ASP.NET Tailspin Spyworks Tutorial

Version 0.8

Joe Stagner - Microsoft

4/28/2010

Contents

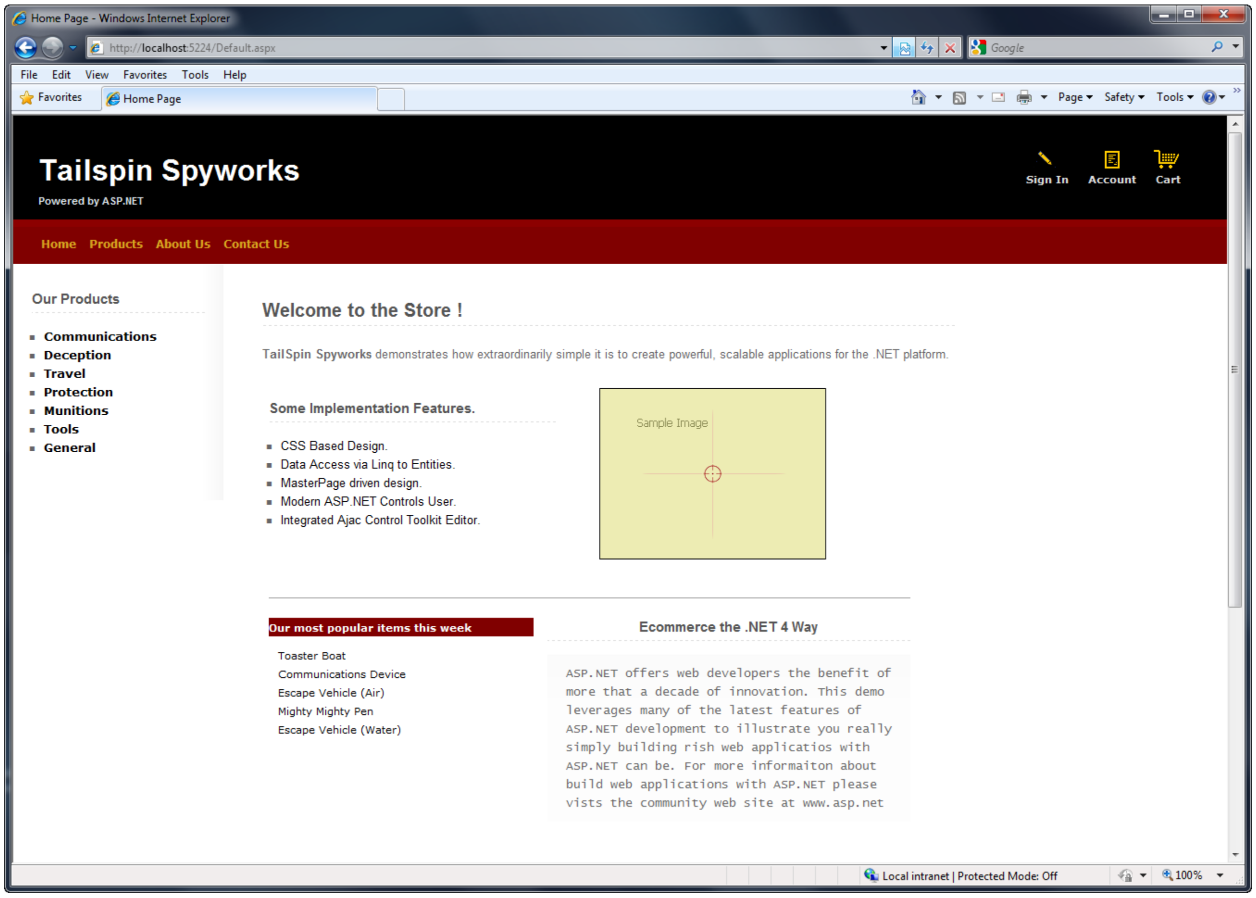
Building Tailspin Spyworks – an ASP.NET WebForms Sample 3
Overview 3
File / New Project 9
Adding the Data Access Layer 14
Adding Some Layout and a Category Menu 21
Listing Products with the GridView Control 26
  Adding Some Business Logic 35
  Working with ASP.NET Membership 49
  Adding Features 60
  Account Review 60
  OrderDetsails.aspx 61
  The Home Page 63
Product Reviews 65
Popular Items Control (Creating User Controls) 70
  "Also Purchased" Control (User Controls with Parameters) 74
Contact Page (Sending email from ASP.NET) 81
About Page 82
Global Exception Handler 83
  Conclusion 84

Building Tailspin Spyworks – an ASP.NET WebForms Sample

Overview

This tutorial is an introduction to ASP.NET WebForms. We'll be starting slowly, so beginner level web development experience is okay.

The application we'll be building is a simple on-line store.


Home Page - Windows Internet Explorer
http://localhost:5224/Default.aspx
File Edit View Favorites Tools Help
Favorites
Home Page
Page Safety Tools
Sign In
Account
Cart
Home
Products
About Us
Contact Us
Local intranet | Protected Mode: Off
100%

Tailspin Spyworks

Powered by ASP.NET

Our Products

- Communications
- Deception
- Travel
- Protection
- Munitions
- Tools
- General

Welcome to the Store !

TailSpin Spyworks demonstrates how extraordinarily simple it is to create powerful, scalable applications for the .NET platform.

Some Implementation Features.

- CSS Based Design.
- Data Access via Linq to Entities.
- MasterPage driven design.
- Modern ASP.NET Controls User.
- Integrated Ajac Control Toolkit Editor.

Sample Image

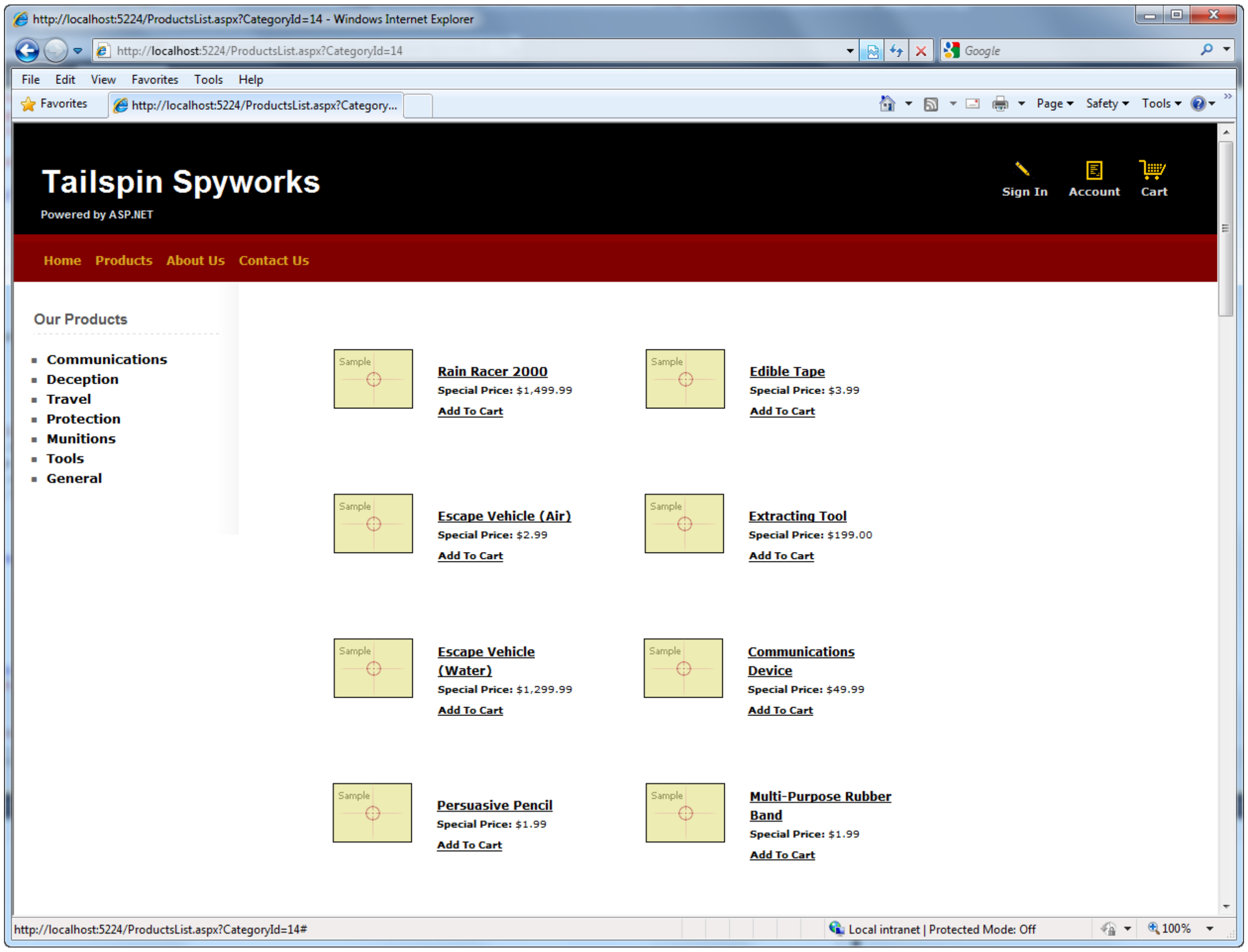
Our most popular items this week

Toaster Boat
Communications Device
Escape Vehicle (Air)
Mighty Mighty Pen
Escape Vehicle (Water)

Ecommerce the .NET 4 Way

ASP.NET offers web developers the benefit of more that a decade of innovation. This demo leverages many of the latest features of ASP.NET development to illustrate you really simply building rish web applicatios with ASP.NET can be. For more informaiton about build web applications with ASP.NET please vists the community web site at www.asp.net

Visitors can browse Products by Category:


http://localhost:5224/ProductsList.aspx?CategoryId=14 - Windows Internet Explorer
File Edit View Favorites Tools Help
Favorites
Page Safety Tools
Sign In
Account
Cart
Home Products About Us Contact Us
Sample
Local intranet | Protected Mode: Off
100%

Tailspin Spyworks

Powered by ASP.NET

Our Products

- Communications
- Deception
- Travel
- Protection
- Munitions
- Tools
- General

Rain Racer 2000
Special Price: $1,499.99
Add To Cart

Edible Tape
Special Price: $3.99
Add To Cart

Escape Vehicle (Air)
Special Price: $2.99
Add To Cart

Extracting Tool
Special Price: $199.00
Add To Cart

Escape Vehicle (Water)
Special Price: $1,299.99
Add To Cart

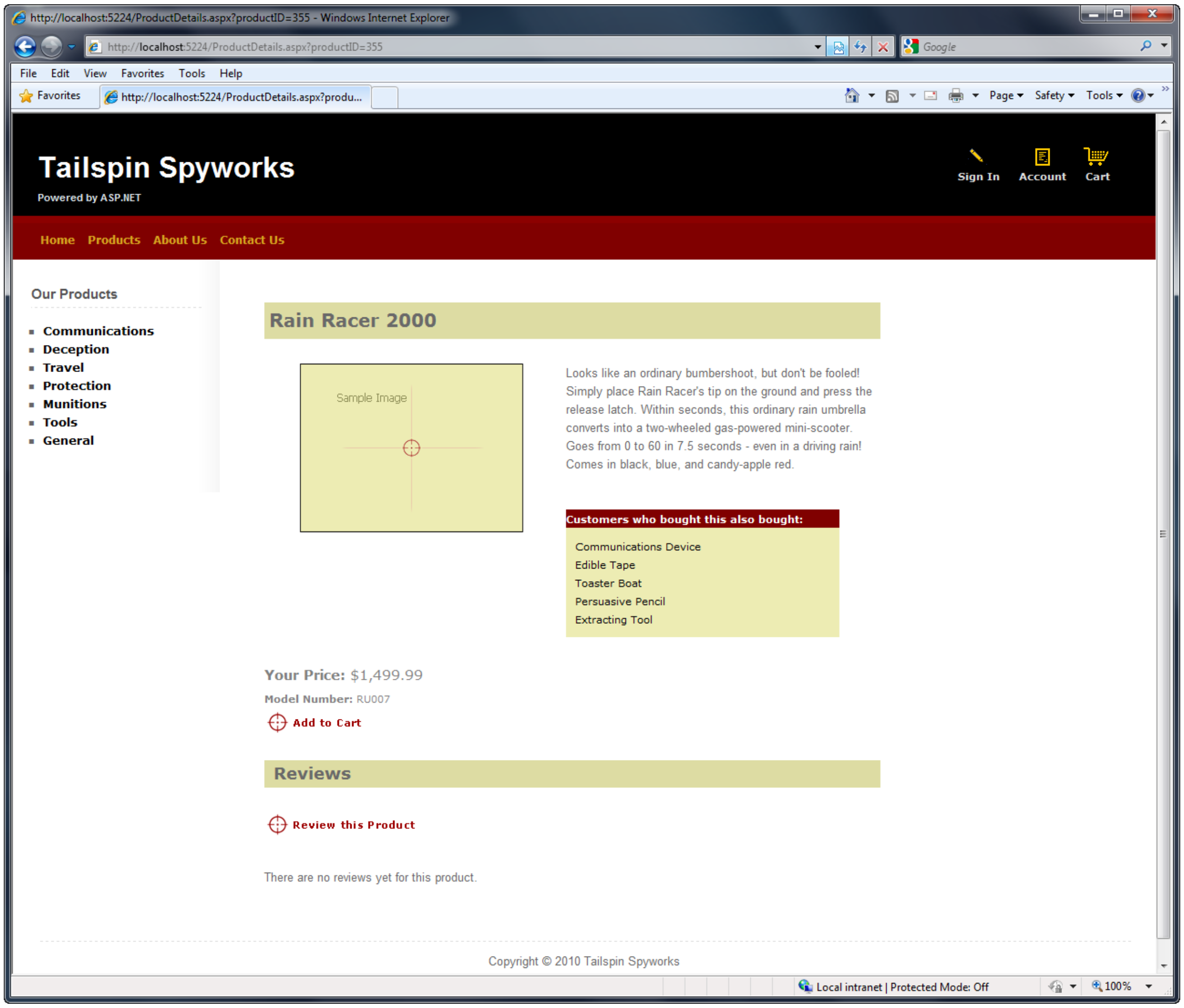
Communications Device
Special Price: $49.99
Add To Cart

Persuasive Pencil
Special Price: $1.99
Add To Cart

Multi-Purpose Rubber Band
Special Price: $1.99
Add To Cart

http://localhost:5224/ProductsList.aspx?CategoryId=14#

They can view a single product and add it to their cart:



Rain Racer 2000

Looks like an ordinary bumbershoot, but don't be fooled! Simply place Rain Racer's tip on the ground and press the release latch. Within seconds, this ordinary rain umbrella converts into a two-wheeled gas-powered mini-scooter. Goes from 0 to 60 in 7.5 seconds - even in a driving rain! Comes in black, blue, and candy-apple red.

Customers who bought this also bought:
- Communications Device
- Edible Tape
- Toaster Boat
- Persuasive Pencil
- Extracting Tool

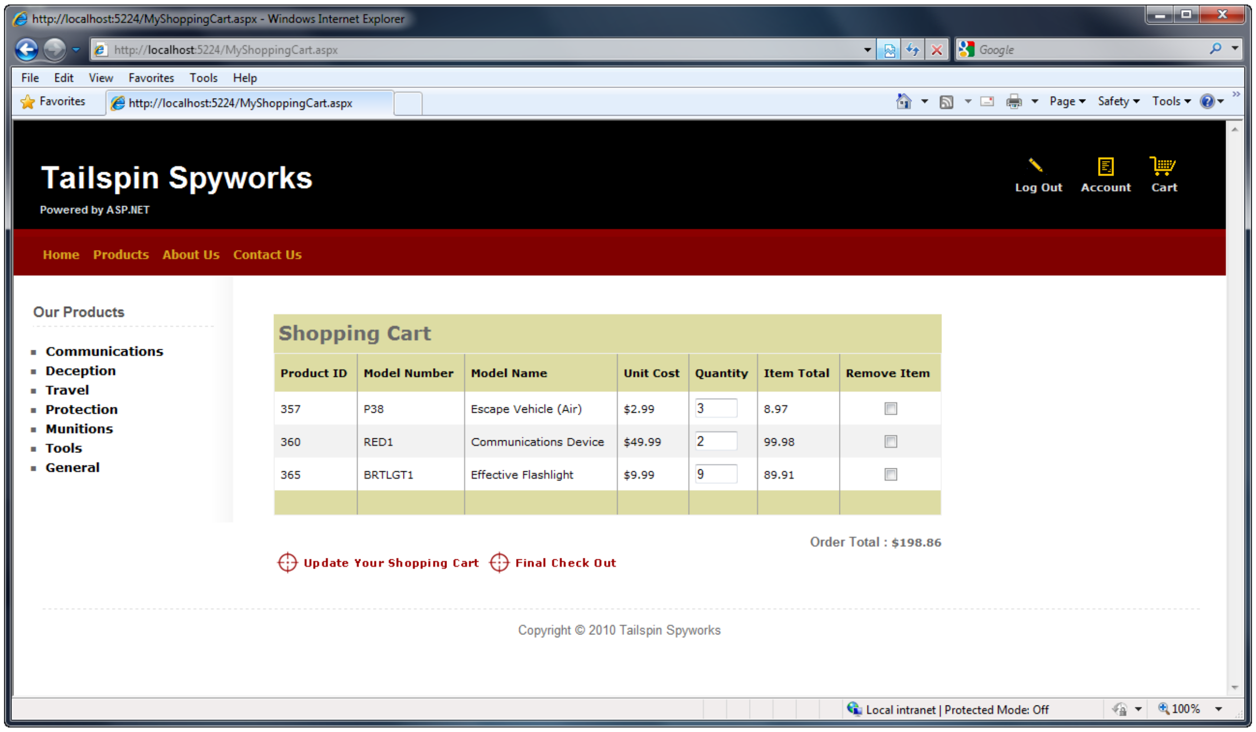
Your Price: $1,499.99
Model Number: RU007

Reviews

There are no reviews yet for this product.

Copyright © 2010 Tailspin Spyworks

They can review their cart, removing any items they no longer want:

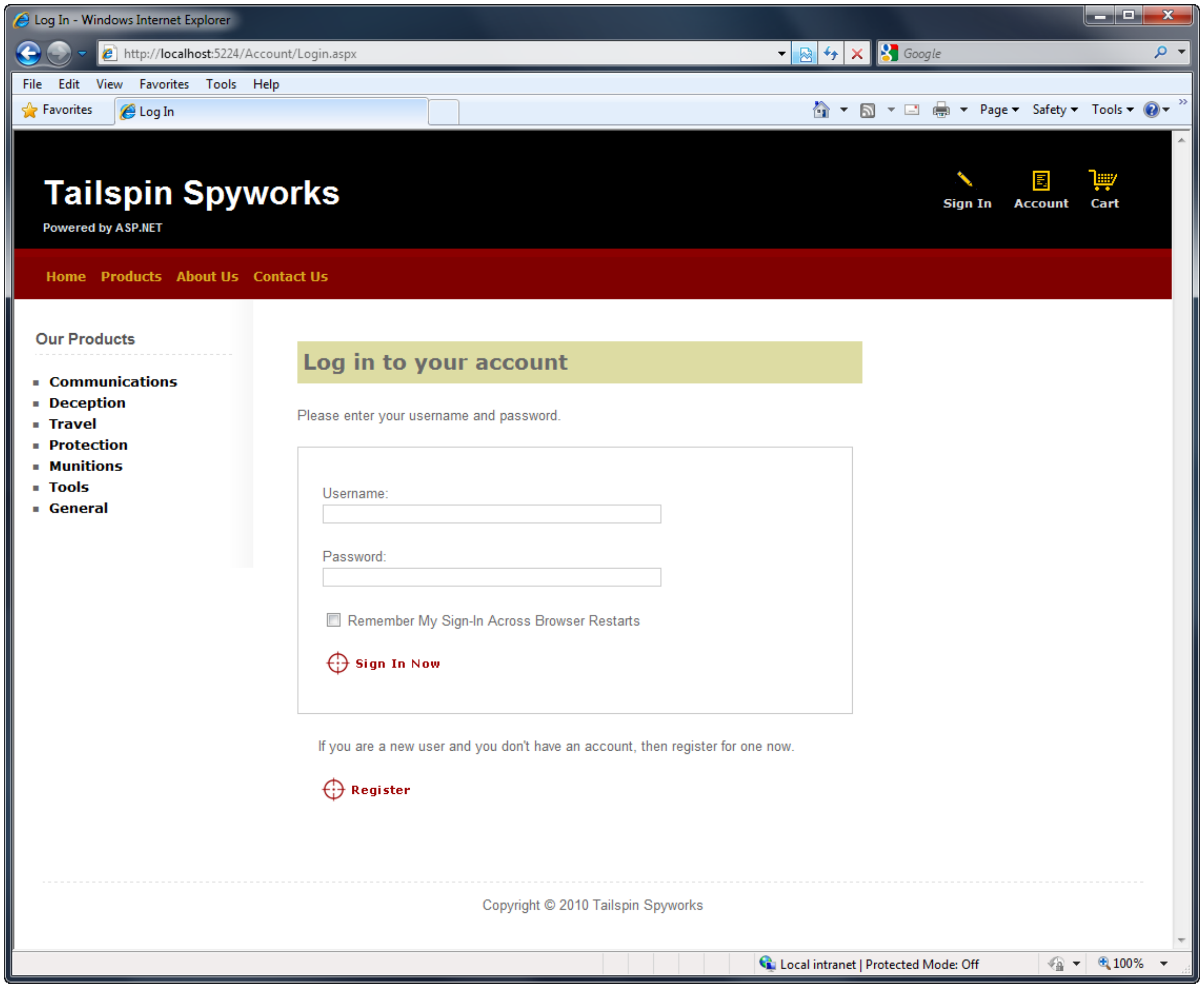


Shopping Cart

Product ID	Model Number	Model Name	Unit Cost	Quantity	Item Total	Remove Item
357	P38	Escape Vehicle (Air)	$2.99	3	8.97	☐
360	RED1	Communications Device	$49.99	2	99.98	☐
365	BRTLGT1	Effective Flashlight	$9.99	9	89.91	☐

Order Total : $198.86

Proceeding to Checkout will prompt them to



Log in to your account

Please enter your username and password.

