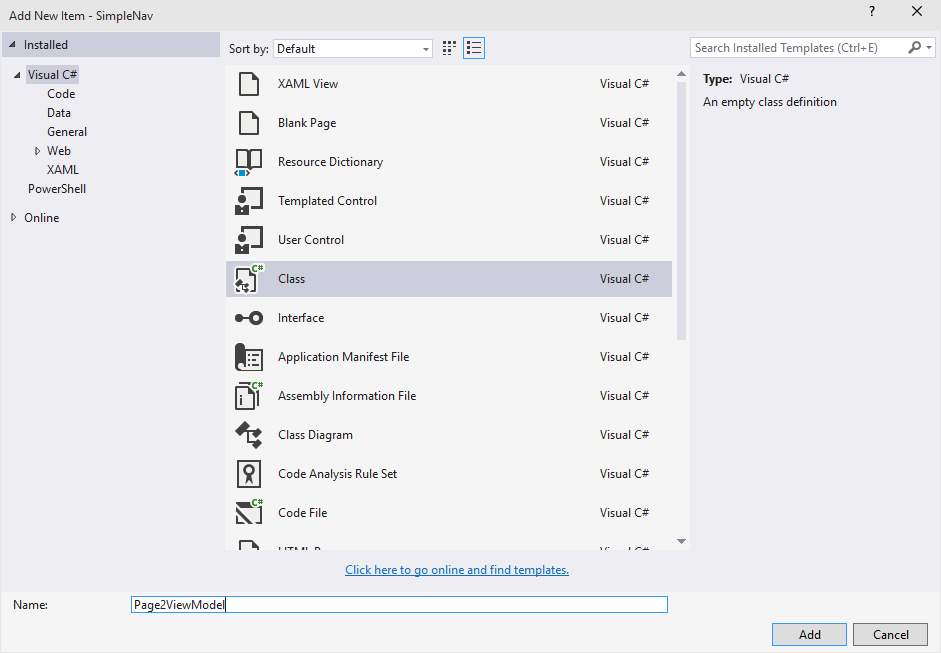


Figure 4

Add a new page

1. Next, you will add the View Model for the page. Expand the **ViewModels** folder, **Add > New Item** and select **Class**. Name it **Page2ViewModel**.
   * 1. 
     2. **Figure 5**
     3. Add a view model for the new page
2. Now we implement the view model by inheriting from **ViewModelBase:**

C#

* + 1. using SimpleNav.Mvvm;
    2. using Windows.UI.Xaml.Navigation;
    3. using System;
    4. using System.Collections.Generic;
    5. using System.Linq;
    6. using System.Text;
    7. using System.Threading.Tasks;
    8. namespace SimpleNav.ViewModels
    9. {
    10. class Page2ViewModel : ViewModelBase
    11. {
    12. }
    13. }

1. Next, you will add and initialize a property. This property will be used to hold the value supplied as a parameter during navigation. Make sure to make the class public.

C#

* + 1. using SimpleNav.Mvvm;
    2. using System;
    3. using System.Collections.Generic;
    4. using System.Linq;
    5. using System.Text;
    6. using System.Threading.Tasks;
    7. namespace SimpleNav.ViewModels
    8. {
    9. public class Page2ViewModel : ViewModelBase
    10. {
    11. public Page2ViewModel()
    12. {
    13. this.MyParameter = "Set from constructor";
    14. }
    15. private string \_myParameter = default(string);
    16. public string MyParameter { get { return \_myParameter; } set { Set(ref \_myParameter, value); } }
    17. }
    18. }

1. You now need to add code that will respond to the **OnNavigatedTo** event and assign the supplied parameter value to your property. Note the use of the C# ? and ?? operators to deal with null value scenarios.

C#

* + 1. using SimpleNav.Mvvm;
    2. using System;
    3. using System.Collections.Generic;
    4. using System.Linq;
    5. using System.Text;
    6. using System.Threading.Tasks;
    7. namespace SimpleNav.ViewModels
    8. {
    9. public class Page2ViewModel : ViewModelBase
    10. {
    11. public Page2ViewModel()
    12. {
    13. this.MyParameter = "Set from constructor";
    14. }
    15. public override void OnNavigatedTo(string parameter, NavigationMode mode, IDictionary<string, object> state)
    16. {
    17. this.MyParameter = parameter?.ToString() ?? "Empty";
    18. }
    19. private string \_myParameter;
    20. public string MyParameter { get { return \_myParameter; } set { Set(ref \_myParameter, value); } }
    21. }
    22. }

1. Open **Page2.xaml**. You will now customize your Page2 view so that it leverages the Page2ViewModel. You will add the view model namespace and create an instance of Page2ViewModel and assign it to the Page DataContext. We will then replace the Grid control with the Hub controls and bind the **Header** property to the **MyParameter** property of our view model. In the design view, you should note that the Hub header is set to the value assigned in the constructor of our Page2ViewModel.

