Hands-On Lab

What’s New in ASP.NET MVC 4

Lab version: 1.2.0

Last updated: 3/9/2012

* + 1. 

Contents

[Overview 3](#_Toc319078306)

[Exercise 1: New MVC4 Project Templates 5](#_Toc319078307)

[Task 1 – Exploring the Internet Application Template 5](#_Toc319078308)

[Task 2 – Exploring the Mobile Application Template 12](#_Toc319078309)

[Task 3 – Using Adaptive Rendering 16](#_Toc319078310)

[Exercise 2: Creating the Photo Gallery Web Application 20](#_Toc319078311)

[Task 1 – Creating a Mock Photo Service 20](#_Toc319078312)

[Task 2 – Displaying the Photo Gallery 22](#_Toc319078313)

[Exercise 3: Adding support for mobile devices 24](#_Toc319078314)

[Task 1 – Installing jQuery Mobile in an ASP.NET MVC 4 Application 25](#_Toc319078315)

[Task 2 – Using Recipes for Code Generation 29](#_Toc319078316)

[Task 3 – jQuery Mobile Themes 35](#_Toc319078317)

[Task 4 – Using the View-Switcher Component and the Browser Overriding Features 37](#_Toc319078318)

[Task 5 – Adding the View-Switcher in the Desktop View 39](#_Toc319078319)

[Task 6 – Creating New Display Modes 42](#_Toc319078320)

[Exercise 4: Using Asynchronous Controllers 45](#_Toc319078321)

[Task 1 – Implementing an Asynchronous Controller 46](#_Toc319078322)

[Task 2 – Handling Time-Outs with Cancellation Tokens 48](#_Toc319078323)

[Appendix: Using Code Snippets 51](#_Toc319078324)

1. To give feedback please write to [VSKitFdbk@Microsoft.com](mailto:VSKitFdbk@Microsoft.com).
2. Copyright © 2012 by Microsoft Corporation. All rights reserved.

Overview

* 1. ASP.NET MVC 4 is a framework for building scalable, standards-based web applications using well-established design patterns and the power of the ASP.NET and the .NET framework. This new, fourth version of the framework focuses on making mobile web application development easier.
  2. To begin with, when you create a new MVC 4 project there is now a mobile application project template you can use to build a standalone app specifically for mobile devices. Additionally, MVC 4 integrates with jQuery Mobile through a jQuery.Mobile.MVC NuGet package. jQuery Mobile is an HTML5-based framework for developing web apps that are compatible with all popular mobile device platforms, including Windows Phone, iPhone, Android and so on. However, if you need specialization, MVC 4 also enables you to serve different views for different devices and provide device-specific optimizations.
  3. In this hands-on lab, you will start with the MVC 4 “Internet Application” project template to create a Photo Gallery application. You will progressively enhance the app using jQuery Mobile and MVC 4’s new features to make it compatible with different mobile devices and desktop web browsers. You will also learn about new code recipes for code generation and how MVC 4 makes it easier for you to write asynchronous action methods by supporting Task<ActionResult> return types.

# Objectives

* 1. In this hands-on lab, you will learn how to:
  + Take advantage of the enhancements to the ASP.NET MVC project templates ― including the new mobile application project template
  + Use the HTML5 viewport attribute and CSS media queries to improve the display on mobile devices
  + Use jQuery Mobile for progressive enhancements and for building touch-optimized web UI
  + Create mobile-specific views
  + Use the view-switcher component to toggle between mobile and desktop views in the application
  + Create asynchronous controllers using task support

# Prerequisites

* + [Microsoft Visual Studio 11 Ultimate Beta](http://go.microsoft.com/fwlink/?LinkId=240160)
  + ASP.NET MVC 4 (included in the Microsoft Visual Studio 11 Beta installation)
  + NuGet Visual Studio Extension (included in the Microsoft Visual Studio 11 Beta installation)
  + [Windows Phone 7 Emulator](http://go.microsoft.com/fwlink/?LinkId=226403)
  + Optional - [Electric Mobile Simulator](http://www.electricplum.com/dlsim.html) only for Exercise 3 used to browse the web application with an iPhone simulator
  1. **Note:**  By the time this lab was released, the required phone emulators were unsupported on Windows 8 Consumer Preview.

# Setup

* 1. Throughout the lab document, you will be instructed to insert code blocks. For your convenience, most of that code is provided as Visual Studio Code Snippets, which you can use from within Visual Studio to avoid having to add it manually.
  2. To install the code snippets:
  3. Open a Windows Explorer window and browse to the lab’s **Source\Setup** folder.
  4. Double-click the **Setup.cmd** file in this folder to install the Visual Studio code snippets.

If you are not familiar with the Visual Studio Code Snippets, and want to learn how to use them, you can refer to the appendix from this document “Appendix: Using Code Snippets”.

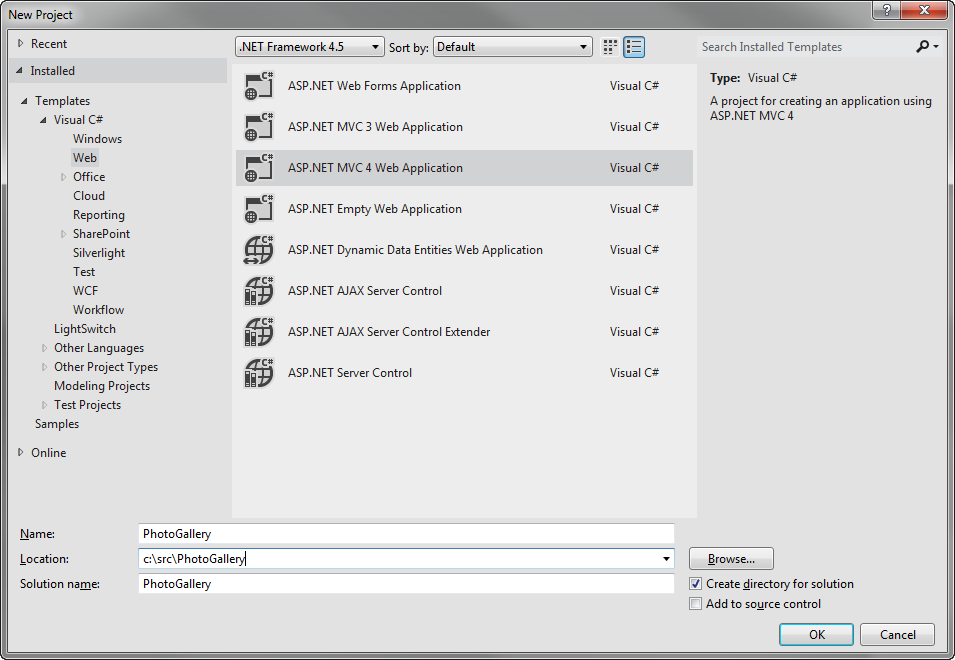
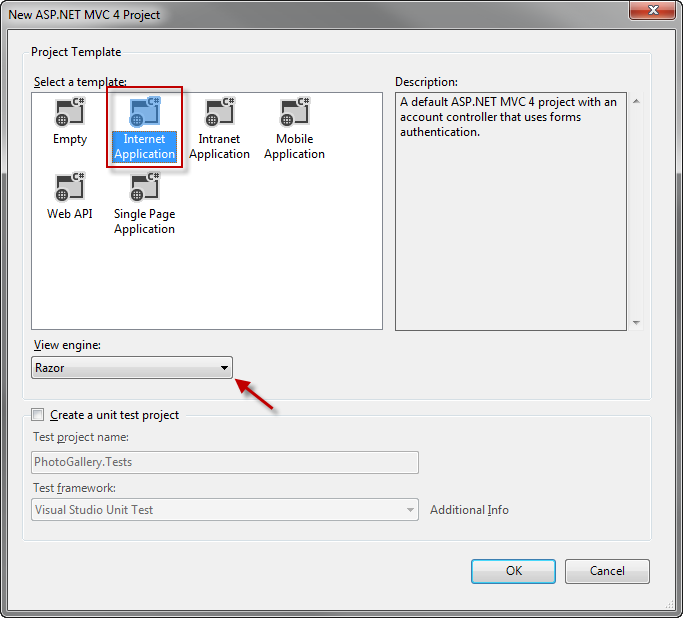
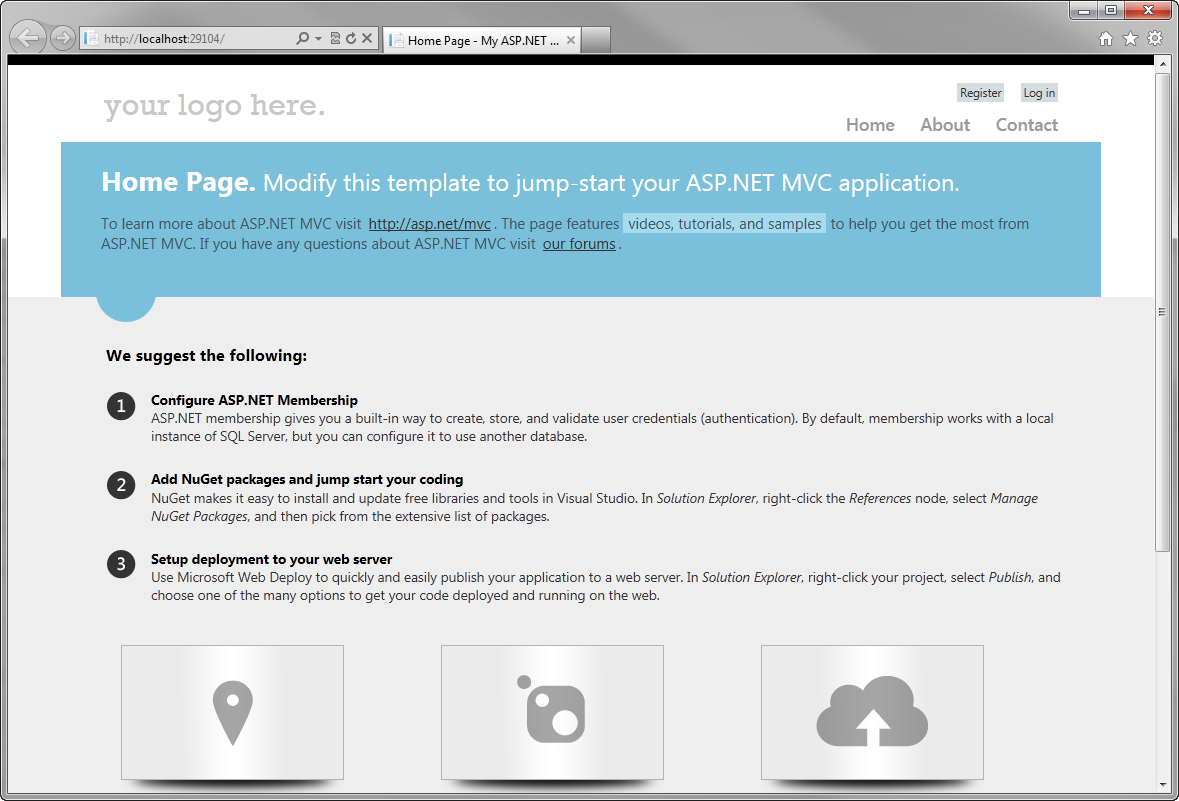
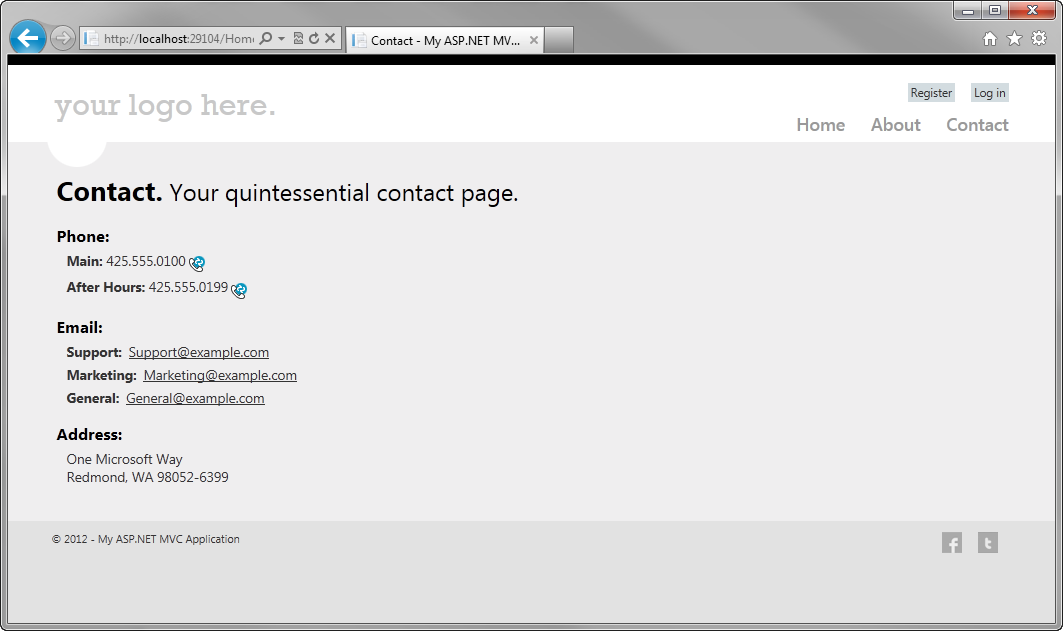
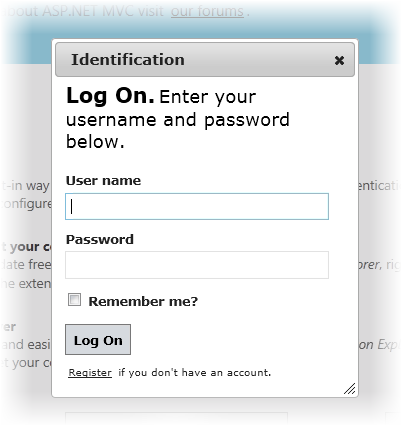
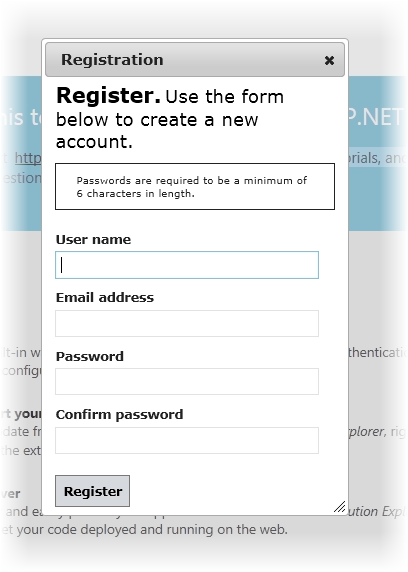
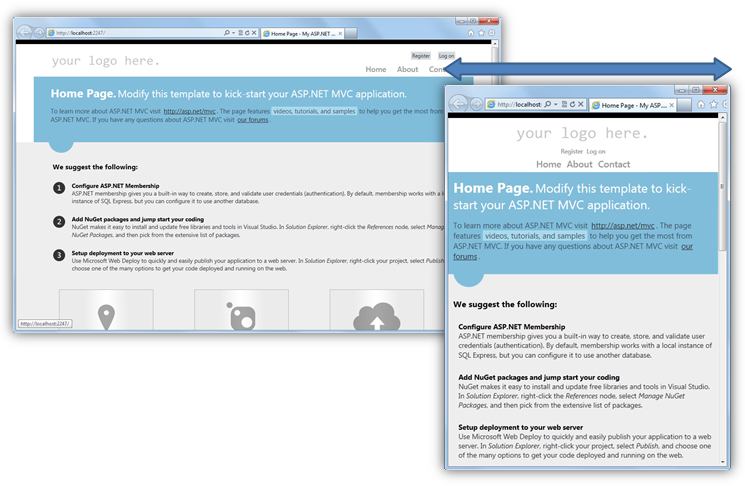
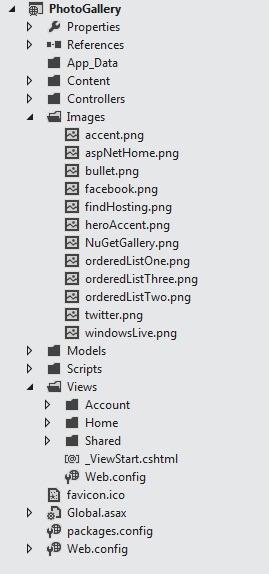
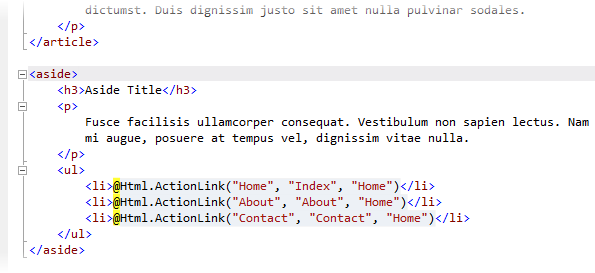
# Exercises

