* 1. 

Hands-On Lab

Introduction to ASP.NET 4 Web Forms

* 1. Lab version: 1.0.0
  2. Last updated: 1/4/2011
  3. 

# Contents

[Overview 3](#_Toc280375238)

[Exercise 1: New ASP.NET Project Templates 5](#_Toc280375239)

[Task 1 - Creating a new Web Application 6](#_Toc280375240)

[Task 2 - Exploring Out-of-the-Box Authentication Mechanism 11](#_Toc280375241)

[Task 3 - Exploring Minified Web.config 16](#_Toc280375242)

[Task 4 - Using Out-of-the-Box jQuery Scripts 17](#_Toc280375243)

[Exercise 1: Verification 20](#_Toc280375244)

[Exercise 2: Controlling Server Control ClientIDs 22](#_Toc280375245)

[Task 1 – Assigning Static ClientID to ASP.NET Controls 24](#_Toc280375246)

[Task 2 – Assigning Predictable ClientID to ASP.NET Controls 26](#_Toc280375247)

[Task 3 – Assigning Inherit ClientID to ASP.NET Controls 29](#_Toc280375248)

[Task 4 – Targeting Static ClientIDs via CSS and JavaScript 30](#_Toc280375249)

[Exercise 2: Verification 34](#_Toc280375250)

[Exercise 3: Enabling Bi-Directional Routing Support 38](#_Toc280375251)

[Task 1 – Enabling ASP.NET Routing in your Application 39](#_Toc280375252)

[Task 2 – Using RouteUrlExpressionBuilder to Modify Navigation Links 41](#_Toc280375253)

[Task 3 – Retrieving Route Parameter Values 44](#_Toc280375254)

[Task 4 – Using RouteValueExpressionBuilder to Retrieve Route Values 45](#_Toc280375255)

[Exercise 3: Verification 47](#_Toc280375256)

[Exercise 4: Granular View State 52](#_Toc280375257)

[Task 1 – Disabling ViewState on Controls 52](#_Toc280375258)

[Task 2 – Enabling Granular ViewState on Child Controls 54](#_Toc280375259)

[Task 3 – Retrieving ViewState Values from Code 55](#_Toc280375260)

[Exercise 4: Verification 58](#_Toc280375261)

[Summary 63](#_Toc280375262)

Overview

* 1. ASP.NET 4 Web Forms provides enhancements in targeted areas and includes some new features. This Lab is also available online at <http://msdn.microsoft.com/trainingcourses> and covers the following features:
  + **New ASP.NET Project Templates**: ASP.NET 4 Web Forms includes updated templates for creating new web applications and web sites. These templates include common functionality already implemented, which helps reducing the development time and provides guidance on best practices for building ASP.NET 4 Web applications.
  + **Client IDs**: Developers can now manage control IDs that affect rendered client IDs. The **Control** class now provides a new property called **ClientIDMode** that allows you to specify what behavior you want the runtime to take when determining whether or not to refactor the client ID upon rendering. This removes the previous bloat in the client ID.
  + **URL Routing**: ASP.NET 4 Web Forms introduces the new **PageRouteHandler** class that integrates URL Routing with Web Form Pages. URL routing in ASP.NET enables you to use URLs that do not have to map to specific files in a Web site. Because the URL does not have to map to a file, you can use URLs in a Web application that are descriptive of the user's action and are more easily understood by users and search engines. In URL routing, you define URL patterns that contain placeholders for values that are used when you handle URL requests. At run time, the pieces of the URL that follow the application name are parsed into discrete values, based on a URL pattern that you have defined.
  + **View State**: ASP.NET 4 Web Forms provides a more granular control over the View State. Developers can now disable the View State on a Page and enable it on specific server controls, and also disable it on a control an enable it on its childs.

# Objectives

* 1. In this Hands-On Lab, you will learn how to:
  + Take advantage of the new ASP.NET Project Templates
  + Control server control ClientIds
  + Enable bi-directional routing support
  + Control the View State at application and page level

# System Requirements

* 1. You must have the following items to complete this lab:
  + Microsoft Visual Studio 2010
  + .NET Framework 4
  + Microsoft SQL Server 2008 (Express edition or above)
  + Adventure Works sample database
    1. **Note:** the Dependency Checker will copy the Adventure Works sample database file on each exercise solution folder. If you wish you can copy it manually from the Assets folder.

# Setup

* 1. All the requisites for this lab are verified using the **Configuration Wizard**. To make sure that everything is correctly configured, follow these steps:
  2. **Note:** To perform the setup steps you need to run the scripts in a command window with administrator privileges.
  3. Run the **Configuration Wizard** for the Training Kit if you have not done it previously. To do this, browse to **Source\Setup** folder of this lab, and double-click the **Dependencies.dep** file. Install any pre-requisites that are missing (rescanning if necessary) and complete the wizard.
     1. **Note:** The Configuration Wizard is used for checking dependencies and setting up the environment. If the Configuration Wizard is not installed on your machine, you must install it running the DependencyChecker.msi file located on the %VS2010TKInstallationFolder%\Assets\DependencyChecker folder (e.g. by default the Training Kit is installed under C:\VS2010TrainingKit).
     2. For convenience, much of the code you will be managing along this lab is available as Visual Studio code snippets. The **Dependencies.dep** file launches the Visual Studio installer file that installs the code snippets.

# Exercises

* 1. This Hands-On Lab is comprised by the following exercises:
  2. Discovering the New ASP.NET Project Templates
  3. Controlling Server Control ClientIds
  4. Enabling Bi-Directional Routing Support
  5. Controlling the View State at Application and Page Levels
  6. Estimated time to complete this lab: **90 minutes**.
  7. **Note:** Each exercise is accompanied by an **End** folder containing the resulting solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

**Note:** Each exercise contains a Visual Basic and a C# version; Inside the **End/Begin** solution folder you will find two folders: **VB**, containing the Visual Basic version of the exercise, and **C#**, containing the C# version of it.

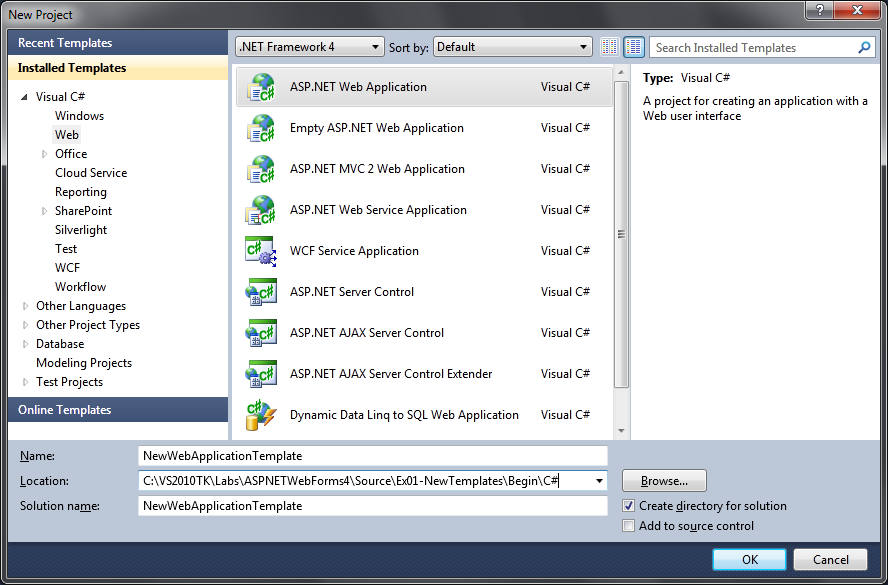
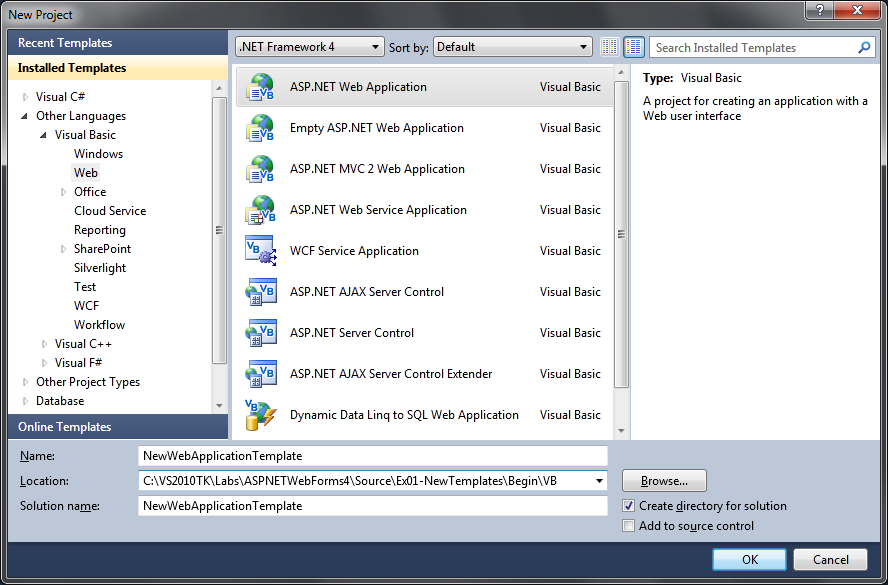
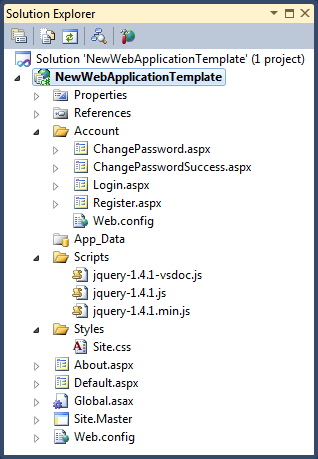
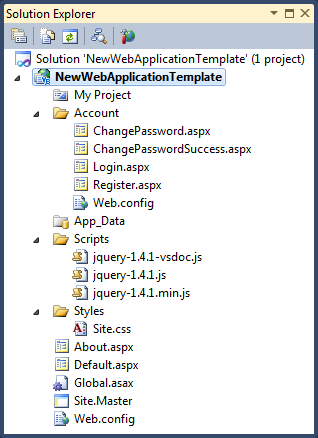
# Next Step

* 1. Exercise 1: New ASP.NET Project Templates

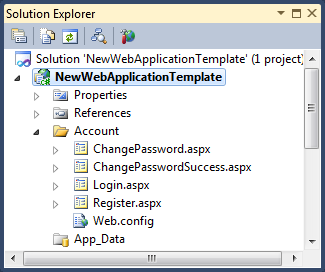
Exercise 1: New ASP.NET Project Templates

* 1. In this exercise, you will explore the new ASP.NET 4 Web Forms project templates provided in Visual Studio 2010. In earlier versions of ASP.NET, the project templates provided only a simple structure with very little guidance on how to build a production Web application. Developers had to implement certain common scenarios, like simple forms authentication, each time from scratch.
  2. ASP.NET 4 introduces three new templates, one for an **Empty Web application** project, and one each for a **Web Application** and **Web Site** project.
  3. The Empty Web Application template is a stripped-down Web Application project, very similar to earlier versions, including a minimal Web.config file.
  4. The other new templates contain major changes, for example:
  + **Basic Membership Functionality**: Most of the web sites or applications require some kind of security and authentication. The new templates have a simple implementation of a security module which lets you quickly get started in securing access to the new application.
  + **Default Master Page**: Frequently, master pages are used to define common rendering across a web application, like headers, menus, login status, etc. The new templates include a master page used by the default page.
  + **Default CSS file**: UI can be easily modified if the site was built using CSS. All the UI components that compose the project created by the new template make use of the provided cascading style sheet file definition, named Site.css. Moreover, ASP.NET 4 Web Forms include improvements on CSS support in the Web.config pages attribute controlRenderingCompatibilityVersion which indicates the level of backward compatibility.
  + **Minified Web.config**: With the Microsoft .NET Framework 4, all the configuration required for each module that is not application-specific, can be inferred from the **machine.config** file located inside the .NET Framework directory. This helps having a simpler Web.config which only includes the data that is application-specific, avoiding the needs of duplicating setting and thus having a much simpler and consumable configuration file.
  + **jQuery Integration**: The jQuery library is a very popular open-source JavaScript library that is included with both ASP.NET Web Forms and ASP.NET MVC. The Microsoft Ajax Library was designed to appeal to jQuery developers. You can mix jQuery plug-ins and Microsoft Ajax client controls seamlessly within the same Ajax application.
  1. On this exercise, you will create a web application using the new ASP.NET 4 WebForms template, explore the created project to identify the elements mentioned above, and implement a simple example using jQuery.

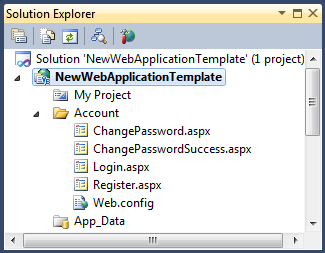
Task 1 - Creating a new Web Application

* 1. In this task, you will create a new Web Application using the project template provided with ASP.NET 4 Web Forms.
  2. Open **Microsoft Visual Studio 2010**. Click **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.
  3. From the **File** menu, select **New** | **Project**.
  4. In the *New Project* dialog, select the **ASP.NET Web Application** template, located under the **Web** templates.
  5. Type **NewWebApplicationTemplate** as **Name** and set its location to **Ex01-NewTemplates\Begin\ (Choosing the folder that matches the language of your preference)** inside the **Source** folder of this lab. Make sure that **Create directory for the solution** is checked, and click **OK** to create the project.
     1. 
     2. Figure 1
     3. Creating a Web Application using the new templates(C#)
     4. 
     5. Figure 2
     6. Creating a Web Application using the new templates(VB)
  6. Your solution should look like the following:
     1. 
     2. Figure 3
     3. Web Application created using the new Project Template (C#)
     4. 
     5. Figure 4
     6. Web Application created using the new Project Template (VB)
     7. You can easily identify the new features mentioned in the introduction by inspecting the project structure:
     8. **Note:** You will go through each of these items later on this exercise
     + **Basic Membership Functionality**: All the security functionality is implemented inside the Account folder of the project.
     + **Default Master Page**: The provided master page called **Site.Master** can be found at the root of the Web Application.
     + **Default CSS file**: A Styles folder with a **Site.css** file inside is automatically created by the template. This is where the styles for the whole site are defined.
     + **Minified Web.config**: Opening the **Web.config** file at the root of the Web Application will show a really simple and clean configuration file.
     + **jQuery Integration**: jQuery JavaScript library files are located inside the new **Scripts** folder.
       1. **Note:** You will notice that there are 3 .js files inside the Scripts folder. All of them are different versions of the jQuery Library:
       2. **- jquery-1.4.1-vsdoc.js:** This file has comments inline to support Visual Studio IntelliSense. You should not use it in your web site; it is intended for use at design time by Visual Studio.
       3. **- jquery-1.4.1.js:** This file is the jQuery library itself. You will use it while develop your application.
       4. **- jquery-1.4.1.min.js:** This file is the minified version of the jquery-1.3.2.js. Unnecessary blank-spaces have been removed and variables names were collapsed from the previous one. This file is optimized to be used on production environment saving user’s bandwidth.
     1. **Note:** Visual Studio 2010 also ships with an empty template if you not need all the features that the default template provides. To create an empty web application, simply choose the **Empty ASP.NET Web Application** template. This template only includes the Web.config.