XAML

* + 1. <Page
    2. x:Class="SimpleNav.Views.Page2"
    3. xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    4. xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    5. xmlns:local="using:SimpleNav.Views"
    6. xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    7. xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    8. xmlns:vm="using:SimpleNav.ViewModels"
    9. mc:Ignorable="d">
    11. <Page.DataContext>
    12. <vm:Page2ViewModel />
    13. </Page.DataContext>
    14. <Hub Header="{Binding MyParameter}" Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    15. </Hub>
    16. </Page>

1. Now you will add navigation to the **MainPage** by adding a **Button** control to the existing content **StackPanel**. Open **MainPage.xaml** and update it as follows:

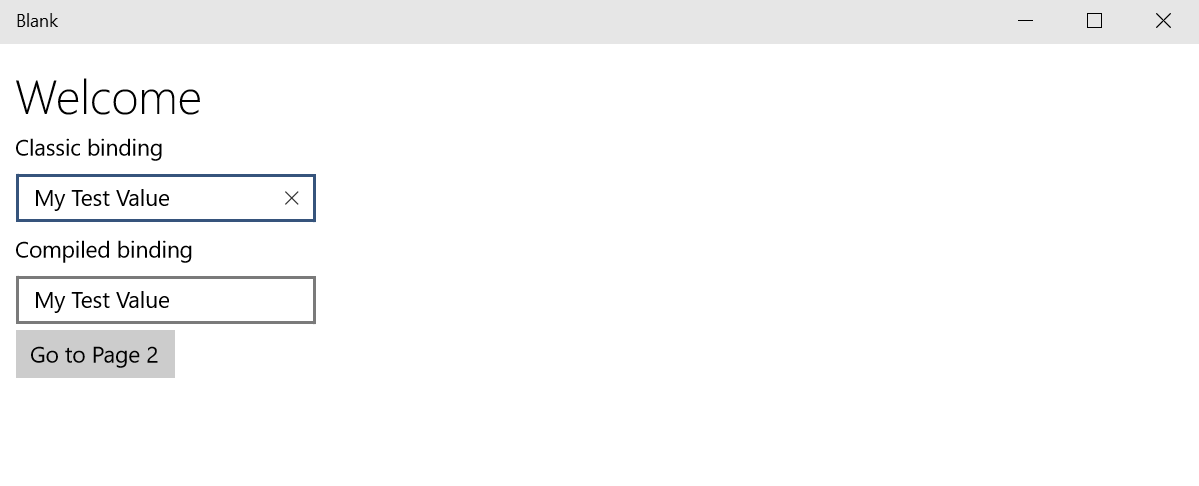
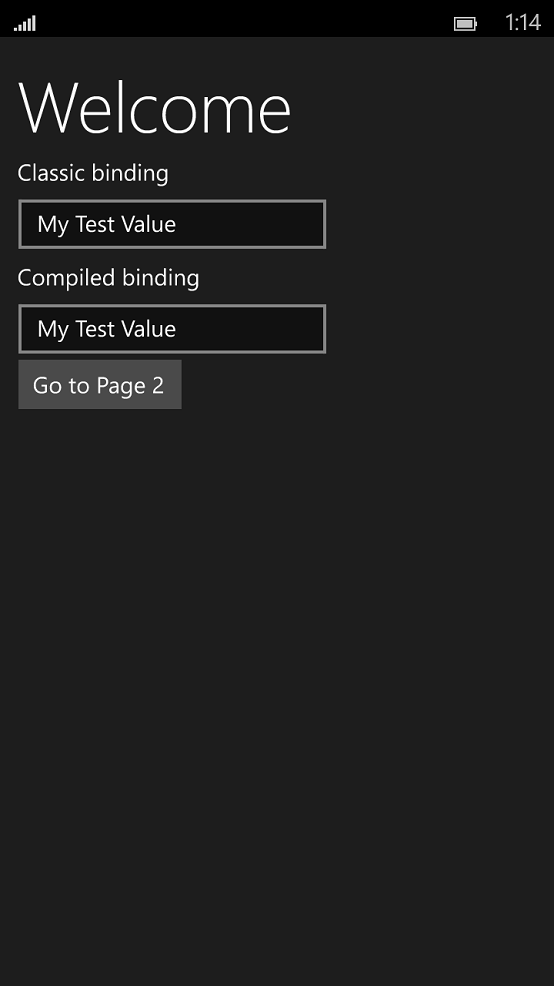
XAML