* 1. This hands-on lab includes the following exercises:
  2. New MVC 4 Project Templates
  3. Creating the Photo Gallery Web Application
  4. Adding Support for Mobile Devices
  5. Using Asynchronous Controllers
  6. **Note:** Each exercise is accompanied by a starting solution—located in the Begin folder of the exercise—that allows you to follow each exercise independently of the others. Please be aware that the code snippets that are added during an exercise are missing from these starting solutions and that they will not necessarily work until you complete the exercise.
  7. Inside the source code for an exercise, you will also find an End folder containing a Visual Studio solution with the resulting code from completing the steps in the corresponding exercise. You can use these solutions as guidance if you need additional help as you work through this hands-on lab.

Exercise 1: New MVC4 Project Templates

* 1. In this exercise, you will explore the enhancements in the ASP.NET MVC4 Project templates. In addition to the Internet Application template, already present in MVC 3, this version now includes a separate template for Mobile applications. First, you will look at some relevant features of each of the templates. Then, you will work on rendering your page properly on the different platforms by using the right approach.

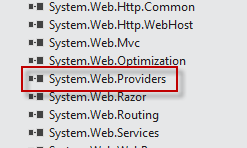
Task 1 – Exploring the Internet Application Template

* 1. Open **Visual Studio 11**.
  2. Select the **File | New | Project** menu command. In the **New Project** dialog, select the **Visual C#|Web** template on the left pane tree, and choose **ASP**.**NET MVC 4 Web Application.** Name the project **PhotoGallery**, select a location (or leave the default) and click **OK**.
     1. **Note:** You will later customize the PhotoGallery MVC 4 solution you are now creating.
     2. 
     3. Figure 1
     4. Creating a new project
  3. In the **New ASP.NET MVC 4 Project** dialog, select the **Internet Application** project template and click **OK**. Make sure you have selected Razor as the view engine.
     1. 
     2. Figure 2
     3. Creating a new MVC4 Internet Application
     4. **Note:** Razor syntax has been introduced in ASP.NET MVC 3. Its goal is to minimize the number of characters and keystrokes required in a file, enabling a fast and fluid coding workflow. Razor leverages existing C#/VB (or other) language skills and delivers a template markup syntax that enables an awesome HTML construction workflow.
  4. Press **F5** to run the solution and see the renewed templates. You can check out the following features:
     + **Modern-style templates**
       1. The templates have been renewed, providing more modern-looking styles.
       2. 
       3. Figure 3
       4. MVC4 restyled templates
       5. 
       6. Figure 4
       7. New Contact page
     + **Richer UI with JavaScript**
       1. Another enhancement to default project templates is the use of JavaScript to provide a more interactive JavaScript. The Login and Register links used in the template exemplify how to use the jQuery UI Dialog to display a fancy login screen.
       2. 
       3. Figure 5
       4. Log On dialog
       5. 
       6. Figure 6
       7. Registration dialog
     + **Adaptive Rendering**
       1. Check out resizing the browser window and notice how the page layout dynamically adapts to the new window size. These templates use the adaptive rendering technique to render properly in both desktop and mobile platforms without any customization.
       2. 
       3. Figure 7
       4. MVC 4 project template in different browser sizes
  5. Close the browser to stop the debugger and return to Visual Studio.
  6. Now you are able to explore the solution and check out some of the new features introduced by ASP.NET MVC 4 in the project template.
     1. 
     2. Figure 8
     3. The MVC4 Internet Application Project Template
     + **HTML 5 Markup**
       1. Browse template views to find out the new theme markup.
       2. 
       3. Figure 9
       4. New template, using Razor and HTML5 markup (About.cshtml).
     + **Updated JavaScript libraries**
       1. The MVC4 default template now includes KnockoutJS, a JavaScript MVVM framework that lets you create rich and highly responsive web applications using JavaScript and HTML. Like in MVC3, jQuery and jQuery UI libraries are also included in ASP.NET MVC 4.
       2. **Note:** You can get more information about KnockOutJS library in this link: <http://learn.knockoutjs.com/>.

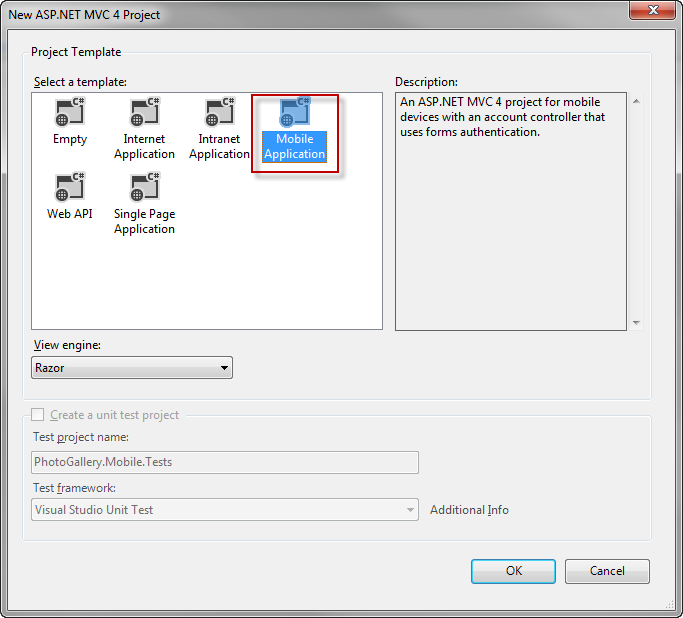
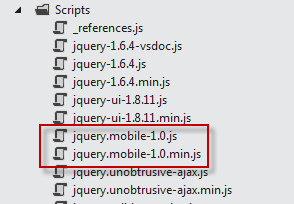
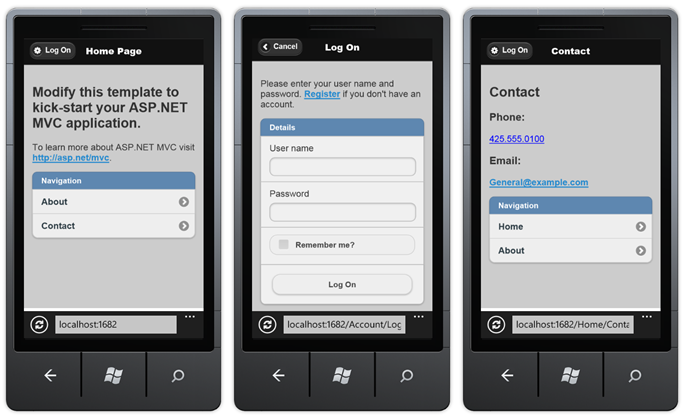
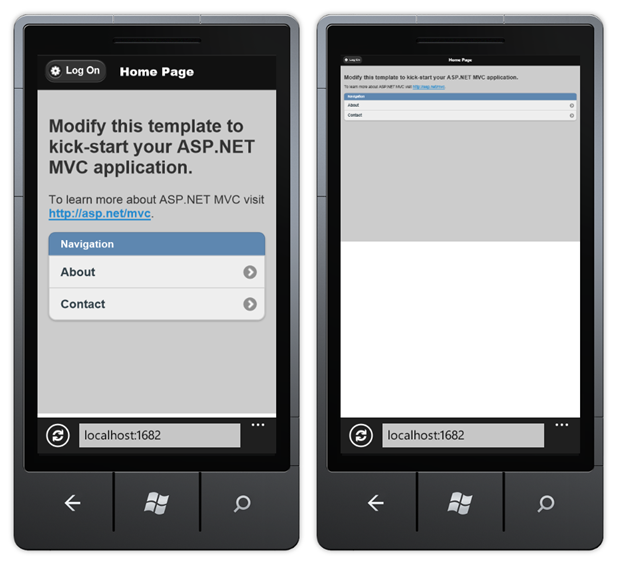
Additionally, you can learn about jQuery and jQuery UI in <http://docs.jquery.com/>.