Task 2 - Exploring Out-of-the-Box Authentication Mechanism

* 1. You are already aware that the new ASP.NET 4 Web Forms templates provide basic membership functionality to secure your web application; in this task, you will explore the implementation, identifying the key elements of it.
  2. In the Solution Explorer, expand the Account folder. You should see 4 pages and a Web.config file inside it.
     1. 
     2. Figure 5

Account folder inside the new Project Template (C#)

* + 1. 
    2. Figure 6
    3. Account folder inside the new Project Template (VB)
    4. **Note:** the **Web.config** file located inside the Account folder allows unauthenticated users to access the Register.aspx page. Access to other pages is limited to authenticated users only.
    5. Permissions for the Login.aspx page is not required since it is defined as the Forms Authentication login page making it automatically accessible to any user (authenticated or not.)
  1. In the Solution Explorer, double-click on the **Login.aspx** file. This is the default login page which contains a **Login** control with a custom layout template.
     1. HTML
     2. <asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
     3. <h2>
     4. Log In
     5. </h2>
     6. <p>
     7. Please enter your username and password.
     8. <asp:HyperLink ID="RegisterHyperLink" runat="server" EnableViewState="false">Register</asp:HyperLink> if you don't have an account.
     9. </p>
     10. **<asp:Login ID="LoginUser" runat="server" EnableViewState="false" RenderOuterTable="false">**
     11. **<LayoutTemplate>**
     12. **...**
     13. **</LayoutTemplate>**
     14. **</asp:Login>**
     15. </asp:Content>
  2. Open the **Site.Master** file, located at root level of the NewWebApplicationTemplate project; and locate the LoginView control.
     1. **Note:** The LoginView control is used to show the login status, and provide the link to login/logout. Adding this control to the Master Page implies it will be rendered in all the web pages across the web application that has this master page set.
     2. **Note:** The control is configured to show the name of the logged in user, or a link to the login page when an anonymous user access the site.
     3. HTML
     4. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
     5. <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
     6. <head runat="server">
     7. <title></title>
     8. <link href="~/Styles/Site.css" rel="stylesheet" type="text/css" />
     9. <asp:ContentPlaceHolder ID="HeadContent" runat="server">
     10. </asp:ContentPlaceHolder>
     11. </head>
     12. <body>
     13. <form runat="server">
     14. <div class="page">
     15. <div class="header">
     16. <div class="title">
     17. <h1>
     18. My ASP.NET Application
     19. </h1>
     20. </div>
     21. <div class="loginDisplay">
     22. **<asp:LoginView ID="HeadLoginView" runat="server" EnableViewState="false">**
     23. **<AnonymousTemplate>**
     24. **[ <a href="~/Account/Login.aspx" ID="HeadLoginStatus" runat="server">Log In</a> ]**
     25. **</AnonymousTemplate>**
     26. **<LoggedInTemplate>**
     27. **Welcome <span class="bold"><asp:LoginName ID="HeadLoginName" runat="server" /></span>!**
     28. **[ <asp:LoginStatus ID="HeadLoginStatus" runat="server" LogoutAction="Redirect" LogoutText="Log Out" LogoutPageUrl="~/"/> ]**
     29. **</LoggedInTemplate>**
     30. **</asp:LoginView>**
     31. </div>
     32. <div class="clear hideSkiplink">
     33. <asp:Menu ID="NavigationMenu" runat="server" CssClass="menu" EnableViewState="false" IncludeStyleBlock="false" Orientation="Horizontal">
     34. <Items>
     35. <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home"/>
     36. <asp:MenuItem NavigateUrl="~/About.aspx" Text="About"/>
     37. </Items>
     38. </asp:Menu>
     39. </div>
     40. </div>
     41. <div class="main">
     42. <asp:ContentPlaceHolder ID="MainContent" runat="server"/>
     43. </div>
     44. <div class="clear">
     45. </div>
     46. </div>
     47. <div class="footer">
     49. </div>
     50. </form>
     51. </body>
     52. </html>
     53. **Note:** Register.aspx and ChangePassword.aspx pages are also implemented using ASP.NET Login Controls.
     54. Register.aspx uses the **CreateUserWizard** control to guide the user through the registration process, while ChangePassword.aspx uses the **ChangePassword** control.
     55. All Login controls are configured to use Forms Authentication with a SQL Membership, Role and Profile provider. This configuration can be found in the Web.config located in the root directory of your application.
  3. In the Solution Explorer, double-click on the Web.config located in the root directory of your application to open it.
  4. Locate the **<authentication>** element located inside the **<system.web>** configuration section. This section configures the forms login authentication as explained above.
     1. XML
     2. <authentication mode="Forms">
     3. <forms loginUrl="~/Account/Login.aspx" timeout="2880" />
     4. </authentication>
  5. Below the authentication section, explore the membership, profile and role providers’ definition. By default, the template has all providers configured to use ASP.NET profiles database.
     1. XML
     2. <membership>
     3. <providers>
     4. <clear/>
     5. <add name="AspNetSqlMembershipProvider" type="System.Web.Security.SqlMembershipProvider" connectionStringName="ApplicationServices"
     6. enablePasswordRetrieval="false" enablePasswordReset="true" requiresQuestionAndAnswer="false" requiresUniqueEmail="false"
     7. maxInvalidPasswordAttempts="5" minRequiredPasswordLength="6" minRequiredNonalphanumericCharacters="0" passwordAttemptWindow="10"
     8. applicationName="/" />
     9. </providers>
     10. </membership>
     11. <profile>
     12. <providers>
     13. <clear/>
     14. <add name="AspNetSqlProfileProvider" type="System.Web.Profile.SqlProfileProvider" connectionStringName="ApplicationServices" applicationName="/"/>
     15. </providers>
     16. </profile>
     17. <roleManager enabled="false">
     18. <providers>
     19. <clear/>
     20. <add name="AspNetSqlRoleProvider" type="System.Web.Security.SqlRoleProvider" connectionStringName="ApplicationServices" applicationName="/" />
     21. <add name="AspNetWindowsTokenRoleProvider" type="System.Web.Security.WindowsTokenRoleProvider" applicationName="/" />
     22. </providers>
     23. </roleManager>
     24. You can get detailed information about ASP.NET security in the following MSDN articles:
     25. - [ASP.NET Login Controls Overview](http://msdn.microsoft.com/en-us/library/ms178329.aspx)
     26. - [Explained: Forms Authentication in ASP.NET 2.0](http://msdn.microsoft.com/en-us/library/aa480476.aspx)
     27. - [Introduction to Membership](http://msdn.microsoft.com/en-us/library/yh26yfzy.aspx)
     28. - [ASP.NET Profile Properties Overview](http://msdn.microsoft.com/en-us/library/2y3fs9xs.aspx)
     29. - [Managing Authorization Using Roles](http://msdn.microsoft.com/en-us/library/9ab2fxh0.aspx)

Task 3 - Exploring Minified Web.config

In the Microsoft .NET Framework 4, the major configuration elements have been moved to the machine.config file, and applications now inherit these settings. This allows the Web.config file in ASP.NET 4 applications either to be empty or to contain just the a few lines.

ASP.NET 4 Web Forms new templates take advantage of this new feature by removing redundant configuration from the configuration file. The Web.config file then inherits the common configuration, such as AJAX, routing, and integration with IIS 7, from the machine.config file by default.

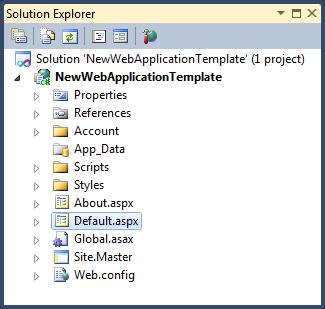
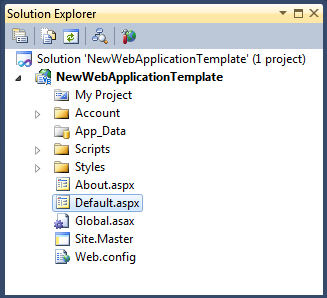
In this task, you will explore the default Web.config file detecting which elements are required.

* 1. In the Solution Explorer, double-click in the **Web.config** file, located in the root directory of your **NewWebApplicationTemplate** project to open it.
     1. You will notice that it is shorter than previous versions ASP.NET’s Web.config files. In an Empty Web Forms application, the only required element in the Web.config file is the **compilation** element located in the **<system.web>** configuration section. This element indicates to the compiler the framework version that the application is targeted to.
     2. XML
     3. <!-- ... -->
     4. <system.web>
     5. **<compilation debug="true" targetFramework="4.0" />**
     6. <!-- ... -->
     7. </system.web>
     8. </configuration>
     9. Since you have created an application that includes the authentication module, you will see the configuration sections described in the previous task. These elements are not required on an application that does not use ASP.NET authentication
     10. The following table shows to which component each configuration element applies:

|  |  |
| --- | --- |
| Component | Configuration element |
| Forms Authentication | **configuration/system.web/authentication** |
| ASP.NET Membership | **configuration/system.web/membership** |
| ASP.NET Profile | **configuration/system.web/profile** |
| ASP.NET Role Management | **configuration/system.web/roleManager** |

* + 1. Additionally, you will see a <connectionString> configuration section. This section defines the connection string that will be used by all preconfigured the SQL Providers.

Task 4 - Using Out-of-the-Box jQuery Scripts

* 1. As it was explained, jQuery is a very popular JavaScript library that provides developers with a framework for interacting with the UI components rendered in the web page. The new templates include this library out-of-the-box.
  2. In this task, you will benefit from jQuery to change the color of the web page title.
  3. In the Solution Explorer, double-click on the **Default.aspx** file to open it.
     1. 
     2. Figure 7
     3. Opening Default.aspx file inside the Web Application (C#)
     4. 
     5. Figure 8
     6. Opening Default.aspx file inside the Web Application (VB)
  4. In the **MainContent** content, add a button that will be in charge of changing the color of the “Welcome to ASP.NET” title. To do this, paste the following bolded code at the bottom of the page markup.
     1. HTML
     2. <asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
     3. <h2>
     4. Welcome to ASP.NET!
     5. </h2>
     6. <p>
     7. To learn more about ASP.NET visit <a href="http://www.asp.net" title="ASP.NET Website">www.asp.net</a>.
     8. </p>
     9. <p>
     10. You can also find <a href="http://go.microsoft.com/fwlink/?LinkID=152368&amp;clcid=0x409"
     11. title="MSDN ASP.NET Docs">documentation on ASP.NET at MSDN</a>.
     12. </p>
     13. **<input type="button" id="btnChangeTitleStyle" value="Change Title Style" />**
     14. </asp:Content>
  5. To use jQuery, you must add a reference to the jQuery Library. To do this, add the following script element, which references the jQuery source file, inside the **HeaderContent** Content tag.
     1. HTML
     2. <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
     3. **<script type="text/javascript" src="Scripts/jquery-1.4.1.js"></script>**
     4. </asp:Content>
     5. **Note:** jQuery libraries are also available in the Microsoft AJAX Content Delivery Network (CDN). It provides catching support for the most common JavaScript libraries like jQuery and AJAX.
     6. If you use it, users browsing your application will download the JavaScript libraries from the CDN instead from your Web Server.
     7. If you want to use the jQuery version provided by the CDN, you have to update the script element for:
     8. <script type="text/javascript" src="**http://ajax.microsoft.com/ajax/jquery/jquery-1.4.1.js**"></script>
  6. Inside the **HeaderContent** element, add the JavaScript implementation to change the color of the title. To do this, paste the following bolded JavaScript code below the jQuery reference.
     1. HTML
     2. <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
     3. <script type="text/javascript" src="Scripts/jquery-1.4.1.js"></script>
     4. **<script type="text/javascript">**
     5. **$(document).ready(function () {**
     6. **$("#btnChangeTitleStyle").click(function () {**
     7. **$("h2").css("color", "red");**
     8. **});**
     9. **});**
     10. **</script>**
     11. </asp:Content>
     12. **Note:** The above implementation is fully implemented using jQuery, below you can find a brief explanation about what the preceding code does:
     13. When the document is loaded, an anonymous function is registered that in turn registers the click event functionality for the **btnChangeTitleStyle** button. When the button is clicked, the function uses [jQuery selectors](http://docs.jquery.com/Selectors) to set the color property on all the **h2** elements to red.
     14. You can check the [official jQuery web site](http://jquery.com/) where you can find a lot of [tutorials](http://docs.jquery.com/Tutorials) to get started with it.

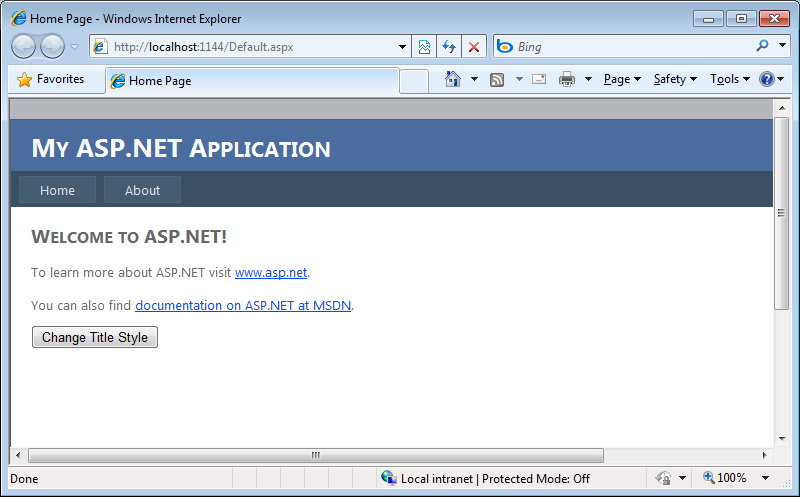
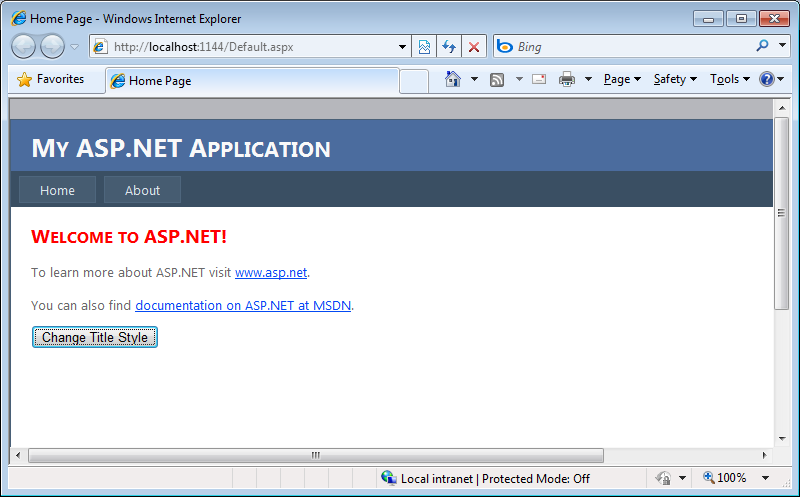
# Next Step

* 1. Exercise 1: Verification

## Exercise 1: Verification

* 1. In order to verify that you have correctly performed all steps of exercise one, proceed as follows:

### Verification 1

* 1. In this verification you will see how your code implementation works changing the Title color to red.
  2. Press **F5** to debug your Web Application. Your default browser should be launched displaying the default page.
     1. 
     2. Figure 9
     3. Default page of your Web Application created with the New Project Template
     4. **Note:** Notice that the style of your application is not the same as the provided in previous version of Visual Studio. This style is defined in the Site.css file, which is provided as part of the new templates.
  3. Click the **Change Title Style** button at the bottom of the page.
  4. Check that the “Welcome to ASP.NET!” title’s color has changed to red.
     1. 
     2. Figure 10
     3. Title’s color modified to red using JavaScript and jQuery