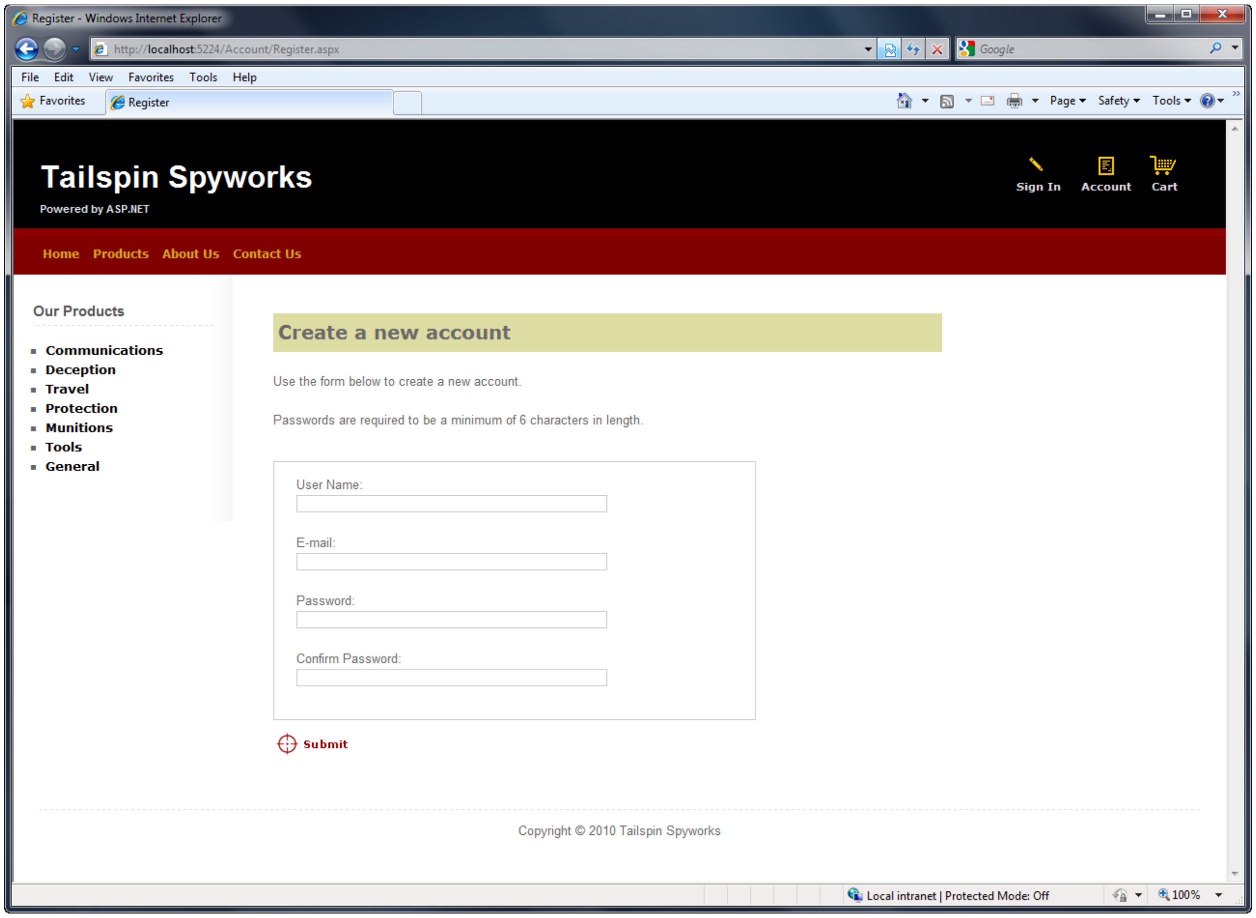
Username:

Password:

Remember My Sign-In Across Browser Restarts

If you are a new user and you don't have an account, then register for one now.

Copyright © 2010 Tailspin Spyworks


Register - Windows Internet Explorer

http://localhost:5224/Account/Register.aspx

Tailspin Spyworks

Powered by ASP.NET

Sign In Account Cart

Home Products About Us Contact Us

Our Products

- Communications
- Deception
- Travel
- Protection
- Munitions
- Tools
- General

Create a new account

Use the form below to create a new account.

Passwords are required to be a minimum of 6 characters in length.

User Name:

E-mail:

Password:

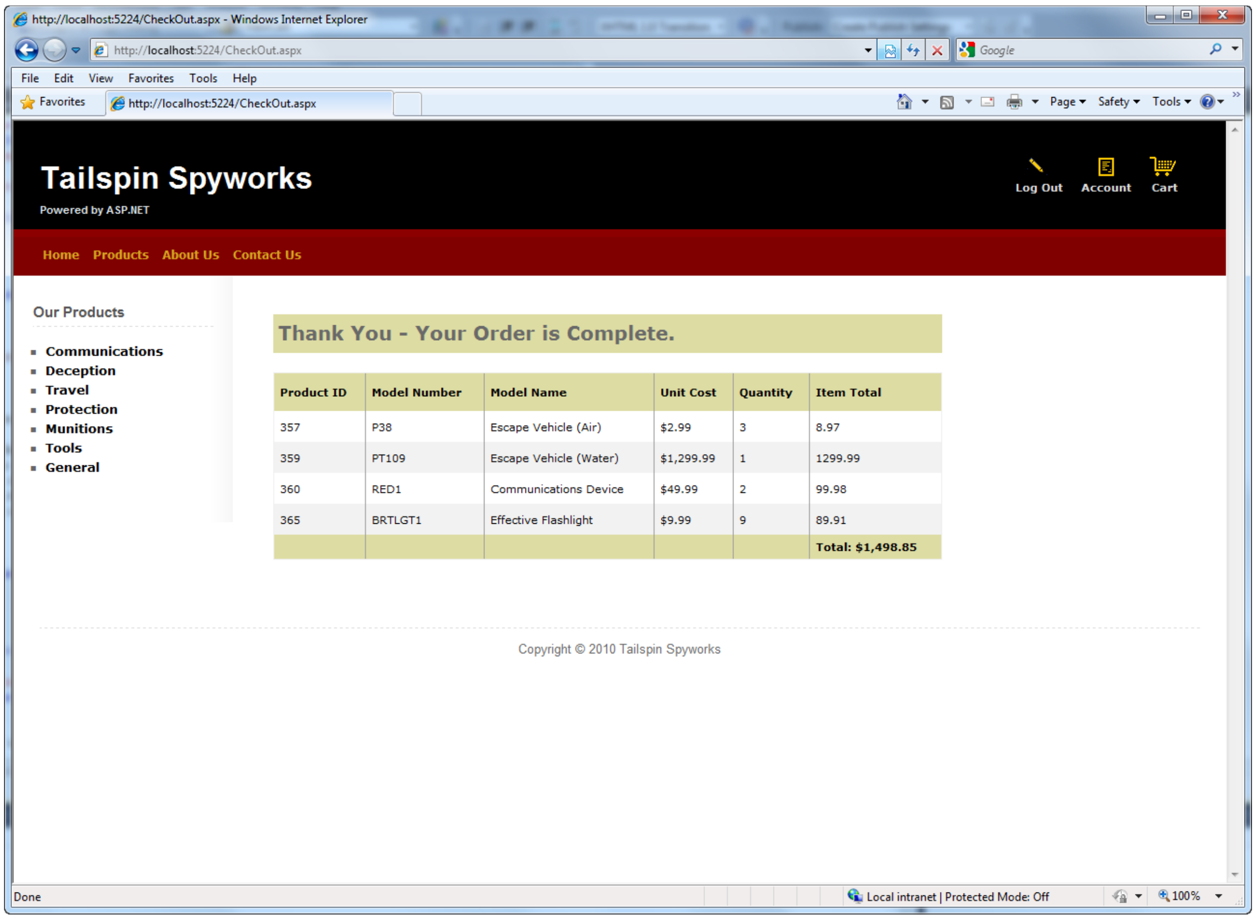
Confirm Password:

Submit

Copyright © 2010 Tailspin Spyworks

Local intranet | Protected Mode: Off

After ordering, they see a simple confirmation screen:



Thank You - Your Order is Complete.

Product ID	Model Number	Model Name	Unit Cost	Quantity	Item Total
357	P38	Escape Vehicle (Air)	$2.99	3	8.97
359	PT109	Escape Vehicle (Water)	$1,299.99	1	1299.99
360	RED1	Communications Device	$49.99	2	99.98
365	BRTLGT1	Effective Flashlight	$9.99	9	89.91
					Total: $1,498.85

Copyright © 2010 Tailspin Spyworks

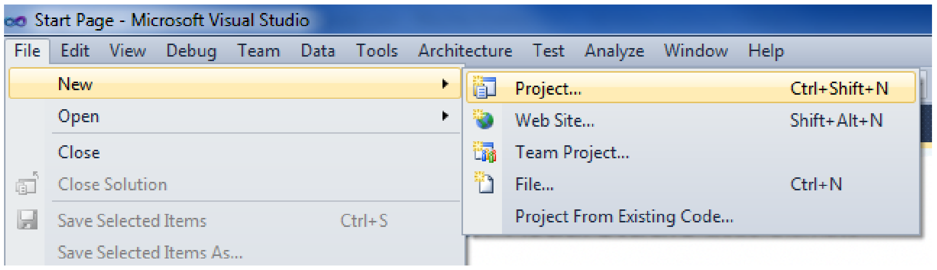
We'll begin by creating a new ASP.NET WebForms project in Visual Studio 2010, and we'll incrementally add features to create a complete functioning application. Along the way, we'll cover database access, list and grid views, data update pages, data validation, using master pages for consistent page layout, AJAX, validation, user membership, and more.

You can follow along step by step, or you can download the completed application from http://tailspinspyworks.codeplex.com/

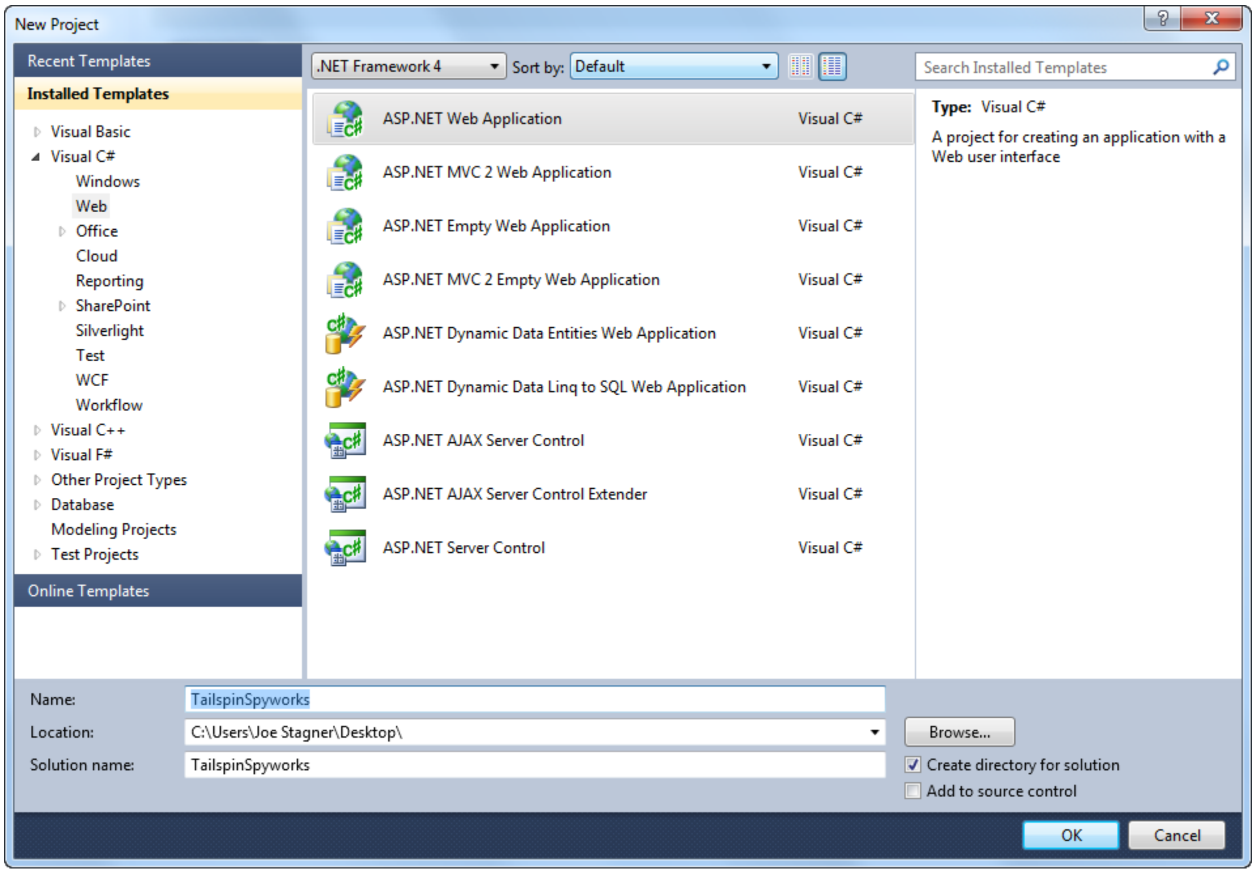
You can use either Visual Studio 2010 or the free Visual Web Developer 2010 from http://www.microsoft.com/express/Web/. To build the application, you can use either SQL Server or the free SQL Server Express to host the database.

File / New Project

We'll start by selecting the New Project from the File menu in Visual Studio. This brings up the New Project dialog.


Start Page - Microsoft Visual Studio
File Edit View Debug Team Data Tools Architecture Test Analyze Window Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Project... Ctrl+Shift+N
Web Site... Shift+Alt+N
Team Project...
File... Ctrl+N
Project From Existing Code...

We'll select the Visual C# / Web Templates group on the left, and then choose the "ASP.NET Empty Web Application" template in the center column. Name your project Tailspin Spyworks and press the OK button.



ASP.NET Web Application	Visual C#
ASP.NET MVC 2 Web Application	Visual C#
ASP.NET Empty Web Application	Visual C#
ASP.NET MVC 2 Empty Web Application	Visual C#
ASP.NET Dynamic Data Entities Web Application	Visual C#
ASP.NET Dynamic Data Linq to SQL Web Application	Visual C#
ASP.NET AJAX Server Control	Visual C#
ASP.NET AJAX Server Control Extender	Visual C#
ASP.NET Server Control	Visual C#

Type: Visual C#

A project for creating an application with a Web user interface

Name: TailspinSpyworks

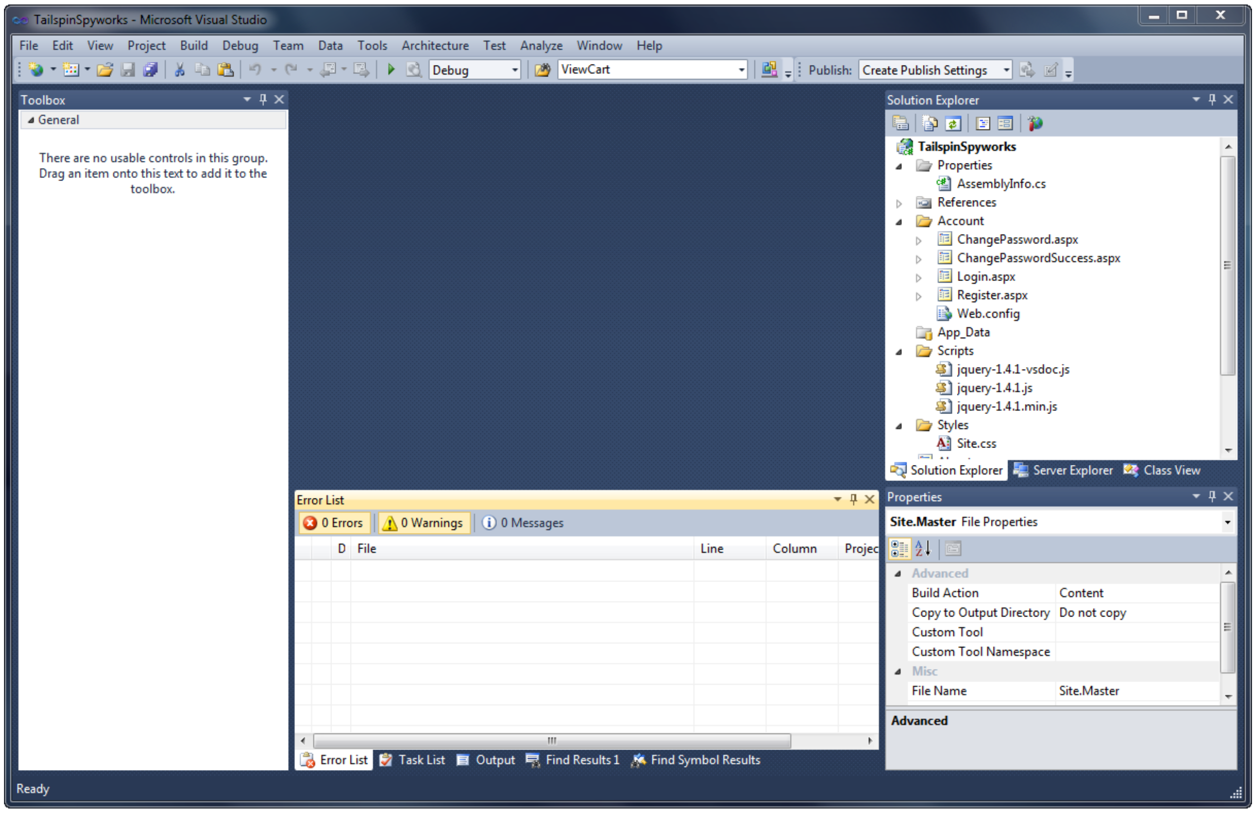
Location: C:\Users\Joe Stagner\Desktop\

Solution name: TailspinSpyworks

[✓] Create directory for solution

[ ] Add to source control

This will create our project. Let's take a look at the folders that are included in our application in the Solution Explorer on the right side.


TailspinSpyworks - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Tools Architecture Test Analyze Window Help
Debug
ViewCart
Publish: Create Publish Settings
Toolbox
General
Solution Explorer
Error List
Properties
Error List Task List Output Find Results 1 Find Symbol Results
Solution Explorer Server Explorer Class View
Ready