* + - **ASP.NET Universal providers included in the solution**
      1. ASP.NET Universal Providers extend Session, Membership, Roles and Profile support to SQL Compact Edition and SQL Azure. By only configuring the right connection string, your application will be able to work with SQL Server (plus Express), SQL Server Compact or SQL Azure.
      2. **Note:** ASP.NET Universal Providers Library is also available for MVC3 projects in the NuGet public feed.

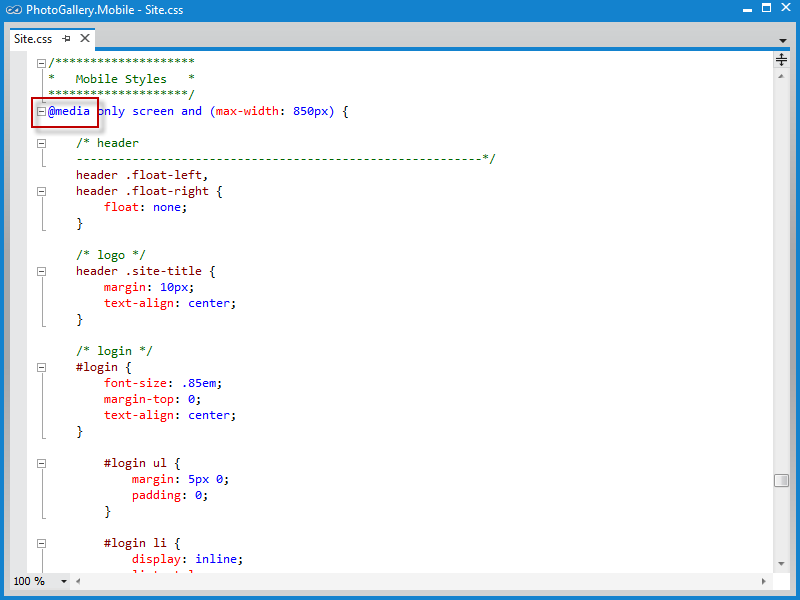
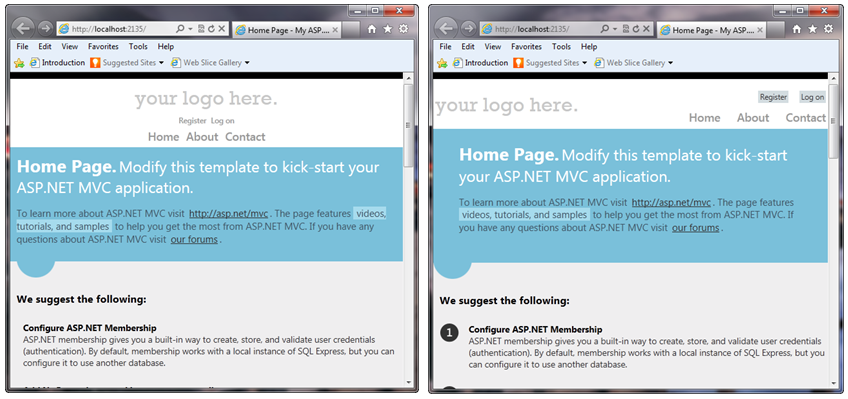
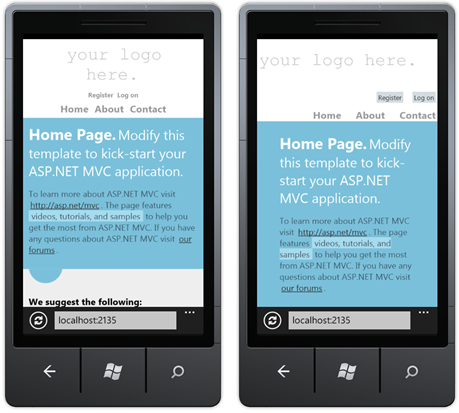
You can learn more about SQL Azure in this link: <http://msdn.microsoft.com/en-us/wazplatformtrainingcourse_sqlazure_unit>.

* + - 1. 
      2. Figure 10
      3. ASP.NET Universal Providers reference is now included

Task 2 – Exploring the Mobile Application Template

* 1. ASP.NET MVC 4 facilitates the development of websites for mobile and tablet browsers. This template has the same application structure as the Internet Application template (notice that the controller code is practically identical), but its style was modified to render properly in touch-based mobile devices.
  2. Select the **File | New | Project** menu command. In the **New Project** dialog, select the **Visual C#|Web** template on the left pane tree, and choose the **ASP**.**NET MVC4 Web Application.** Name the project **PhotoGallery.Mobile**, select a location (or leave the default) and click **OK**.
  3. In the **New ASP.NET MVC 4 Project** dialog, select the **Mobile Application** project template and click **OK**. Make sure you have selected Razor as the view engine.
     1. 
     2. Figure 11
     3. Creating a new MVC4 Mobile Application
  4. Now you are able to explore the solution and check out some of the new features introduced by the MVC 4 solution template for mobile:
     + **jQuery Mobile Library** 
       1. The Mobile Application project template includes the jQuery Mobile library, which is an open source library for mobile browser compatibility. jQuery Mobile applies progressive enhancement to mobile browsers that support CSS and JavaScript. Progressive enhancement enables all browsers to display the basic content of a web page, while it only enables the most powerful browsers to display the rich content. The JavaScript and CSS files, included in the jQuery Mobile style, help mobile browsers to fit the content in the screen without making any change in the page markup.
       2. 
       3. Figure 12
       4. jQuery mobile library included in the template
     + **HTML5 based markup**
       1. 
       2. Figure 13
       3. Mobile application template using HTML5 markup, (LogOn.cshtml and index.cshtml)
  5. Press **F5** to run the solution.
  6. Open the **Windows Phone 7 Emulator,** located in **Start Menu | All Programs | Windows Phone SDK 7.1 | Windows Phone Emulator.**
  7. In the phone start screen, open Internet Explorer. Check out the URL where the desktop application started and browse to that URL from the phone (e.g. *http://localhost:6385/*).
  8. Now you are able to enter the login page or check out the about page. Notice that the style of the website is based on the new Metro app for mobile. The ASP.NET MVC 4 project template is correctly displayed on mobile devices, making sure all the elements of the page are visible and enabled. Notice that the links on the header are big enough to be clicked or tapped.
     1. 
     2. Figure 14
     3. Project template pages in a mobile device
  9. The new template also uses the **Viewport meta tag**. Most mobile browsers define a width for a virtual browser window or “viewport”, which is larger than the actual width of the mobile device. This enables mobile browsers to display the entire web page inside the virtual display. The **Viewport meta tag** allows web developers to set the width, height and scale of the browser area on mobile devices**.** The MVC 4 template for Mobile Applications sets the viewport to the device width (“width=device-width”) in the layout template (*Views\Shared\\_Layout.cshtml*), so that all the pages will have their viewport set to the device screen width. Notice that the Viewport tag will not change the default browser view.
  10. Open **\_Layout.cshtml**, located in the Views | Shared folder, and comment the Viewport tag. Run the application, if not already opened, and check out the differences.
      1. CSHTML
      2. ...
      3. <meta charset="utf-8" />
      4. <title>@ViewBag.Title</title>
      5. **@\* <meta name="viewport" content="width=device-width" /> \*@**
      6. ...
      7. 
      8. Figure 15
      9. The site after commenting the viewport meta tag
  11. In Visual Studio, press **SHIFT**+**F5** to stop debugging the application.
  12. Uncomment the viewport tag.
      1. CSHTML
      2. ...
      3. <meta charset="utf-8" />
      4. <title>@ViewBag.Title</title>
      5. **<meta name="viewport" content="width=device-width" />**
      6. ...

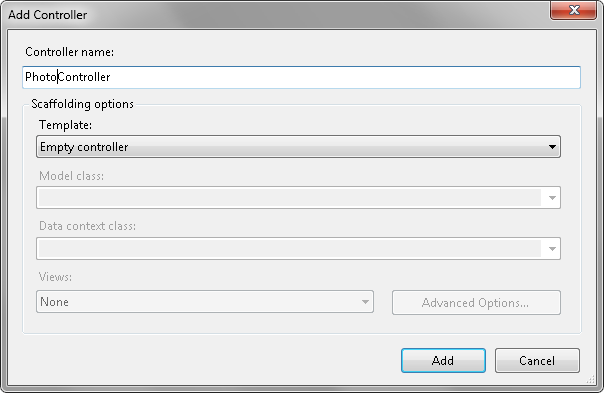
Task 3 – Using Adaptive Rendering

* 1. In this task, you will learn another method to render a Web page correctly on mobile devices and Web browsers at the same time without any customization. You have already used Viewport meta tag with a similar purpose. Now you will meet another powerful method: *adaptive rendering*.
  2. Adaptive rendering is a technique that uses **CSS3 media queries** to customize the style applied to a page. Media queries define conditions inside a style sheet, grouping CSS styles under a certain condition. Only when the condition is true, the style is applied to the declared objects.
  3. The flexibility provided by the adaptive rendering technique enables any customization for displaying the site on different devices. You can define as many styles as you want on a single style sheet without writing logic code to choose the style. Therefore, it is a very neat way of adapting page styles, as it reduces the amount of duplicated code and logic for rendering purposes. On the other hand, bandwidth consumption would increase, as the size of your CSS files could grow marginally.
  4. By using the adaptive rendering technique, your site will be **displayed properly, regardless of the browser.** However,you should consider if the bandwidth extra load is a concern.
  5. **Note:** The basic format of a media query is: @media [Scope: all | handheld | print | projection | screen] ([property:value] and … [property:value])
  6. Examples of media queries:
  7. **@media all and (max-width: 1000px) and (min-width: 700px) {}:   
     For a**ll the resolutions between 700px and 1000px.
  8. **@media screen and (min-width: 400px) and (max-width: 700px) { … }:**Only for screens. The resolution must be between 400 and 700px.
  9. **@media handheld and (min-width: 20em), screen and (min-width: 20em) { ... }:**For handhelds(mobile and devices) and screens. The minimum width must be greater than 20em.
  10. You can find more information about this on the [W3C site](http://www.w3.org/TR/css3-mediaqueries/).
  11. You will now explore how the adaptive rendering works, improving the readability of the MVC 4 default website template.
  12. Open the **PhotoGallery.sln** Internet application you have created at Task 1. Press **F5** to run the solution.
  13. Resize the browser’s width, setting the windows to half or to less than a quarter of its original size. Notice what happens with the items in the header: Some elements will not appear in the visible area of the header.
  14. Open **Site.css** file from the Visual Studio Solution explorer, located in **Content** project folder. Press **CTRL+F** to open Visual Studio integrated search, and write **@media** to locate the **CSS media query**.
      1. The media query condition defined in this template works in this way: When the browser’s window size is below **850px**, the CSS rules applied are the ones defined inside this media block.
      2. 
      3. Figure 16
      4. Locating the media query
  15. Replace the max-width attribute value set in 850px with 10px, in order to disable the adaptive rendering, and press **CTRL + S** to save the changes. Return to the browser and press **CTRL+F5** to refresh the page with the changes you have made. Notice the differences in both pages when adjusting the width of the window.
      1. 
      2. Figure 17
      3. In the left, the page is applying the @media style, in the right, the style is omitted
      4. Now, let’s check out what happens on mobile devices:
      5. 
      6. Figure 18
      7. In the left, the page is applying the @media style, in the right, the style is omitted
      8. Although you will notice that the changes when the page is rendered in a Web browser are not very significant, when using a mobile device the differences become more obvious. On the left side of the image, we can see that the custom style improved the readability.
      9. Adaptive rendering can be used in many more scenarios, making it easier to apply conditional styling to a Web site and solving common style issues with a neat approach.
      10. The Viewport meta tag and CSS media queries are not specific to ASP.NET MVC 4, so you can take advantage of these features in any web application.
  16. In Visual Studio, press **SHIFT**+**F5** to stop debugging the application.