* + 1. <!-- content -->
    2. <StackPanel Margin="12" HorizontalAlignment="Left" VerticalAlignment="Top">
    3. <TextBlock Text="Welcome" Style="{StaticResource PageHeaderStyle}" />
    4. <TextBox Width="200" Margin="0,4" Header="Classic binding" Text="{Binding Value, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
    5. <TextBox Width="200" Margin="0,4" Header="Compiled binding" Text="{x:Bind ViewModel.Value, Mode=TwoWay}" />
    6. <Button Content="Go to Page 2" Click="GotoPage2" />
    7. </StackPanel>

1. We now need to implement the click handler in the code behind file. Hit F7 to open the code behind and add the following code to **MainPage.xaml.cs**. You will note that we are passing in the **ViewModel.Value** as the parameter.

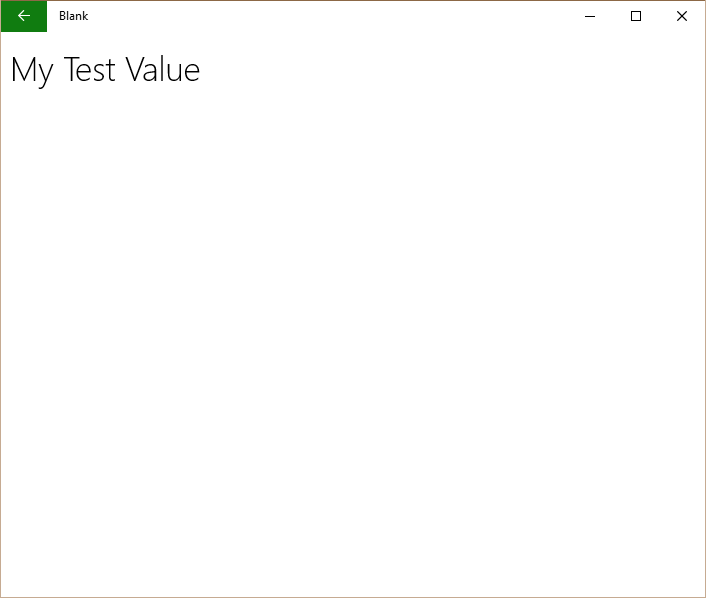
C#

* + 1. namespace SimpleNav.Views
    2. {
    3. public sealed partial class MainPage : Page
    4. {
    5. public MainPage()
    6. {
    7. InitializeComponent();
    8. this.DataContextChanged += (s, e) => ViewModel = DataContext as ViewModels.MainPageViewModel;
    9. }
    10. // Strongly-typed view models enable x:Bind
    11. public ViewModels.MainPageViewModel ViewModel { get; set; }
    12. private void GotoPage2(object sender, Windows.UI.Xaml.RoutedEventArgs e)
    13. {
    14. var app = App.Current as Common.BootStrapper;
    15. app.NavigationService.Navigate(typeof(Views.Page2), this.ViewModel.Value);
    16. }
    17. }
    18. }

1. Compile and run the application. Change the value in the **Classic binding** template to “My Test Value” and hit the **Go to Page 2** button. The application will then navigate to page 2 and the header will be set to “My Test Value”:
   * 1. 
     2. Figure 6
     3. Set the value and click.
     4. 
     5. Figure 7
     6. The Page 2 header reflects the value.
2. At this point we have implemented the ability to navigate to **Page2** but we have no UI control to go back in the desktop version of the app. However, if we press **ALT-LEFT** we will go back. Also, if we now run the app in a mobile emulator, we can use the hardware back button, and the app will automatically return to the main page. This is due to the fact that the **Bootstrapper** class handles the universal **BackRequested** event and the **NavigationService** leverages the **KeyBoardService** to add keyboard handling.
   * 1.  
     2. Figure 8
     3. The app running in a Mobile Emulator
3. Return to Visual Studio and stop debugging.