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
TailspinSpyworks
  Properties
    AssemblyInfo.cs
  References
  Account
    ChangePassword.aspx
    ChangePasswordSuccess.aspx
    Login.aspx
    Register.aspx
    Web.config
  App_Data
  Scripts
    jquery-1.4.1-vsdoc.js
    jquery-1.4.1.js
    jquery-1.4.1.min.js
  Styles
    Site.css
```

0 Errors | 0 Warnings | 0 Messages

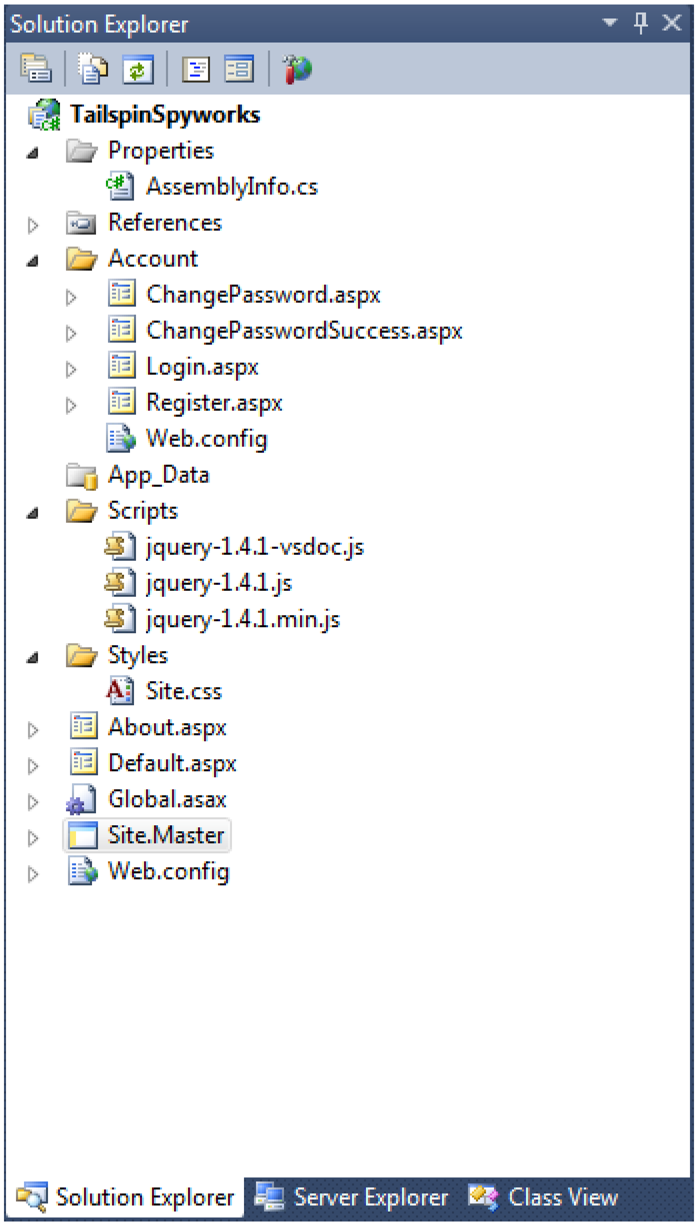
	D	File	Line	Column	Projec

Site.Master File Properties

Advanced	
Build Action	Content
Copy to Output Directory	Do not copy
Custom Tool	
Custom Tool Namespace	
Misc	
File Name	Site.Master

Advanced

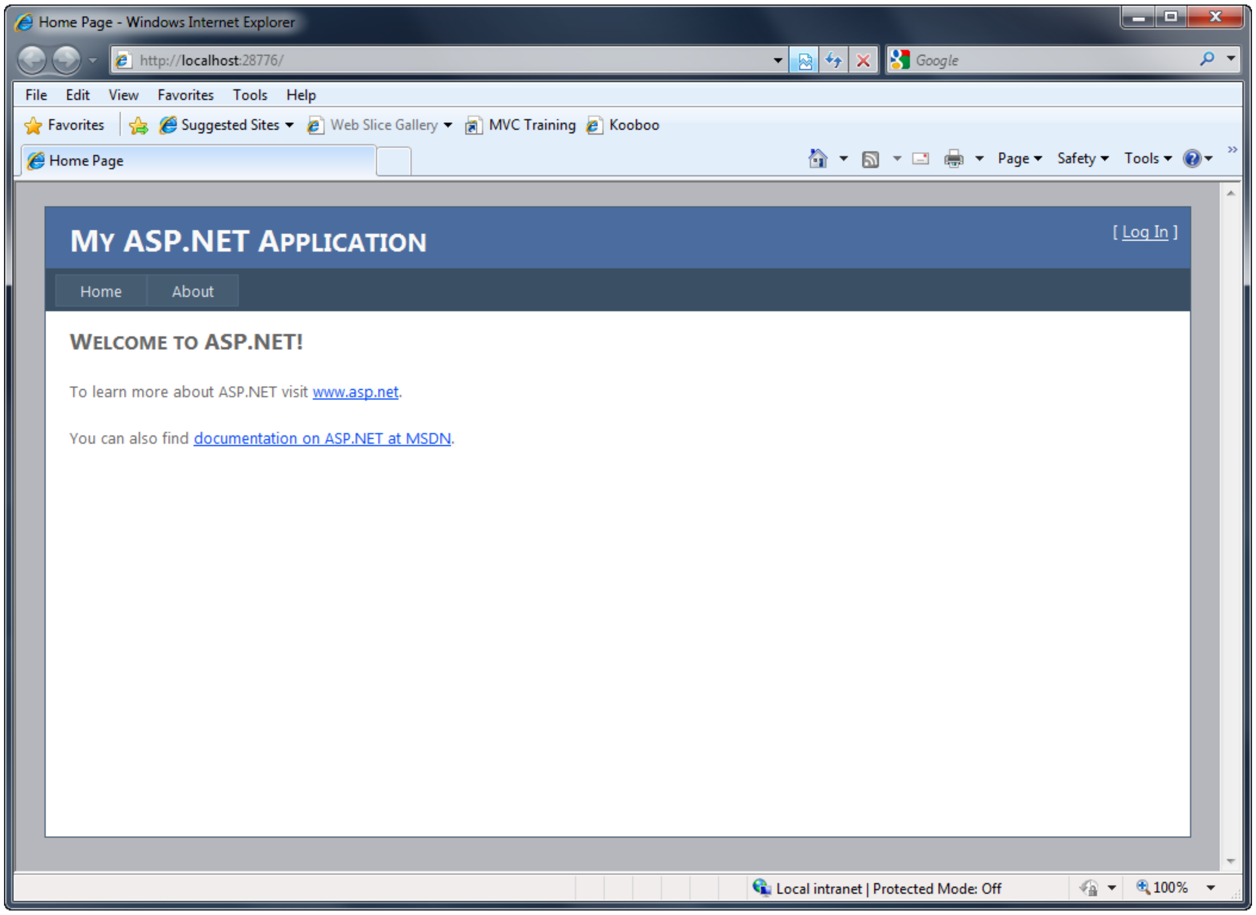
The Empty Solution isn’t completely empty – it adds a basic folder structure:



Note the conventions implemented by the ASP.NET 4 default project template.

- The “Account” folder implements a basic user interface for ASP.NET’s membership subsystem.
- The “Scripts” folder serves as the repository for client side JavaScript files and the core jQuery .js files are made available by default.
- The “Styles” folder is used to organize our web site visuals (CSS Style Sheets)

When we press F5 to run our application and render the default.aspx page we see the following.



MY ASP.NET APPLICATION

[Log In]

Home About

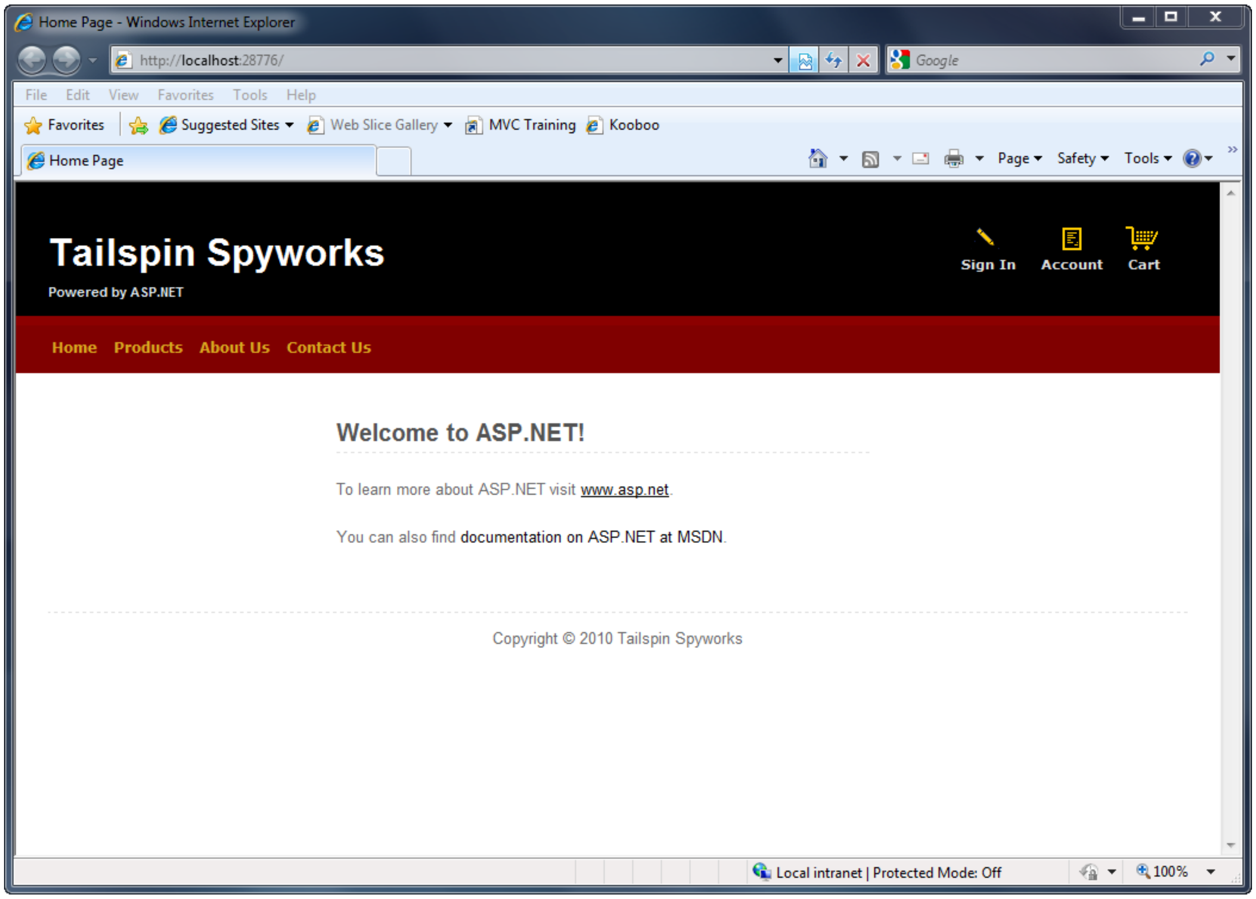
WELCOME TO ASP.NET!

To learn more about ASP.NET visit www.asp.net.

You can also find documentation on ASP.NET at MSDN.

Our first application enhancement will be to replace the Style.css file from the default WebForms template with the CSS classes and associated image files that will render the visual asthetics that we want for our Tailspin Spyworks application.

After doing so our default.aspx page renders like this.



Tailspin Spyworks

Powered by ASP.NET

Welcome to ASP.NET!

To learn more about ASP.NET visit www.asp.net.

You can also find documentation on ASP.NET at MSDN.

Copyright © 2010 Tailspin Spyworks

Notice the image links at the top right of the page and the menu items that have been added to the master page. Only the "Sign In" and "Account" links point to pages that exist (generated by the default template) and the rest of the pages we will implement as we build our application.

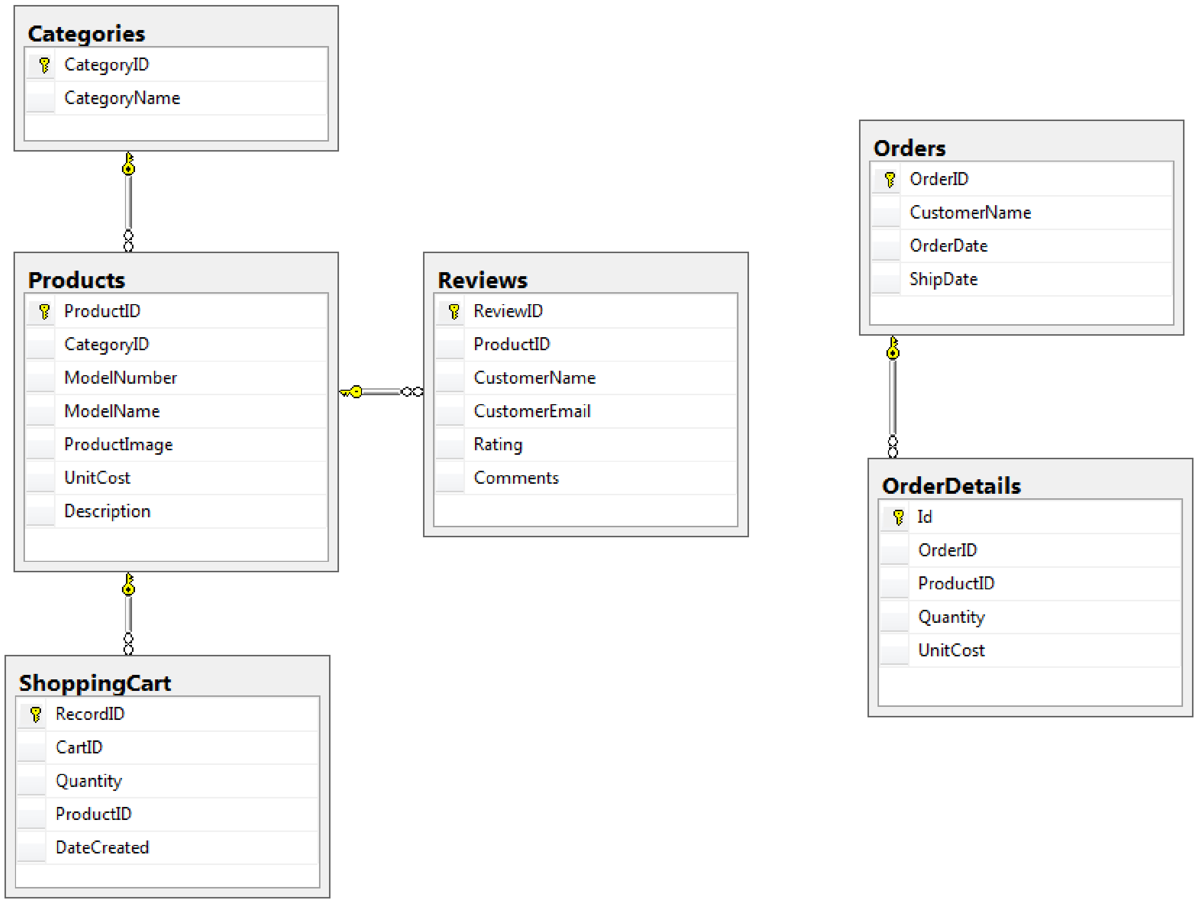
We're also going to relocate the Master Page to the Styles directory. Though this is only a preference it may make things a little easier if we decide to make our application "skinable" in the future.

After doing this we'll need to change the master page references in all the .aspx files generated by the default ASP.NET WebForms pages.

Adding the Data Access Layer

Our ecommerce application will depend on two databases.

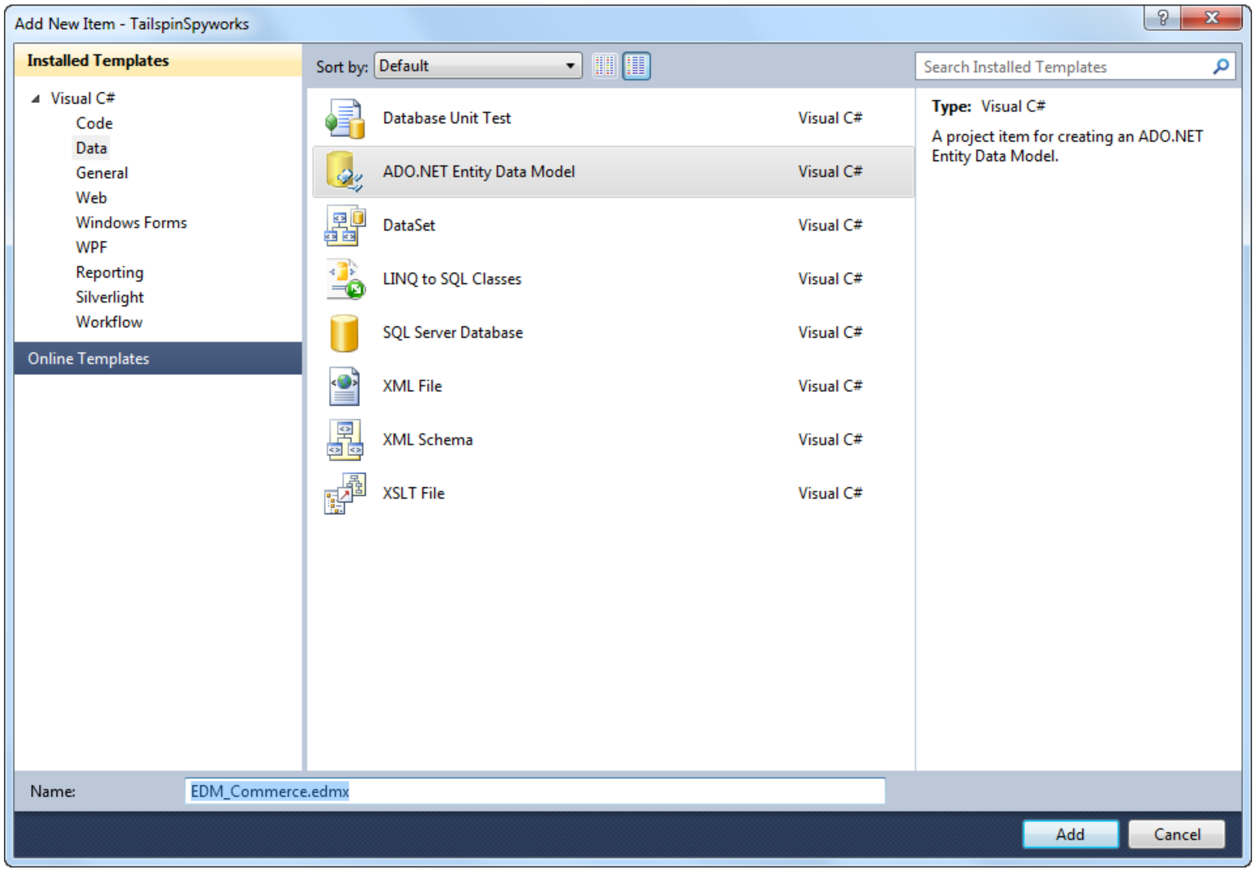
For customer information we'll use the standard ASP.NET Membership database. For our shopping cart and product catalog we'll implement a SQL Express database as follows.



Having created the database (Commerce.mdf) in the application's App_Data folder we can proceed to create our Data Access Layer using the .NET Entity Framework.

We'll create a folder named "Data_Access" and then right click on that folder and select "Add New Item".

In the "Installed Templates" item and then select "ADO.NET Entity Data Model" enter EDM_Commerce.edmx as the name and click the "Add" button.

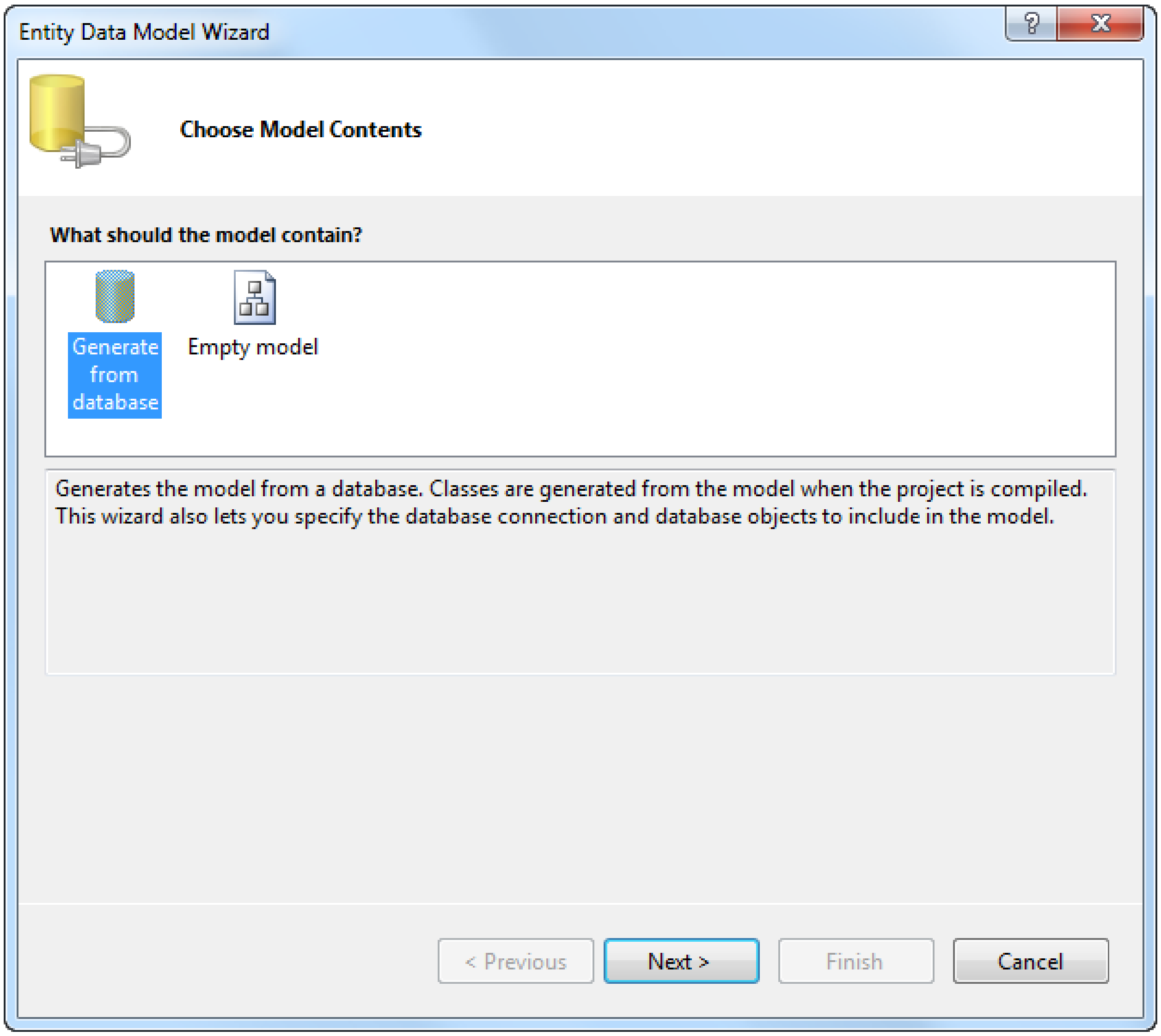


Type: Visual C#

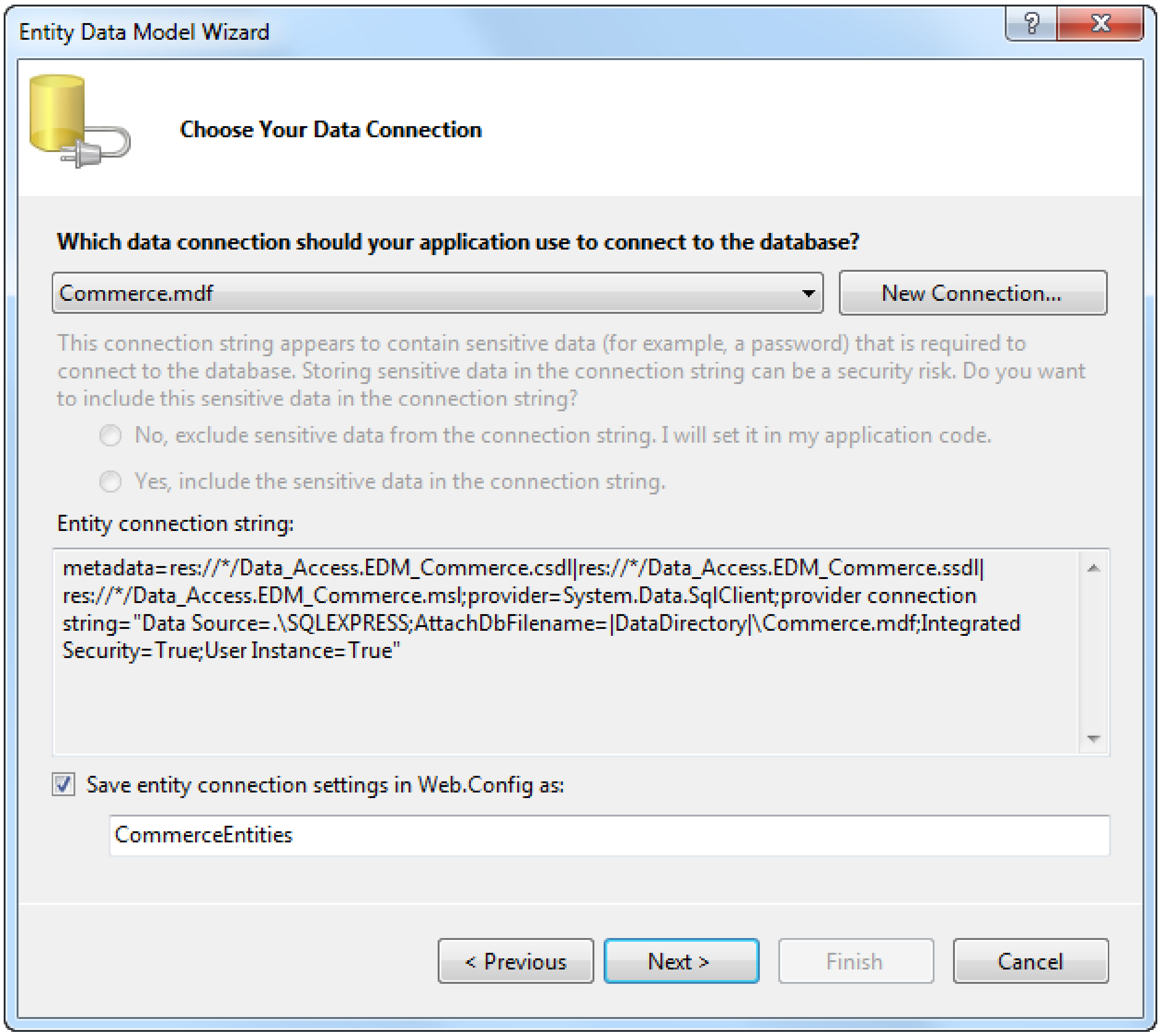
A project item for creating an ADO.NET Entity Data Model.

Name: EDM_Commerce.edmx

Choose “Generate from Database”.



Generates the model from a database. Classes are generated from the model when the project is compiled. This wizard also lets you specify the database connection and database objects to include in the model.



Choose Your Data Connection

Which data connection should your application use to connect to the database?

Commerce.mdf

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

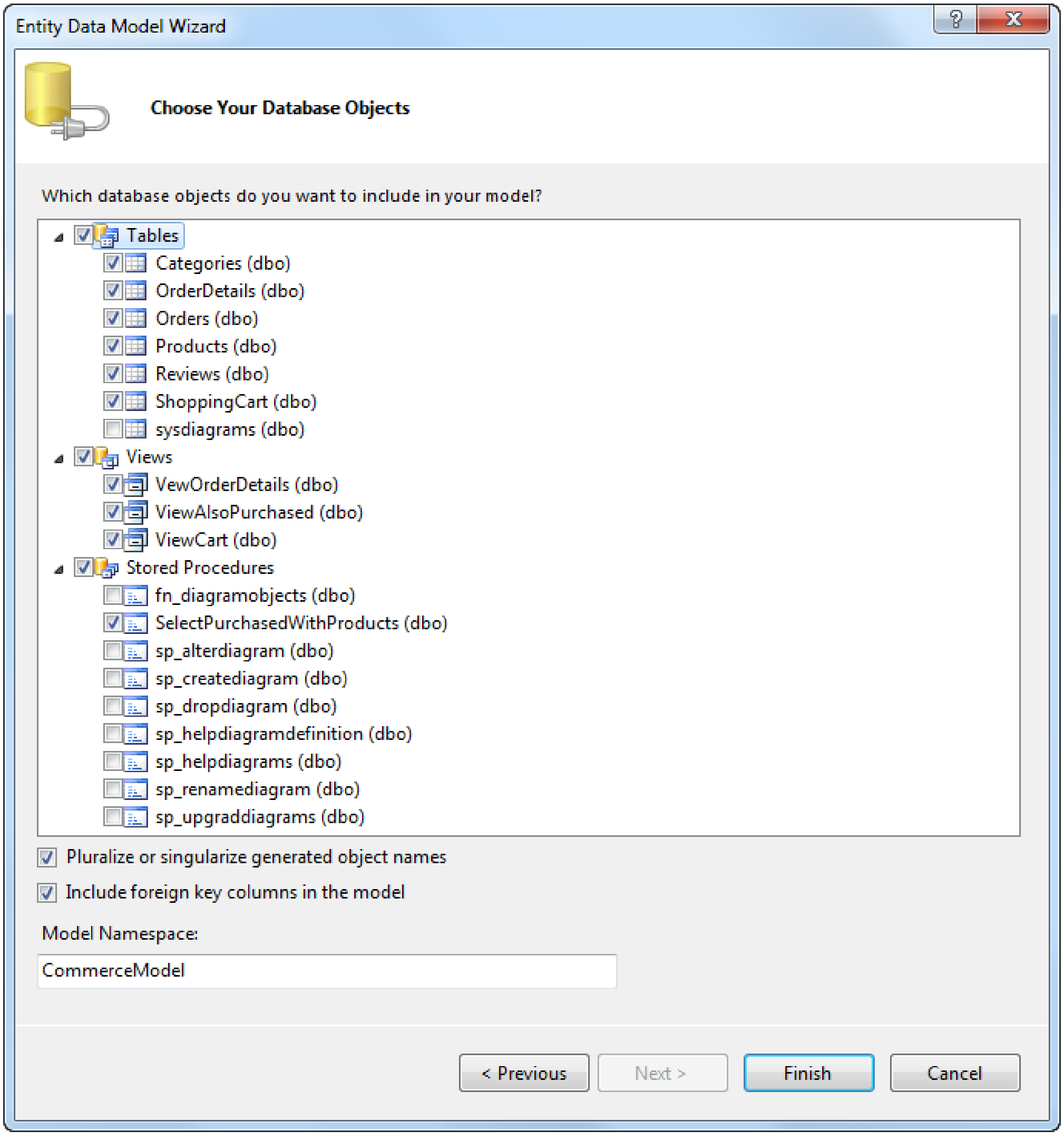
- No, exclude sensitive data from the connection string. I will set it in my application code.
- Yes, include the sensitive data in the connection string.

Entity connection string:

```
metadata=res://*/Data_Access.EDM_Commerce.csdl|res://*/Data_Access.EDM_Commerce.ssdl|
res://*/Data_Access.EDM_Commerce.msl;provider=System.Data.SqlClient;provider connection
string="Data Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\Commerce.mdf;Integrated
Security=True;User Instance=True"
```

[✓] Save entity connection settings in Web.Config as:

CommerceEntities



Entity Data Model Wizard

Choose Your Database Objects

Which database objects do you want to include in your model?

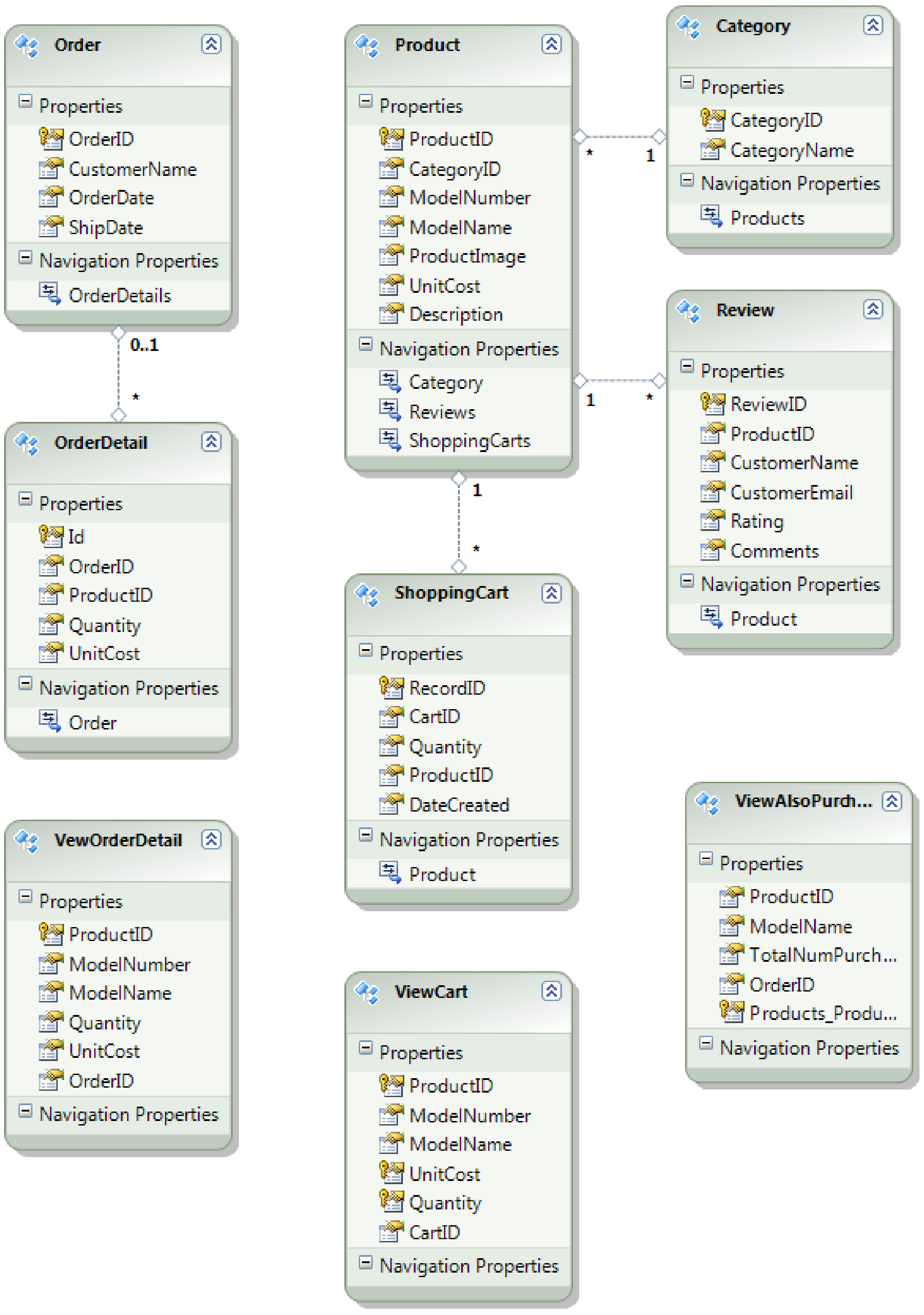
- [✓] Tables
  - [✓] Categories (dbo)
  - [✓] OrderDetails (dbo)
  - [✓] Orders (dbo)
  - [✓] Products (dbo)
  - [✓] Reviews (dbo)
  - [✓] ShoppingCart (dbo)
  - [ ] sysdiagrams (dbo)
- [✓] Views
  - [✓] VewOrderDetails (dbo)
  - [✓] ViewAlsoPurchased (dbo)
  - [✓] ViewCart (dbo)
- [✓] Stored Procedures
  - [ ] fn_diagramobjects (dbo)
  - [✓] SelectPurchasedWithProducts (dbo)
  - [ ] sp_alterdiagram (dbo)
  - [ ] sp_creatediagram (dbo)
  - [ ] sp_dropdiagram (dbo)
  - [ ] sp_helpdiagramdefinition (dbo)
  - [ ] sp_helpdiagrams (dbo)
  - [ ] sp_renamediagram (dbo)
  - [ ] sp_upgraddiagrams (dbo)

[✓] Pluralize or singularize generated object names

[✓] Include foreign key columns in the model

Model Namespace:

CommerceModel



Save and build.

Now we are ready to add our first feature – a product category menu.

Adding Some Layout and a Category Menu

In our site master page we'll add a div for the left side column that will contain our product category menu.

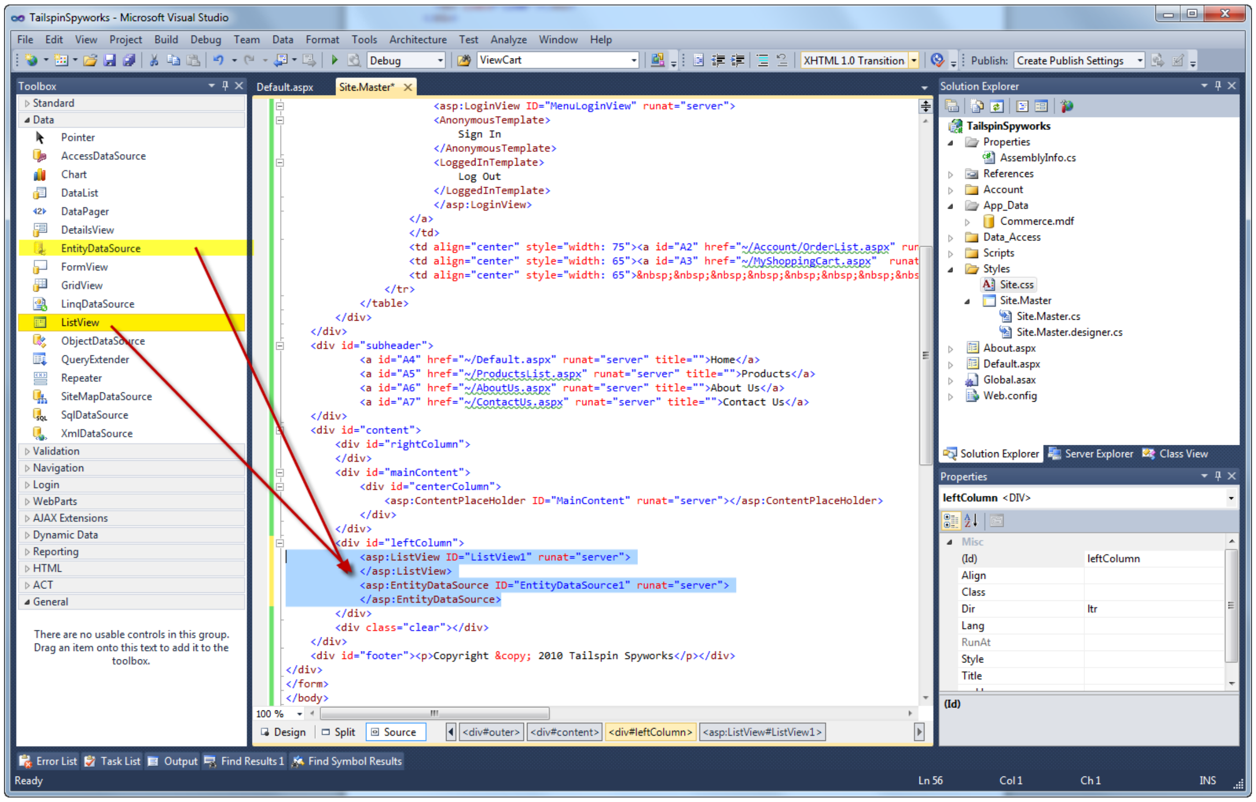
```
<div id="content">
  <div id="rightColumn"></div>
  <div id="mainContent">
    <div id="centerColumn">
       <asp:ContentPlaceHolder ID="MainContent" runat="server"></asp:ContentPlaceHolder>
    </div>
  </div>
  <div id="leftColumn">
  <!—Our menu will go here. →
  </div>
  <div class="clear"></div>
</div>
```

Note that the desired aligning and other formatting will be provided by the CSS class that we added to our Style.css file.

```
#leftColumn
{
       position: relative;
       float: left;
       width: 14em;
       padding: 2em 1.5em 2em;
       background: #fff url('images/a1.gif') repeat-y right top;
       top: 1px;
       left: 0px;
       height: 100%;
}
```

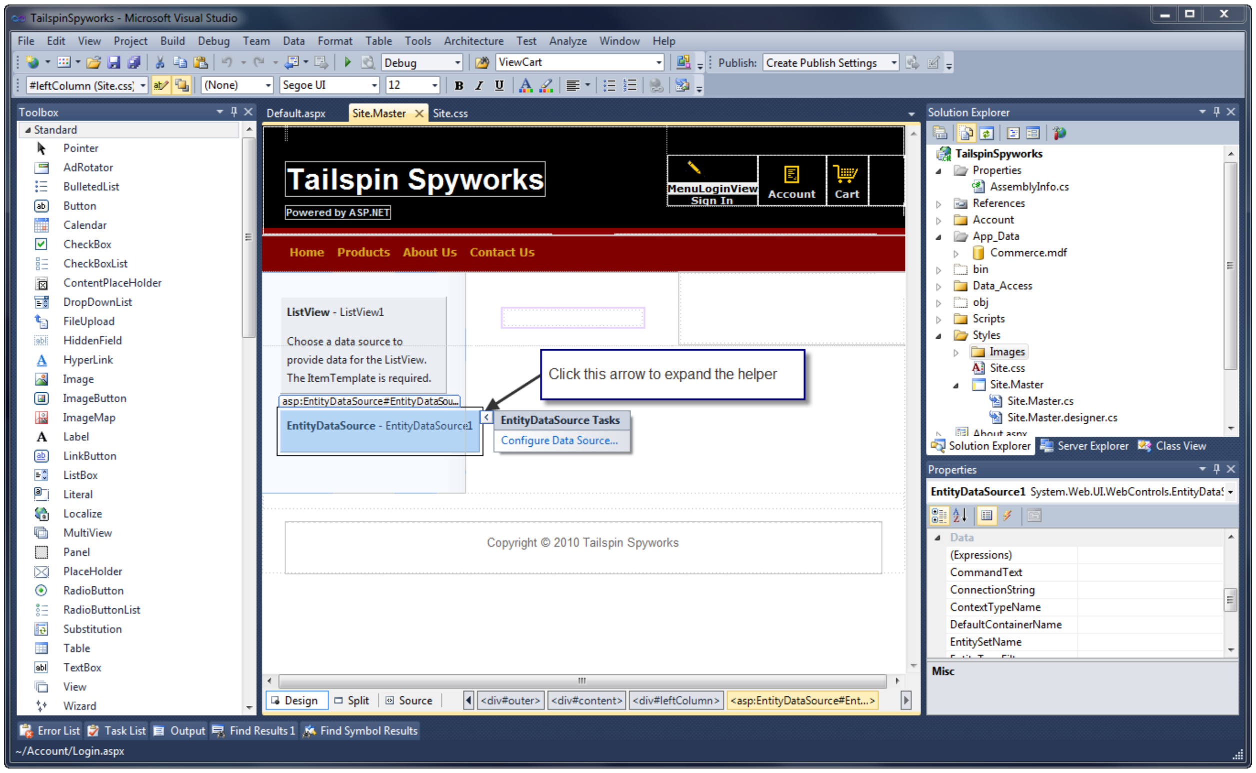
The product category menu will be dynamically created at runtime by querying the Commerce database for existing product categories and creating the menu items and corresponding links.

To accomplish this we will use two of ASP.NET's powerful data controls. The "Entity Data Source" control and the "ListView" control.

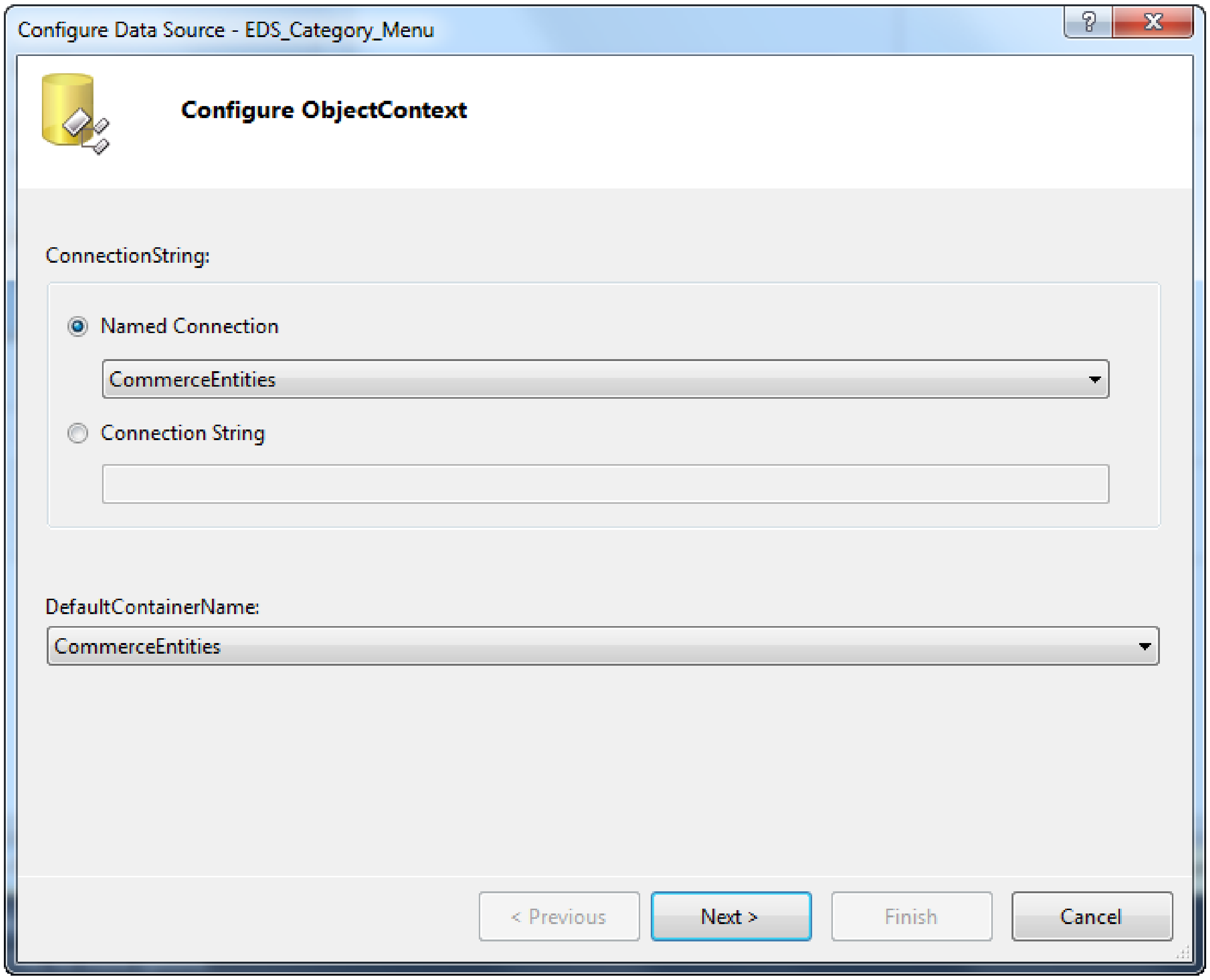