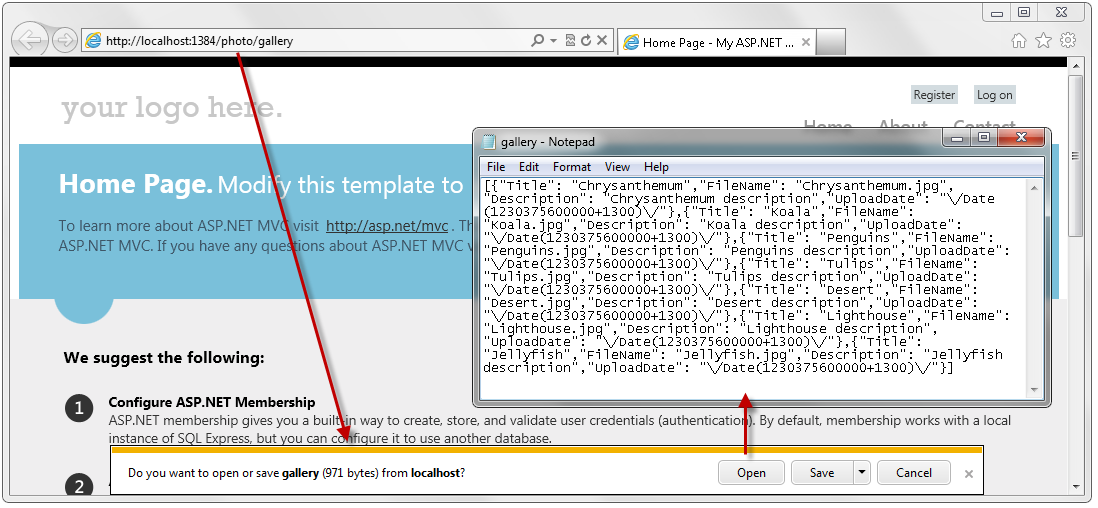
Exercise 2: Creating the Photo Gallery Web Application

* 1. In this exercise, you will work on a Photo Gallery application to display photos. You will start with the ASP.NET MVC 4 project template, and then you will add a feature to retrieve photos from a service and display them in the home page.
  2. In the following exercise, you will update this solution to enhance the way it is displayed on mobile devices.

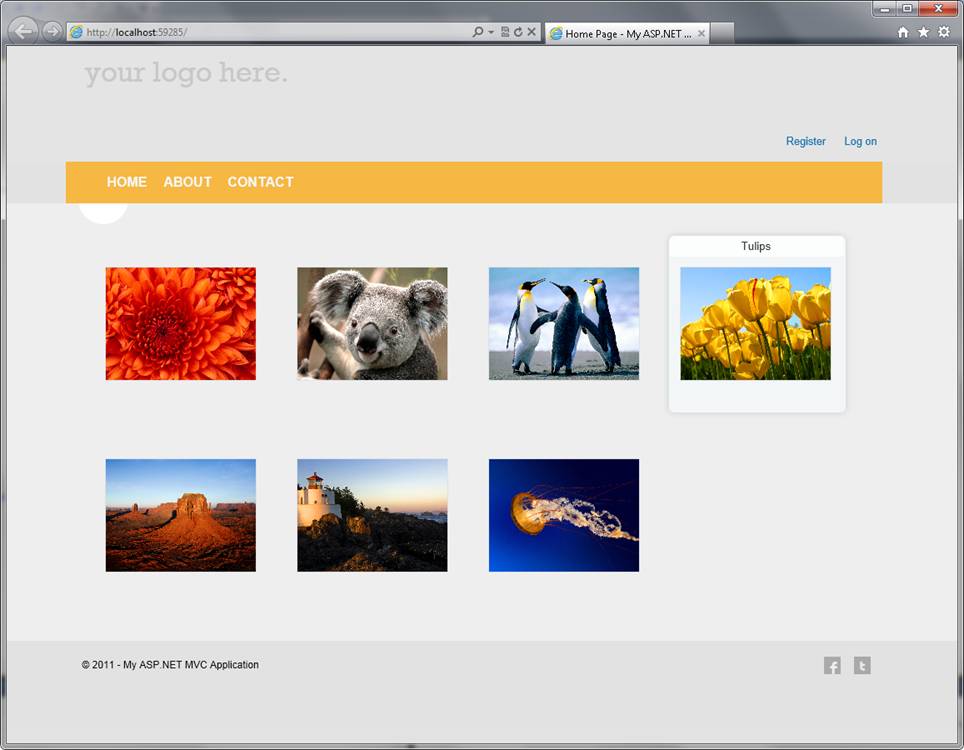
Task 1 – Creating a Mock Photo Service

* 1. In this task, you will create a mock of the photo service to retrieve the content that will be displayed in the gallery. To do this, you will add a new controller that will simply return a JSON file with the data of each photo.
  2. Open **Visual Studio 11** if not already opened.
  3. Select the **File | New | Project** menu command. In the **New Project** dialog, select the **Visual C# | Web** template on the left pane tree, and choose **ASP**.**NET MVC 4 Web Application.** Name the project **PhotoGallery**, select a location (or leave the default) and click **OK**. Alternatively, you can continue working from your existing MVC 4 **Internet Application** solution from **Exercise 1** and skip the next step.
  4. In the **New ASP.NET MVC 4 Project** dialog box, select the **Internet Application** project template and click **OK**. Make sure you have Razor selected as the View Engine.
  5. In the **Solution Explorer**, right-click the **App\_Data** folder of your project, and select **Add | Existing Item**. Browse to the **Source\Assets\App\_Data** folder of this lab and add the **Photos.json** file.
  6. Create a new controller with the name **PhotoController**. To do this, right-click on the **Controllers** folder, go to **Add** and select **Controller.** Complete the controller name, leave the **Empty controller** template and click **Add**.
     1. 
     2. Figure 19

Adding the PhotoController

* 1. Replace the **Index** method with the following **Gallery** action, and return the content from the JSON file you have recently added to the project.
     1. (Code Snippet – *MVC4 Lab - Ex02 - PhotoController*)
     2. C#
     3. public class PhotoController : Controller
     4. {
     5. **public ActionResult Gallery()**
     6. **{**
     7. **return this.File("~/App\_Data/Photos.json", "application/json");**
     8. **}**
  2. Press **F5** to run the solution, and then browse to the following URL in order to test the mocked photo service: **http://localhost:[port]/photo/gallery** (the [port] value depends on the current port where the application was launched). The request to this URL should retrieve the content of the *Photos.json* file.
     1. 
     2. Figure 20
     3. Testing the mocked photo service

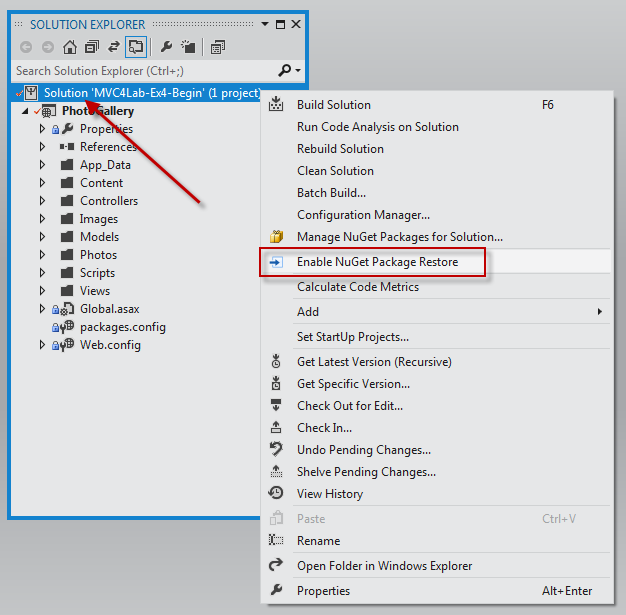
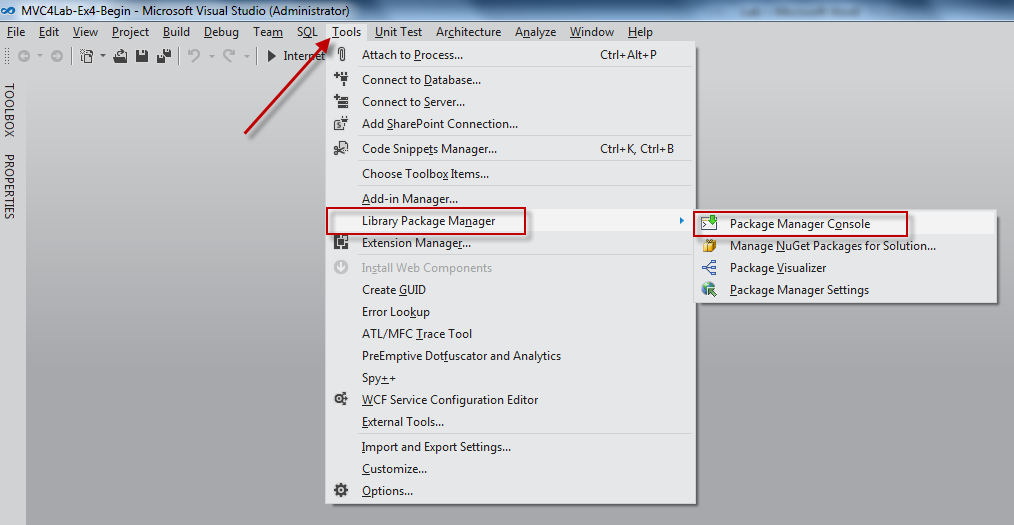
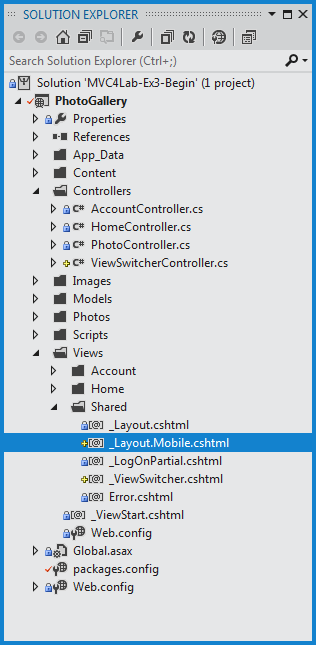
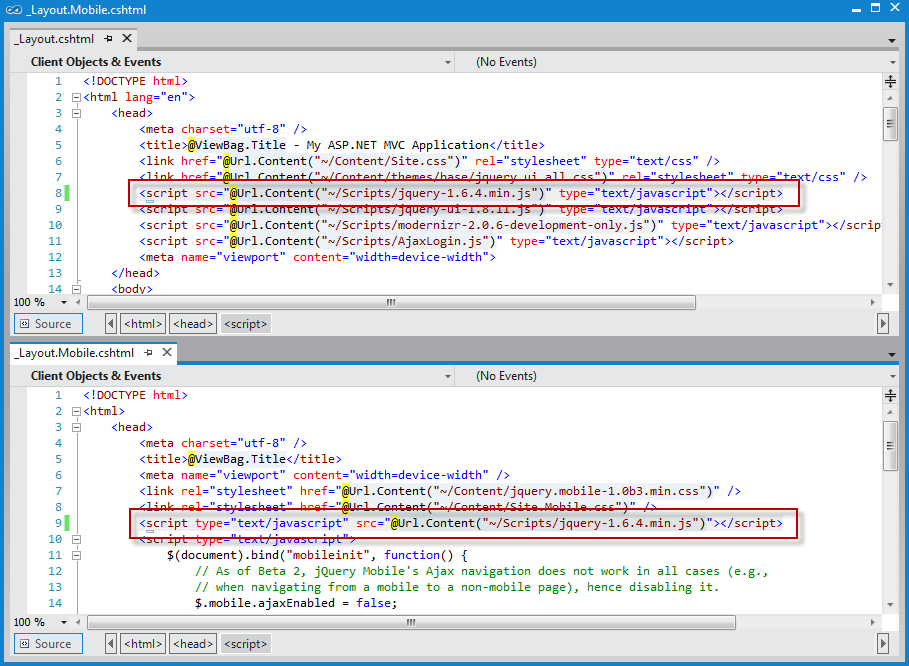
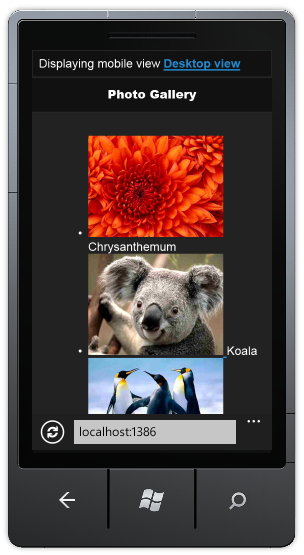
Task 2 – Displaying the Photo Gallery