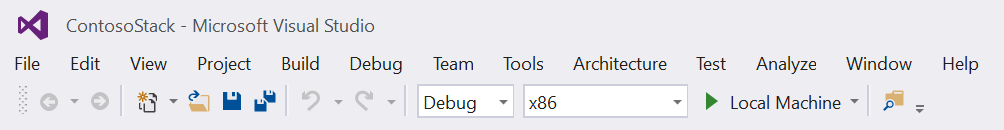
Task 3 – Add a back button

* 1. In this task, you will add a back navigation control to the desktop version of your app.
  2. **Note:** The platform provides support for displaying a System UI for Back navigation on the Desktop and Tablets. View the session **Navigation and Windowing in Universal Apps** - <http://channel9.msdn.com/Events/Build/2015/3-779> for more details.

1. You will implement the functionality for the shell back button in the code behind **Page2.xaml.cs**. This functionality will be implemented in two steps. First, add code to the **OnNavigatedTo** event handler to detect whether the Back Button should be displayed:
   * 1. **C#**
     2. protected override void OnNavigatedTo(NavigationEventArgs e)
     3. {
     4. var deviceFamily = default(string);
     5. var qualifiers = Windows.ApplicationModel.Resources.Core.ResourceContext.GetForCurrentView().QualifierValues;
     6. if (qualifiers.ContainsKey("DeviceFamily"))
     7. deviceFamily = qualifiers["DeviceFamily"];
     8. else
   1. deviceFamily = "";
   2. SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility = Frame.CanGoBack ? AppViewBackButtonVisibility.Visible : AppViewBackButtonVisibility.Collapsed;
   3. }
2. Second, add code to handle the click event and navigate back:
   * 1. **C#**
     2. private void GoBack(object sender, RoutedEventArgs e)
     3. {
     4. if (Frame.CanGoBack)
     5. {
     6. Frame.GoBack();
     7. }
     8. }
3. Now compile and run the project in the **Local Machine**. You will see the back button display on Page 2. If you then close the app and re-launch it in an emulator, the Back button will not display as we already have a back button built in (either hardware or software).
   * 1. 
     2. **Figure 9**
     3. The virtual back button displays in the desktop app view.
     4. 
     5. **Figure 10**
     6. The virtual back button does not display in the emulator view.
4. Return to Visual Studio and stop debugging.

Exercise 2: Saving State when Suspending

1. In this exercise we will use the state management capabilites of Template 10 to save application state such as the current navigaiton stack and values the user has entered into a UI in case our application is suspended and terminated.
2. Task 1 – Opening the starter project
   1. The first task is to open the starter project we have created for you.
3. Open the file location where you installed the hands-on labs. Navigate to the **\Lab 4\Solution\Suspend** folder and open **Suspend.sln** in Visual Studio 2015.
4. To prepare to build and run your app, use the Solution Configurations dropdown to choose the Debug configuration and use the Solution Platforms dropdown to target x86 (Figure 11). To run the app on the local machine, select Local Machine from the drop-down list next to the Start Debugging button on the debugger Standard toolbar.
   1. Note:  is the Start Debugging button.



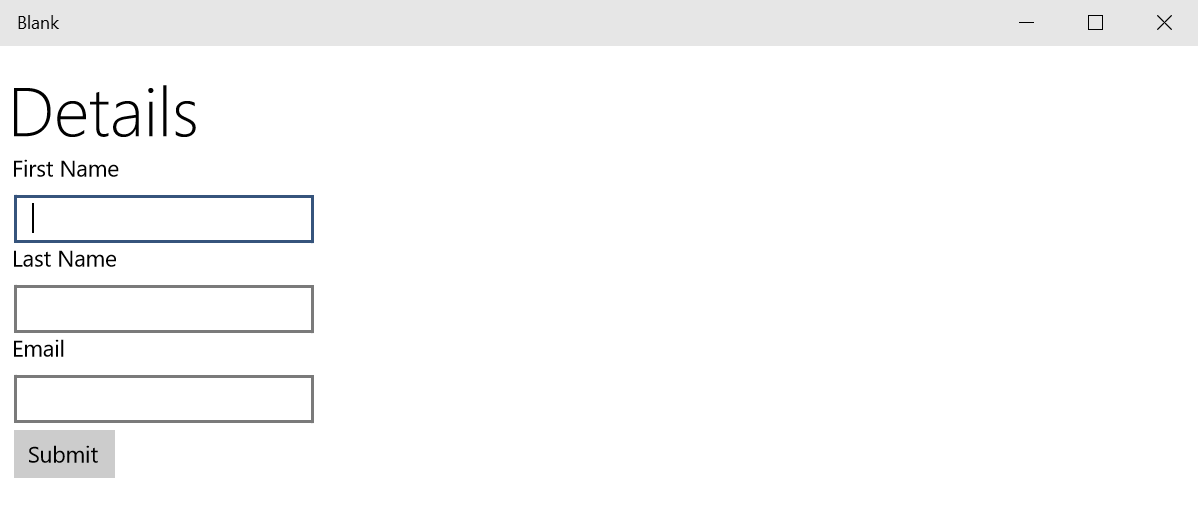
* + 1. Figure 11
    2. Select options for building and running the app.