```
                        <asp:LoginView ID="MenuLoginView" runat="server">
                        <AnonymousTemplate>
                            Sign In
                        </AnonymousTemplate>
                        <LoggedInTemplate>
                            Log Out
                        </LoggedInTemplate>
                        </asp:LoginView>
                    </a>
                    </td>
                    <td align="center" style="width: 75"><a id="A2" href="~/Account/OrderList.aspx" run
                    <td align="center" style="width: 65"><a id="A3" href="~/MyShoppingCart.aspx"  runat
                    <td align="center" style="width: 65">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbs
                </tr>
            </table>
        </div>
    </div>
    <div id="subheader">
            <a id="A4" href="~/Default.aspx" runat="server" title="">Home</a>
            <a id="A5" href="~/ProductsList.aspx" runat="server" title="">Products</a>
            <a id="A6" href="~/AboutUs.aspx" runat="server" title="">About Us</a>
            <a id="A7" href="~/ContactUs.aspx" runat="server" title="">Contact Us</a>
    </div>
    <div id="content">
        <div id="rightColumn">
        </div>
        <div id="mainContent">
            <div id="centerColumn">
                <asp:ContentPlaceHolder ID="MainContent" runat="server"></asp:ContentPlaceHolder>
            </div>
        </div>
        <div id="leftColumn">
            <asp:ListView ID="ListView1" runat="server">
            </asp:ListView>
            <asp:EntityDataSource ID="EntityDataSource1" runat="server">
            </asp:EntityDataSource>
        </div>
        <div class="clear"></div>
    </div>
    <div id="footer"><p>Copyright &copy; 2010 Tailspin Spyworks</p></div>
</div>
</form>
</body>
```

Let's switch to "Design View" and use the helpers to configure our controls.


Tailspin Spyworks
Powered by ASP.NET
MenuLoginView Sign In
Account
Cart
Home Products About Us Contact Us
ListView - ListView1
Choose a data source to provide data for the ListView. The ItemTemplate is required.
asp:EntityDataSource#EntityDataSou...
EntityDataSource - EntityDataSource1
EntityDataSource Tasks
Configure Data Source...
Click this arrow to expand the helper
Copyright © 2010 Tailspin Spyworks

Let's set the EntityDataSource ID property to EDS_Category_Menu and click on "Configure Data Source".



Configure Data Source - EDS_Category_Menu

Configure ObjectContext

ConnectionString:

(●) Named Connection

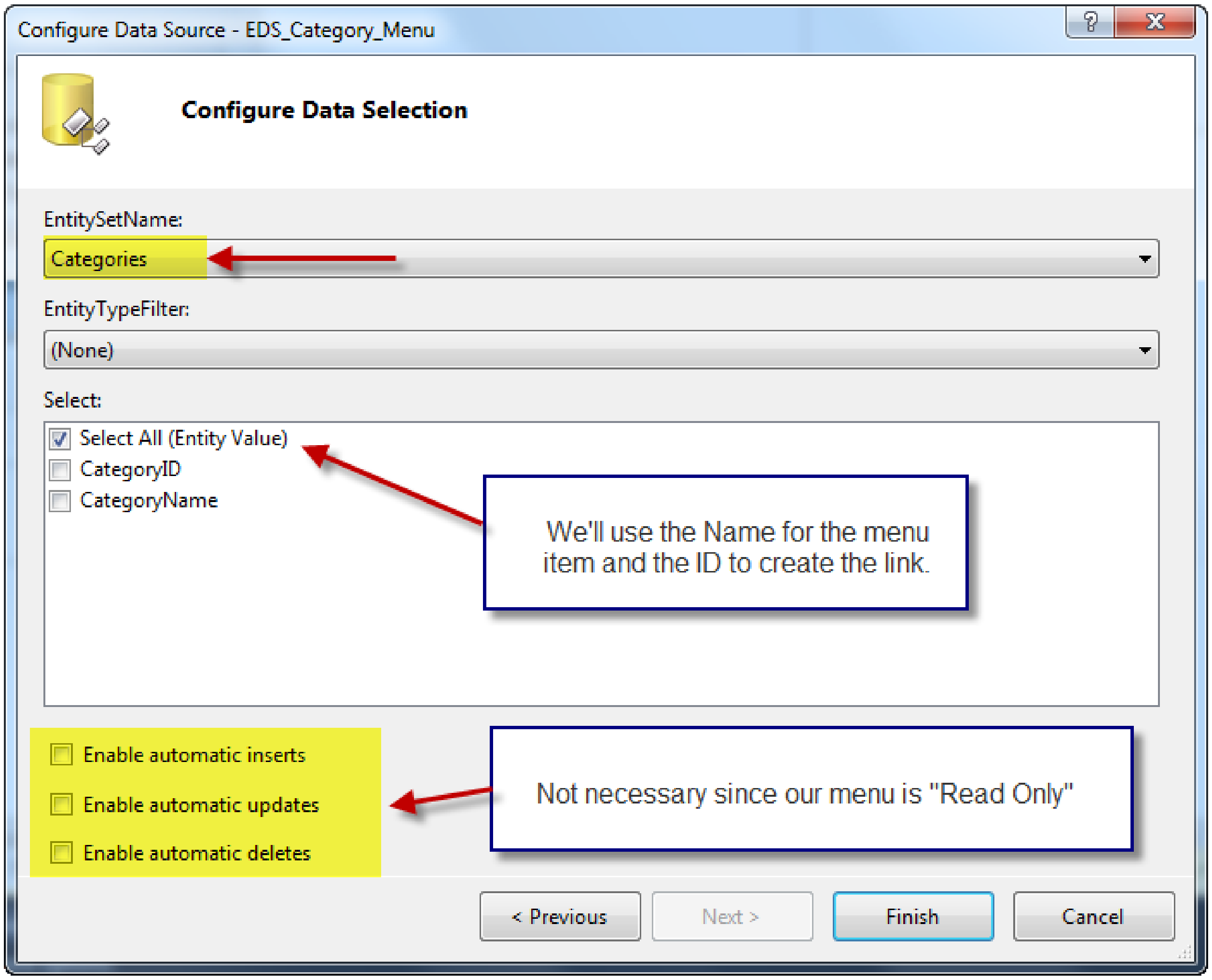
CommerceEntities

( ) Connection String

DefaultContainerName:

CommerceEntities

Select the CommerceEntities Connection that was created for us when we created the Entity Data Source Model for our Commerce Database and click "Next".


Configure Data Source - EDS_Category_Menu

Configure Data Selection

EntitySetName: Categories

EntityTypeFilter: (None)

Select:
- [✓] Select All (Entity Value)
- [ ] CategoryID
- [ ] CategoryName

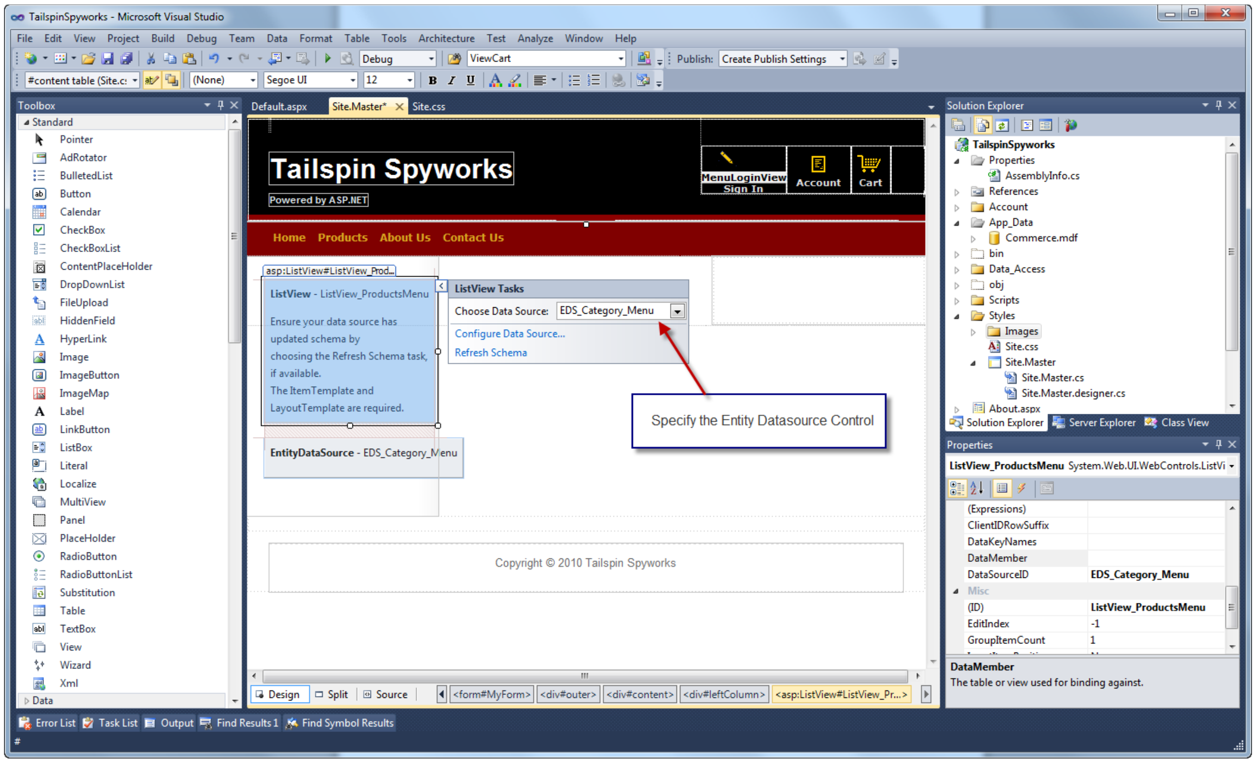
We'll use the Name for the menu item and the ID to create the link.

- [ ] Enable automatic inserts
- [ ] Enable automatic updates
- [ ] Enable automatic deletes

Not necessary since our menu is "Read Only"

Select the "Categories" Entity set name and leave the rest of the options as default. Click "Finish".

Now let's set the ID property of the ListView control instance that we placed on our page to ListView_ProductsMenu and activate its helper.


ListView Tasks
Choose Data Source: EDS_Category_Menu
Configure Data Source...
Refresh Schema
EntityDataSource - EDS_Category_Menu

ListView - ListView_ProductsMenu

Ensure your data source has updated schema by choosing the Refresh Schema task, if available. The ItemTemplate and LayoutTemplate are required.

Specify the Entity Datasource Control

Though we could use control options to format the data item display and formatting, our menu creation will only require simple markup so we will enter the code in the source view.

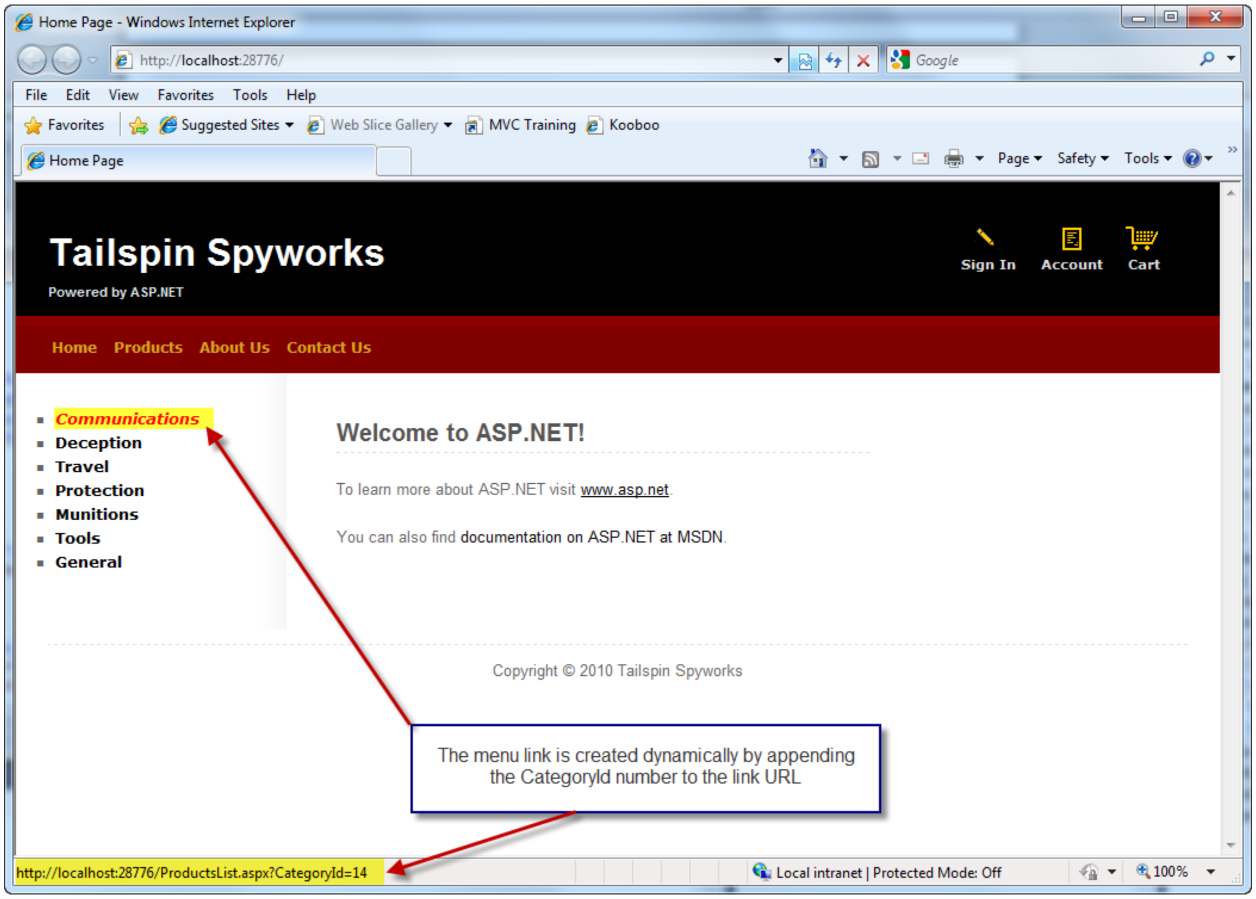
```
<asp:ListView ID="ListView_ProductsMenu" runat="server" DataKeyNames="CategoryID"
              DataSourceID="EDS_Category_Menu">
   <EmptyDataTemplate>No Menu Items.</EmptyDataTemplate>
   <ItemSeparatorTemplate></ItemSeparatorTemplate>
   <ItemTemplate>
      <li>
         <a href='<%# VirtualPathUtility.ToAbsolute("~/ProductsList.aspx?CategoryId=" +
                                    Eval("CategoryID")) %>'><%# Eval("CategoryName") %></a>
      </li>
   </ItemTemplate>
   <LayoutTemplate>
      <ul ID="itemPlaceholderContainer" runat="server"
                           style="font-family: Verdana, Arial, Helvetica, sans-serif;">
         <li runat="server" id="itemPlaceholder" />
      </ul>
      <div style="text-align: center;background-color: #FFCC66;font-family: Verdana,
                                          Arial, Helvetica, sans-serif;color: #333333;">
      </div>
   </LayoutTemplate>
</asp:ListView>
```

Please note the "Eval" statement : `<%# Eval("CategoryName") %>`

The ASP.NET syntax `<%# %>` is a shorthand convention that instructs the runtime to execute whatever is contained within and output the results "in Line".

The statement Eval("CategoryName") instructs that, for the current entry in the bound collection of data items, fetch the value of the Entity Model item names “CatagoryName”. This is concise syntax for a very powerful feature.

Lets run the application now.

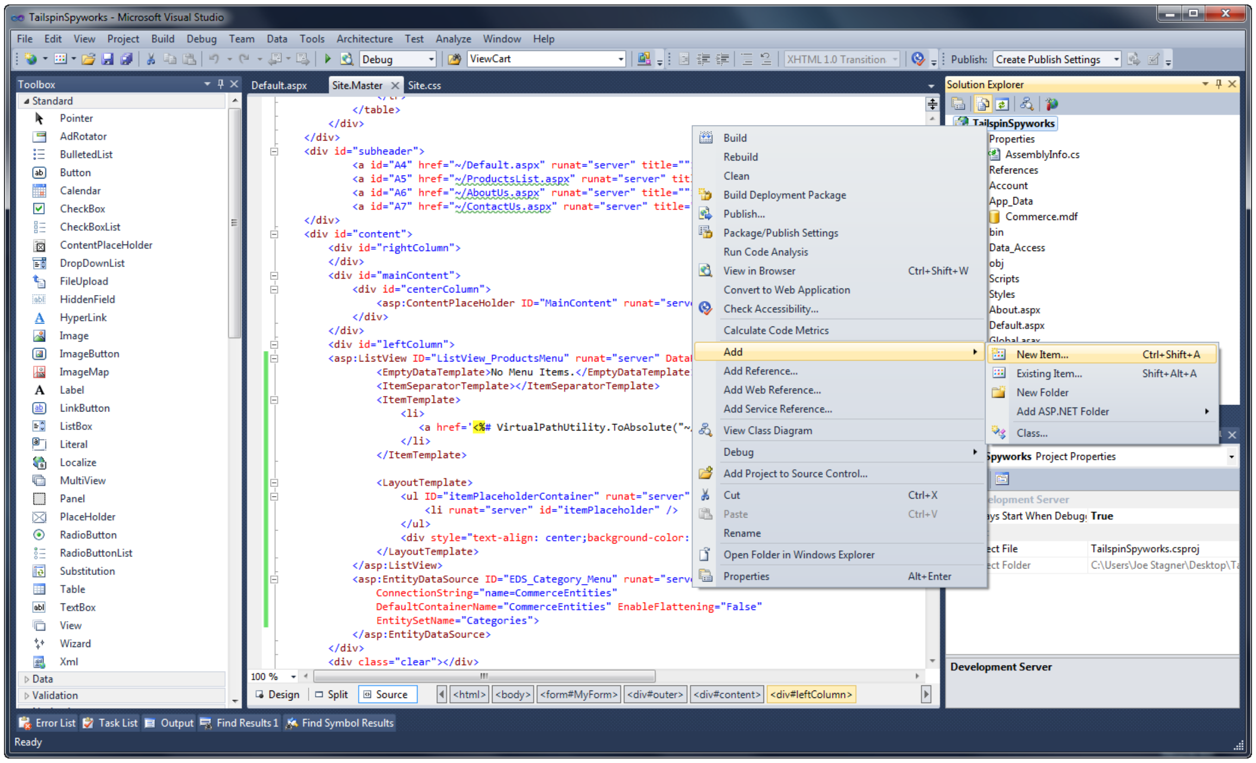


Home Page - Windows Internet Explorer
http://localhost:28776/
File Edit View Favorites Tools Help
Favorites Suggested Sites Web Slice Gallery MVC Training Kooboo
Home Page
Page Safety Tools
Tailspin Spyworks
Powered by ASP.NET
Sign In Account Cart
Home Products About Us Contact Us
Communications
Deception
Travel
Protection
Munitions
Tools
General
Welcome to ASP.NET!
To learn more about ASP.NET visit www.asp.net.
You can also find documentation on ASP.NET at MSDN.
Copyright © 2010 Tailspin Spyworks
The menu link is created dynamically by appending the CategoryId number to the link URL
http://localhost:28776/ProductsList.aspx?CategoryId=14
Local intranet | Protected Mode: Off
100%

Note that our product category menu is now displayed and when we hover over one of the category menu items we can see the menu item link points to a page we have yet to implement named ProductsList.aspx and that we have built a dynamic query string argument that contains the category id.

Listing Products with the GridView Control

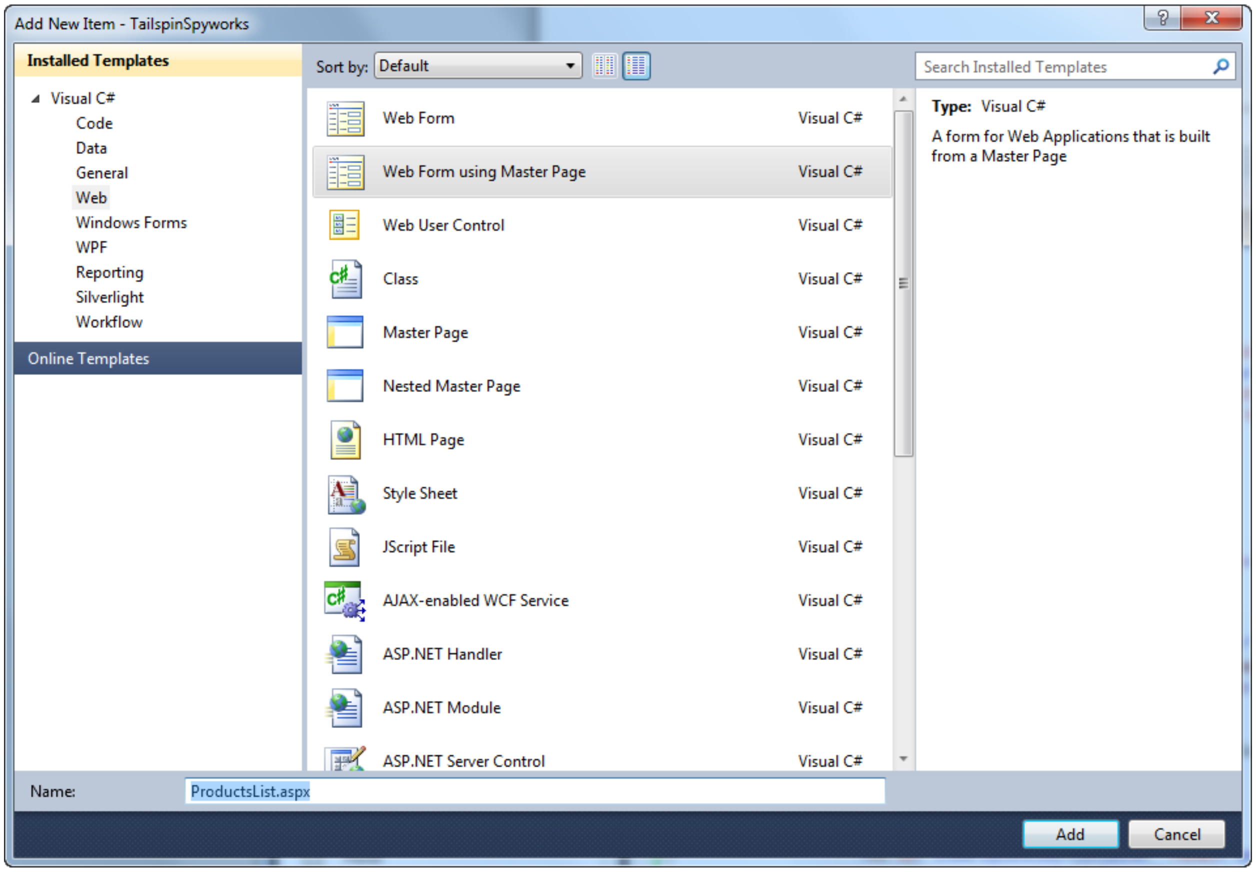
Let’s begin implementing our ProductsList.aspx page by “Right Clicking” on our solution and selecting “Add” and “New Item”.



```
                </table>
            </div>
        </div>
        <div id="subheader">
                <a id="A4" href="~/Default.aspx" runat="server" title=""
                <a id="A5" href="~/ProductsList.aspx" runat="server" tit
                <a id="A6" href="~/AboutUs.aspx" runat="server" title=""
                <a id="A7" href="~/ContactUs.aspx" runat="server" title=
        </div>
        <div id="content">
            <div id="rightColumn">
            </div>
            <div id="mainContent">
                <div id="centerColumn">
                    <asp:ContentPlaceHolder ID="MainContent" runat="serv
                </div>
            </div>
            <div id="leftColumn">
            <asp:ListView ID="ListView_ProductsMenu" runat="server" Data
                    <EmptyDataTemplate>No Menu Items.</EmptyDataTemplate
                    <ItemSeparatorTemplate></ItemSeparatorTemplate>
                    <ItemTemplate>
                        <li>
                            <a href='<%# VirtualPathUtility.ToAbsolute("~
                        </li>
                    </ItemTemplate>