* 1. In this task, you will update the Home page to show the photo gallery by using the mocked service you created in the first task of this exercise. You will add model files and update the gallery views.
  2. In Visual Studio, press **SHIFT**+**F5** to stop debugging the application.
  3. Create the **Photo** class in the **Models** folder. To do this, right-click on the **Models** folder, select **Add** and click **Class**. Then, set the name to **Photo.cs** and click **Add**.
  4. Add the following members to the **Photo** class.
     1. (Code Snippet – *MVC4 Lab - Ex02 - Photo model*)
     2. C#
     3. public class Photo
     4. {
     5. **public string Title { get; set; }**
     6. **public string FileName { get; set; }**
     7. **public string Description { get; set; }**
     8. **public DateTime UploadDate { get; set; }**
     9. }
  5. Open the **HomeController.cs** file from the **Controllers** folder.
  6. Add the following using statements.
     1. (Code Snippet – *MVC4 Lab - Ex02 - HomeController Usings*)
     2. C#
     3. **using System.Net;**
     4. **using System.Web.Script.Serialization;**
     5. **using PhotoGallery.Models;**
  7. Update the **Index** action to use **WebClient** to retrieve the gallery data, and then use the **JavaScriptSerializer** to deserialize it to the view model.
     1. (Code Snippet – *MVC4 Lab - Ex02 - Index Action*)
     2. C#
     3. public ActionResult Index()
     4. {
     5. **var client = new HttpClient();**
     6. **var response = client.Get(Url.Action("gallery", "photo", null, Request.Url.Scheme));**
     8. **var jss = new JavaScriptSerializer();**
     9. **var result = jss.Deserialize<List<Photo>>(response.Content.ReadAsString());**
     11. **return View(result);**
     12. }
  8. Open the **Index.cshtml** file located under the **Views\Home** folder and replace all the content with the following code.
     1. This code loops through all the photos retrieved from the service and displays them into an unordered list.
     2. (Code Snippet – *MVC4 Lab - Ex02 – Photo List*)
     3. HTML
     4. **@model List<PhotoGallery.Models.Photo>**
     5. **@{**
     6. **ViewBag.Title = "Photo Gallery";**
     7. **}**
     8. **<ul class="thumbnails">**
     9. **@foreach (var photo in Model)**
     10. **{**
     11. **<li class="item">**
     12. **<a href="@Url.Content("~/photos/" + photo.FileName)">**
     13. **<img alt="@photo.Description" src="@Url.Content("~/photos/" + photo.FileName)" class="thumbnail-border" width="180" />**
     14. **</a>**
     15. **<span class="image-overlay">@photo.Title</span>**
     16. **</li>**
     17. **}**
     18. **</ul>**
  9. In the **Solution Explorer**, right-click the **Content** folder of your project, and select **Add | Existing Item**. Browse to the **Source\Assets\Content** folder of this lab and add the **Site.css** file. You will have to confirm its replacement. If you have the **Site.css** file open, you will have to confirm to reload the file also.
  10. Open Windows Explorer and copy the entire **Photos** folder located under the **Source\Assets** folder of this lab to the root folder of your project in Solution Explorer.
  11. Run the application. You should now see the home page displaying the photos in the gallery.
      1. 
      2. Figure 21
      3. Photo Gallery
  12. In Visual Studio, press **SHIFT**+**F5** to stop debugging the application.

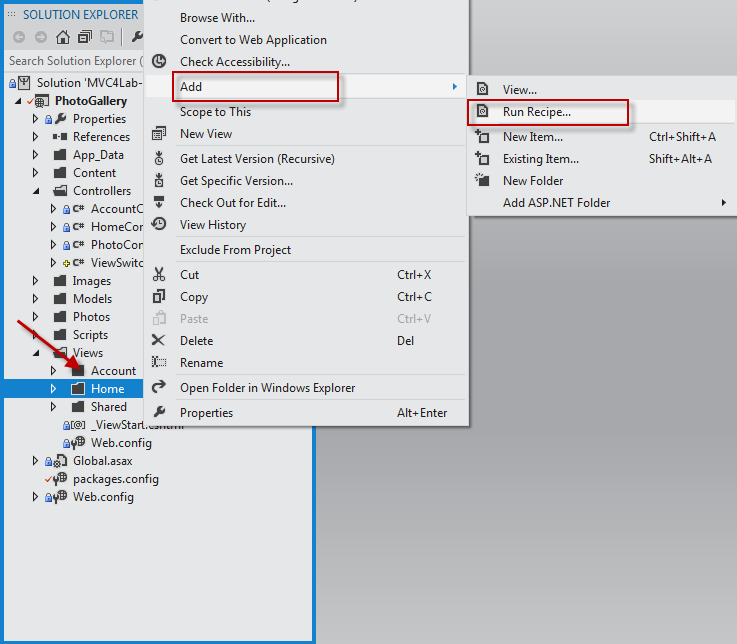
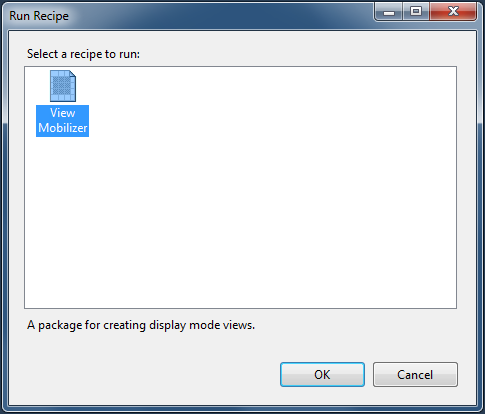
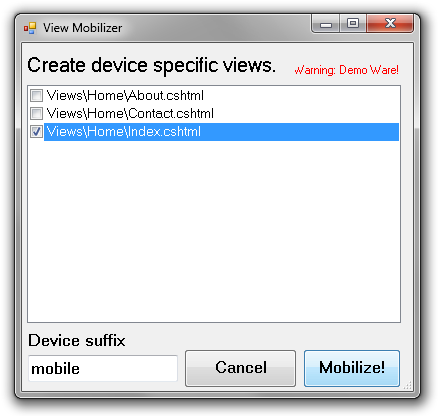
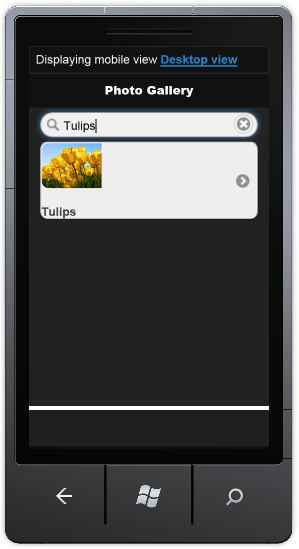
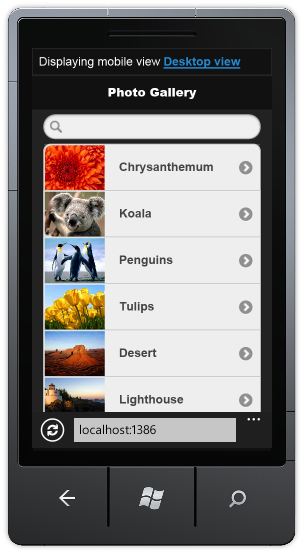
Exercise 3: Adding support for mobile devices

* 1. One of the key updates in ASP.NET MVC 4 is the support for mobile development. In this exercise, you will explore MVC4 new features for mobile applications by extending the PhotoGallery solution you have created in the previous exercise.