# Next Step

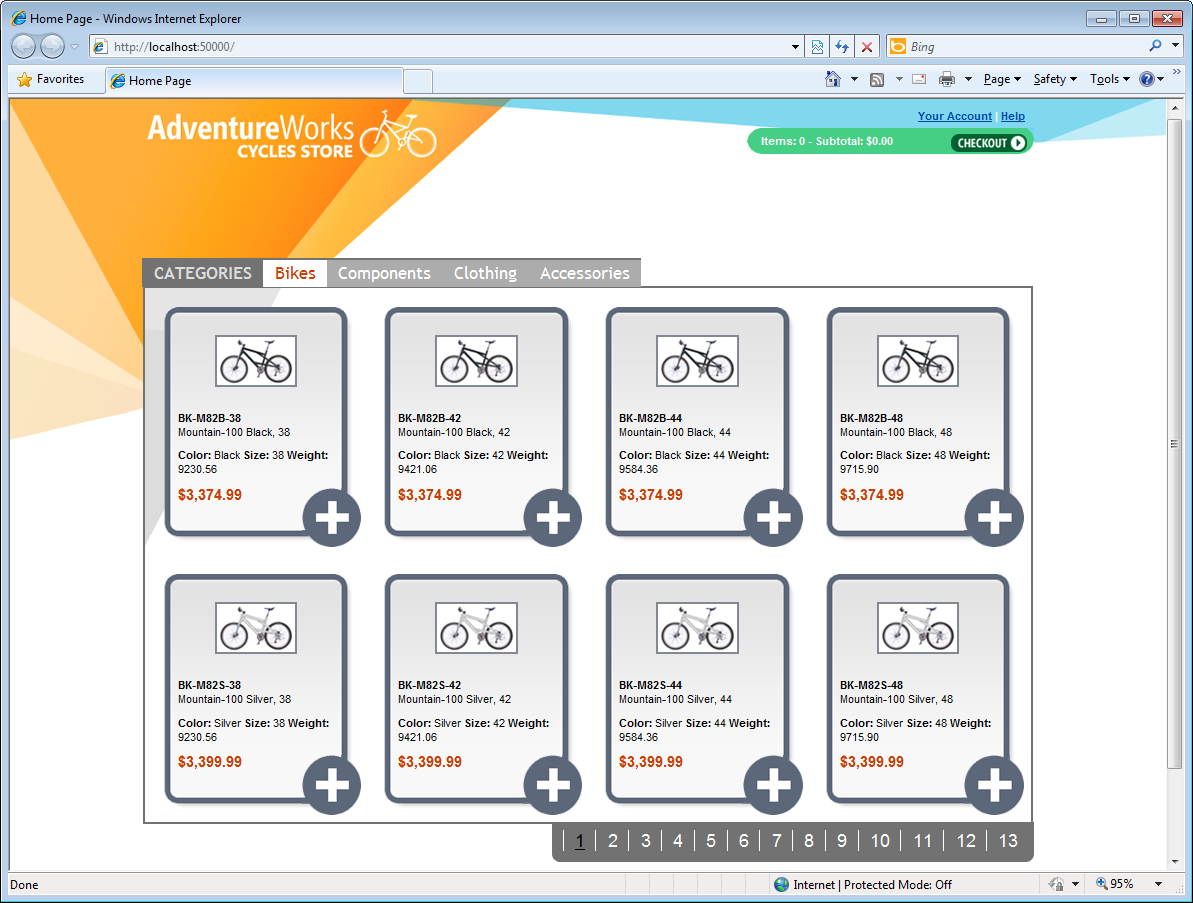
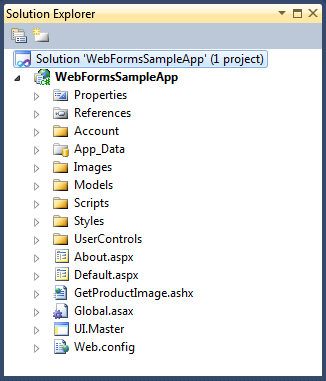
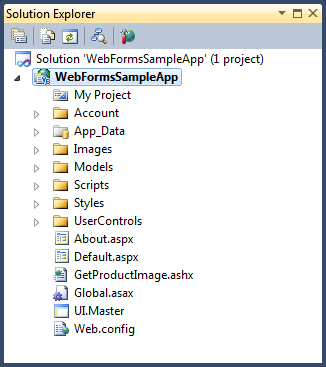
* 1. Exercise 2: Controlling Server Control ClientIDs

Exercise 2: Controlling Server Control ClientIDs

* 1. In this exercise, you will learn how to control the client side IDs that are generated from ASP.NET server controls by the framework. Previously the framework would modify the client side IDs to uniquely identify each control. This sometimes left you with the ID you defined in markup or sometimes left you with something that looks like this, "ctl00\_MasterPageBody\_ctl01\_Textbox1".
  2. The modification of the client side id property works great to ensure that each element is uniquely identified, however, to anyone that has tried to do any sort of client side scripting this becomes very frustrating. Chances are that if you have worked in ASP.NET for any time at all you have run into this issue. The problem is that until runtime you do not know what the client side ID could be, making it difficult to do any kind of client side scripting. In addition, any modification of the page, adding removing controls, can result in a different client side ID being generated.
  3. Again, if you have worked with ASP.NET for any amount of time you know there is a work around for this issue. Each control has a property called **ClientID** that is a read only and supplies the unique client side ID. You could use this in a code behind when dynamically adding scripts or more commonly use inline code (old ASP style) to supply the value to and client side scripts.
  4. JavaScript
  5. <script type="text/javascript">
  6. function DoSomething(){
  7. alert('<%= Control.ClientID %>');
  8. }
  9. </script>
  10. ASP.NET 4 Web Forms addresses this need by providing four ClientID ‘modes’, giving the user everything from existing behavior to full control. The controls ID property is modified according to the **ClientIDMode** mode and then used as the client side id.

1. The four modes are:
   * **AutoID:** The default value if **ClientIDMode** is not set anywhere in the control hierarchy. This causes client side IDs to behave the way they did in version 2.0 (3.0 and 3.5 did not change this code path) of the framework. This mode will generate an ID similar to "ctl00\_MasterPageBody\_ctl01\_Textbox1".
   * **Inherit:** This is the default behavior for every control. This looks to the controls parent to get its value for **ClientIDMode**. You do not need to set this on every control as it is the default, this is used only when the **ClientIDMode** has been changed and the new desired behavior is to inherit from the controls parent.
   * **Static:** This mode does exactly what you think it would; it makes the client side ID static. Meaning that what you put for the ID is what will be used for the client side ID. Warning, this means that if a static **ClientIDMode** is used in a repeating control the developer is responsible for ensuring client side ID uniqueness.
   * **Predictable:** This mode is used when the framework needs to ensure uniqueness but it needs to be done so in a predictable way. The most common use for this mode is on databound controls. The framework will traverse the control hierarchy prefixing the supplied ID with its parent control ID until it reaches a control in the hierarchy whose **ClientIDMode** is defined as static. In the event that the control is placed inside a databound control a suffix with a value that identifies that instance will also be added to the supplied ID. The **ClientIDRowSuffix** property is used to control the value that will be used as a suffix. This mode will generate an ID similar to "Gridview1\_Label1\_0".
   1. For more information, see [Setting Client IDs](http://www.asp.net/LEARN/whitepapers/aspnet4/default.aspx#_TOC3_5).
   2. **Note:** To verify that each step is correctly performed, it is recommended to build the solution at the end of each task.

Task 1 – Assigning Static ClientID to ASP.NET Controls

* 1. In this task, you will enable the **Static** **ClientID** mode in several ASP.NET controls within the web application. By doing this, you will be able to reference them seamlessly from client-side code, and from a CSS in future steps.
  2. Open Microsoft Visual Studio 2010 as an Administrator. Right-click on **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.and select Run as administrator.
  3. Open the solution file **WebFormsSampleApp.sln** located under **\Ex02-ClientId\begin\ (Choosing the folder that matches the language of your preference)**.
     1. **Note:** The lab scenario consists of a single page that lists the products filtered by category from **AdventureWorksLT** database, allowing the user to add them to a cart and then submit them by clicking Check Out. The **AdventureWorksLT.mdf** file must be copied from **\AspNetWebForms4\Source\Assets** folder to the **App\_Data** folder of this project.
     2. 
     3. 
     4. Figure 11
     5. Viewing the Web Application in Solution Explorer (C#)
     6. 
     7. Figure 12
     8. Viewing the Web Application in Solution Explorer (VB)
  4. Open the **ShoppingCart** user control in **Source** mode. To do this, in **Solution Explorer**, right-click the **ShoppingCart.ascx** file under the **UserControls** folder, and select **View Markup**.
     1. **Note:** This is the shopping cart where the user will be placing orders. This user control when rendered will have the ability to expand and collapse by taking advantage of the **ClientID** property (**Static** mode) via client side script.
  5. Enable **ClientID** **Static** mode in **ShopCartCollapsed** ASP.NET **Panel**. To do this, replace the current **ShopCartCollapsed asp:Panel** definition with the following highlighted code.
     1. **Note:** This **Panel** will be rendered to the client as a div with the same **Id** as the server control, in this case, **ShopCartCollapsed**.
     2. ASP.NET
     3. **<asp:Panel ID="ShopCartCollapsed" ClientIDMode="Static" runat="server">**
  6. Do the same as in the previous step but with the **ShopCartExpanded** ASP.NET **Panel**.
     1. ASP.NET
     2. **<asp:Panel ID="ShopCartExpanded" ClientIDMode="Static" runat="server">**

Task 2 – Assigning Predictable ClientID to ASP.NET Controls

* 1. In this task, you will assign the **Predictable** **ClientID** mode to the product list items that are retrieved from the database, setting the product id as the **ClientIDRowSuffix**.
  2. **Note:** ASP.NET previously generated its unique IDs to prevent ID collisions, and the most common place for these types of collisions was inside databound controls. **Predictable** mode was principally designed to tackle this problem while working with databound controls.
  3. **Predictable** mode output follows the pattern **[Prefix]\_[ID]\_[Suffix]**, where each parameter represents the following:
  4. - **Prefix:** Underscore-separated list of all parent controls with an explicit ID/ClientID
  5. - **ID:** The repeated item server control Id
  6. - **Suffix:** An optional auto-incrementing number used for repeated items (only applicable when using an **IDataKeysControl**). This parameter is assigned by setting the **ClientIDRowSuffix** property of the databound server control (not on the repeated items). If this property is not set or is not available, the row index will be used in its place.
  7. Setting the **ClientIDRowSuffix** property is only supported by controls that implements a new interface called **IDataKeysControl** (currently implemented by **GridView** and **ListView**). This interface provides the ability to set the **ClientIDRowSuffix** of a child element whose value is based on the data keys of each row.
  8. Assign the **ClientIDRowSuffix** property to the **ListView** that shows the items that were placed in the shopping cart. To do this, open **ShoppingCart.ascx** in **Source** mode, locate the **ShoppingCartItemsLists ListView**, and replace the current control definition with the following highlighted code.
     1. **Note:** The **ProductId** is a property of the class which items will be repeated (**ShoppingCartItem**), and is automatically inserted into the data keys collection when the data source is bound.
     2. ASP.NET
     3. **<asp:ListView ID="ShoppingCartItemsLists" runat="server" ClientIDMode="Static" ClientIDRowSuffix="ProductId">**
     4. **Note:** There are three ways to use the **Predictable** mode, each one of these is defined through the **ClientIDRowSuffix** property that specifies the suffix for each instance.
     5. 1- With no **ClientIDRowSuffix** defined. This is also the behavior for databound controls without a data keys collection (for example, **Repeater** Control). To construct the **ClientId**, ASP.NET will suffix the ID with the row index.
     6. 2- With a **ClientIDRowSuffix** defined. It looks in the databound server control’s data keys collection for the value, and then suffixes the ID with that value.
     7. 3- With a **ClientIDRowSuffix** defined, but using a compound value instead of just one value. Exhibits the same behavior as with one value, but it will suffix the ID with both concatenated values. (For example, ClientIDRowSuffix="ID, Name").
  9. Bound the cart items to the **ShoppingCartItemLists** control. To do this, open the **ShoppingCart.ascx** code-behind file, and add the following highlighted code at the bottom of the **Page\_PreRender** method of the **ShoppingCartControl** class.
     1. **Note:** If you browse the **ShoppingCartItem** class you will view the **ProductId** property that is used to set the **ClientIDRowSuffix** property of the **ListView**.
     2. *(Code Snippet – ASP.NET 4 Web Forms Lab – Page\_PreRender method – C#)*
     3. C#
     4. protected void Page\_PreRender(object sender, EventArgs e)
     5. {
     6. ShoppingCart cart = ShoppingCartFactory.GetInstance();
     7. ExpandedItemsCountLabel.Text = cart.TotalItems.ToString();
     8. CollapsedItemsCountLabel.Text = cart.TotalItems.ToString();
     9. ExpandedTotalLabel.Text = cart.Subtotal.ToString("c");
     10. CollapsedTotalLabel.Text = cart.Subtotal.ToString("c");
     11. this.ShopCartExpandedEmpty.Visible = cart.TotalItems == 0;
     12. this.ShopCartExpandedNonEmpty.Visible = cart.TotalItems != 0;
     13. **ShoppingCartItemsLists.DataSource = cart.Items;**
     14. **ShoppingCartItemsLists.DataBind();**
     15. }
     16. *(Code Snippet – ASP.NET 4 Web Forms Lab – Page\_PreRender method – VB)*
     17. VB
     18. Protected Sub Page\_PreRender(ByVal sender As Object, ByVal e As EventArgs)
     19. Dim cart As ShoppingCart = ShoppingCartFactory.GetInstance()
     20. ExpandedItemsCountLabel.Text = cart.TotalItems.ToString()
     21. CollapsedItemsCountLabel.Text = cart.TotalItems.ToString()
     22. ExpandedTotalLabel.Text = cart.Subtotal.ToString("c")
     23. CollapsedTotalLabel.Text = cart.Subtotal.ToString("c")
     24. Me.ShopCartExpandedEmpty.Visible = cart.TotalItems = 0
     25. Me.ShopCartExpandedNonEmpty.Visible = cart.TotalItems <> 0
     26. **ShoppingCartItemsLists.DataSource = cart.Items**
     27. **ShoppingCartItemsLists.DataBind()**
     28. End Sub
  10. Enable **Predictable** **ClientId** mode in the child elements of the shopping cart **ListView**. To do this, switch back to the markup view of the ShoppingCartControl user control and replace the current **Quantity** and **TotalPrice** **asp:Labels** definition contained in the **ShoppingCartItemLists ListView**, with the following highlighted code.
      1. **Note:** These labels will be rendered to the client as divs, once for each item in the shopping cart. For example, the **ClientId** of the **Quantity** label will be something like "ctrl0\_Quantity\_12", where 12 is the **ProductId** and ctrl0 is the parent control id.
      2. ASP.NET
      3. <asp:ListView ID="ShoppingCartItemsLists" runat="server" ClientIDMode="Static" ClientIDRowSuffix="ProductId">
      4. <ItemTemplate>
      5. <asp:Panel ID="ShoppingCartItem" ClientIDMode="Static" runat="server">
      6. <div class="productColumn">
      7. **<asp:Label ID="Quantity" ClientIDMode="Predictable" runat="server">**
      8. <%#Eval("ProductName")%>&nbsp;(<%#Eval("Quantity")%>)</asp:Label>
      9. </div>
      10. <div class="priceColumn">
      11. **<asp:Label ID="TotalPrice" ClientIDMode="Predictable" runat="server">**
      12. <%# string.Format(System.Globalization.CultureInfo.CurrentUICulture, "{0:c}", Eval("TotalPrice"))%></asp:Label>
      13. </div>
      14. </asp:Panel>
      15. </ItemTemplate>
      16. </asp:ListView>

Task 3 – Assigning Inherit ClientID to ASP.NET Controls

* 1. In this task, you will assign the **Inherit** **ClientID** mode to a **Panel** server control contained in the shopping cart. This will let the control inherit the **ClientIdMode** from the first parent server control that implements the **INamingContainer** interface (in this case the **ShoppingCart** User Control).
  2. **Note:** The **INamingContainer** interface identifies a container control that creates a new ID namespace within a **Page** object's control hierarchy. Any control that implements this interface creates a new namespace in which all child control ID attributes are guaranteed to be unique within an entire application. This is a marker interface only.
  3. For more information, see [INamingContainer](http://msdn.microsoft.com/en-us/library/system.web.ui.inamingcontainer.aspx) interface.
  4. Define the **ClientIDMode** of the parent **ShoppingCart** user control that will be inherited by the child **Panel** server control. To do this, open the **UI.Master** page in **Source** mode, and replace the current **ShoppingCart1** user control definition with the following highlighted code**.**
     1. ASP.NET
     2. ...
     3. **<uc1:ShoppingCart ID="ShoppingCart1" runat="server" ClientIDMode="Static" />**
     4. ...
     5. **Note:** If no **ClientIDMode** had been set, all its child control would have inheritedthe default **ClientIDMode** which is **AutoID**.
  5. Enable **Inherit** **ClientId** mode in the child **Panel** server control. To do this, open **ShoppingCart.ascx** in **Source** mode, and replace the current **ShopCartExpandedNonEmpty** **asp:Panel** definition with the following highlighted code.
     1. ASP.NET
     2. ...
     3. **<asp:Panel ID="ShopCartExpandedNonEmpty" ClientIDMode="Inherit" runat="server">**
     4. <p class="items">
     5. <span>Your cart items:</span>
     6. </p>
     7. ...
     8. **Note**: There are two other possible ways to set the **ClientIdMode**:
     9. - At **Page** level: Defines the default **ClientIdMode** for all controls within the current page. For example:
     10. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" ClientIdMode="Static"%>
     11. - At **Web.config** level: It is also possible to set the **ClientIdMode** in the config section at either machine or application level. For example, this defines the default **ClientIdMode** for all controls within all pages in the application:
     12. <system.web>
     13. <pages clientIdMode="Predictable"></pages>
     14. </system.web>

Task 4 – Targeting Static ClientIDs via CSS and JavaScript

In this task, you will take advantage of the static **ClientID** mode, by manipulating the rendered control with the static **ClientID** through JavaScript, and by applying CSS styles directly to the Id rather than creating an unnecessary CssClass.