                    <LayoutTemplate>
                        <ul ID="itemPlaceholderContainer" runat="server"
                            <li runat="server" id="itemPlaceholder" />
                        </ul>
                        <div style="text-align: center;background-color:
                    </LayoutTemplate>
                </asp:ListView>
                <asp:EntityDataSource ID="EDS_Category_Menu" runat="serv
                    ConnectionString="name=CommerceEntities"
                    DefaultContainerName="CommerceEntities" EnableFlattening="False"
                    EntitySetName="Categories">
                </asp:EntityDataSource>
            </div>
            <div class="clear"></div>
```

Choose “Web Form Using Master Page” and enter a page name of ProductsList.aspx”.

Click “Add”.

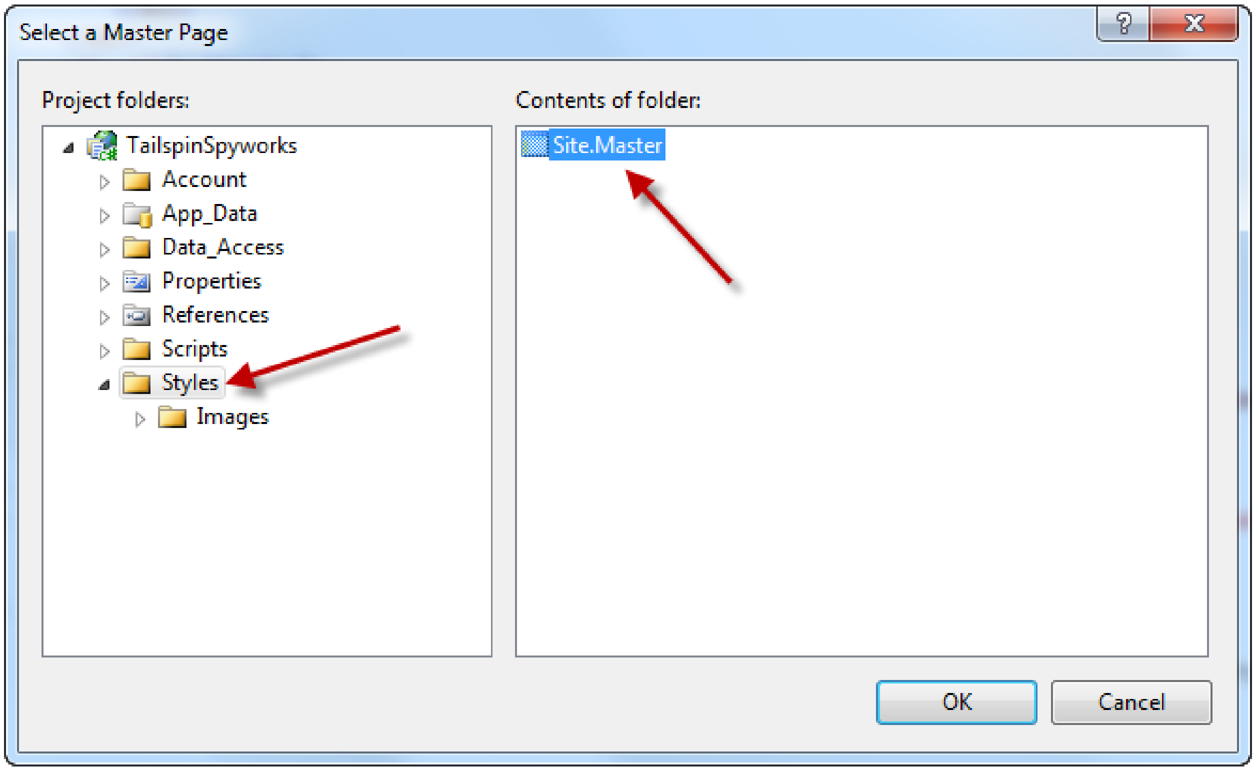


Type: Visual C#

A form for Web Applications that is built from a Master Page

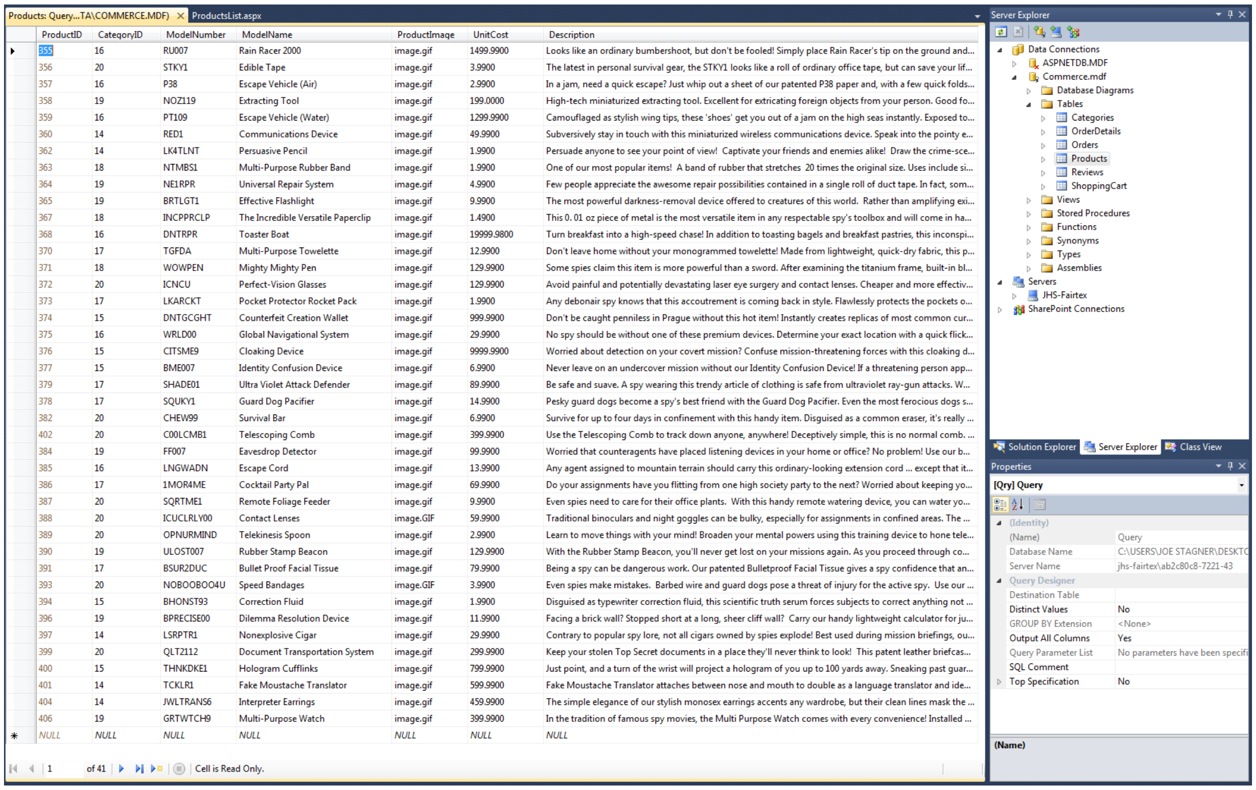
Name: ProductsList.aspx

Next choose the “Styles” folder where we placed the Site.Master page and select it form the “Contents of folder” window.



Click "Ok" to create the page.

Our database is populated with product data as seen below.



ProductID	CategoryID	ModelNumber	ModelName	ProductImage	UnitCost	Description
355	16	RU007	Rain Racer 2000	image.gif	1499.9900	Looks like an ordinary bumbershoot, but don't be fooled! Simply place Rain Racer's tip on the ground and...
356	20	STKY1	Edible Tape	image.gif	3.9900	The latest in personal survival gear, the STKY1 looks like a roll of ordinary office tape, but can save your lif...
357	16	P38	Escape Vehicle (Air)	image.gif	2.9900	In a jam, need a quick escape? Just whip out a sheet of our patented P38 paper and, with a few quick folds...
358	19	NOZ119	Extracting Tool	image.gif	199.0000	High-tech miniaturized extracting tool. Excellent for extricating foreign objects from your person. Good fo...
359	16	PT109	Escape Vehicle (Water)	image.gif	1299.9900	Camouflaged as stylish wing tips, these 'shoes' get you out of a jam on the high seas instantly. Exposed to...
360	14	RED1	Communications Device	image.gif	49.9900	Subversively stay in touch with this miniaturized wireless communications device. Speak into the pointy e...
362	14	LK4TLNT	Persuasive Pencil	image.gif	1.9900	Persuade anyone to see your point of view! Captivate your friends and enemies alike! Draw the crime-sce...
363	18	NTMBS1	Multi-Purpose Rubber Band	image.gif	1.9900	One of our most popular items! A band of rubber that stretches 20 times the original size. Uses include si...
364	19	NE1RPR	Universal Repair System	image.gif	4.9900	Few people appreciate the awesome repair possibilities contained in a single roll of duct tape. In fact, som...
365	19	BRTLGT1	Effective Flashlight	image.gif	9.9900	The most powerful darkness-removal device offered to creatures of this world. Rather than amplifying exi...
367	18	INCPPRCLP	The Incredible Versatile Paperclip	image.gif	1.4900	This 0. 01 oz piece of metal is the most versatile item in any respectable spy's toolbox and will come in ha...
368	16	DNTRPR	Toaster Boat	image.gif	19999.9800	Turn breakfast into a high-speed chase! In addition to toasting bagels and breakfast pastries, this inconspi...
370	17	TGFDA	Multi-Purpose Towelette	image.gif	12.9900	Don't leave home without your monogrammed towelette! Made from lightweight, quick-dry fabric, this p...
371	18	WOWPEN	Mighty Mighty Pen	image.gif	129.9900	Some spies claim this item is more powerful than a sword. After examining the titanium frame, built-in bl...
372	20	ICNCU	Perfect-Vision Glasses	image.gif	129.9900	Avoid painful and potentially devastating laser eye surgery and contact lenses. Cheaper and more effectiv...
373	17	LKARCKT	Pocket Protector Rocket Pack	image.gif	1.9900	Any debonair spy knows that this accoutrement is coming back in style. Flawlessly protects the pockets o...
374	15	DNTGCGHT	Counterfeit Creation Wallet	image.gif	999.9900	Don't be caught penniless in Prague without this hot item! Instantly creates replicas of most common cur...
375	16	WRLD00	Global Navigational System	image.gif	29.9900	No spy should be without one of these premium devices. Determine your exact location with a quick flick...
376	15	CITSME9	Cloaking Device	image.gif	9999.9900	Worried about detection on your covert mission? Confuse mission-threatening forces with this cloaking d...
377	15	BME007	Identity Confusion Device	image.gif	6.9900	Never leave on an undercover mission without our Identity Confusion Device! If a threatening person app...
379	17	SHADE01	Ultra Violet Attack Defender	image.gif	89.9900	Be safe and suave. A spy wearing this trendy article of clothing is safe from ultraviolet ray-gun attacks. W...
378	17	SQUKY1	Guard Dog Pacifier	image.gif	14.9900	Pesky guard dogs become a spy's best friend with the Guard Dog Pacifier. Even the most ferocious dogs s...
382	20	CHEW99	Survival Bar	image.gif	6.9900	Survive for up to four days in confinement with this handy item. Disguised as a common eraser, it's really ...
402	20	C00LCMB1	Telescoping Comb	image.gif	399.9900	Use the Telescoping Comb to track down anyone, anywhere! Deceptively simple, this is no normal comb. ...
384	19	FF007	Eavesdrop Detector	image.gif	99.9900	Worried that counteragents have placed listening devices in your home or office? No problem! Use our b...
385	16	LNGWADN	Escape Cord	image.gif	13.9900	Any agent assigned to mountain terrain should carry this ordinary-looking extension cord ... except that it...
386	17	1MOR4ME	Cocktail Party Pal	image.gif	69.9900	Do your assignments have you flitting from one high society party to the next? Worried about keeping yo...
387	20	SQRTME1	Remote Foliage Feeder	image.gif	9.9900	Even spies need to care for their office plants. With this handy remote watering device, you can water yo...
388	20	ICUCLRLY00	Contact Lenses	image.GIF	59.9900	Traditional binoculars and night goggles can be bulky, especially for assignments in confined areas. The ...
389	20	OPNURMIND	Telekinesis Spoon	image.gif	2.9900	Learn to move things with your mind! Broaden your mental powers using this training device to hone tele...
390	19	ULOST007	Rubber Stamp Beacon	image.gif	129.9900	With the Rubber Stamp Beacon, you'll never get lost on your missions again. As you proceed through co...
391	17	BSUR2DUC	Bullet Proof Facial Tissue	image.gif	79.9900	Being a spy can be dangerous work. Our patented Bulletproof Facial Tissue gives a spy confidence that an...
393	20	NOBOOBOO4U	Speed Bandages	image.GIF	3.9900	Even spies make mistakes. Barbed wire and guard dogs pose a threat of injury for the active spy. Use our ...
394	15	BHONST93	Correction Fluid	image.gif	1.9900	Disguised as typewriter correction fluid, this scientific truth serum forces subjects to correct anything not ...
396	19	BPRECISE00	Dilemma Resolution Device	image.gif	11.9900	Facing a brick wall? Stopped short at a long, sheer cliff wall? Carry our handy lightweight calculator for ju...
397	14	LSRPTR1	Nonexplosive Cigar	image.gif	29.9900	Contrary to popular spy lore, not all cigars owned by spies explode! Best used during mission briefings, ou...
399	20	QLT2112	Document Transportation System	image.gif	299.9900	Keep your stolen Top Secret documents in a place they'll never think to look! This patent leather briefcas...
400	15	THNKDKE1	Hologram Cufflinks	image.gif	799.9900	Just point, and a turn of the wrist will project a hologram of you up to 100 yards away. Sneaking past guar...
401	14	TCKLR1	Fake Moustache Translator	image.gif	599.9900	Fake Moustache Translator attaches between nose and mouth to double as a language translator and ide...
404	14	JWLTRANS6	Interpreter Earrings	image.gif	459.9900	The simple elegance of our stylish monosex earrings accents any wardrobe, but their clean lines mask the ...
406	19	GRTWTCH9	Multi-Purpose Watch	image.gif	399.9900	In the tradition of famous spy movies, the Multi Purpose Watch comes with every convenience! Installed ...
NULL	NULL	NULL	NULL	NULL	NULL	NULL

1 of 41 Cell is Read Only.

Data Connections
- ASPNETDB.MDF
- Commerce.mdf
  - Database Diagrams
  - Tables
    - Categories
    - OrderDetails
    - Orders
    - Products
    - Reviews
    - ShoppingCart
  - Views
  - Stored Procedures
  - Functions
  - Synonyms
  - Types
  - Assemblies

Servers
- JHS-Fairtex

SharePoint Connections

Properties

[Qry] Query

(Identity)	
(Name)	Query
Database Name	C:\USERS\JOE STAGNER\DESKTC
Server Name	jhs-fairtex\ab2c80c8-7221-43
Query Designer	
Destination Table	
Distinct Values	No
GROUP BY Extension	<None>
Output All Columns	Yes
Query Parameter List	No parameters have been specifi
SQL Comment	
Top Specification	No

(Name)

After our page is created we'll again use an Entity Data Source to access that product data, but in this instance we need to select the Product Entities and we need to restrict the items that are returned to only those for the selected Category.

To accomplish this we'll tell the EntityDataSource to Auto Generate the WHERE clause and we'll specify the WhereParameter.

You'll recall that when we created the Menu Items in our "Product Category Menu" we dynamically built the link by adding the CatagoryID to the QueryString for each link. We will tell the Entity Data Source to derive the WHERE parameter from that QueryString parameter.

```
<asp:EntityDataSource ID="EDS_ProductsByCategory" runat="server"
                      EnableFlattening="False" AutoGenerateWhereClause="True"
                      ConnectionString="name=CommerceEntities"
                      DefaultContainerName="CommerceEntities"
                      EntitySetName="Products">
       <WhereParameters>
              <asp:QueryStringParameter Name="CategoryID"
                                        QueryStringField="Category Id"
                                        Type="Int32" />
       </WhereParameters>
</asp:EntityDataSource>
```

Next, we'll configure the ListView control to display a list of products. To create an optimal shopping experience we'll compact several concise features into each individual product displayed in our ListVew.

- The product name will be a link to the product's detail view.
- The product's price will be displayed.
- An image of the product will be displayed and we'll dynamically select the image from a catalog images directory in our application.
- We will include a link to immediately add the specific product to the shopping cart.

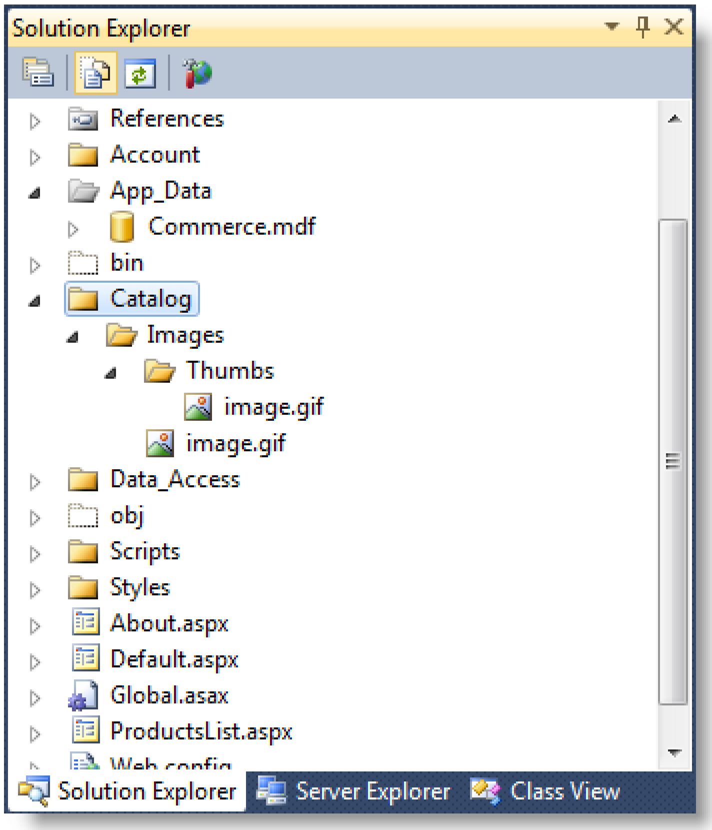
Here is the markup for our ListView control instance.

```
<asp:ListView ID="ListView_Products" runat="server"
              DataKeyNames="ProductID"
              DataSourceID="EDS_ProductsByCategory"
              GroupItemCount="2">
   <EmptyDataTemplate>
      <table runat="server">
        <tr>
          <td>No data was returned.</td>
        </tr>
     </table>
  </EmptyDataTemplate>
  <EmptyItemTemplate>
     <td runat="server" />
  </EmptyItemTemplate>
  <GroupTemplate>
```

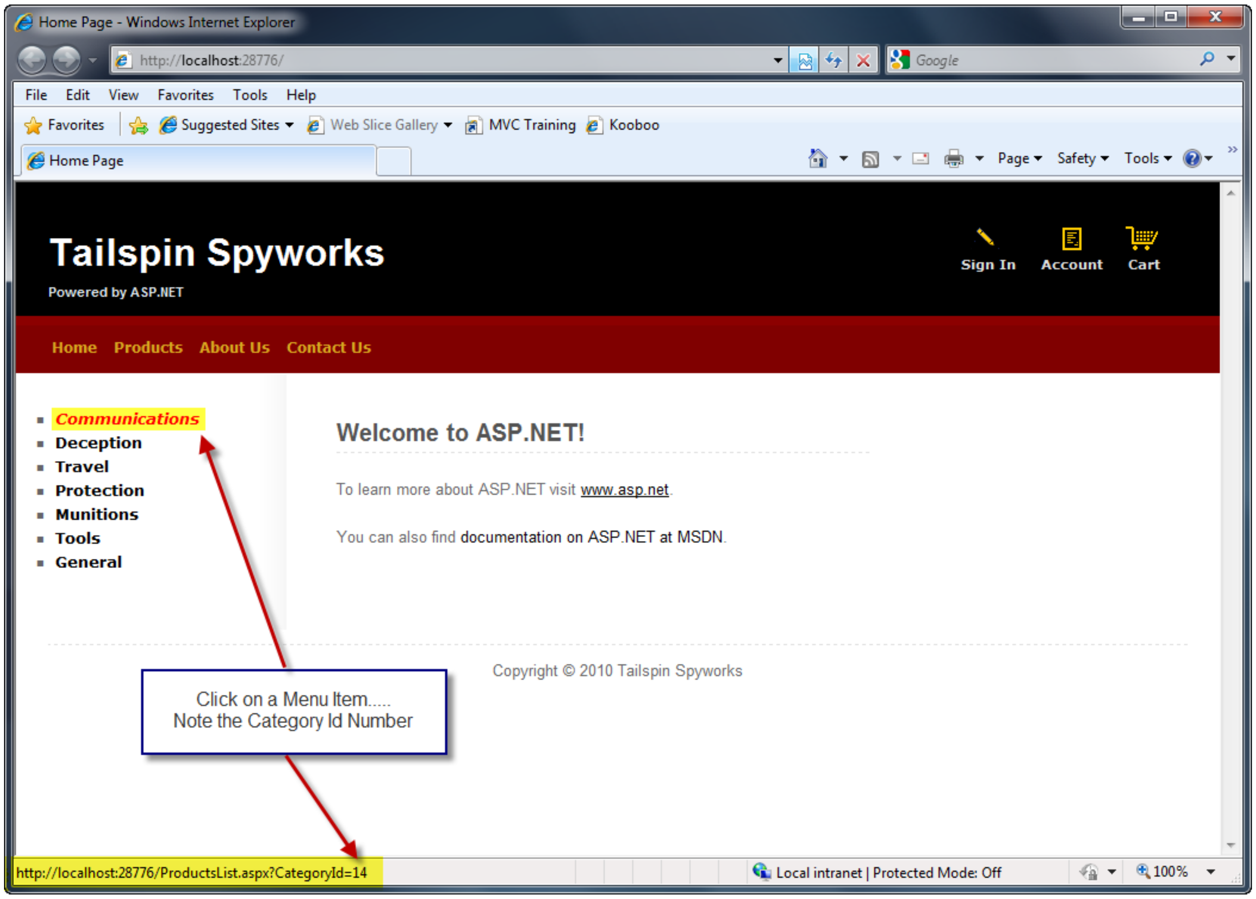
```
    <tr ID="itemPlaceholderContainer" runat="server">
      <td ID="itemPlaceholder" runat="server"></td>
    </tr>
  </GroupTemplate>
  <ItemTemplate>
    <td runat="server">
      <table border="0" width="300">
        <tr>
          <td style="width: 25px;">&nbsp</td>
          <td style="vertical-align: middle; text-align: right;">
            <a href='ProductDetails.aspx?productID=<%# Eval("ProductID") %>'>
               <image src='Catalog/Images/Thumbs/<%# Eval("ProductImage") %>'
                      width="100" height="75" border="0">
            </a>&nbsp;&nbsp
          </td>
          <td style="width: 250px; vertical-align: middle;">
            <a href='ProductDetails.aspx?productID=<%# Eval("ProductID") %>'><span
               class="ProductListHead"><%# Eval("ModelName") %></span><br>
            </a>
            <span class="ProductListItem">
              <b>Special Price: </b><%# Eval("UnitCost", "{0:c}")%>
            </span><br />
            <a href='AddToCart.aspx?productID=<%# Eval("ProductID") %>'>
               <span class="ProductListItem"><b>Add To Cart<b></font></span>
            </a>
          </td>
        </tr>
      </table>
    </td>
  </ItemTemplate>
  <LayoutTemplate>
    <table runat="server">
      <tr runat="server">
        <td runat="server">
          <table ID="groupPlaceholderContainer" runat="server">
            <tr ID="groupPlaceholder" runat="server"></tr>
          </table>
        </td>
      </tr>
      <tr runat="server"><td runat="server"></td></tr>
    </table>
  </LayoutTemplate>
</asp:ListView>
```

We are dynamically building several links for each displayed product.

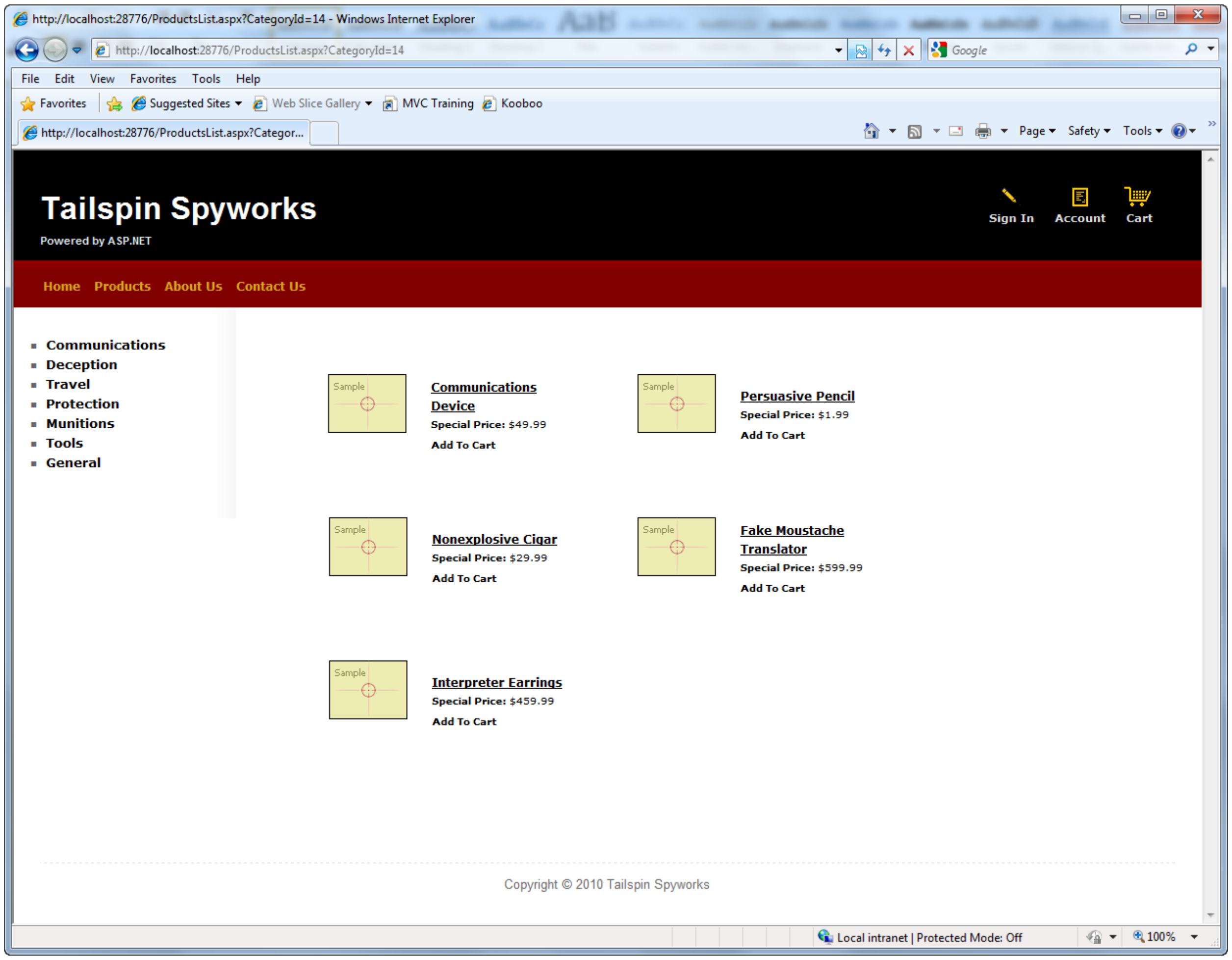
Also, before we test own new page we need to create the directory structure for the product catalog images as follows.



Once our product images are accessible we can test our product list page.


Home Page - Windows Internet Explorer
http://localhost:28776/
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Web Slice Gallery
MVC Training
Kooboo
Home Page
Page Safety Tools
Tailspin Spyworks
Powered by ASP.NET
Sign In
Account
Cart
Home
Products
About Us
Contact Us
Communications
Deception
Travel
Protection
Munitions
Tools
General
Welcome to ASP.NET!
To learn more about ASP.NET visit www.asp.net.
You can also find documentation on ASP.NET at MSDN.
Copyright © 2010 Tailspin Spyworks
Click on a Menu Item.....
Note the Category Id Number
http://localhost:28776/ProductsList.aspx?CategoryId=14
Local intranet | Protected Mode: Off
100%

From the site's home page, click on one of the Category List Links.



Now we need to implement the ProductDetials.apsx page and the AddToCart functionality.

Use File->New to create a page name ProductDetails.aspx using the site Master Page as we did previously.

We will again use an EntityDataSource control to access the specific product record in the database and we will use an ASP.NET FormView control to display the product data as follows.