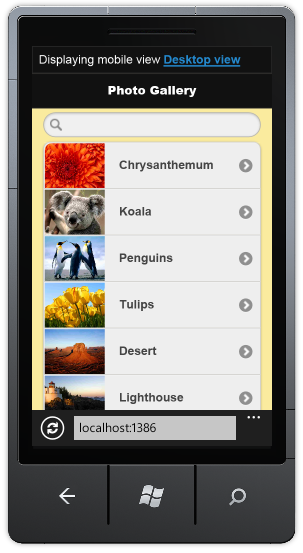
Task 1 – Installing jQuery Mobile in an ASP.NET MVC 4 Application

* 1. Open **Visual Studio 11** if not already opened.
  2. Open the **MVC4Lab-Ex3-Begin.sln** solution located in **Source\Ex3-MobileSupport\Begin** from this lab’s folder. Alternatively, you can continue working on your existing solution from the previous exercise.
  3. Right-click the solution and select the menu item **Enable NuGet Package Restore** to update all of the NuGet packages in each of the projects in the solution.
     1. 
     2. Figure 22
     3. Updating solution NuGet packages
        1. **Note:** One of the advantages of using NuGet is that you don’t have to ship all the libraries in your project, reducing the project size. With NuGet Power Tools, by specifying the package versions in the packages.config file, you will be able to download all the required libraries the first time you run the project. This is why you will have to run these steps after you open a new solution in this lab.
        2. For more information see the following article: <http://docs.nuget.org/docs/workflows/using-nuget-without-committing-packages>.
  4. Open the **Package Manager Console** by clicking the **Tools** > **Library Package Manager > Package Manager Console** menu option.
     1. 
     2. Figure 23
     3. Opening the NuGet Package Manager Console
  5. In the Package Manager Console run the following command to install the **jQuery.Mobile.MVC** package. The installation process also upgrades jQuery from version 1.6.2 to 1.6.4.
     1. jQuery Mobile is an open source library for building touch-optimized web UI. The jQuery.Mobile.MVC NuGet package includes helpers to use jQuery Mobile with an ASP.NET MVC 4 application.
     2. **Note:** By running the following command, you will be downloading the jQuery.Mobile.MVC library from Nuget.
     3. PM
     4. Install-Package jQuery.Mobile.MVC
     5. This command installs jQuery Mobile and some helper files, including the following:
     + **Views/Shared/\_Layout.Mobile.cshtml**: is a jQuery Mobile-based layout optimized for a smaller screen. When the website receives a request from a mobile browser, it will replace the original layout (\_Layout.cshtml) with this one.
     + A view-switcher component: consists of the **Views/Shared/\_ViewSwitcher.cshtml** partial view and the **ViewSwitcherController.cs** controller. This component will show a link on mobile browsers to enable users to switch to the desktop version of the page.
     1. 
     2. Figure 24
     3. Photo Gallery project with mobile support
  6. Open both **\_Layout.cshtml** and **\_Layout.Mobile.cshtml** (both under the Views | Shared folder) and make sure that the reference to jQuery matches the 1.6.4 version.
     1. 
     2. Figure 25
     3. Updating the jQuery reference
  7. Run the application using a desktop web browser.
  8. Open the **Windows Phone 7 Emulator,** located in **Start Menu | All Programs | Windows Phone SDK 7.1 | Windows Phone Emulator.**
  9. In the phone start screen, open Internet Explorer. Check out the URL where the application started and browse to that URL with the phone browser (e.g. *http://localhost:1385/*).
     1. You will notice that your application will look different in the Windows Phone emulator, as the jQuery.Mobile.MVC has created new assets in your project that show views optimized for mobile devices.
     2. Notice the message at the top of the phone, showing the link that switches to the Desktop view. Additionally, the \_Layout.Mobile.cshtml layout that was created by the package you have installed is including a different layout in the application.
     3. **Note:** So far, there is no link to get back to mobile view. It will be included in later versions.
     4. 
     5. Figure 26
     6. Mobile view of the Photo Gallery Home page
  10. In Visual Studio, press **SHIFT**+**F5** to stop debugging the application.

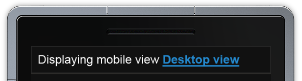
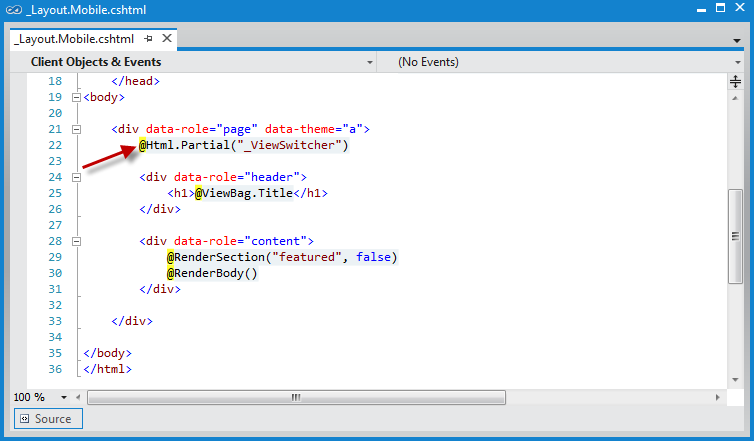
Task 2 – Using Recipes for Code Generation

* 1. The new recipes feature in ASP.NET MVC 4 enables Visual Studio to generate solution-specific code based on packages that you can install using NuGet. This recipes framework makes it easy for developers to write code-generation plugins, which have the ability to replace the built-in code generators for Add Area, Add Controller, and Add View. Since recipes are deployed as NuGet packages, they can easily be checked into source control and shared with all developers in the project automatically. They are also available on a per-solution basis.
  2. In this task, you will install and use a recipe to generate mobile versions of existing views.
  3. Open the NuGet **Package Manager Console**. To do this, use the menu **Tools | Library Package Manager | Package Manager Console**.
  4. In the **Package Manager Console** run the following command to install the **MvcHaack.ViewMobilizer** package.
     1. PM
     2. Install-Package MvcHaack.ViewMobilizer
     3. **Note:** In this lab you will use a sample code recipe available from NuGet. If you want to learn how to create a code recipe, read the following article: <http://haacked.com/archive/2011/09/22/writing-a-recipe-for-asp-net-mvc-4-developer-preview.aspx>
  5. Right-click the **Views** | **Home** folder, select **Add** | **Run Recipe**.
     1. 
     2. Figure 27
     3. Running the code recipe
  6. In the **Run Recipe** dialog select the **View Mobilizer** recipe and click **OK**.
     1. 
     2. Figure 28
     3. Run Recipe dialog box
  7. Select the **Views\Home\Index.cshtml** view, leave the default “mobile” Device suffix, and then click **Mobilize!**.
     1. 
     2. Figure 29
     3. View Mobilizer
  8. Open the new generated **Index.mobile.cshtml** view and replace the existing <ul> tag with this code. By doing this, you will be updating the <ul> tag with jQuery Mobile data annotations to use the mobile themes from jQuery.
     1. HTML
     2. **<ul data-role="listview" data-inset="true" data-filter="true">**
     3. **Note:**
     4. Notice that:
     5. - The **data-role** attribute set to **listview** will render the list using the listview styles.
     6. - The **data-inset** attribute set to true will show the list with rounded border and margin.
     7. - The **data-filter** attribute set to **true** will generate a search box.
     8. You can learn more about jQuery Mobile conventions in the project documentation: <http://jquerymobile.com/demos/1.0rc2/>
  9. Press **CTRL+ S** to save the changes.
  10. Switch to the **Windows Phone Emulator** and refresh the site. Notice the new look and feel of the gallery list, as well as the new search box located on the top. Then, type a word in the search box (for instance, **Tulips**) to start a search in the photo gallery.
      1. 
      2. Figure 30
      3. Gallery using listview style with filtering
      4. To summarize, you have used the View Mobilizer recipe to create a copy of the Index view with the “mobile” suffix. This suffix indicates to MVC that every request generated from a mobile device will use this copy of the index. Additionally, you have updated the mobile version of the Index view to use jQuery Mobile for enhancing the site look and feel in mobile devices.
  11. Go back to Visual Studio and open **Site.Mobile.css** located under the **Content** folder.
  12. Fix the positioning of the photo title to make it show at the right side of the image. To do this, add the following code to the **Site.Mobile.css** file.
      1. CSS
      2. **.ui-li .ui-btn-inner a.ui-link-inherit, .ui-li-static.ui-li {**
      3. **padding: 0px;**
      4. **}**
      5. **li.item span.image-overlay**
      6. **{**
      7. **position:relative;**
      8. **left:100px;**
      9. **top:-40px;**
      10. **height:0px;**
      11. **display:block;**
      12. **}**
  13. Press **CTRL + S** to save the changes.
  14. Switch back to the **Windows Phone Emulator** and refresh the site. Notice the photo title is properly positioned now.
      1. 
      2. Figure 31
      3. Title positioned on the right side of the image

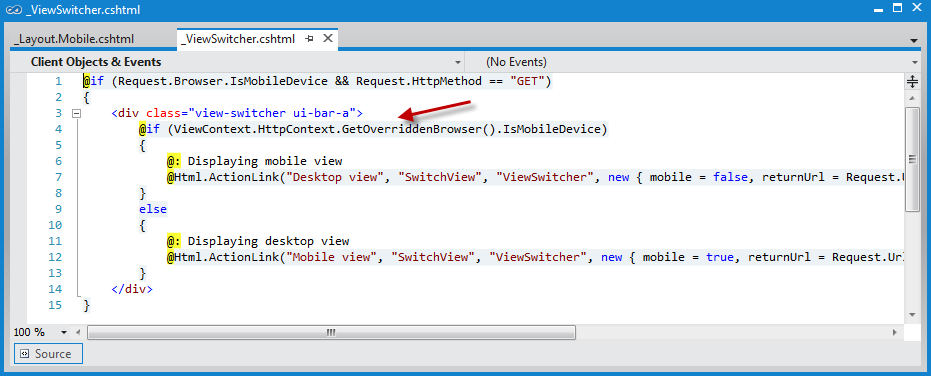
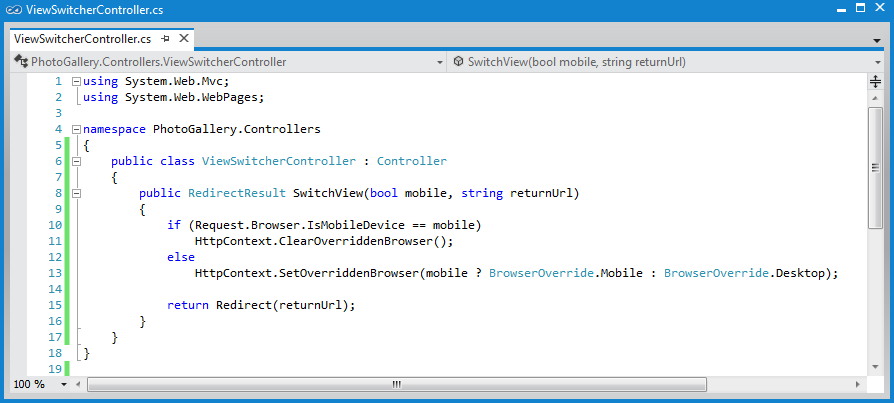
Task 3 – jQuery Mobile Themes

* 1. Every layout and widget in jQuery Mobile is designed around a new object-oriented CSS framework that makes it possible to apply a complete unified visual design theme to sites and applications.
  2. jQuery Mobile's default Theme includes 5 swatches that are given letters (a, b, c, d, e) for quick reference.
  3. In this task, you will update the mobile layout to use a different theme than the default.
  4. Switch back to Visual Studio.
  5. Open the **\_Layout.Mobile.cshtml** file located in **Views\Shared**.
  6. Find the div element with the data-role set to “page” and update the **data-theme** to “**e**”.
     1. XML
     2. **<div data-role="page" data-theme="e">**
  7. Press **CTRL + S** to save the changes.
  8. Refresh the site in the **Windows Phone Emulator** and notice the new colors scheme.
     1. 
     2. Figure 32
     3. Mobile layout with a different color scheme