* 1. Apply styles to the controls with static **ClientID**s. To do this, open the **main.css** file located under **Styles** folder, and add the following highlighted code:
     1. **Note:** **ShopCartCollapsed** and **ShopCartExpanded** ID attributes are being referencedapplying ID selectors rather than Class selectors. The essential difference is that while class selectors apply to one or more elements on a page, ID selectors apply to exactly one element. For more information, see <http://www.w3.org/TR/CSS21/selector.html#id-selectors>.
     2. CSS
     3. **#ShopCartCollapsed {**
     4. **display: none;**
     5. **background-color: #00FF80;**
     6. **padding: 10px 15px;**
     7. **height: 10px;**
     8. **background: url(../Images/shopcart\_collapsed\_bg.png) no-repeat top left;**
     9. **}**
     10. **#ShopCartCollapsed p.summary span {**
     11. **position: relative; top:-4px;**
     12. **}**
     13. **#ShopCartCollapsed a.checkoutLink {**
     14. **position: absolute; top: 6px; right: 8px;**
     15. **}**
     16. **#ShopCartExpanded {**
     17. **display: none;**
     18. **padding: 10px 15px;**
     19. **height: 90px;**
     20. **background: url(../Images/shopcart\_bg.gif) no-repeat top left;**
     21. **}**
     22. **#ShopCartExpanded p.items {**
     23. **font-size: 11px;**
     24. **color: #666;**
     25. **}**
     26. **#ShopCartExpanded ul.items {**

**height:60px;overflow:auto;**

* + 1. **margin: 0; padding: 0; list-style: none;**
    2. **}**
    3. **#ShopCartExpanded ul.items li {**

**display: inline; margin: 0; padding: 0;**

* + 1. **}**
    2. **#ShopCartExpanded div.ShoppingCartItem {**
    3. **display: block;**
    4. **}**
    5. **#ShopCartExpanded div.productColumn {**
    6. **float: left;**
    7. **width: 150px;**
    8. **}**
    9. **#ShopCartExpanded div.priceColumn {**
    10. **float: right;**
    11. **width: auto;**
    12. **}**
    13. **#ShopCartExpanded p.summary span {**
    14. **position: relative; top:+6px;**
    15. **}**
    16. **#ShopCartExpanded p.empty {**
    17. **text-align: center;**
    18. **font-weight: bold; color: #50d48f;**
    19. **padding-top: 15px;**
    20. **}**
    21. **#ShopCartExpanded a.checkoutLink {**
    22. **position: absolute; top: 89px; right: 8px;**
    23. **}**
  1. Implement the necessary JavaScript code to create the effects for the Shopping Cart using JQuery. To do this, open the **shoppingCart.Effects.js** file located under **Scripts** folder, and follow these steps:
     1. Add the following highlighted code to **CollapseCart** function:
        1. JS
        2. function CollapseCart(withAnimation) {
        3. **if (withAnimation) {**
        4. **$("#ShopCartExpanded").hide();**
        5. **$("#ShopCartCollapsed").show("slow");**
        6. **}**
        7. **else {**
        8. **$("#ShopCartExpanded").css("display", "none");**
        9. **$("#ShopCartCollapsed").css("display", "block");**
        10. **}**
        11. **$("#ShoppingCartState").val("collapsed");**
        12. }
     2. Add the following highlighted code to **ExpandCart** function:
        1. JS
        2. function ExpandCart(withAnimation) {
        3. **if (withAnimation) {**
        4. **$("#ShopCartCollapsed").hide();**
        5. **$("#ShopCartExpanded").show("slow");**
        6. **}**
        7. **else {**
        8. **$("#ShopCartCollapsed").css("display", "none");**
        9. **$("#ShopCartExpanded").css("display", "block");**
        10. **}**
        11. **$("#ShoppingCartState").val("expanded");**
        12. }
     3. Add the following highlighted code to **$(document).ready** function:
        1. JS
        2. $(document).ready(function() {
        3. // Preload expanded Shopping Cart background image
        4. $("<img>").attr("src", "Images/shopcart\_bg.gif");
        5. **$("#ShopCartCollapsed").click(function() { ExpandCart(true) });**
        6. **$("#ShopCartExpanded").click(function() { CollapseCart(true) });**
        7. if ($("#ShoppingCartState").val() == "expanded") {
        8. ExpandCart(false);
        9. }
        10. else {
        11. CollapseCart(false);
        12. }
        13. });
  2. Add a reference to **ShoppingCart.Effects.js** in **Default.aspx**. To do this, open **Default.aspx** in **Source** mode, and add the following highlighted code inside the first **asp:Content** tag.
     1. ASP.NET
     2. ...
     3. <%@ MasterType TypeName="WebFormsSampleApp.Master.UI" %>
     4. <asp:Content ContentPlaceHolderID="HeadContentPlaceHolder" runat="server">
     5. <link type="text/css" rel="Stylesheet" media="screen" href="/Styles/products.css" />
     6. **<script type="text/javascript" src="/Scripts/shoppingCart.Effects.js"></script>**
     7. </asp:Content>
     8. ...

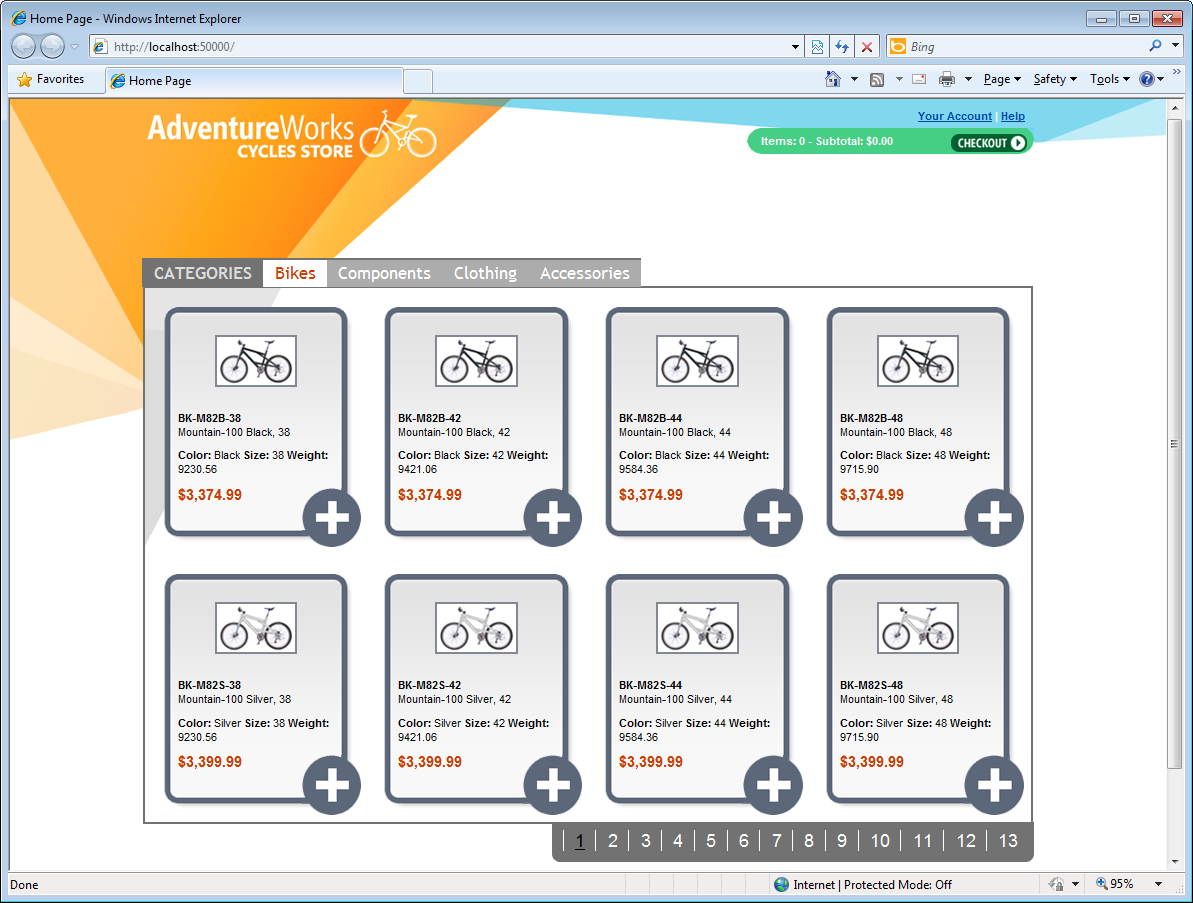
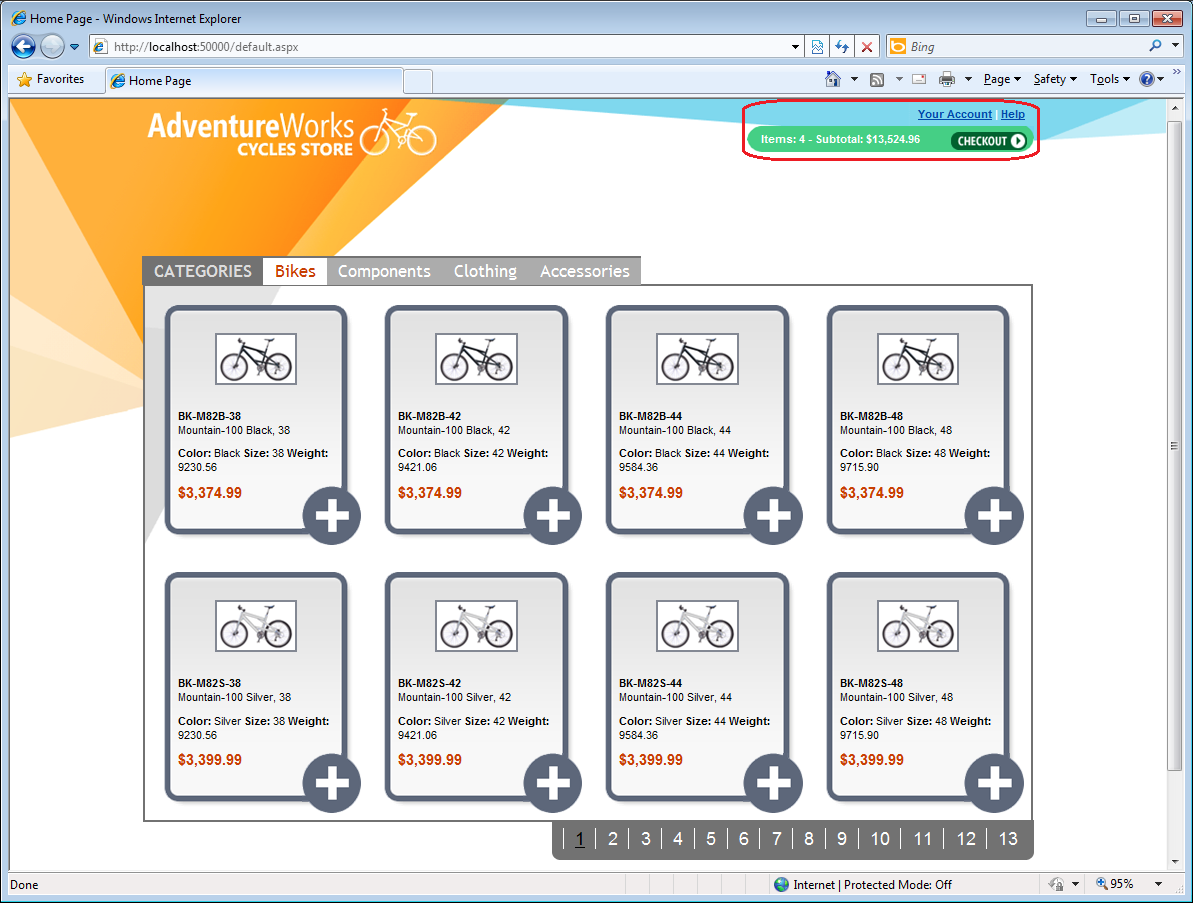
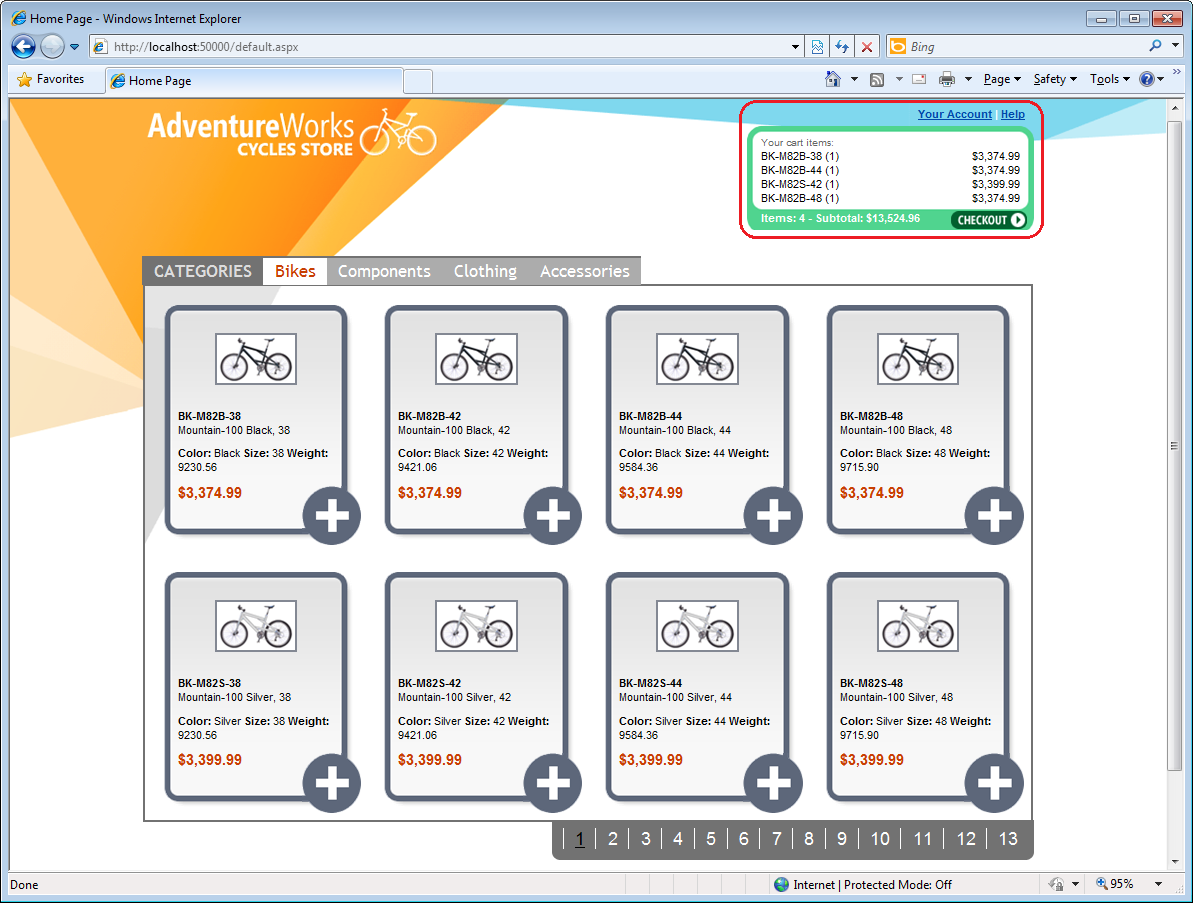
# Next Step