```
<asp:FormView ID="FormView_Product" runat="server" DataKeyNames="ProductID"
                                                  DataSourceID="EDS_Product">
  <ItemTemplate>
    <div class="ContentHead"><%# Eval("ModelName") %></div><br />
      <table  border="0">
        <tr>
          <td style="vertical-align: top;">
            <img src='Catalog/Images/<%# Eval("ProductImage") %>'  border="0"
                                                       alt='<%# Eval("ModelName") %>' />
          </td>
          <td style="vertical-align: top"><%# Eval("Description") %>
            <br /><br /><br />
          </td>
        </tr>
      </table>
      <span class="UnitCost"><b>Your Price:</b>&nbsp;<%# Eval("UnitCost", "{0:c}")%>
      <br />
      <span class="ModelNumber">
        <b>Model Number:</b>&nbsp;<%# Eval("ModelNumber") %>
      </span><br />
      <a href='AddToCart.aspx?ProductID=
        <%# Eval("ProductID") %>' style="border: 0 none white">
        <img id="Img1" src="~/Styles/Images/add_to_cart.gif" runat="server"
             alt="" style="border-width: 0" />
      </a>
      <br /><br />
    </ItemTemplate>
  </asp:FormView>
  <asp:EntityDataSource ID="EDS_Product" runat="server" AutoGenerateWhereClause="True"
                        EnableFlattening="False"
                        ConnectionString="name=CommerceEntities"
                        DefaultContainerName="CommerceEntities"
                        EntitySetName="Products"
                        EntityTypeFilter=""
                        Select="" Where="">
    <WhereParameters>
      <asp:QueryStringParameter Name="ProductID"
                                QueryStringField="productID"  Type="Int32" />
    </WhereParameters>
  </asp:EntityDataSource>
```

Don't worry if the formatting looks a bit funny to you. The markup above leaves room in the display layout for a couple of features we'll implement later on.

The Shopping Cart will represent the more complex logic in our application. To get started, use File->New to create a page called MyShoppingCart.aspx.

Note that we are not choosing the name ShoppingCart.aspx.

Our database contains a table named "ShoppingCart". When we generated an Entity Data Model a class was created for each table in the database. Therefore, the Entity Data Model generated an Entity Class named "ShoppingCart". We could edit the model so that we could use that name for our shopping cart implementation or extend it for our needs, but we will opt instead to simply select a name that will avoid the conflict.

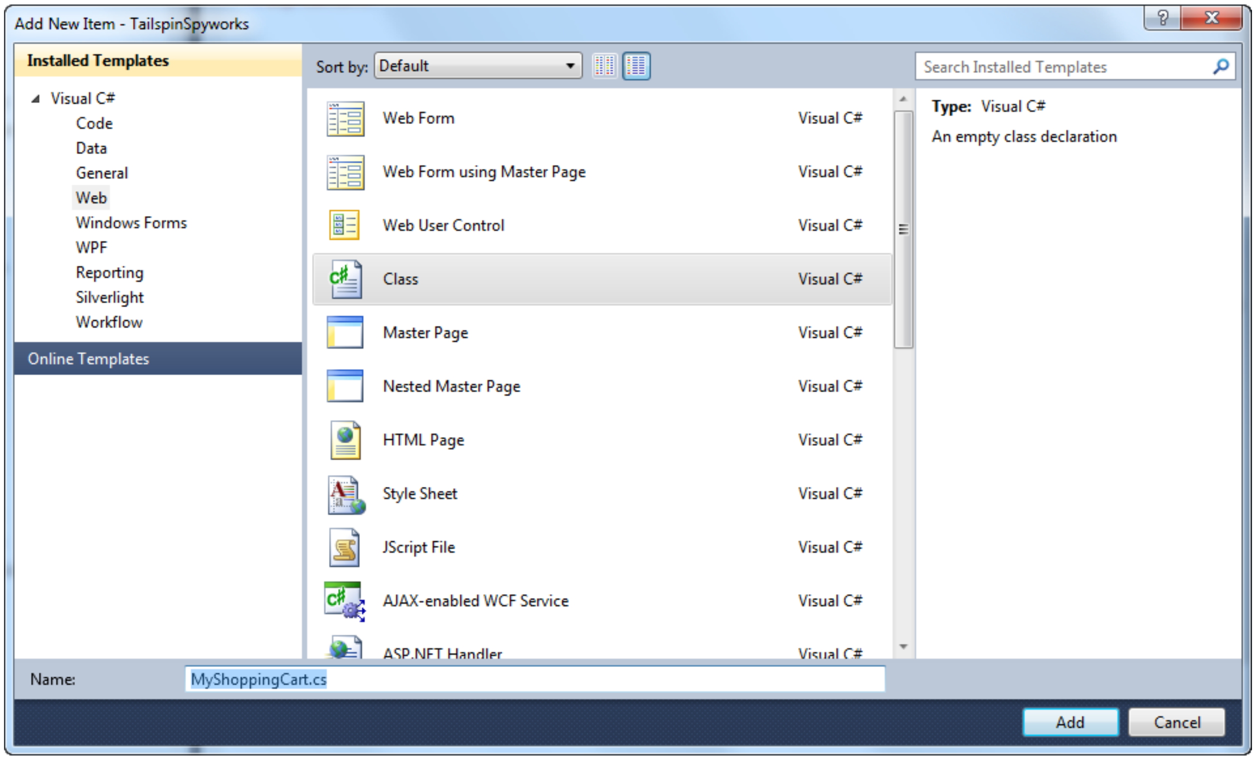
It's also worth noting that we will be creating a simple shopping cart and embedding the shopping cart logic with the shopping cart display. We might also choose to implement our shopping cart in a completely separate Business Layer.

Adding Some Business Logic

We want our shopping experience to be available whenever someone visits our web site. Visitors will be able to browse and add items to the shopping cart even if they are not registered or logged in. When they are ready to check out they will be given the option to authenticate and if they are not yet members they will be able to create an account.

This means that we will need to implement the logic to convert the shopping cart from an anonymous state to a “Registered User” state.

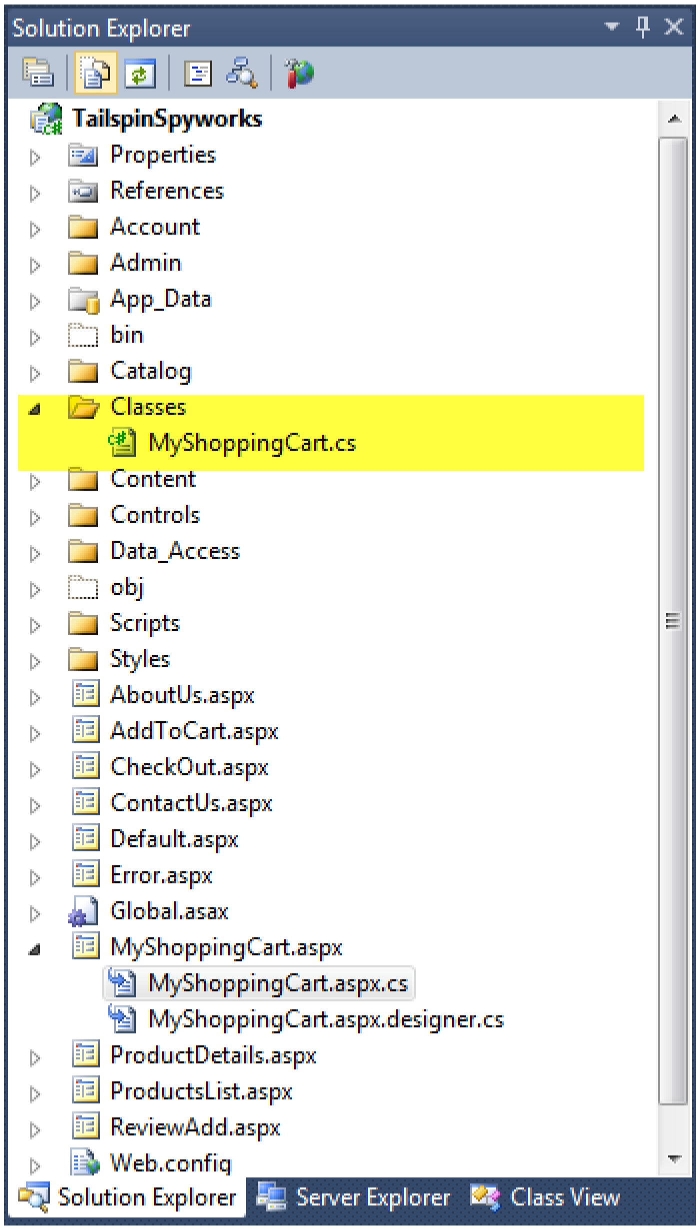
Let’s create a directory named “Classes” then Right-Click on the folder and create a new “Class” file named MyShoppingCart.cs



Type: Visual C#

An empty class declaration

Name: MyShoppingCart.cs



As previously mentioned we will be extending the class that implements the MyShoppingCart.aspx page and we will do this using .NET's powerful "Partial Class" construct.

The generated call for our MyShoppingCart.aspx.cs file looks like this.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace TailspinSpyworks
{
    public partial class MyShoppingCart : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
    }
}
```

Note the use of the “partial” keyword.

The class file that we just generated looks like this.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace TailspinSpyworks.Classes
{
    public class MyShoppingCart
    {
    }
}
```

We will merge our implementations by adding the partial keyword to this file as well.

Our new class file now looks like this.

```
namespace TailspinSpyworks.Classes
{
    public partial class MyShoppingCart
    {
    }
}
```

The first method that we will add to our class is the “AddItem” method. This is the method that will ultimately be called when the user clicks on the “Add to Art” links on the Product List and Product Details pages.

Append the following to the using statements at the top of the page.

```
using TailspinSpyworks.Data_Access;
```

And add this method to the MyShoppingCart class.

```
//------------------------------------------------------------------------------------+
public void AddItem(string cartID, int productID, int quantity)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
        var myItem = (from c in db.ShoppingCarts where c.CartID == cartID &&
                                c.ProductID == productID select c).FirstOrDefault();
        if(myItem == null)
          {
          ShoppingCart cartadd = new ShoppingCart();
          cartadd.CartID = cartID;
          cartadd.Quantity = quantity;
          cartadd.ProductID = productID;
          cartadd.DateCreated = DateTime.Now;
          db.ShoppingCarts.AddObject(cartadd);
          }
        else
          {
          myItem.Quantity += quantity;
          }
        db.SaveChanges();
        }
      catch (Exception exp)
        {
        throw new Exception("ERROR: Unable to Add Item to Cart - " +
                                                   exp.Message.ToString(), exp);
        }
    }
}
```

We are using LINQ to Entities to see if the item is already in the cart. If so, we update the order quantity of the item, otherwise we create a new entry for the selected item

In order to call this method we will implement an AddToCart.aspx page that not only calls this method but also displays the current shopping cart after the item has been added.

Right-Click on the solution name in the solution explorer and add and new page named AddToCart.aspx as we have done previously.

While we could use this page to display interim results like low stock issues, etc, in our implementation, the page will not actually render, but rather call the "Add" logic and redirect.

To accomplish this we'll add the following code to the Page_Load event.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Diagnostics;

namespace TailspinSpyworks
{
    public partial class AddToCart : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            string rawId = Request.QueryString["ProductID"];
            int productId;
            if (!String.IsNullOrEmpty(rawId) && Int32.TryParse(rawId, out productId))
            {
                MyShoppingCart usersShoppingCart = new MyShoppingCart();
                String cartId = usersShoppingCart.GetShoppingCartId();
                usersShoppingCart.AddItem(cartId, productId, 1);
            }
            else
            {
                Debug.Fail("ERROR : We should never get to AddToCart.aspx
                                                   without a ProductId.");
                throw new Exception("ERROR : It is illegal to load AddToCart.aspx
                                                   without setting a ProductId.");
            }
            Response.Redirect("MyShoppingCart.aspx");
        }
    }
}
```

Note that we are retrieving the product to add to the shopping cart from a QueryString parameter and calling the AddItem method of our class.

Assuming no errors are encountered control is passed to the ShoppingCart.aspx page which we will fully implement next. If there should be an error we throw an exception.

Currently we have not yet implemented a global error handler so this exception would go unhandled by our application but we will remedy this shortly.

Note also the use of the statement Debug.Fail() (available via `using System.Diagnostics;`)

If the application is running inside the debugger, this method will display a detailed dialog with information about the applications state along with the error message that we specify.

When running in production the Debug.Fail() statement is ignored.

You will note in the code above a call to a method in our shopping cart class named "GetShoppingCartId".

Add the code to implement the method as follows.

Note that we've also added update and checkout buttons and a label where we can display the cart "total".

```
public const string CartId = "TailSpinSpyWorks_CartID";

//--------------------------------------------------------------------------------------+
public String GetShoppingCartId()
{
  if (Session[CartId] == null)
     {
     Session[CartId] = System.Web.HttpContext.Current.Request.IsAuthenticated ?
                                 User.Identity.Name : Guid.NewGuid().ToString();
     }
  return Session[CartId].ToString();
}
```

We can now add items to our shopping cart but we have not implemented the logic to display the cart after a product has been added.

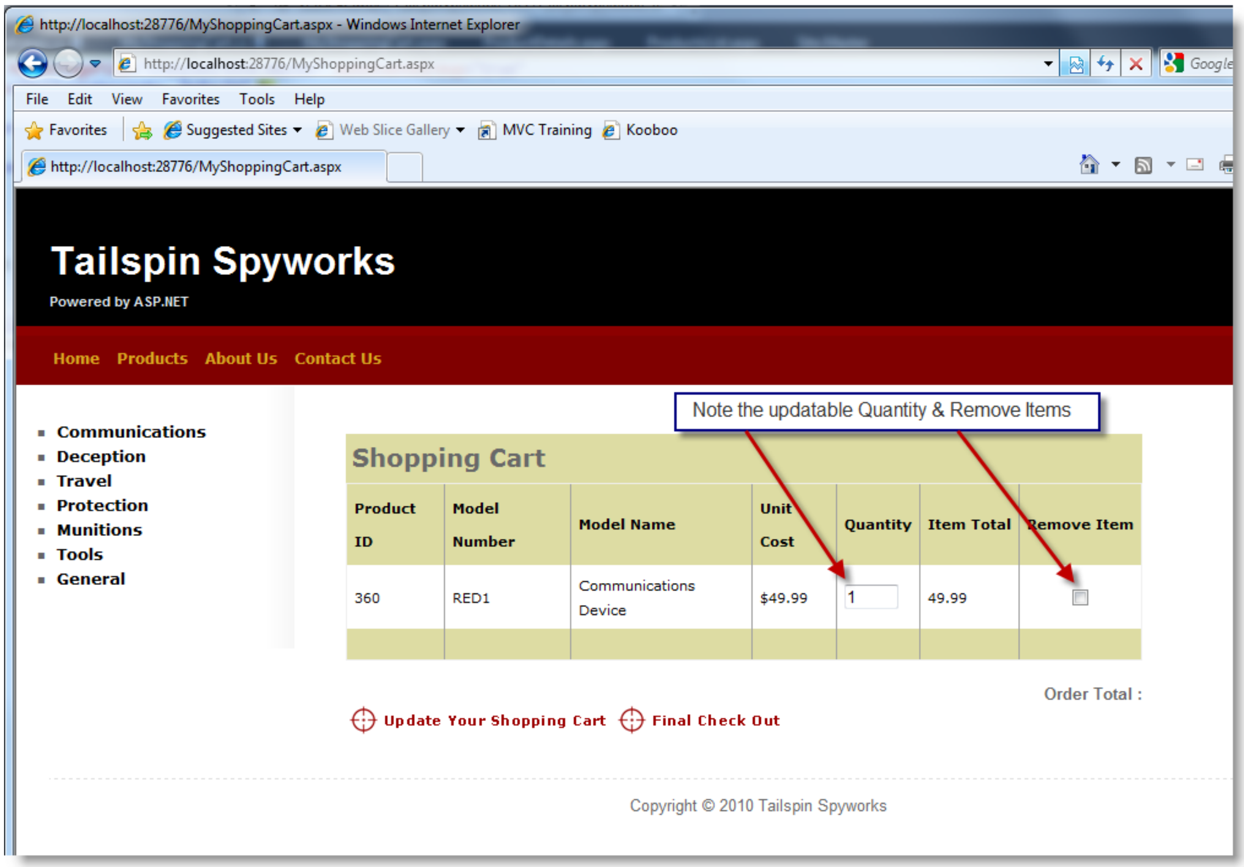
So, in the MyShoppingCart.aspx page we'll add an EntityDataSource control and a GridView control as follows.

```
<div id="ShoppingCartTitle" runat="server" class="ContentHead">Shopping Cart</div>
<asp:GridView ID="MyList" runat="server" AutoGenerateColumns="False" ShowFooter="True"
                          GridLines="Vertical" CellPadding="4"
                          DataSourceID="EDS_Cart"
                          DataKeyNames="ProductID,UnitCost,Quantity"
                          CssClass="CartListItem">
  <AlternatingRowStyle CssClass="CartListItemAlt" />
  <Columns>
    <asp:BoundField DataField="ProductID" HeaderText="Product ID" ReadOnly="True"
                                          SortExpression="ProductID"  />
    <asp:BoundField DataField="ModelNumber" HeaderText="Model Number"
                                            SortExpression="ModelNumber" />
    <asp:BoundField DataField="ModelName" HeaderText="Model Name"
                                          SortExpression="ModelName"  />
    <asp:BoundField DataField="UnitCost" HeaderText="Unit Cost" ReadOnly="True"
                                         SortExpression="UnitCost"
                                         DataFormatString="{0:c}" />
    <asp:TemplateField>
      <HeaderTemplate>Quantity</HeaderTemplate>
      <ItemTemplate>
         <asp:TextBox ID="PurchaseQuantity" Width="40" runat="server"
                      Text='<%# Bind("Quantity") %>'></asp:TextBox>
      </ItemTemplate>
    </asp:TemplateField>
    <asp:TemplateField>
      <HeaderTemplate>Item&nbsp;Total</HeaderTemplate>
      <ItemTemplate>
        <%# (Convert.ToDouble(Eval("Quantity")) *
             Convert.ToDouble(Eval("UnitCost")))%>
      </ItemTemplate>
    </asp:TemplateField>
    <asp:TemplateField>
    <HeaderTemplate>Remove&nbsp;Item</HeaderTemplate>
      <ItemTemplate>
        <center>
          <asp:CheckBox id="Remove" runat="server" />
        </center>
      </ItemTemplate>
    </asp:TemplateField>
  </Columns>
  <FooterStyle CssClass="CartListFooter"/>
  <HeaderStyle  CssClass="CartListHead" />
</asp:GridView>

<div style="float: right">
  <strong>
    <asp:Label ID="LabelTotalText" runat="server" Text="Order Total : ">
    </asp:Label>
    <asp:Label CssClass="NormalBold" id="lblTotal" runat="server"
                                                  EnableViewState="false">
    </asp:Label>
  </strong>
</div>
<br />
<asp:imagebutton id="UpdateBtn" runat="server" ImageURL="Styles/Images/update_cart.gif"
                                onclick="UpdateBtn_Click"></asp:imagebutton>
<asp:imagebutton id="CheckoutBtn" runat="server"
                                  ImageURL="Styles/Images/final_checkout.gif"
                                  PostBackUrl="~/CheckOut.aspx">
</asp:imagebutton>
<asp:EntityDataSource ID="EDS_Cart" runat="server"
                      ConnectionString="name=CommerceEntities"
                      DefaultContainerName="CommerceEntities" EnableFlattening="False"
                      EnableUpdate="True" EntitySetName="ViewCarts"
                      AutoGenerateWhereClause="True" EntityTypeFilter="" Select=""
                      Where="">
  <WhereParameters>
    <asp:SessionParameter Name="CartID" DefaultValue="0"
                                        SessionField="TailSpinSpyWorks_CartID" />
  </WhereParameters>
</asp:EntityDataSource>
```

Call up the form in the designer so that you can double click on the Update Cart button and generate the click event handler that is specified in the declaration in the markup.

We'll implement the details later but doing this will let us build and run our application without errors.

When you run the application and add an item to the shopping cart you will see this.



Note the updatable Quantity & Remove Items

Product ID	Model Number	Model Name	Unit Cost	Quantity	Item Total	Remove Item
360	RED1	Communications Device	$49.99	1	49.99	☐

Order Total :

Note that we have deviated from the “default” grid display by implementing three custom columns.

The first is an Editable, “Bound” field for the Quantity:

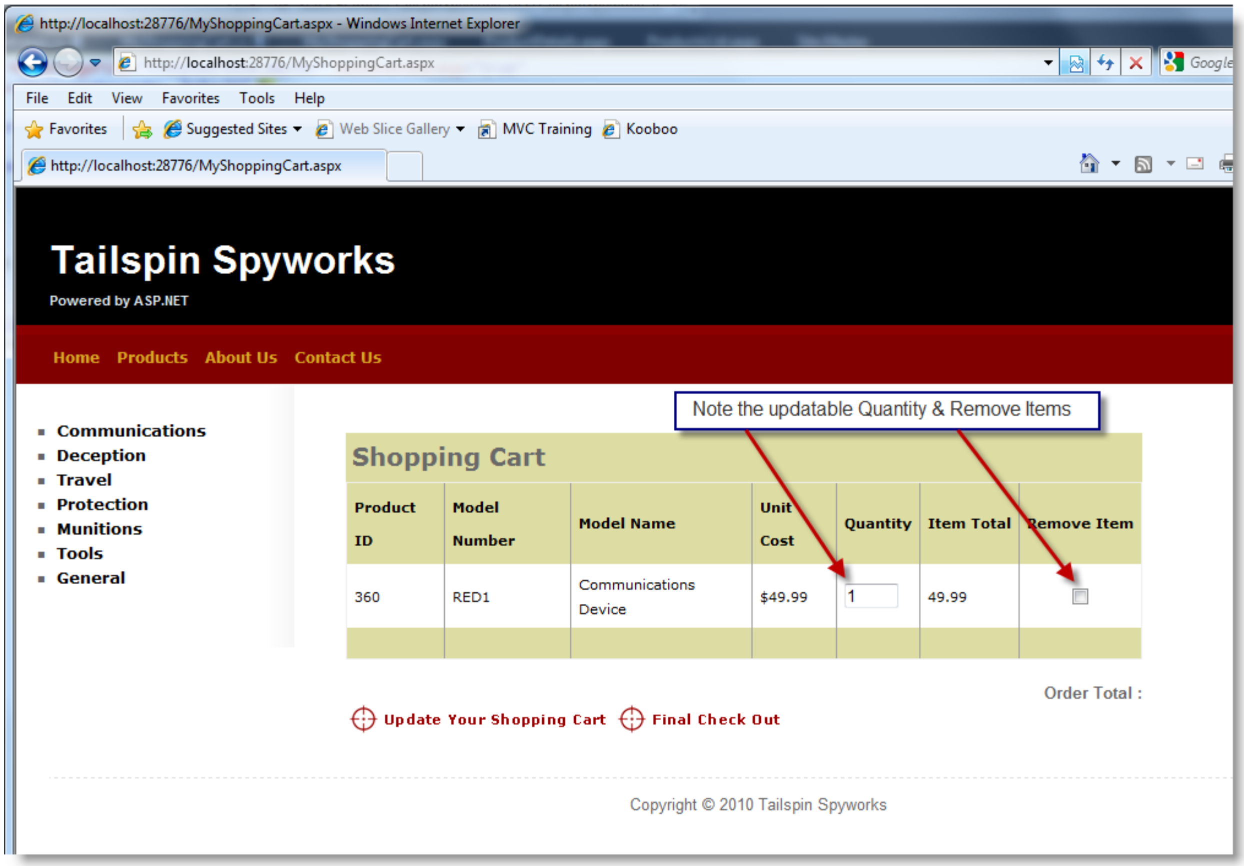
```
<asp:TemplateField>
  <HeaderTemplate>Quantity</HeaderTemplate>
  <ItemTemplate>
     <asp:TextBox ID="PurchaseQuantity" Width="40" runat="server"
                  Text='<%# Bind("Quantity") %>'></asp:TextBox>
  </ItemTemplate>
</asp:TemplateField>
```

The next is a “calculated” column that displays the line item total (the item cost times the quantity to be ordered):

```
 <asp:TemplateField>
  <HeaderTemplate>Item&nbsp;Total</HeaderTemplate>
  <ItemTemplate>
    <%# (Convert.ToDouble(Eval("Quantity")) *
         Convert.ToDouble(Eval("UnitCost")))%>
  </ItemTemplate>
</asp:TemplateField>
```

Lastly we have a custom column that contains a CheckBox control that the user will use to indicate that the item should be removed from the shopping chart.

```
    <asp:TemplateField>
    <HeaderTemplate>Remove&nbsp;Item</HeaderTemplate>
      <ItemTemplate>
        <center>
          <asp:CheckBox id="Remove" runat="server" />
        </center>
      </ItemTemplate>
    </asp:TemplateField>
```


http://localhost:28776/MyShoppingCart.aspx - Windows Internet Explorer
File Edit View Favorites Tools Help
Favorites Suggested Sites Web Slice Gallery MVC Training Kooboo
Tailspin Spyworks
Powered by ASP.NET
Home Products About Us Contact Us
Communications
Deception
Travel
Protection
Munitions
Tools
General
Note the updatable Quantity & Remove Items
Update Your Shopping Cart
Final Check Out
Copyright © 2010 Tailspin Spyworks

Shopping Cart

Product ID	Model Number	Model Name	Unit Cost	Quantity	Item Total	Remove Item
360	RED1	Communications Device	$49.99	1	49.99	☐

Order Total :

As you can see, the Order Total line is empty so let's add some logic to calculate the Order Total.