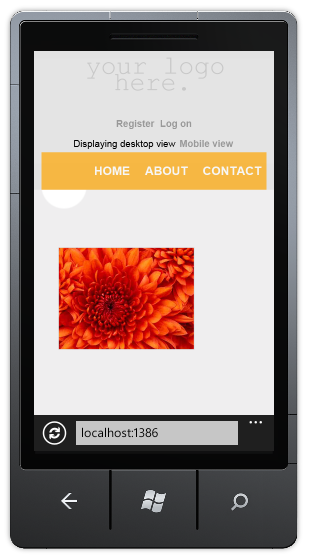
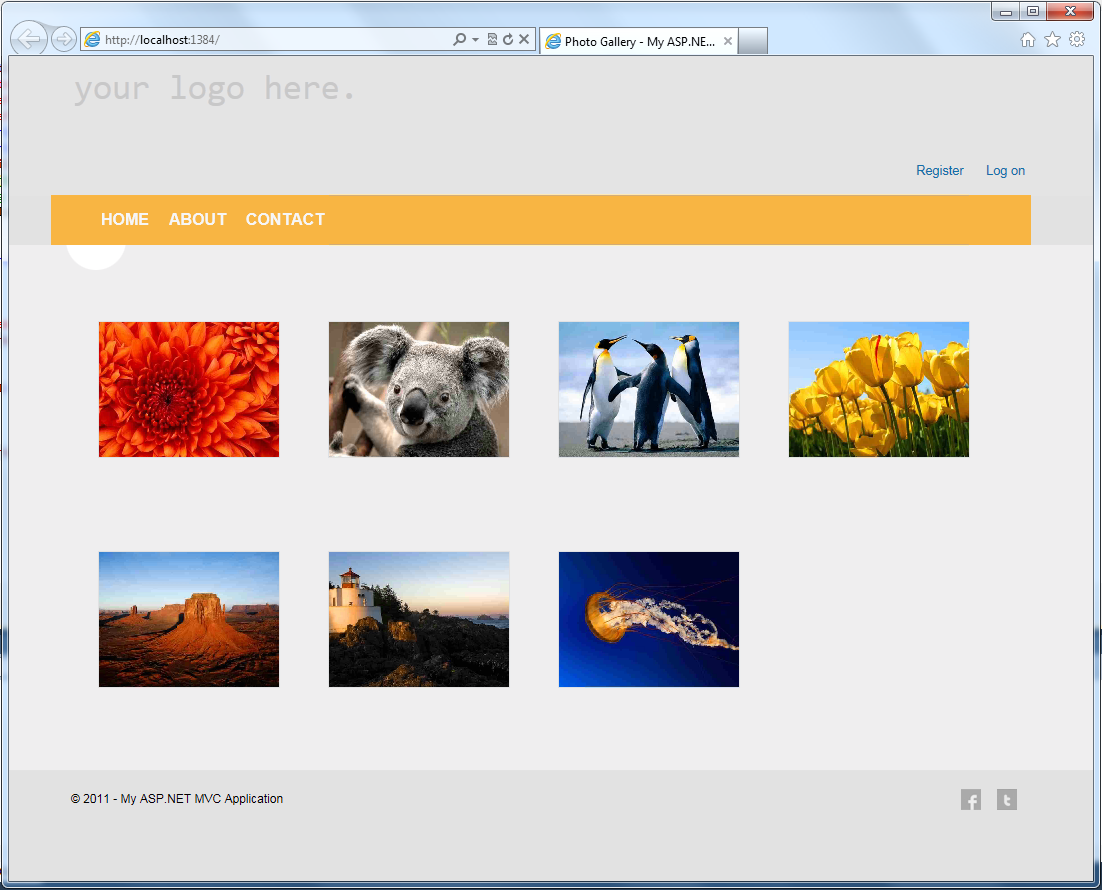
Task 4 – Using the View-Switcher Component and the Browser Overriding Features

* 1. A convention for mobile-optimized web pages is to add a link whose text is something like Desktop view or Full site mode that lets users switch to a desktop version of the page. The jQuery.Mobile.MVC package includes a sample **view-switcher** component for this purpose used in the \_Layout.Mobile.cshtml view.
  2. 
  3. Figure 33
  4. Link to switch to Desktop View
  5. The view switcher uses a new feature called **Browser Overriding**. This feature lets your application treat requests as if they were coming from a different browser (user agent) than the one they are actually coming from.
  6. In this task, you will explore the sample implementation of a view-switcher added by jQuery.Mobile.MVC and the new browser overriding features in ASP.NET MVC 4.
  7. Switch back to Visual Studio.
  8. Open the **\_Layout.Mobile.cshtml** view located under the **Views\Shared** folder and notice the view-switcher component being referenced as a partial view.
     1. 
     2. Figure 34
     3. Mobile layout using View-Switcher component
  9. Open the **\_ViewSwitcher.cshtml** partial view.
     1. The partial view uses the new method **ViewContext.HttpContext.GetOverriddenBrowser()** to determine the origin of the web request and show the corresponding link to switch either to the Desktop or Mobile views.

The **GetOverridenBrowser** method returns an **HttpBrowserCapabilitiesBase** instance that corresponds to the user agent currently set for the request (actual or overridden). You can use this value to get properties such as **IsMobileDevice**.

* + 1. 
    2. Figure 35
    3. \_ViewSwitcher partial view
  1. Open the **ViewSwitcherController.cs** class located in the **Controllers** folder. Check out that SwitchView action is called by the link in the ViewSwitcher component, and notice the new HttpContext methods.
     + The **HttpContext.ClearOverridenBrowser()** method removes any overridden user agent for the current request.
     + The **HttpContext.SetOverridenBrowser()** method overrides the request's actual user agent value using the specified user agent.
     1. 
     2. Figure 36
     3. ViewSwitcher Controller
     4. Browser Overriding is a core feature of ASP.NET MVC 4, which is also available even if you do not install the jQuery.Mobile.MVC package. However, this feature affects only view, layout, and partial-view, and it does not affect any of the features that depend on the Request.Browser object.

Task 5 – Adding the View-Switcher in the Desktop View

* 1. In this task, you will update the desktop layout to include the view-switcher. This will allow mobile users to go back to the mobile view when browsing the desktop view.
  2. Refresh the site in the **Windows Phone Emulator**.
  3. Click on the **Desktop view** link at the top of the gallery. Notice there is no view-switcher in the desktop view to allow you return to the mobile view.
  4. Go back to Visual Studio and open the **\_Layout.cshtml** view.
  5. Find the login section and insert a call to render the **\_ViewSwitcher** partial view below the **\_LogOnPartial** partial view. Then, press **CTRL + S** to save the changes.
     1. HTML
     2. <div class="float-right">
     3. <section id="login">
     4. @Html.Partial("\_LogOnPartial")
     6. **@Html.Partial("\_ViewSwitcher")**
     7. </section>
     8. <nav>
  6. Press **CTRL + S** to save the changes.
  7. Refresh the page in the Windows Phone Emulator and double-click the screen to zoom in. Notice that the home page now shows the **Mobile view** link that switches from mobile to desktop view.
     1. 
     2. Figure 37
     3. View Switcher rendered in desktop view
  8. Switch to the Mobile view again and browse to **About** page. Notice that, even if you haven’t created an About.Mobile.cshtml view, the About page is displayed using the mobile layout (\_Layout.Mobile.cshtml).
     1. 
     2. Figure 38
     3. About page
  9. Finally, open the site in a desktop Web browser. Notice that none of the previous updates has affected the desktop view.
     1. 
     2. Figure 39
     3. PhotoGallery desktop view

Task 6 – Creating New Display Modes

* 1. The new display modes feature lets an application select views depending on the browser that is generating the request. For example, if a desktop browser requests the Home page, the application will return the **Views\Home\Index.cshtml** template. Then, if a mobile browser requests the Home page, the application will return the **Views\Home\Index.mobile.cshtml** template.
  2. In this task, you will create a customized layout for iPhone devices, and you will have to simulate requests from iPhone devices. To do this, you can use either an iPhone emulator/simulator (like [Electric Mobile Simulator](http://www.electricplum.com/dlsim.html)) or a browser with add-ons that modify the user agent. For instructions on how to set the user agent string in an Safari browser to emulate an iPhone, see [How to let Safari pretend it's IE](http://www.davidalison.com/2008/05/how-to-let-safari-pretend-its-ie.html) in David Alison's blog.
  3. **Notice that this task is optional and you can continue throughout the lab without executing it.**
  4. In Visual Studio, press **SHIFT**+**F5** to stop debugging the application.
  5. Open **Global.asax.cs** and add the following using statement.
     1. C#
     2. **using System.Web.WebPages;**
  6. Add the following highlighted code into the Application\_Start method.
     1. (Code Snippet – *MVC4 Lab - Ex03 - iPhone DisplayMode*)
     2. C#
     3. protected void Application\_Start()
     4. {
     5. AreaRegistration.RegisterAllAreas();
     6. RegisterGlobalFilters(GlobalFilters.Filters);
     7. RegisterRoutes(RouteTable.Routes);
     8. **DisplayModeProvider.Instance.Modes.Insert(0, new DefaultDisplayMode("iPhone")**
     9. **{**
     10. **ContextCondition = context =>**
     11. **context.Request.UserAgent != null &&**
     12. **context.Request.UserAgent.IndexOf("iPhone", StringComparison.OrdinalIgnoreCase) >= 0**
     13. **});**
     14. }
     15. You have registered a new **DefaultDisplayMode named “iPhone”**, within the static **DisplayModeProvider.Instance.Modes** static list, that will be matched against each incoming request. If the incoming request contains the string "iPhone", ASP.NET MVC will find the views whose name contain the "iPhone" suffix. The 0 parameter indicates how specific is the new mode; for instance, this view is more specific than the general “.mobile” rule that matches requests from mobile devices.
     16. After this code runs, when an iPhone browser generates a request, your application will use the **Views\Shared\\_Layout.iPhone.cshtml** layout you will create in the next steps.
     17. **Note:** This way of testing the request for iPhone has been simplified for demo purposes and might not work as expected for every iPhone user agent string (for example test is case sensitive).
  7. Create a copy of the **\_Layout.Mobile.cshtml** file in the **Views\Shared** folder and rename the copy to “**\_Layout.iPhone.csthml**”.
  8. Open **\_Layout.iPhone.csthml** you created in the previous step.
  9. Find the div element with the data-role attribute set to **page** and change the **data-theme** attribute to “**a**”.
     1. XML
     2. <body>
     3. **<div data-role="page" data-theme="a">**
     4. @Html.Partial("\_ViewSwitcher")
     5. ...
     6. Now you have 3 layouts in your MVC application:
     7. **\_Layout.cshtml**: default layout used for desktop browsers.
     8. **\_Layout.mobile.cshtml**: default layout used for mobile devices.
     9. **\_Layout.iPhone.cshtml**: specific layout for iPhone devices, using a different color scheme to differentiate from \_Layout.mobile.cshtml.
  10. Press **F5** to run the application and browse the site in the **Windows Phone Emulator**.
  11. Open an **iPhone simulator**, as the [Electric Mobile Simulator](http://www.electricplum.com/dlsim.html), and browse to the site too. Notice that each phone is using the specific template.
      1. 
      2. Figure 40
      3. Using different views for each mobile device

Exercise 4: Using Asynchronous Controllers

1. Microsoft .NET Framework 4.5 introduces new language features in C# and Visual Basic to provide a new foundation for asynchrony in .NET programming. This new foundation makes asynchronous programming similar to – and about as straightforward as – synchronous programming.