* 1. Exercise 2: Verification

## Exercise 2: Verification

* 1. In order to verify that you have correctly performed all steps of exercise two, proceed as follows:

### Verification 1

* 1. In this verification you will see how **Static** **ClientID**s are being targeted from CSS and JavaScript. You will see how styles are applied to the shopping cart, and how it is expanded and collapsed with fancy effects. All this, by leveraging the capability of referencing the rendered controls static id with client side code.
  2. Start a new instance of the **WebFormsSampleApp** project. To do this, in **Solution Explorer** right-click **WebFormsSampleApp** project, point to **Debug** and select **Start new instance**.
     1. **Note:** If the **Debugging Not Enabled** dialog appears, select **Modify the Web.config file to enable debugging**, and click **OK**.
     2. 
     3. Figure 13
     4. Viewing the Default page
  3. Inspect the page’s source code to check that the **asp:Panels** with static **ClientID** mode were indeed rendered with the same Id as the server control. To do this, right-click over the page on your browser, and select **View Source**.
     1. **Note:** A text editor should appear, showing the source code of the rendered **Default.aspx** page.
  4. Find in the page’s source code two **<div>** elements with Id **ShopCartCollapsed** and **ShopCartExpanded**. These **<div>** elements represent the shopping cart’s expanded and collapsed views.
     1. **Note:** ASP.NET has rendered the **asp:Panel** server controls to **<div>** html elements, preserving the Id that uniquely identifies them.
     2. 
     3. Figure 14
     4. Viewing the rendered <div> elements with static ClientID
  5. Place several products into the shopping cart by clicking the plus () symbol next to them. Notice how the collapsed view of the shopping cart is updated with information of the new products.
     1. 
     2. Figure 15
     3. Adding products to the shopping cart
  6. Expand and collapse the shopping cart. To do this, make a click over the green panel at the top-right of the page.
     1. **Note:** Expanding and collapsing the shopping cart can be done thanks that you know beforehand how these **<div>** elements are going to be called. In this case, apart from applying CSS styles directly using ID selectors, you are also referencing the **<div>** elements from JavaScript to expand and collapse them with animation.
     2. 
     3. Figure 16
     4. Expanding and collapsing the shopping cart

### Verification 2

* 1. In this verification, you will see how **Predictable** and **Inherit** **ClientID**s are being rendered, by inspecting the source from client side.
  2. Make sure that the shopping cart contains several products.
  3. Inspect the default page’s source code to view how **Predictable** **ClientID**s are being rendered. To do this, follow these steps:
     1. Find in the page’s source a **<div>** element with Id **ShopCartExpandedNonEmpty**.
        1. **Note:** Thiselement represents the **asp:Panel** that contained the **ListView** with the **ClientIDSuffix** (set to *ProductId*) to be assigned in the child elements. While rendering the page, the **ListView** iterated over its items, and replaced their ids using the predictable pattern seen before.
        2. Also notice that **ShopCartExpandedNonEmpty** was the panel in which was applied the **Inherit** **ClientID** mode. The fact that the id of this **<div>** element preserved the original **asp:Panel** id, demonstrates that the control had inherited the **ClientIDMode** of the parent control which was **Static**.
     2. Inspect all the **ShoppingCartItem**s that were generated by the **ListView**, and view the **<span>** elements (formerly **asp:Labels**) that show **Quantity** and **TotalPrice** information of each product in the shopping cart.
        1. 
        2. Figure 17
        3. Viewing the generated Predictable ClientIDs



# Next Step

* 1. Exercise 3: Enabling Bi-Directional Routing Support

Exercise 3: Enabling Bi-Directional Routing Support

* 1. In this exercise, you will learn how to take full advantage of the common **ASP.NET Routing** engine that allows you to customize the URLs that your application exposes. In addition, you will use the new expression builders that allow generating dynamic URLs that are based on your route definitions, alleviating from having to fixed static links. This feature provides full class support by allowing you to define any custom route to a web form page.
  2. By using **ASP.NET Routing** and the new bi-directional support, users can decouple URLs from a physical Web Form, having friendlier URLs and leveraging the power of search engines to discover and use them.
  3. **Note:** To verify that each step is correctly performed, it is recommended to build the solution at the end of each task.

Task 1 – Enabling ASP.NET Routing in your Application

* 1. In this task, you will enable **ASP.NET Routing** engine in your Web Forms application by adding the **UrlRouting** HTTP Module, and creating routes to specify the Url pattern to match.
  2. Open Microsoft Visual Studio 2010 as an Administrator. Right-click on **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.and select Run as administrator.
  3. Open the solution file WebFormsSampleApp.sln located under **Ex03-Routing\begin\ (choosing the folder that matches the language of your preference)**
     1. **Note:** Alternatively, you may continue working with the solution obtained after completing the previous exercise.
  4. In **Web.config** file, add the **UrlRouting** HTTP Module. To do this, add the following highlighted element inside the **<httpModules>** node.
     1. Web.config
     2. ...
     3. <system.web>
     4. ...
     5. <httpmodule>
     6. **<add name="RoutingModule" type="System.Web.Routing.UrlRoutingModule"/>**
     7. <add name="ScriptModule" type="System.Web.Handlers.ScriptModule, System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
     8. </httpmodule>
     9. ...
     10. <system.web>

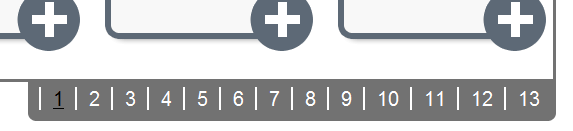
...

* + 1. **Note:** The **UrlRoutingModule** class is a basic HTTP Module that matches an incoming HTTP request to a route in an ASP.NET application. The module iterates through all the defined routes searching for one that has a URL pattern that matches the format of the HTTP request. When the module finds a matching route, it retrieves the **IRouteHandler** object for that route. From the route handler, the module gets an **IHttpHandler** object and uses that as the HTTP handler for the current request.
    2. For more information, see [UrlRoutingModule](http://msdn.microsoft.com/en-us/library/system.web.routing.urlroutingmodule.aspx) class.
  1. In **Global.asax**, replace all the namespace directives created by default with the following code.
     1. C#
     2. **using System;**
     3. **using System.Web.Routing;**
     4. VB
     5. **Imports System**
     6. **Imports System.Web.Routing**
  2. Specify the routes **{category}** and **{category}/{page}** to be handled by the **Default.aspx** page. To do this, add the following bolded code to the **Application\_Start** method.
     1. *(Code Snippet – ASP.NET 4 Web Forms Lab – Application\_Start method – C#)*
     2. C#
     3. protected void Application\_Start(object sender, EventArgs e)
     4. {
     5. **RouteTable.Routes.Add("Category", new Route("{category}", new PageRouteHandler("~/Default.aspx")));**
     6. **RouteTable.Routes.Add("CategoryAndPage", new Route("{category}/{page}", new PageRouteHandler("~/Default.aspx")));**
     7. }
     8. *(Code Snippet – ASP.NET 4 Web Forms Lab – Application\_Start method - VB)*
     9. VB
     10. Private Sub Application\_Start(ByVal sender As Object, ByVal e As EventArgs)
     11. **RouteTable.Routes.Add("Category", New Route("{category}", New PageRouteHandler("~/Default.aspx")))**
     12. **RouteTable.Routes.Add("CategoryAndPage", New Route("{category}/{page}", New PageRouteHandler("~/Default.aspx")))**
     13. End Sub
     14. **Note:** The **RouteTable** class is one of the primary classes in the ASP.NET Routing engine. It is a central location that stores the defined URL routes for an application.
     15. You can add routes to the **RouteTable** by providing a name to identify them, and a concrete implementation of the **RouteBase** class, in this case the **Route** class.
     16. Routes are URL patterns that are used for processing requests, and can be also used to construct URLs dynamically. The **Route** class enables you to specify how routing is processed in an ASP.NET application. You create a **Route** object for each URL pattern that you want to map to a class that can handle requests that correspond to that pattern.
     17. The previous code uses the new **PageRouteHandler** class to map incoming request with a Page. This class is the one that allows the integration of Web Forms and ASP.NET Routing.
     18. For more information, see [RouteTable](http://msdn.microsoft.com/en-us/library/system.web.routing.routetable.aspx), [RouteBase](http://msdn.microsoft.com/en-us/library/system.web.routing.routebase.aspx) and [Route](http://msdn.microsoft.com/en-us/library/system.web.routing.route.aspx) classes.

Task 2 – Using RouteUrlExpressionBuilder to Modify Navigation Links

* 1. In this task, you will modify the navigation links of the application to use the routes defined in the previous task. You are going to take advantage of the new **RouteUrlExpressionBuilder** by adding bi-directional routing support to your application. This means that you will be able to generate dynamic URLs that are based on your route definitions, making it easier to manage all the registered routes within your ASP.NET pages without having to write fixed static links.
  2. Enable the **RouteUrlExpressionBuilder** in your application. To do this, in the **Web.config** file, add the following highlighted **<expressionBuilders>** node, inside the **<compilation>** node.
     1. Web.config
     2. ...
     3. <system.web>
     4. ...
     5. <compilation debug="true" targetFramework="4.0" >
     6. **<expressionBuilders>**
     7. **<add expressionPrefix="RouteUrl" type="System.Web.Compilation.RouteUrlExpressionBuilder"/>**
     8. **</expressionBuilders>**
     9. </compilation>
     10. ...
     11. <system.web>
     12. ...

**Note:** Expression builders parse declarative expressions and create code to retrieve values that are bound to a control property. In no-compile scenarios, an expression builder that supports a no-compile feature evaluates the expression during run time.

* 1. Change the code to generate the category navigation links in the **Default.aspx** page to use the new defined routes. To do this, in **Solution Explorer,** double-click **Default.aspx**, and replace the value of the **NavigateUrl** attribute of all the category **HyperLink** controls with the following.
     1. Default.aspx
     2. ...
     3. <ul id="categoryTabs">
     4. <li>
     5. <asp:HyperLink runat="server"
     6. **NavigateUrl="<%$ RouteUrl:RouteName=Category, category=Bikes %>"**
     7. OnLoad="CategoryLink\_Load" Text="Bikes" />
     9. <asp:HyperLink runat="server"
     10. **NavigateUrl="<%$ RouteUrl:RouteName=Category, category=Components %>"**
     11. OnLoad="CategoryLink\_Load" Text="Components" />
     13. <asp:HyperLink runat="server"
     14. **NavigateUrl="<%$ RouteUrl:RouteName=Category, category=Clothing %>"**
     15. OnLoad="CategoryLink\_Load" Text="Clothing" />
     17. <asp:HyperLink runat="server"
     18. **NavigateUrl="<%$ RouteUrl:RouteName=Category, category=Accessories %>"**
     19. OnLoad="CategoryLink\_Load" Text="Accessories"/>
     20. </li>
     21. </ul>
     22. ...
     23. **Note:** When the page parser encounters an expression that is delimited with the string **<%$ %>**, it creates an expression builder for the expression based on the prefix in the string. The prefix is the portion of the string that is to the left of the colon (:) and is defined in the **Web.config**. In this case, the prefix for the **RouteUrlExpressionBuilder** is **RouteUrl**.
     24. The **RouteUrlExpressionBuilder** then generates a route previously registered in the **RouteTable** based on the parameter to the right of the colon (:), in this case, a **Category** route.
  2. Change the code to generate the pager navigation links in the **Default.aspx** code-behind, to use new defined routes. To do this, in **Solution Explorer** right-click **Default.aspx**, select **View Code**, and replace the last two lines of the **CreatePagerLinks** method with the following highlighted code.
     1. **Note:** As the database retrieves lots of results, is convenient to paginate the results in different pages. Page links are generated dynamically depending on the amount of results retrieved.
     2. ****
     3. The type of route to create in this case will be **CategoryAndName**. For example, a possible link could be **/Products/3**, where *Products* is the category and *3* is the page number to retrieve.
     4. *(Code Snippet – ASP.NET 4 Web Forms Lab – Create Pager Links – C#)*
     5. C#
     6. private void CreatePagerLinks()
     7. {
     8. for (int i = 1; i <= this.TotalPages; i++)
     9. {
     10. HyperLink link = new HyperLink() { Text = i.ToString() };
     11. if (i == this.SelectedPage)
     12. {
     13. link.CssClass = "currentPage";
     14. }
     15. PagerPanel.Controls.Add(link);
     16. **string expression = String.Format("RouteName={0}, category={1}, page={2}", "CategoryAndPage", this.SelectedCategoryName, i);**
     17. **link.NavigateUrl = RouteUrlExpressionBuilder.GetRouteUrl(this, expression);**
     18. }
     19. }
     20. *(Code Snippet – ASP.NET 4 Web Forms Lab – Create Pager Links – VB)*
     21. VB
     22. Private Sub CreatePagerLinks()
     23. For i As Integer = 1 To Me.TotalPages
     24. Dim link As New HyperLink() With {.Text = i.ToString()}
     25. If i = Me.SelectedPage Then
     26. link.CssClass = "currentPage"
     27. End If
     28. PagerPanel.Controls.Add(link)
     29. **Dim expression As String = String.Format(CultureInfo.InvariantCulture, "RouteName={0}, category={1}, page={2}", "CategoryAndPage", Me.SelectedCategoryName, i)**
     30. **link.NavigateUrl = RouteUrlExpressionBuilder.GetRouteUrl(Me, expression)**
     31. Next i
     32. End Sub
     33. **Note:** You can use the **RouteUrlExpressionBuilder** directly from your code behind by calling its **GetRouteUrl** static method. In this way, you can dynamically assign values to your route’s parameters.

Task 3 – Retrieving Route Parameter Values

* 1. In this task, you will change the behavior in which you retrieve the category name and page index parameters in every post back. As you are now working with routes, these parameters will no longer be available in the **QueryString** collection. Instead, you will use a new property defined in the **Page** class called **RouteData**, which have a key-value collection that includes all the parameters that are part of the route.
  2. Open the **Default.aspx** code-behind file. To do this, in **Solution Explorer** right-click **Default.aspx**, and select **View Code**.
  3. Replace the usage of **Request.QueryString** collection with the **RouteData.Values** collection in the **GetCategoryName** and **GetPageIndex** methods.
     1. C#
     2. ...
     3. private string GetCategoryName()
     4. {
     5. **string category = RouteData.Values["category"] as string;**
     6. AdventureWorksRepository repository = new AdventureWorksRepository();
     7. if (category != null)
     8. {
     9. return category;
     10. }
     11. return repository.GetCategories()[0].Name;
     12. }
     13. private int GetPageIndex()
     14. {
     15. **string page = RouteData.Values["page"] as string;**
     16. if (page != null)
     17. return Convert.ToInt32(page);
     18. return 1;
     19. }
     20. ...
     21. VB
     22. Private Function GetCategoryName() As String
     23. **Dim category As String = TryCast(RouteData.Values("category"), String)**
     24. Dim repository As New AdventureWorksRepository()
     25. If category IsNot Nothing Then
     26. Return category
     27. End If
     28. Return repository.GetCategories()(0).Name
     29. End Function
     30. Private Function GetPageIndex() As Integer
     31. **Dim page As String = TryCast(RouteData.Values("page"), String)**
     32. If page IsNot Nothing Then
     33. Return Convert.ToInt32(page)
     34. End If
     35. Return 1
     36. End Function
     37. **Note:** The key-value collection of the **RouteData** property contains values that are parsed from the URL.
     38. For more information, see [RouteData](http://msdn.microsoft.com/library/system.web.routing.routedata.aspx) class and its [members](http://msdn.microsoft.com/library/system.web.routing.routedata_members.aspx).