1. Build and run the app.

**

**Figure 12**

*The main page of the starter app.*

**

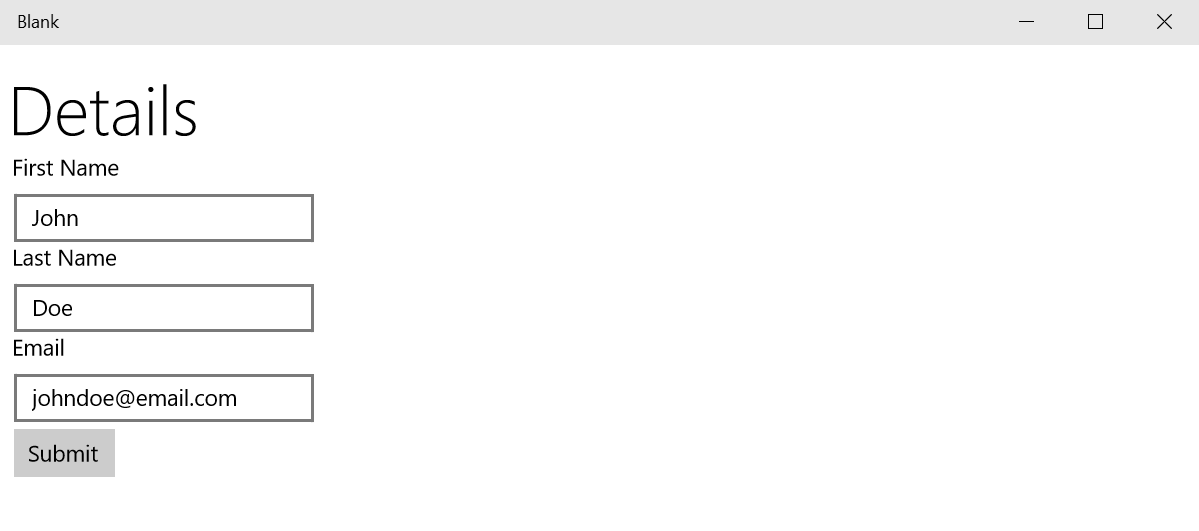
**Figure 13**

*The detail page of the starter app.*

1. Return to Visual Studio and stop debugging.
2. Task 2 – Save state when suspending

In this task, you will add overrides to navigation methods to save state when the app is suspended.

1. Open **DetailPageViewModel.cs**. in ViewModels Folder and add an **OnNavigatedFromAsync** task to save the values of the FirstName, LastName, and Email properties to a state dictionary on suspend.
   * 1. **C#**
   1. public class DetailPageViewModel : ViewModelBase
   2. {
   3. public override Task OnNavigatedFromAsync(IDictionary<string, object> state, bool suspending)
   4. {
   5. if (suspending)
   6. {
   7. state["FirstName"] = FirstName;
   8. state["LastName"] = LastName;
   9. state["Email"] = Email;
   10. }
   11. return base.OnNavigatedFromAsync(state, suspending);
   12. }
2. Add an override for the **OnNavigatedTo** method that will recover state and repopulate the model.
   * 1. **C#**
   1. public override void OnNavigatedTo(string parameter, NavigationMode mode, IDictionary<string, object> state)
   2. {
   3. try
   4. {
   5. FirstName = state["FirstName"]?.ToString();
   6. LastName = state["LastName"]?.ToString();
   7. Email = state["Email"]?.ToString();
   8. }
   9. finally
   10. {
   11. state.Clear();
   12. }
   13. }
   14. **Note:** You will see that we are using the **Null-condition operator** in the line:
   15. FirstName = state["FirstName"]?.ToString()
   16. This new C# 6 operator addresses many of the situations where code tends to drown in null-checking. It lets you access members and elements only when the receiver is not-null, providing a null result otherwise. So, in the above code, the ToString method is only invoked if the state dictionary contains an entry with a key equal to “FirstName”, otherwise FirstName is assigned a null value.
   17. For more information about the new capabilities in C# 6 check here: <http://blogs.msdn.com/b/csharpfaq/archive/2014/11/20/new-features-in-c-6.aspx>
3. Build and run the app. Navigate to the Details page and enter values for First Name, Last Name, and Email.