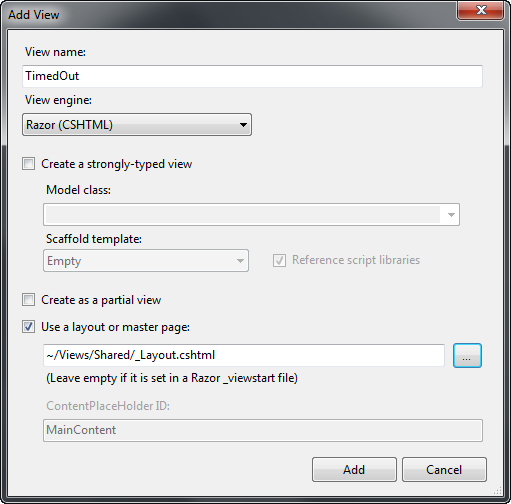
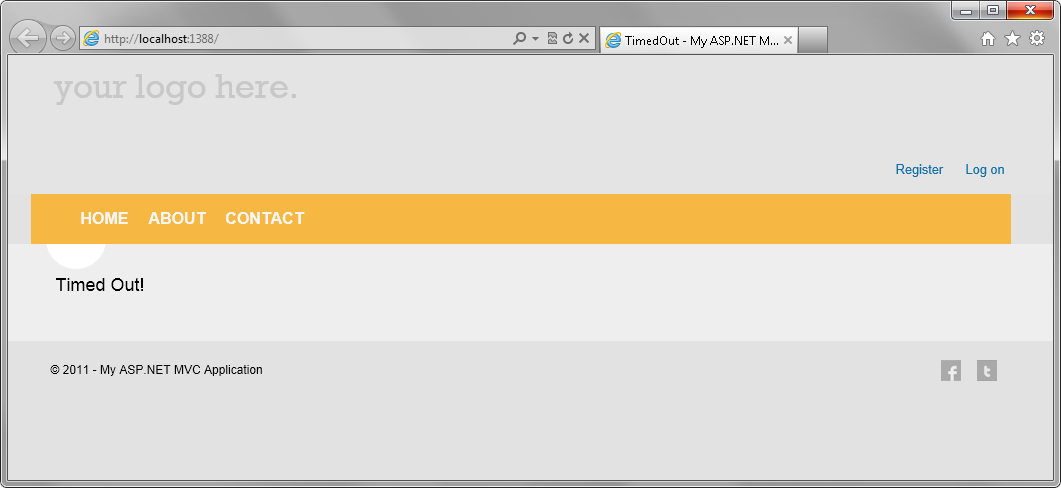
You are now able to write asynchronous action methods in ASP.NET MVC 4 by using the **AsyncController** class. You can use asynchronous action methods for long-running, non-CPU bound requests. This avoids blocking the Web server from performing work while the request is being processed. The AsyncController class is typically used for long-running Web service calls.

This exercise explains the basics of asynchronous operation in ASP.NET MVC 4. If you want a deeper dive, you can check out the following article: <http://msdn.microsoft.com/en-us/library/ee728598.aspx>

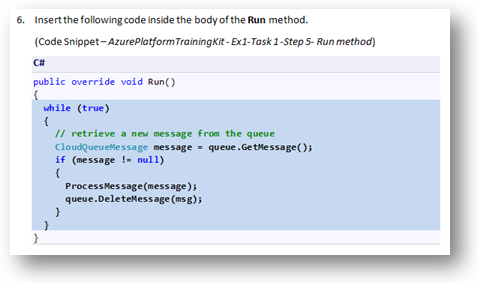
Task 1 – Implementing an Asynchronous Controller

* 1. Open **Visual Studio 11** if not already opened.
  2. Open the **MVC4Lab-Ex4-Begin.sln** solution located in **Source\Ex4-Async\Begin** from this lab’s folder. Alternatively, you can continue working on your existing solution from the previous exercise.
  3. If you opened the **MVC4Lab-Ex4-Begin.sln** solution, right-click the solution and select **Enable NuGet Package Restore**
     1. **Note:** For more information see the following article http://docs.nuget.org/docs/workflows/using-nuget-without-committing-packages
  4. Open the **HomeController.cs** class from the **Controllers** folder.
  5. Add the following using statement.
     1. C#
     2. **using System.Threading.Tasks;**
  6. Update the **HomeController** class to inherit from **AsyncController**. Controllers that derive from AsyncController enable ASP.NET to process asynchronous requests, and they can still service synchronous action methods.
     1. C#
     2. **public class HomeController : AsyncController**
     3. {
  7. Add the **async** keyword to the **Index** method and make it return the type **Task<ActionResult>**.
     1. C#
     2. **public async Task<ActionResult> Index()**
     3. {
     4. ...
     5. **Note:** The **async** keyword is one of the new keywords the .NET Framework 4.5 provides; it tells the compiler that this method contains asynchronous code.
     6. A **Task** object represents an asynchronous operation that may complete at some point in the future.
  8. Replace the **client.Get()** calls with the async version **client.GetAsync()** as shown below.
     1. (Code Snippet – *MVC4 Lab - Ex04 - GetAsync*)
     2. C#
     3. public async Task<ActionResult> Index()
     4. {
     5. var client = new HttpClient();
     6. **var response = await client.GetAsync(Url.Action("gallery", "photo", null, Request.Url.Scheme));**
     7. ...
     8. **Note:** In the previous code, you are using the asynchronous version of the **Get** method - **GetAsync** - which is a new method added by the .NET Framework 4.5 and has different signature since it returns a **Task** type object.
     9. Adding the **await** keyword tells the compiler to asynchronously wait for the task returned from the method call. This means that the rest of the code will be executed as a callback only after the awaited method completes. Another thing to notice is that you do not need to change your try-catch block in order to make this work: the exceptions that happen in background or in foreground will still be caught without any extra work using a handler provided by the framework.
  9. Change the code to continue with the asynchronous implementation by commenting out the latter two lines of the method and replacing them with the new code as shown below
     1. (Code Snippet – *MVC4 Lab - Ex04 - ReadAsStringAsync*)
     2. C#
     3. //var jss = new JavaScriptSerializer();
     4. //var result = jss.Deserialize<List<Photo>>(response.Content.ReadAsString());
     5. **var jss = new JavaScriptSerializer();**
     6. **var responseString = await response.Content.ReadAsStringAsync();**
     7. **var result = jss.Deserialize<List<Photo>>(responseString);**
  10. Run the application. You will notice no major changes, but your code will not block a thread from the thread pool making a better usage of the server resources and improving performance.
      1. **Note:** You can learn more about the new asynchronous programming features in the lab “**Asynchronous Programming in .NET 4.5 with C# and Visual Basic**” included in the Visual Studio 11 Training Kit.

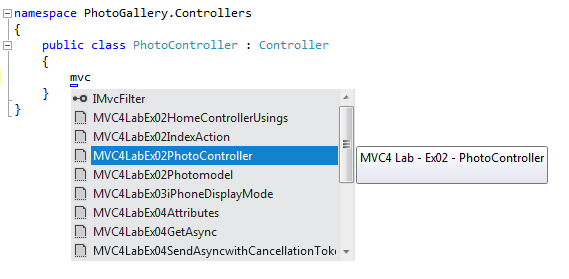
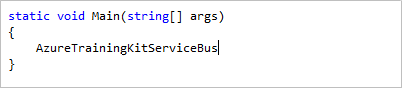
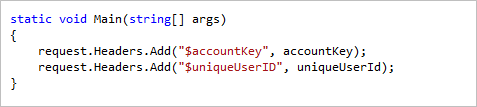
Task 2 – Handling Time-Outs with Cancellation Tokens

* 1. Asynchronous action methods that return Task instances can also support time-outs. In this task, you will update the Index method code to handle a time-out scenario using a cancellation token.
  2. Go back to Visual Studio and press **SHIFT+F5** to stop debugging.
  3. Add the following using statement to the **HomeController.cs** file.
     1. C#
     2. **using System.Threading;**
  4. Update the Index action to receive a **CancellationToken** argument.
     1. C#
     2. public async Task<ActionResult> Index(**CancellationToken cancellationToken**)
     3. {
     4. ...
  5. Refactor the service calls to use **SendAsync** instead of **GetAsync** and receive the cancellation token.
     1. (Code Snippet – *MVC4 Lab - Ex04 - SendAsync with CancellationToken*)
     2. C#
     3. public async Task<ActionResult> Index(CancellationToken cancellationToken)
     4. {
     5. var client = new HttpClient();
     6. **var photosRequest = new HttpRequestMessage(HttpMethod.Get, Url.Action("gallery", "photo", null, Request.Url.Scheme));**
     7. **var response = await client.SendAsync(photosRequest, cancellationToken);**
     9. var jss = new JavaScriptSerializer();
     10. var result = jss.Deserialize<List<Photo>>(response.Content.ReadAsString());
     11. return View(result);
     12. }
  6. Decorate the *Index* method with an **AsyncTimeout** attribute set to 500 milliseconds and a **HandleError** attribute configured to handle **TaskCanceledException** by redirecting to a **TimedOut** view.
     1. (Code Snippet – *MVC4 Lab - Ex04 - Attributes*)
     2. C#
     3. **[AsyncTimeout(500)]**
     4. **[HandleError(ExceptionType = typeof(TaskCanceledException), View = "TimedOut")]**
     5. public async Task<ActionResult> Index(CancellationToken cancellationToken)
     6. {
  7. Open the **PhotoController** class and update the **Gallery** method to delay the execution 1000 miliseconds (1 second) to simulate a long running task.
     1. C#
     2. public ActionResult Gallery()
     3. {
     4. **System.Threading.Thread.Sleep(1000);**
     5. return this.File("~/App\_Data/Photos.json", "application/json");
     6. }
  8. Open the **Web.config** file and enable custom errors by editing the following property.
     1. C#
     2. <system.web>
     3. **<customErrors mode="On"></customErrors>**
     4. ...
  9. Create a new view in **Views\Shared** named **TimedOut** and use the default layout. In the Solution Explorer, right-click the **Views\Shared** folder and select **Add | View**.
     1. 
     2. Figure 41
     3. Using different views for each mobile device
  10. Update the **TimedOut** view content as shown below.
      1. HTML
      2. @{
      3. ViewBag.Title = "TimedOut";
      4. Layout = "~/Views/Shared/\_Layout.cshtml";
      5. }
      6. **<h2>Timed Out!</h2>**
  11. Run the application and navigate to the root URL. As you have added a **Thread.Sleep** of 1000 milliseconds, you will get a time-out error, generated by the **AsyncTimeout** attribute and catch by the **HandleError** attribute.
      1. 
      2. Figure 42
      3. Time-out exception handled

Appendix: Using Code Snippets

* 1. With code snippets, you have all the code you need at your fingertips. The lab document will tell you exactly when you can use them, as shown in the following figure.
  2. 
  3. Figure 43
  4. Using Visual Studio code snippets to insert code into your project

#### To add a code snippet using the keyboard (C# only)

* 1. Place the cursor where you would like to insert the code.
  2. Start typing the snippet name (without spaces or hyphens).
  3. Watch as IntelliSense displays matching snippets' names.
  4. Select the correct snippet (or keep typing until the entire snippet's name is selected).
  5. Press the Tab key twice to insert the snippet at the cursor location.
  6. 
  7. Figure 44
  8. Start typing the snippet name
  9. 
  10. Figure 45
  11. Press Tab to select the highlighted snippet
  12. 
  13. Figure 46
  14. Press Tab again and the snippet will expand

#### To add a code snippet using the mouse (C#, Visual Basic and XML)

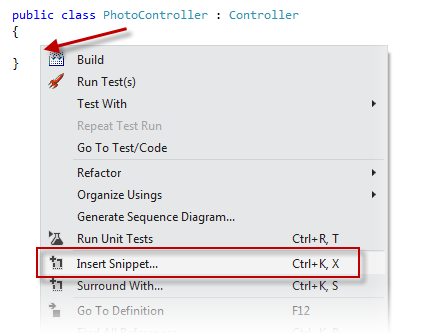
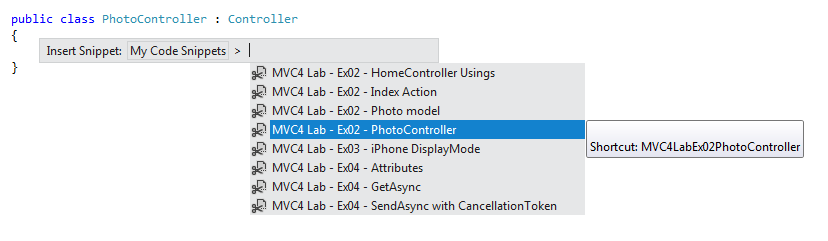
* 1. Right-click where you want to insert the code snippet.
  2. Select **Insert Snippet** followed by **My Code Snippets**.
  3. Pick the relevant snippet from the list, by clicking on it.
  4. 
  5. Figure 47
  6. Right-click where you want to insert the code snippet and select Insert Snippet
  7. 

Figure 48

* 1. Pick the relevant snippet from the list, by clicking on it