Task 4 – Using RouteValueExpressionBuilder to Retrieve Route Values

* 1. In this task, you will learn how to obtain the values of the route parameters directly from your ASP.NET page. To show this feature, you will add some messages to the **Default.aspx** to be displayed every time a requested product is not found, or when a requested page is out of bounds. You are going to take advantage of the new **RouteValueExpressionBuilder** to get those values from the current route and warn the user with a friendly message.
  2. In **Web.config** file, add the **RouteValueExpressionBuilder** to the **<expressionBuilders>** node.
     1. Web.config
     2. ...
     3. <system.web>
     4. ...
     5. <compilation debug="true" targetFrameworkMoniker=".NETFramework,Version=v4.0">
     6. <expressionBuilders>
     7. <add expressionPrefix="RouteUrl" type="System.Web.Compilation.RouteUrlExpressionBuilder"/>
     8. **<add expressionPrefix="RouteValue" type="System.Web.Compilation.RouteValueExpressionBuilder" />**
     9. </expressionBuilders>
     10. </compilation>
     11. ...
     12. <system.web>
     13. ...
  3. Add messages for non-existing category and page number in the **Default.aspx** page using the **RouteValue** expression builder. To do this, open **Default.aspx** in **Source** view, and replace the content of the **PageIndexOverflowPanel** and **NoProductsFoundPanel** panels with the following.
     1. **Note:** If you inspect the code-behind file you will see a method called **ApplyProductsFilter** that contains the logic to make visible one of these panels accordingly.
     2. Default.aspx
     3. ...
     4. <asp:Panel ID="PageIndexOverflowPanel" runat="server" Visible="false">
     5. <div class="noResults">
     6. **The <strong><asp:Literal runat="server" Text="<%$ RouteValue:category%>" /></strong> category does not have the page <strong><asp:Literal ID="Literal1" runat="server" Text="<%$ RouteValue:page%>" /></strong>.**
     7. </div>
     8. </asp:Panel>
     9. <asp:Panel ID="NoProductsFoundPanel" runat="server" Visible="false">
     10. <div class="noResults">
     11. **No products were found matching the <strong><asp:Literal runat="server" Text="<%$ RouteValue:category%>" /></strong> category you have selected.**
     12. </div>
     13. </asp:Panel>
     14. ...
     15. **Note:** The **RouteValueExpressionBuilder** allows you to get a route parameter by the name you defined when you registered the route. This name is the one to the right of the colon (:). For example, the expression **<%$ RouteValue:category%>** can be understood as, give me the value of the parameter whose name is *category* on the current route.

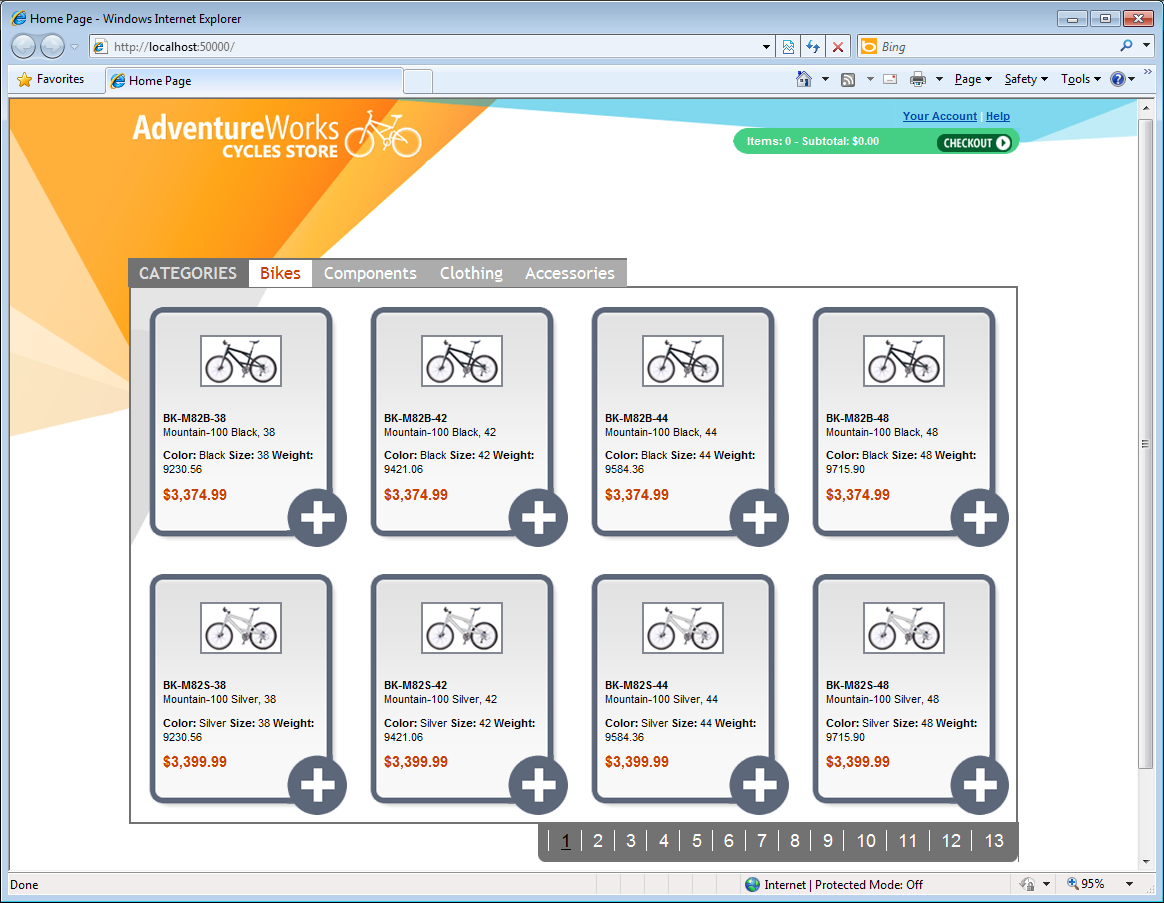
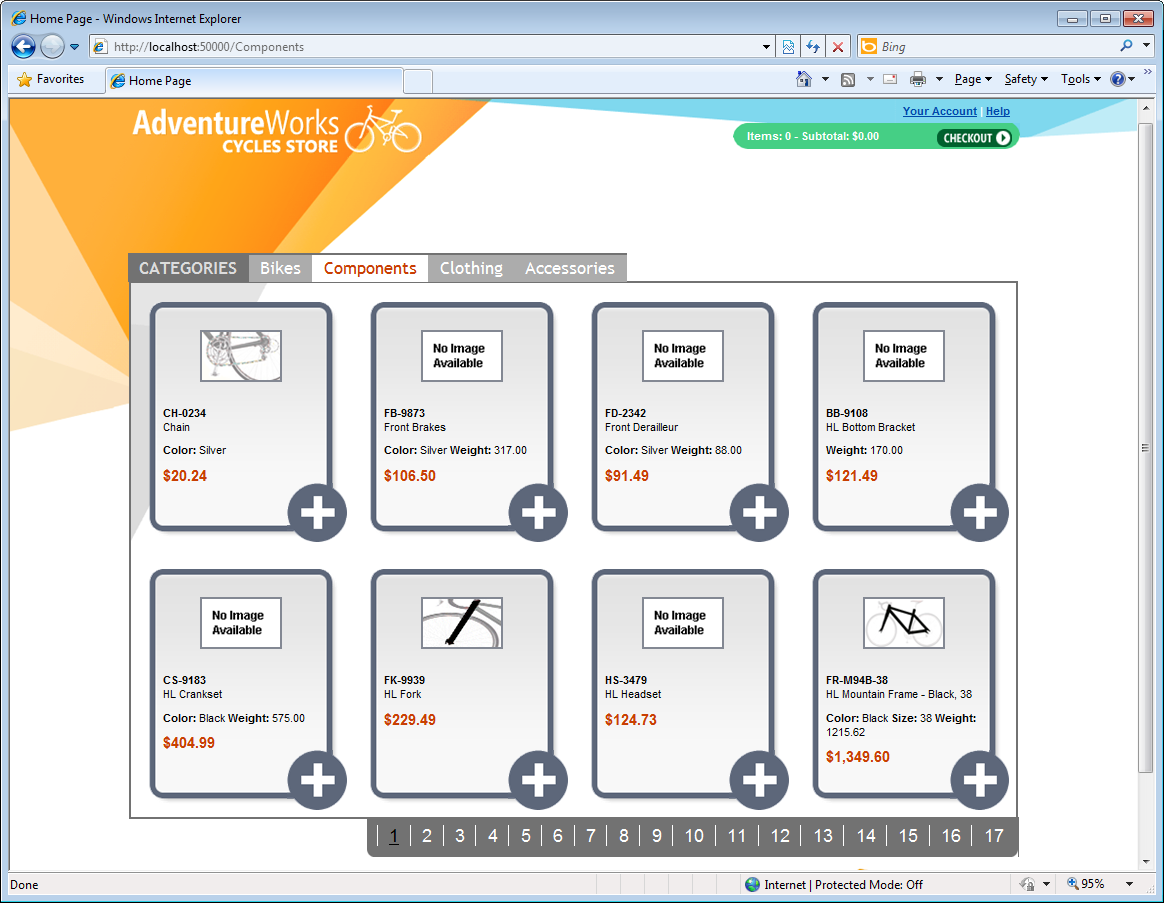
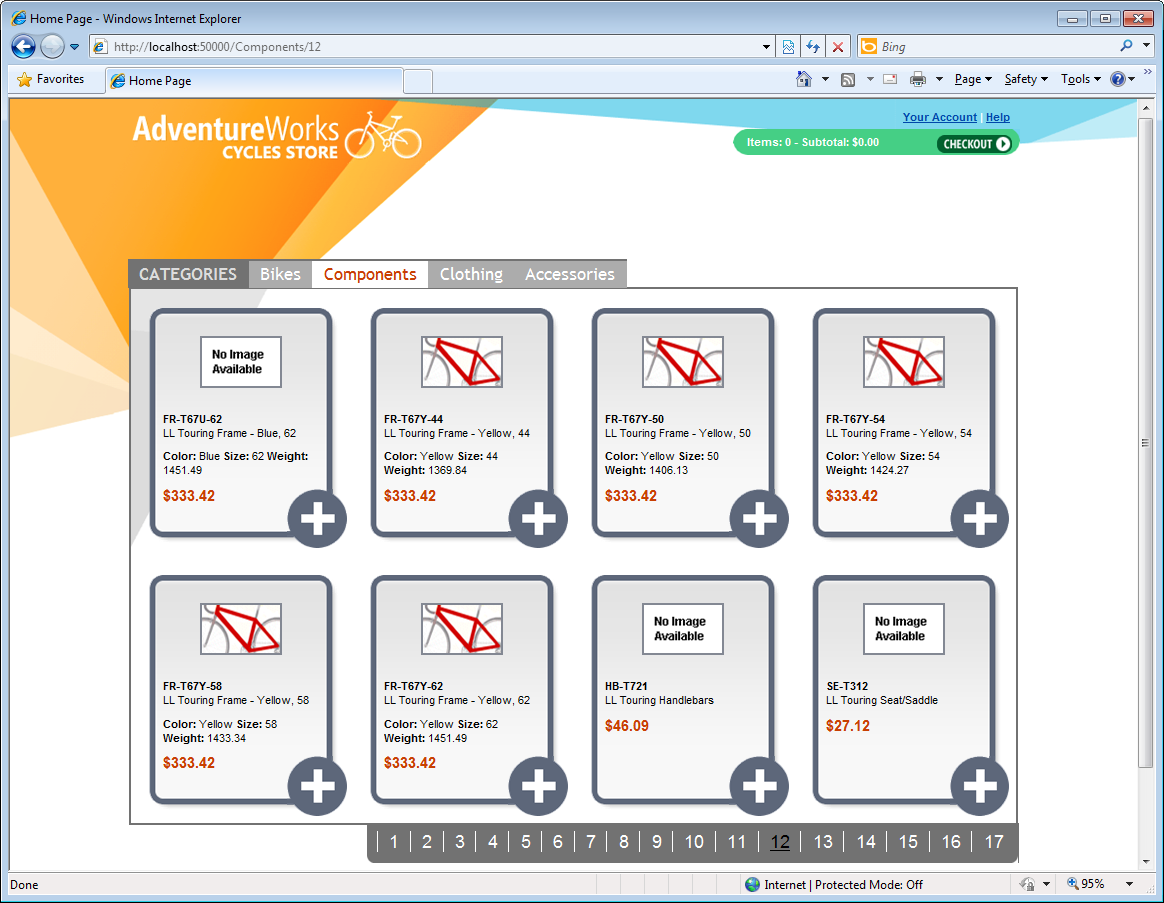
# Next Step