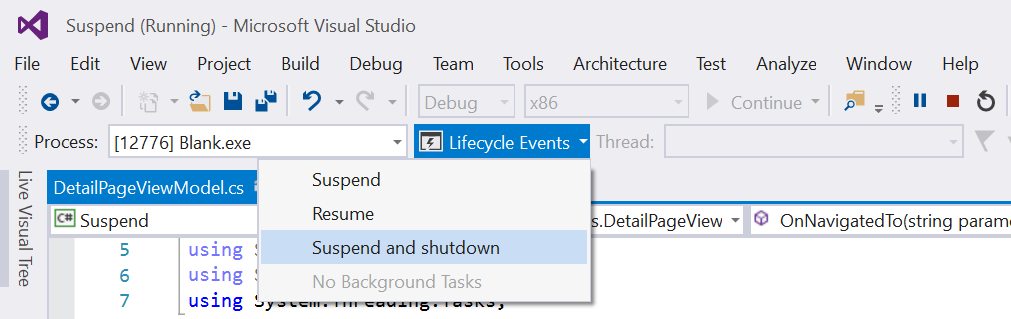
**

**Figure 14**

*Enter values on the Details page.*

1. While the app is still running, return to Visual Studio and expand the **Lifecycle Events** dropdown menu. Select **Suspend and shutdown**.

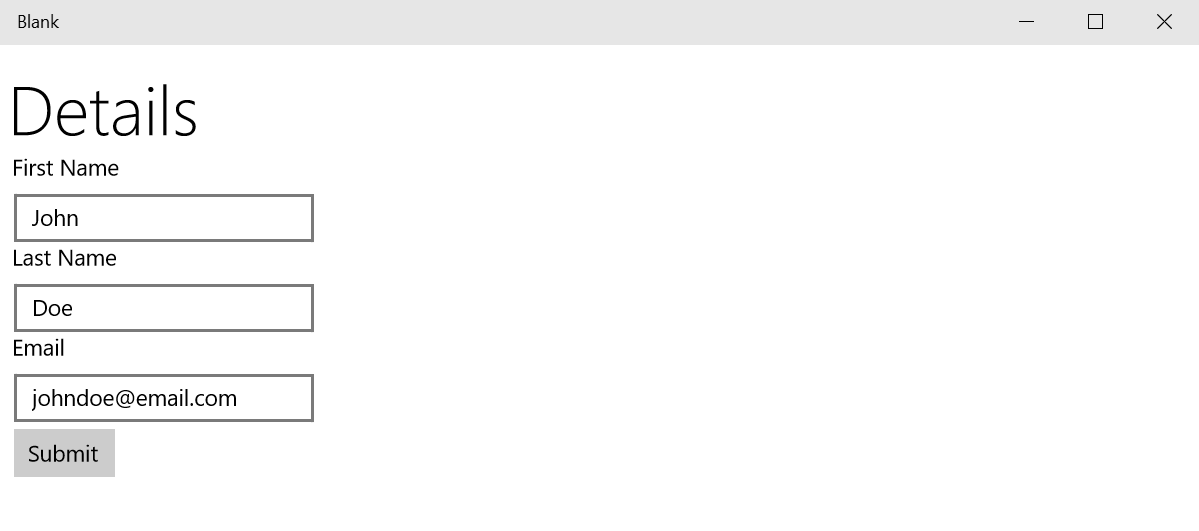
To add the toolbar (View > Toolbars> Debug Location)

**

**Figure 15**

*Suspend and shutdown the app.*

1. Once the app has suspended and shut down, the Start Debugging button will be available again. Start debugging to resume the app from its suspended state. When you view the Detail page, the fields for First Name, Last Name, and Email will repopulate with the fields you entered previously.

**

**Figure 16**

*The values repopulate.*

1. Stop debugging and return to Visual Studio.

Summary

* 1. In this lab we covered simple navigation and state management using Template 10.