Exercise 3: Verification

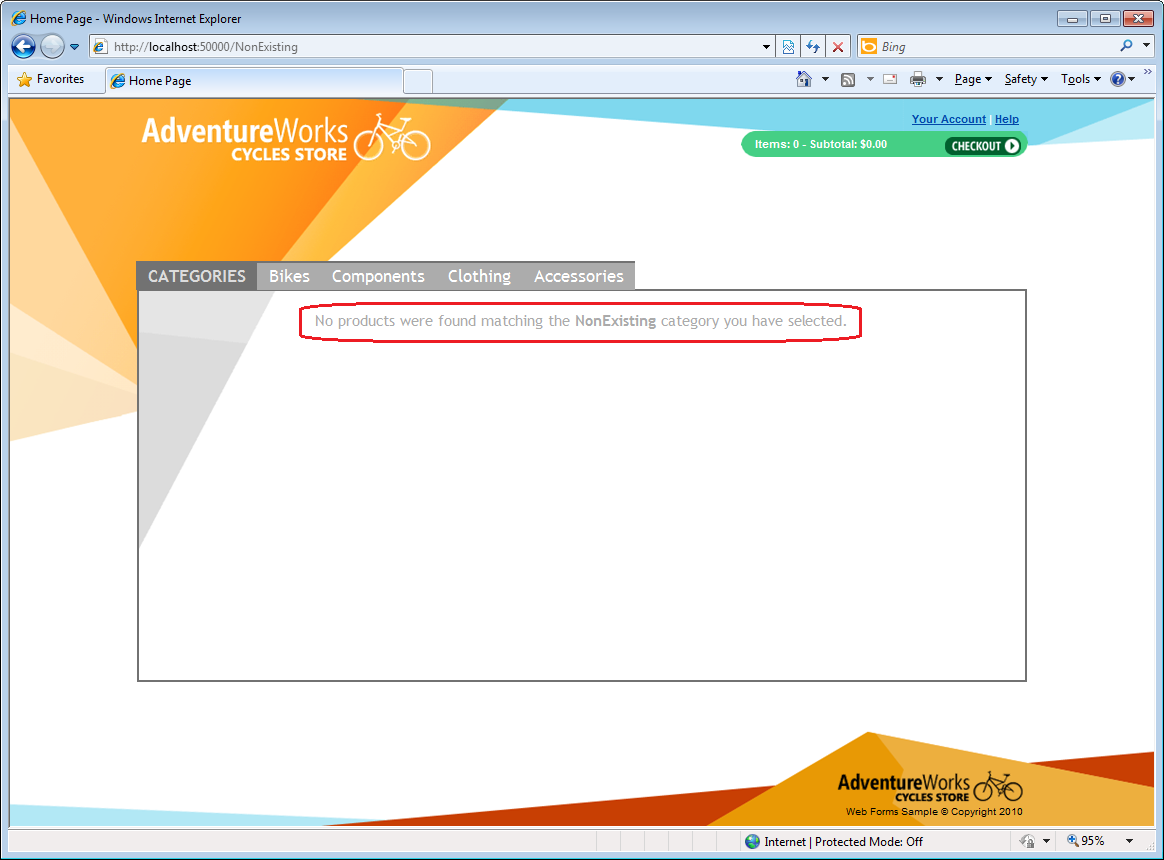
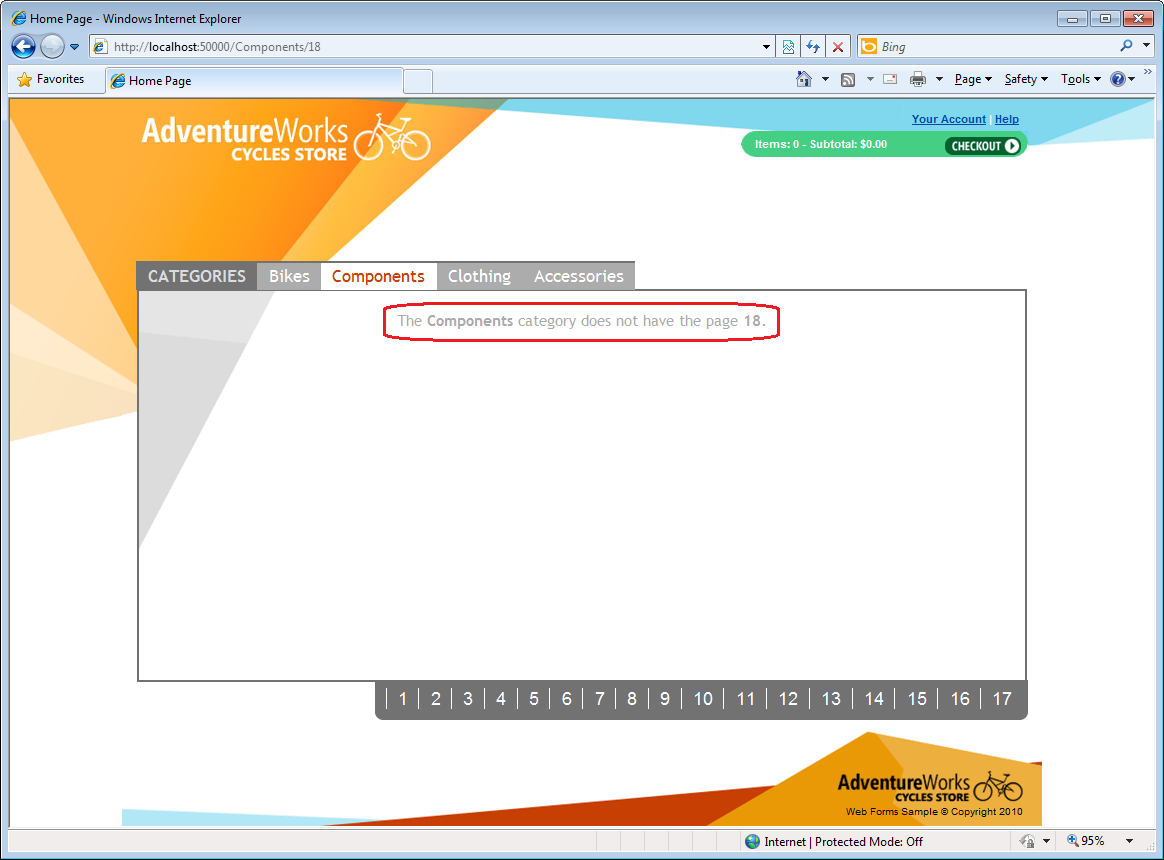
## Exercise 3: Verification

* 1. In order to verify that you have correctly performed all steps of exercise three, proceed as follows:

### Verification 1

* 1. In this verification, you will **navigate** through the **WebFormsSample** application using the updated category and pager links to see the newly introduced routes.
  2. Start a new instance of the **WebFormsSampleApp** project. To do this, in **Solution Explorer** right-click **WebFormsSampleApp** project, point to **Debug** and select **Start New Instance**.
     1. **Note:** If the dialog **Debugging Not Enabled** appears, select **Modify the Web.config file to enable debugging** and click **OK**.
     2. 
     3. Figure 18
     4. Viewing the Default page
  3. Browse Components category by clicking **Components** link on page header. You will be redirected to the following address in the web browser <http://localhost:50000/Components>. The following output should appear. Page one is showed by default.
     1. 
     2. Figure 19
     3. Viewing the products of the Components category
     4. **Note:** This route maps with the **Category** route added to the **RouteTable** in the **Global.asax**.
  4. To browse another page of the **Components** category, click on a page link at the bottom of the page. For example, if you click **12**, you will be redirected to the following address in the web browser <http://localhost:50000/Components/12>.
     1. 
     2. Figure 20
     3. Viewing the products in the page 12 of the Components category
     4. **Note:** This route maps with the **CategoryAndPage** route added to the **RouteTable** in the **Global.asax**.

### Verification 2

* 1. In this verification, you will check the non-existing category and non-existing page message errors generated using the **RouteValueExpressionBuilder**. You will request two pages, one with a **Category** that does not exist, and other with an out of range page index.
  2. Type a Url in the Browser that contains a non-existing category. For example request the <http://localhost:50000/NonExisting> page. You will see the message "No products were found matching the **NonExisting** category you have selected".
     1. 
     2. Figure 21
     3. Category not found message
  3. Type a Url in the Browser containing an out of range page index. For example request the <http://localhost:50000/Components/18> page(the Components category contains only 17 pages). You will see the message “The **Components** category does not have the page **18**”.
     1. 
     2. Figure 22
     3. PageIndex not found message



# Next Step

* 1. Exercise 4: Granular View State

Exercise 4: Granular View State

* 1. WebForms 4 includes the new **ViewStateMode** property in the **Control** class to provide a more granular control over the View State. Granular control of View State means that you can turn it off at the page, and just turn it on for the controls you want. Versus turning it on, and turning it off a million places. So it should be much easier to say: I want it off for the page, turn it on for these three controls, and it’s out of your way.

Task 1 – Disabling ViewState on Controls

* 1. In this task, you will disable ViewState at Page level in Default.aspx, and at Control level in ShoppingCart.ascx. This will make possible to have a granular ViewState control in future steps.
  2. Open Microsoft Visual Studio 2010 as an Administrator. Right-click on **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Microsoft Visual Studio 2010**.and select Run as Administrator.
  3. Open the solution file **WebFormsSampleApp.sln** located under **\Ex04-ViewState\begin\** **(choosing the folder that matches the language of your preference)**
     1. **Note:** Alternatively, you may continue working with the solution obtained after completing the previous exercise.
     2. You will be using this unaltered begin solution in the Verification section, so consider working on a copy of this begin solution, in order to conserve the original one intact.
  4. Disable ViewState at **Page** level in **Default.aspx**. To do this, open **Default.aspx** in **Markup** mode, and replace the **<% Page %>** directive for the following one.
     1. **Note:** This will result in disabling ViewState for all the child controls in the page. In later steps, you will see how to leverage granular ViewState by enabling it just for the controls that you need within that page.
     2. ASP.NET (C#)

**<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/UI.master" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="WebFormsSampleApp.\_Default" EnableViewState="false" %>**

* + 1. ASP.NET (VB)

**<%@ Page Title="Home Page" Language="Vb" MasterPageFile="~/UI.master" AutoEventWireup="true" CodeBehind="Default.aspx.vb" Inherits="WebFormsSampleApp.\_Default" EnableViewState="false" %>**

* 1. Disable **ViewState** for the shopping cart user control. To do this, open **ShoppingCart.ascx** in **Markup** mode (located under **UserControls** folder), and replace the **<% Control %>** directive for the following one.
     1. **Note:** Enabling **ViewStateMode** for this control would result in a heavy pay load while rendering the page. To avoid this, you will disable **ViewStateMode** for this control and all its child controls, and handle them from **Session**.
     2. ASP.NET (C#)
     3. **<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="ShoppingCart.ascx.cs" Inherits="WebFormsSampleApp.UserControls.ShoppingCartControl"**
     4. **EnableViewState="false" %>**
     5. ASP.NET (VB)
     6. **<%@ Control Language="VB" AutoEventWireup="true" CodeBehind="ShoppingCart.ascx.vb" Inherits="WebFormsSampleApp.UserControls.ShoppingCartControl"**
     7. **EnableViewState="false" %>**

Task 2 – Enabling Granular ViewState on Child Controls

* 1. In this task, you will modify the way in which some values are stored in ViewState, such as the current category name, the current selected page, and the number of total pages. It is useful to have these values in ViewState for example every time you add an item to the shopping cart, and preserve the exact page state (same category, same page number) after the postback.
  2. You will create three hidden controls, one for each value, and you will take advantage of granular ViewState by enabling ViewState for each of them. As a result, these will be the only fields to be stored in the **ViewState** for the **Default.aspx** page, reducing the total size of the page, and increasing the application performance.
  3. **Note:** Until now, you were storing these values in the **Default.aspx** page’s ViewState, but as this functionality was disabled in the previous task, the ViewState collection for this control is no longer available.
  4. Add three new hidden fields to replace the values that were stored on the **Page**'s **ViewState**. To do this, open **Default.aspx** in **Markup** mode, and add the following highlighted code to the second **<asp:Content>** element.
     1. **Note:** Although you will use these hidden fields’ ViewState to preserve the values between postbacks, you could have also obtained them based on the URL via **ASP.NET Routing** (as shown in Exercise 2: Enabling Bi-Directional Routing Support ). However, for practical purposes you will use the hidden field approach in order to show the new **ViewStateMode** feature.
     2. ASP.NET
     3. ...
     4. <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
     5. **<asp:HiddenField ID="CategoryNameState" runat="server" ViewStateMode="Enabled" />**
     6. **<asp:HiddenField ID="TotalPagesState" runat="server" ViewStateMode="Enabled" />**
     7. **<asp:HiddenField ID="SelectedPageState" runat="server" ViewStateMode="Enabled" />**
     8. <asp:HiddenField ID="ShoppingCartState" ClientIDMode="Static" runat="server" />
     9. ...
  5. Enable ViewState in the hidden field that stores the shopping cart state (indicating if it is expanded or collapsed). To do this, set the **ViewStateMode** property to **Enable** in the ShoppingCartState **HiddenField** declaration as shown in the following code.
     1. ASP.NET
     2. <asp:HiddenField ID="ShoppingCartState" ClientIDMode="Static" runat="server" **ViewStateMode="Enabled"** />

Task 3 – Retrieving ViewState Values from Code

* 1. In this task, you will change the way in which the **SelectedCategoryName**, **TotalPages** and **SelectedPage** properties are getting/setting their values from code. You will also force to reload the product **ListView** in every postback, as a consequence of having disabled the **ViewState** for that control.
  2. Replace the getter and setter implementation of the **SelectedCategoryName**, **TotalPages** and **SelectedPage** properties to get/set the value from/to the hidden fields instead of the **ViewState**. To do this, open **Default.aspx** code-behind, and replace the current property definitions with the following highlighted code.
     1. *(Code Snippet – ASP.NET 4 Web Forms Lab – GranularViewState Properties – C#)*
     2. C#
     3. **public string SelectedCategoryName**
     4. **{**
     5. **get**
     6. **{**
     7. **if (this.CategoryNameState.Value == null)**
     8. **{**
     9. **this.CategoryNameState.Value = "Bikes";**
     10. **}**
     11. **return this.CategoryNameState.Value;**
     12. **}**
     13. **set**
     14. **{**
     15. **this.CategoryNameState.Value = value;**
     16. **}**
     17. **}**
     18. **public int TotalPages**
     19. **{**
     20. **get**
     21. **{**
     22. **if (TotalPagesState.Value == null)**
     23. **{**
     24. **TotalPagesState.Value = "0";**
     25. **}**
     26. **return Convert.ToInt32(TotalPagesState.Value);**
     27. **}**
     28. **set**
     29. **{**
     30. **TotalPagesState.Value = Convert.ToString(value);**
     31. **}**
     32. **}**
     33. **public int SelectedPage**
     34. **{**
     35. **get**
     36. **{**
     37. **if (this.SelectedPageState.Value == null)**
     38. **{**
     39. **this.SelectedPageState.Value = "1";**
     40. **}**
     41. **return Convert.ToInt32(this.SelectedPageState.Value);**
     42. **}**
     43. **set**
     44. **{**
     45. **this.SelectedPageState.Value = Convert.ToString(value);**
     46. **}**
     47. **}**
     48. *(Code Snippet – ASP.NET 4 Web Forms Lab – GranularViewState Properties – VB)*
     49. VB
     50. **Public Property SelectedCategoryName() As String**
     51. **Get**
     52. **If Me.CategoryNameState.Value Is Nothing Then**
     53. **Me.CategoryNameState.Value = "Bikes"**
     54. **End If**
     55. **Return Me.CategoryNameState.Value**
     56. **End Get**
     57. **Set(ByVal value As String)**
     58. **Me.CategoryNameState.Value = value**
     59. **End Set**
     60. **End Property**
     61. **Public Property TotalPages() As Integer**
     62. **Get**
     63. **If TotalPagesState.Value Is Nothing Then**
     64. **TotalPagesState.Value = "0"**
     65. **End If**
     66. **Return Convert.ToInt32(TotalPagesState.Value)**
     67. **End Get**
     68. **Set(ByVal value As Integer)**
     69. **TotalPagesState.Value = Convert.ToString(value)**
     70. **End Set**
     71. **End Property**
     72. **Public Property SelectedPage() As Integer**
     73. **Get**
     74. **If Me.SelectedPageState.Value Is Nothing Then**
     75. **Me.SelectedPageState.Value = "1"**
     76. **End If**
     77. **Return Convert.ToInt32(Me.SelectedPageState.Value)**
     78. **End Get**
     79. **Set(ByVal value As Integer)**
     80. **Me.SelectedPageState.Value = Convert.ToString(value)**
     81. **End Set**
     82. **End Property**
  3. Reload the product **ListView** on every postback. To do this, open **Default.aspx.cs** file, and inside the **Page\_Load** method move the call to the **ApplyProductsFilter** method outside the **if (!PostBack)** conditional clause. The resulting method should look like the following:
     1. **Note:** While **ViewStateMode** in **Default.aspx** was enabled at Page level, all of its child controls were saving **ViewState** by default including the product **ListView**. That is why you did not need to reload the products again during postbacks. This incurred substantial increases in the page size rendered to the user.
     2. *(Code Snippet – ASP.NET 4 Web Forms Lab – Page\_Load Method – C#)*
     3. C#
     4. protected void Page\_Load(object sender, EventArgs e)
     5. {
     6. **if (!IsPostBack)**
     7. **{**
     8. **this.SelectedCategoryName = GetCategoryName();**
     9. **this.SelectedPage = GetPageIndex();**
     10. **}**
     11. **ApplyProductsFilter();**
     12. **CreatePagerLinks();**
     13. }
     14. *(Code Snippet – ASP.NET 4 Web Forms Lab – Page\_Load Method – VB)*
     15. C#
     16. Protected Sub Page\_Load(ByVal sender As Object, ByVal e As EventArgs)
     17. **If (Not IsPostBack) Then**
     18. **Me.SelectedCategoryName = GetCategoryName()**
     19. **Me.SelectedPage = GetPageIndex()**
     20. **End If**
     21. **ApplyProductsFilter()**
     22. **CreatePagerLinks()**
     23. End Sub
     24. **Note:** In this case, you are re-loading the **ListView** (going against the database) in every postback because no ViewState was stored for this control.
     25. In a more realistic scenario you should evaluate whether it is more convenient to have an extra payload to store the **ViewState** for the product’s **ListView**, rather than making a new round-trip to the database to retrieve the values again, or vice versa.

# Next Step

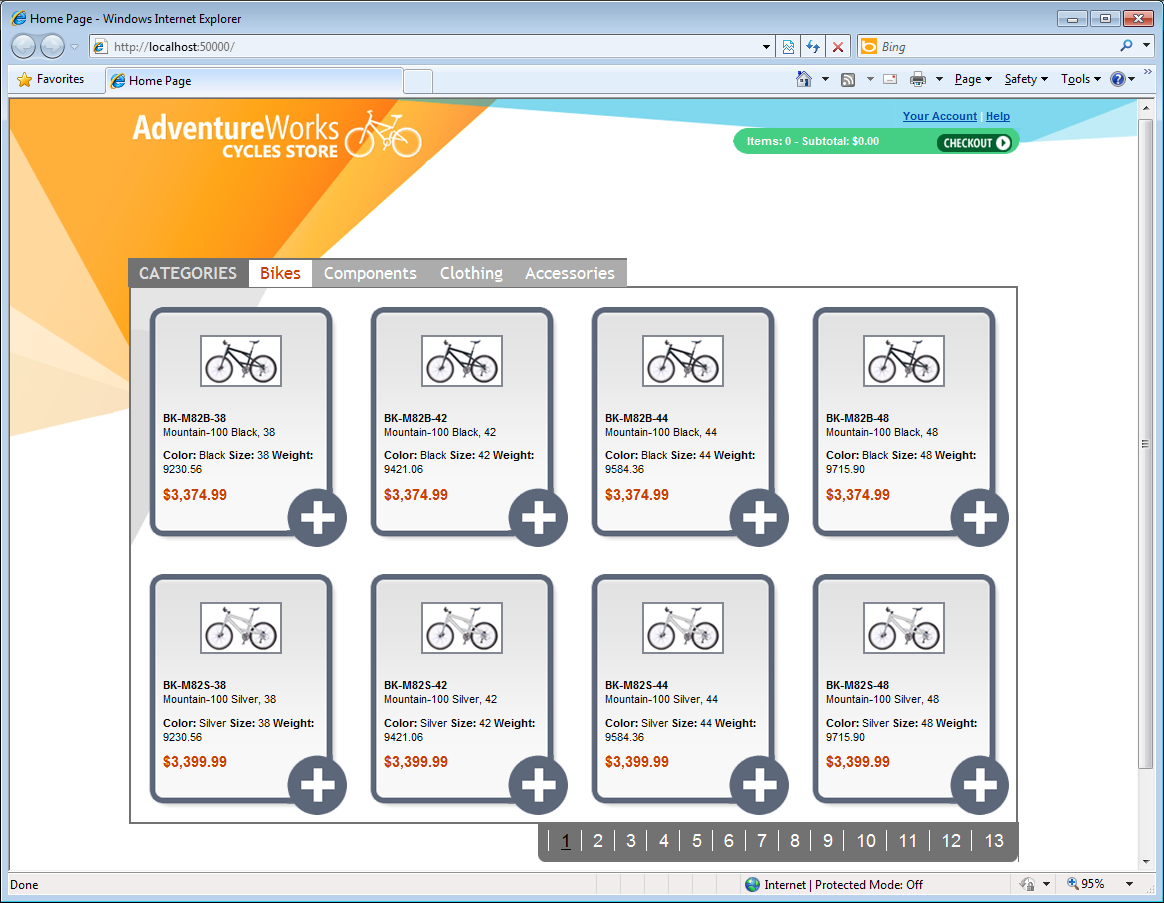
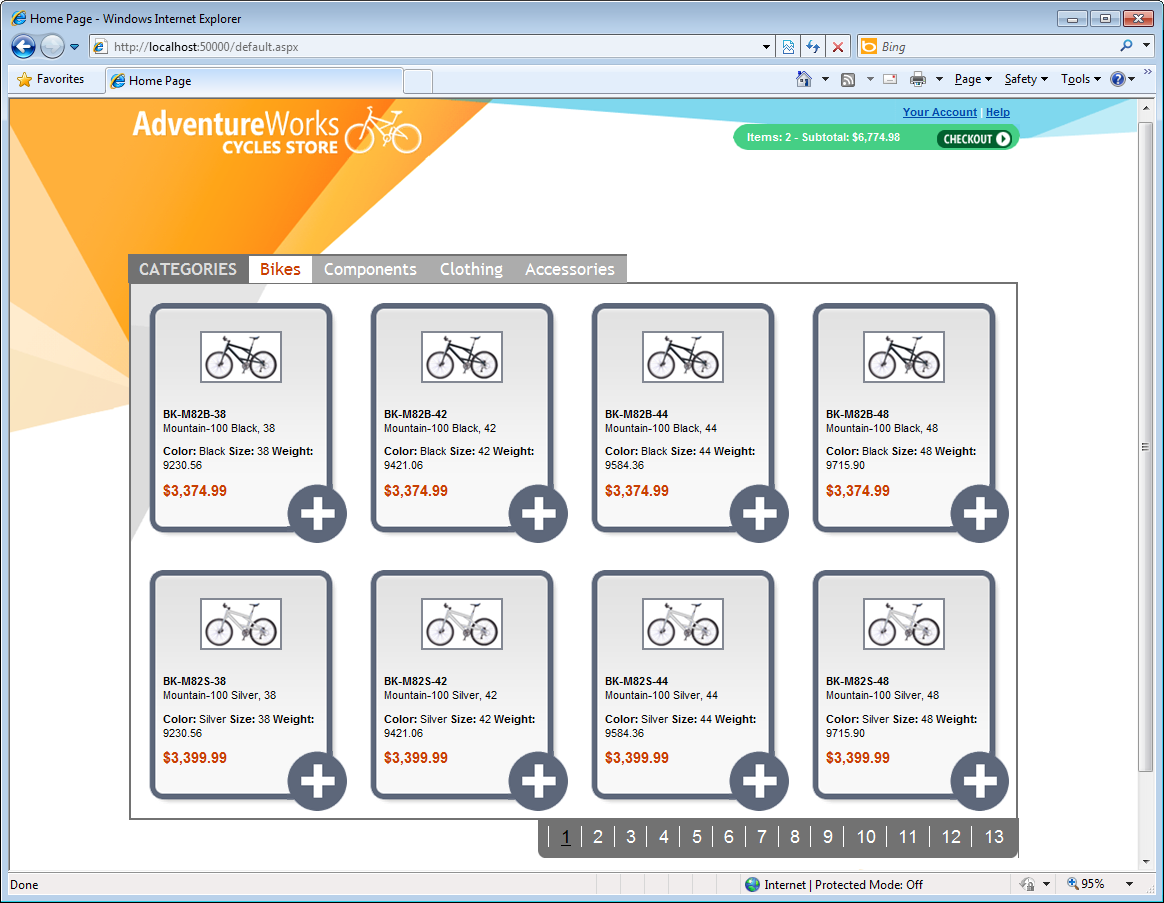
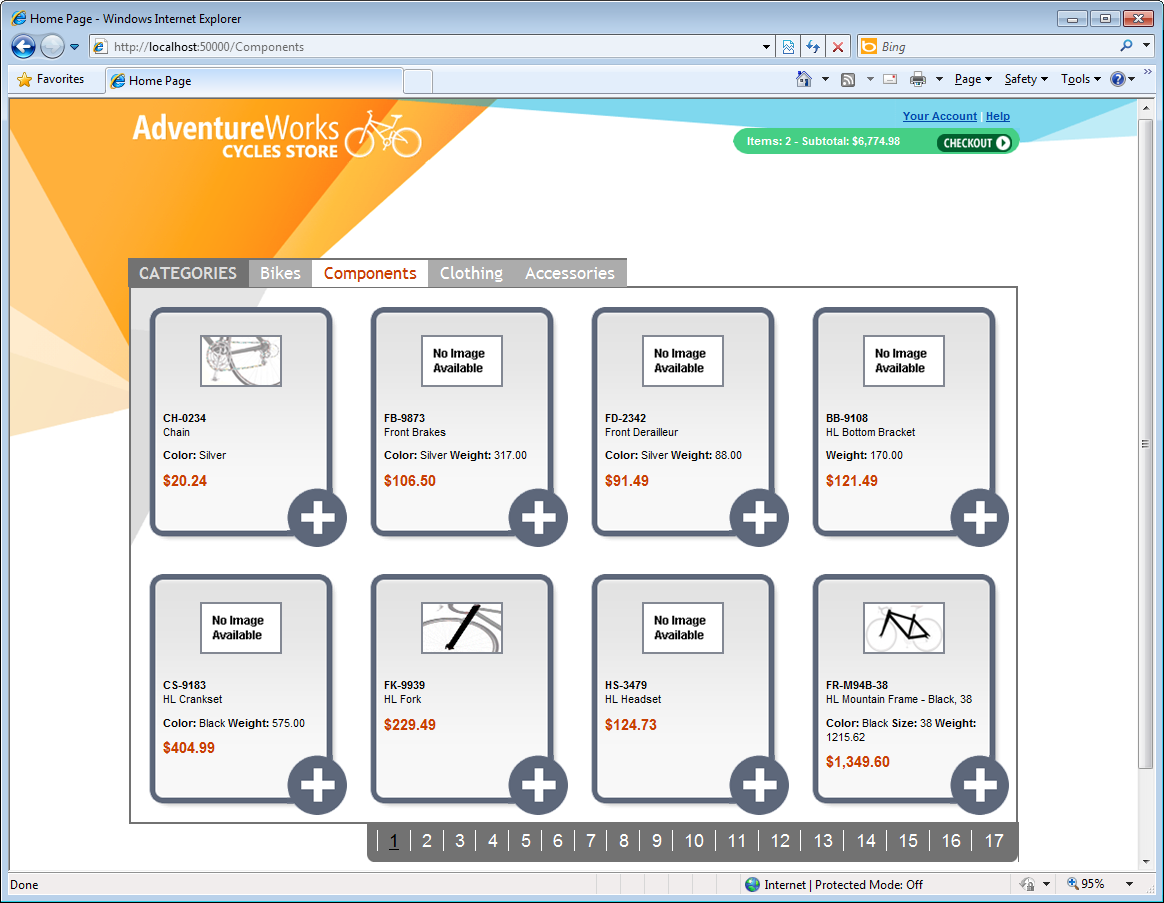
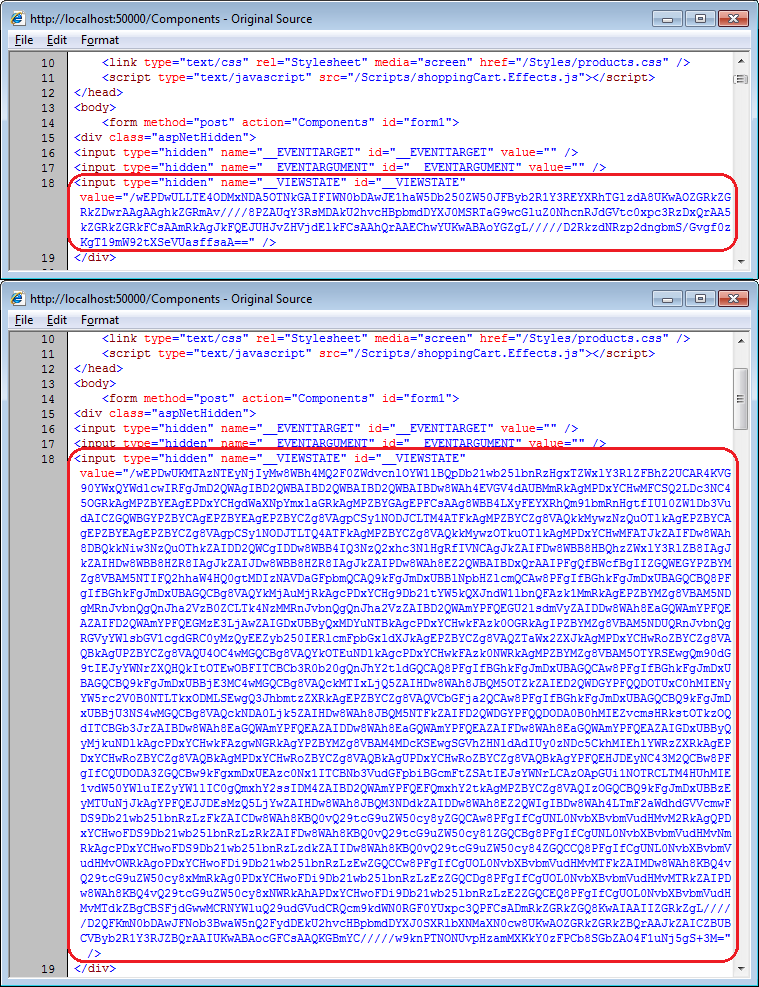
Exercise 4: Verification

## Exercise 4: Verification

In order to verify that you have correctly performed all steps of exercise four, proceed as follows:

### Verification 1

In this verification, you will see how the page size is reduced while taking granular ViewState control in your application. You will place some orders and browse some categories, to finally compare the rendered page size between the one using granular ViewState, and the other with ViewState enabled at the Page level. For the latter one, you will use the unaltered begin solution of this exercise, which does not implement granular ViewState yet.

* 1. Start a new instance of the **WebFormsSampleApp** project. To do this, in **Solution Explorer** right-click **WebFormsSampleApp** project, point to **Debug** and select **Start new instance**.
     1. **Note:** If the dialog **Debugging Not Enabled** appears, select **Modify the Web.config file to enable debugging** and click **OK**.
     2. 
     3. Figure 23
     4. Viewing the Default page
  2. Place one or more products into the shopping cart by clicking the plus () symbol next to them.
     1. **Note:** Make sure to add the same products for both applications (with granular ViewState and without) so as to have a consistent page size comparison.
     2. 
     3. Figure 24
     4. Placing an order into the shopping cart
  3. Browse one or more categories by clicking the links on the **CATEGORIES** header.
     1. **Note:** Make sure to browse the same categories both applications (with granular ViewState and without) so as to have a consistent page size comparison.
     2. 
     3. Figure 25
     4. Browsing to Component category
  4. Open the unaltered begin solution of this exercise (without Granular ViewState), and perform the same exact steps going from 1 to 3.
     1. **Note:** Both browsers should display the same rendered pages with no apparent difference. The subtle difference resides on how ViewState is stored for each page. The page with Granular ViewState with have less bytes spent on saving ViewState, resulting on a more lightweight html rendered to the user.
  5. Inspect both pages’ html source to notice the difference in length of the encoded **ViewState** hidden field for both pages. To do this, right-click over each page on your browser, and select **View Source**. Locate the tag show in the following figure, and compare its length.
     1. **Note:** The following figure shows the html source for both pages. The HTML source at the top belongs to the Granular ViewState page, while the bottom one belongs to the ViewState at page level.
     2. In this case, where the page is relatively small, the difference in bytes between both pages was about 2KB (a 10% of the total page size). In larger pages, the difference would become more significant, causing negative effects in the overall application performance.
     3. 
     4. Figure 26
     5. Viewing the length differences between both ViewState values (Top – Granular View State ; Bottom – ViewState at Page level)

# Next Step

Summary

Summary

In this Lab, you have learnt some of the new feature and enhancements included in ASP.NET 4 Web Forms like the **ClientIDMode** property to remove the bloat in the client ID, the **PageRouteHanlder** class to enable URL Routing, and the **ViewStateMode** property to have a more granular control over ViewState.