We'll first implement a "GetTotal" method to our MyShoppingCart Class.

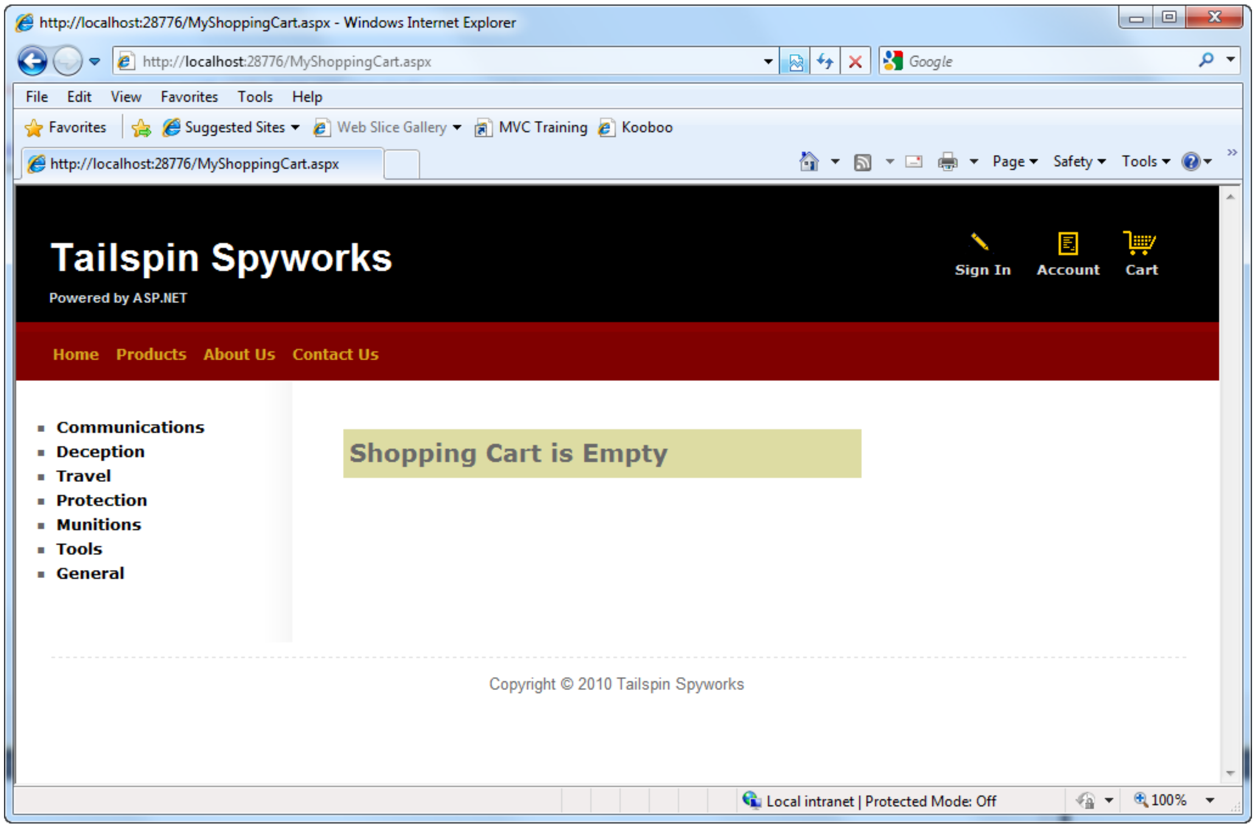
In the MyShoppingCart.cs file add the following code.

```
//------------------------------------------------------------------------------------+
public decimal GetTotal(string cartID)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    decimal cartTotal = 0;
    try
```

```
      {
      var myCart = (from c in db.ViewCarts where c.CartID == cartID select c);
      if (myCart.Count() > 0)
         {
         cartTotal = myCart.Sum(od => (decimal)od.Quantity * (decimal)od.UnitCost);
         }
       }
     catch (Exception exp)
       {
       throw new Exception("ERROR: Unable to Calculate Order Total - " +
                                                  exp.Message.ToString(), exp);
       }
     return (cartTotal);
     }
   }
```

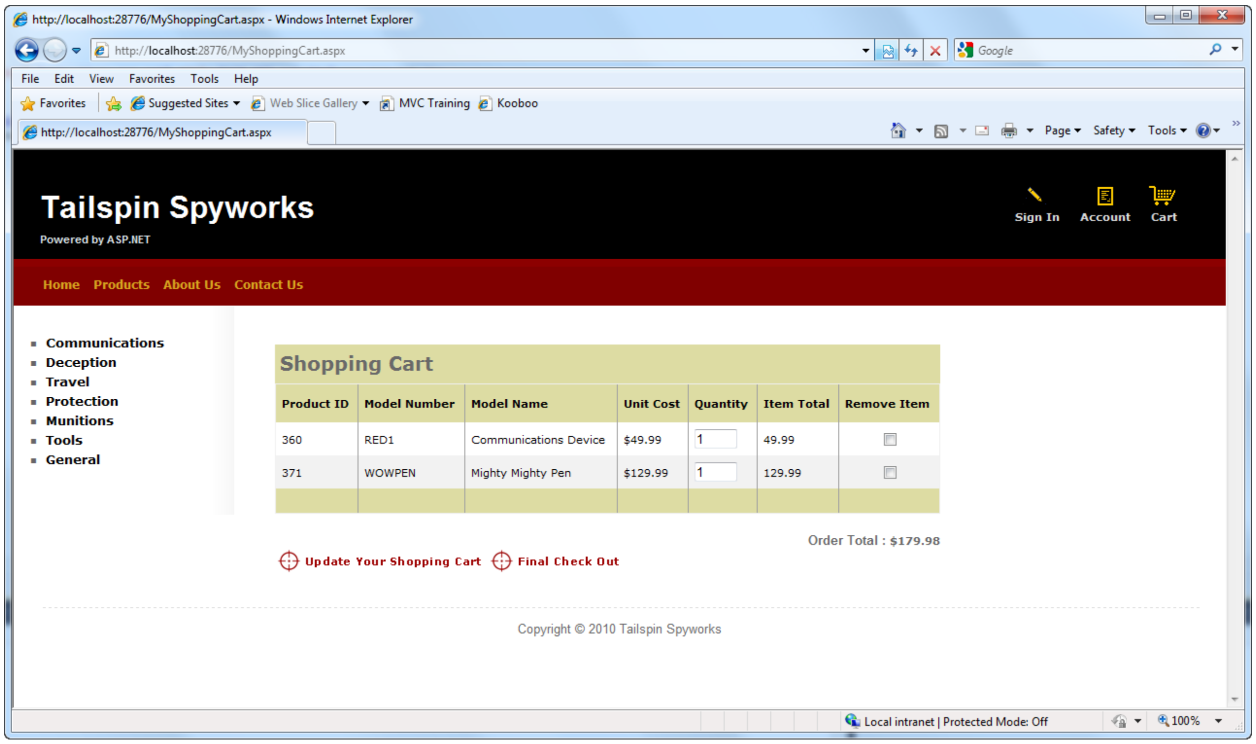
Then in the Page_Load event handler we'll can call our GetTotal method. At the same time we'll add a test to see if the shopping cart is empty and adjust the display accordingly if it is.

Now if the shopping cart is empty we get this:


http://localhost:28776/MyShoppingCart.aspx - Windows Internet Explorer
File Edit View Favorites Tools Help
Tailspin Spyworks
Powered by ASP.NET
Sign In
Account
Cart
Home
Products
About Us
Contact Us
Communications
Deception
Travel
Protection
Munitions
Tools
General
Copyright © 2010 Tailspin Spyworks
Local intranet | Protected Mode: Off

Shopping Cart is Empty

And if not, we see our total.



Shopping Cart

Product ID	Model Number	Model Name	Unit Cost	Quantity	Item Total	Remove Item
360	RED1	Communications Device	$49.99	1	49.99	☐
371	WOWPEN	Mighty Mighty Pen	$129.99	1	129.99	☐

Order Total : $179.98

Update Your Shopping Cart   Final Check Out

Copyright © 2010 Tailspin Spyworks

However, this page is not yet complete.

We will need additional logic to recalculate the shopping cart by removing items marked for removal and by determining new quantity values as some may have been changed in the grid by the user.

Lets add a "RemoveItem" method to our shopping cart class in MyShoppingCart.cs to handle the case when a user marks an item for removal.

```
//------------------------------------------------------------------------------------+
public void RemoveItem(string cartID, int  productID)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
      var myItem = (from c in db.ShoppingCarts where c.CartID == cartID &&
                         c.ProductID == productID select c).FirstOrDefault();
      if (myItem != null)
         {
         db.DeleteObject(myItem);
         db.SaveChanges();
         }
      }
    catch (Exception exp)
      {
      throw new Exception("ERROR: Unable to Remove Cart Item - " +
                                  exp.Message.ToString(), exp);
      }
    }
}
```

Now let's add a method to handle the circumstance when a user simply changes the quantity to be ordered in the GridView.

```
//--------------------------------------------------------------------------------------+
public void UpdateItem(string cartID, int productID, int quantity)
{
   using (CommerceEntities db = new CommerceEntities())
      {
      try
        {
        var myItem = (from c in db.ShoppingCarts where c.CartID == cartID &&
                                c.ProductID == productID select c).FirstOrDefault();
        if (myItem != null)
           {
           myItem.Quantity = quantity;
           db.SaveChanges();
           }
        }
     catch (Exception exp)
        {
        throw new Exception("ERROR: Unable to Update Cart Item - " +
                                                        exp.Message.ToString(), exp);
        }
      }
}
```

With the basic Remove and Update features in place we can implement the logic that actually updates the shopping cart in the database. (In MyShoppingCart.cs)

```
 //--------------------------------------------------------------------------------------+
public void UpdateShoppingCartDatabase(String cartId,
                                       ShoppingCartUpdates[] CartItemUpdates)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
      int CartItemCOunt = CartItemUpdates.Count();
      var myCart = (from c in db.ViewCarts where c.CartID == cartId select c);
      foreach (var cartItem in myCart)
        {
        // Iterate through all rows within shopping cart list
        for (int i = 0; i < CartItemCOunt; i++)
          {
          if (cartItem.ProductID == CartItemUpdates[i].ProductId)
             {
             if (CartItemUpdates[i].PurchaseQantity < 1 ||
                 CartItemUpdates[i].RemoveItem == true)
                {
```

```
                RemoveItem(cartId, cartItem.ProductID);
                }
              else
                {
                UpdateItem(cartId, cartItem.ProductID,
                                   CartItemUpdates[i].PurchaseQantity);
                }
              }
            }
          }
        }
      catch (Exception exp)
        {
        throw new Exception("ERROR: Unable to Update Cart Database - " +
                             exp.Message.ToString(), exp);
        }
    }
}
```

You'll note that this method expects two parameters. One is the shopping cart Id and the other is an array of objects of user defined type.

So as to minimize the dependency of our logic on user interface specifics, we've defined a data structure that we can use to pass the shopping cart items to our code without our method needing to directly access the GridView control.

```
public struct ShoppingCartUpdates
{
   public int ProductId;
   public int PurchaseQantity;
   public bool RemoveItem;
}
```

In our MyShoppingCart.aspx.cs file we can use this structure in our Update Button Click Event handler as follows. Note that in addition to updating the cart we will update the cart total as well.

```
//------------------------------------------------------------------------------------+
protected void UpdateBtn_Click(object sender, ImageClickEventArgs e)
{
  MyShoppingCart usersShoppingCart = new MyShoppingCart();
  String cartId = usersShoppingCart.GetShoppingCartId();

  ShoppingCartUpdates[] cartUpdates = new ShoppingCartUpdates[MyList.Rows.Count];
  for (int i = 0; i < MyList.Rows.Count; i++)
    {
    IOrderedDictionary rowValues = new OrderedDictionary();
    rowValues = GetValues(MyList.Rows[i]);
    cartUpdates[i].ProductId =  Convert.ToInt32(rowValues["ProductID"]);
    cartUpdates[i].PurchaseQantity = Convert.ToInt32(rowValues["Quantity"]);

    CheckBox cbRemove = new CheckBox();
    cbRemove = (CheckBox)MyList.Rows[i].FindControl("Remove");
    cartUpdates[i].RemoveItem = cbRemove.Checked;
    }

   usersShoppingCart.UpdateShoppingCartDatabase(cartId, cartUpdates);
   MyList.DataBind();
   lblTotal.Text = String.Format("{0:c}", usersShoppingCart.GetTotal(cartId));
}
```

Note with particular interest this line of code:

```
rowValues = GetValues(MyList.Rows[i]);
```

GetValues() is a special helper function that we will implement in MyShoppingCart.aspx.cs as follows.

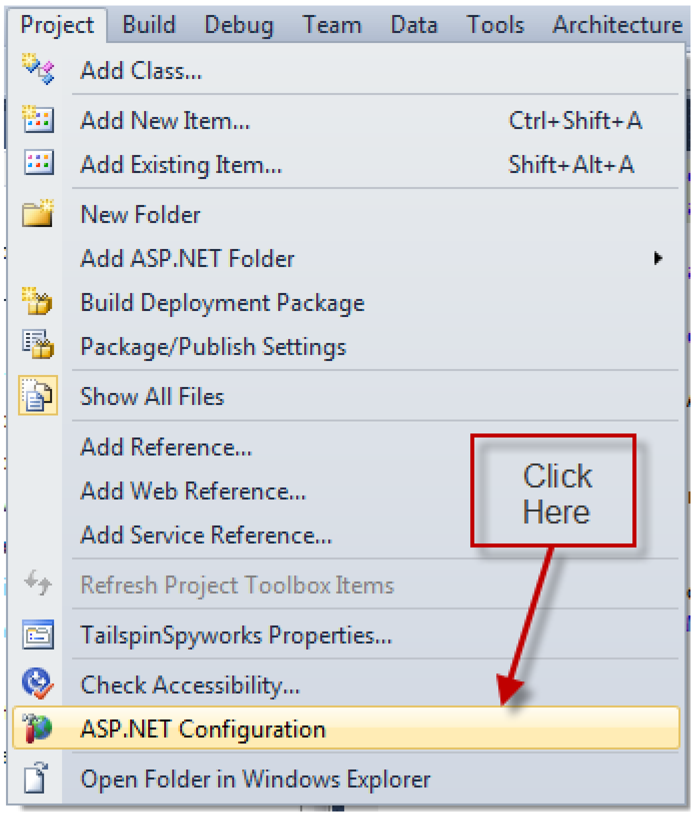
```
//------------------------------------------------------------------------------------+
public static IOrderedDictionary GetValues(GridViewRow row)
{
  IOrderedDictionary values = new OrderedDictionary();
  foreach (DataControlFieldCell cell in row.Cells)
    {
    if (cell.Visible)
      {
      // Extract values from the cell
      cell.ContainingField.ExtractValuesFromCell(values, cell, row.RowState, true);
      }
    }
    return values;
}
```

This provides a clean way to access the values of the bound elements in our GridView control. Since our “Remove Item” CheckBox Control is not bound we’ll access it via the FindControl() method.

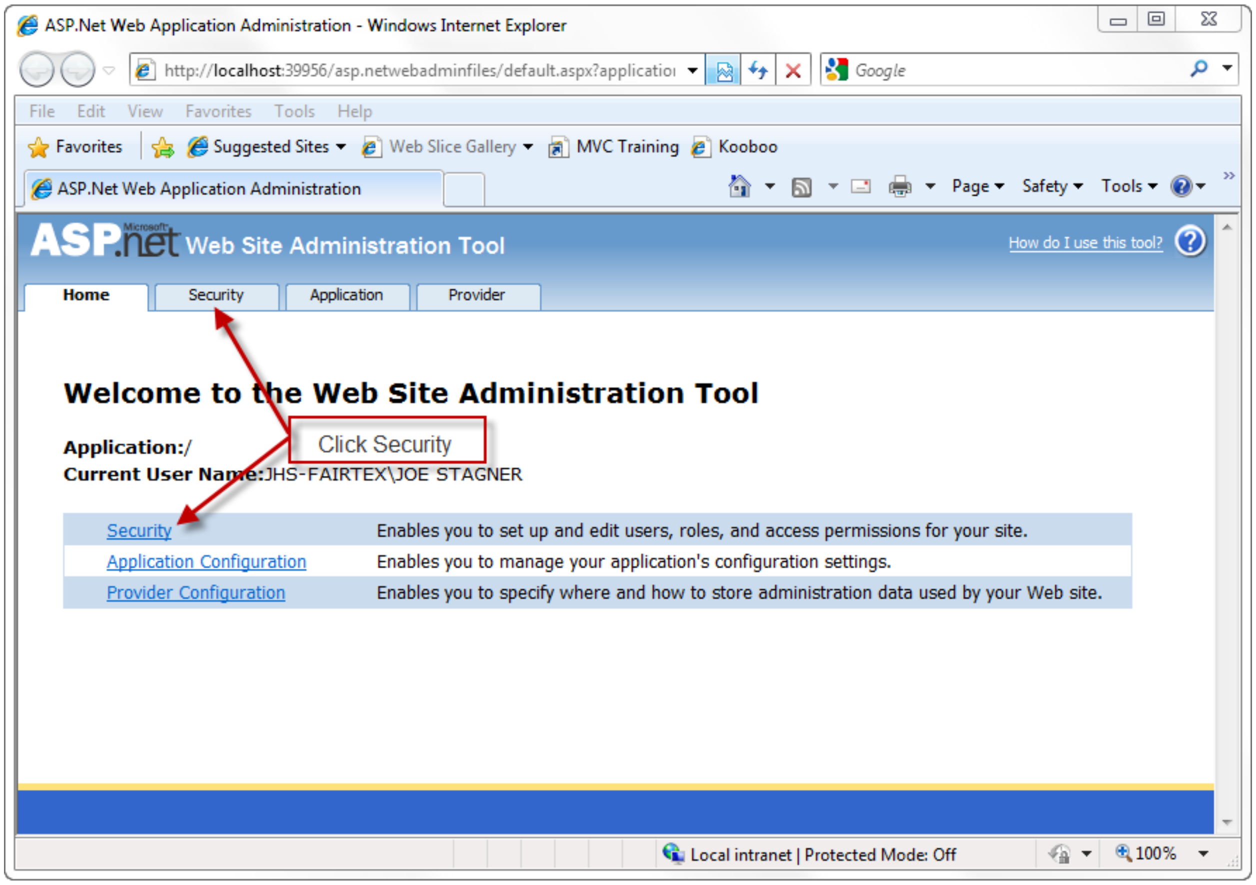
At this stage in your project’s development we are getting ready to implement the checkout process.

Before doing so let’s use Visual Studio to generate the membership database and add a user to the membership repository.

Working with ASP.NET Membership



Click Security

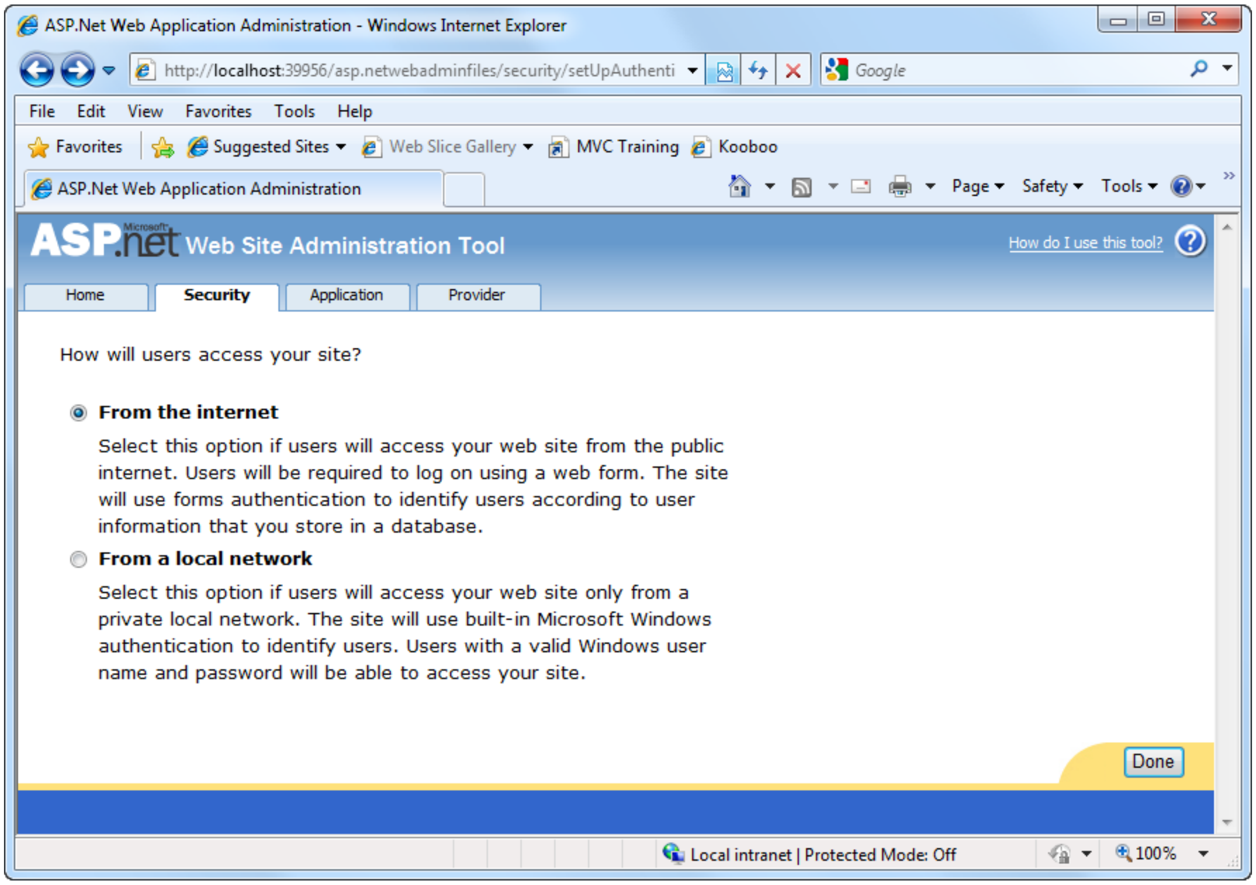

ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:39956/asp.netwebadminfiles/default.aspx?applicatio
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Web Slice Gallery
MVC Training
Kooboo
ASP.Net Web Application Administration
Page
Safety
Tools
Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Click Security
Local intranet | Protected Mode: Off
100%

Welcome to the Web Site Administration Tool

Application:/
Current User Name:JHS-FAIRTEX\JOE STAGNER

Security	Enables you to set up and edit users, roles, and access permissions for your site.
Application Configuration	Enables you to manage your application's configuration settings.
Provider Configuration	Enables you to specify where and how to store administration data used by your Web site.

Make sure that we are using forms authentication.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:39956/asp.netwebadminfiles/security/setUpAuthenti
File Edit View Favorites Tools Help
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Done
Local intranet | Protected Mode: Off
100%

How will users access your site?

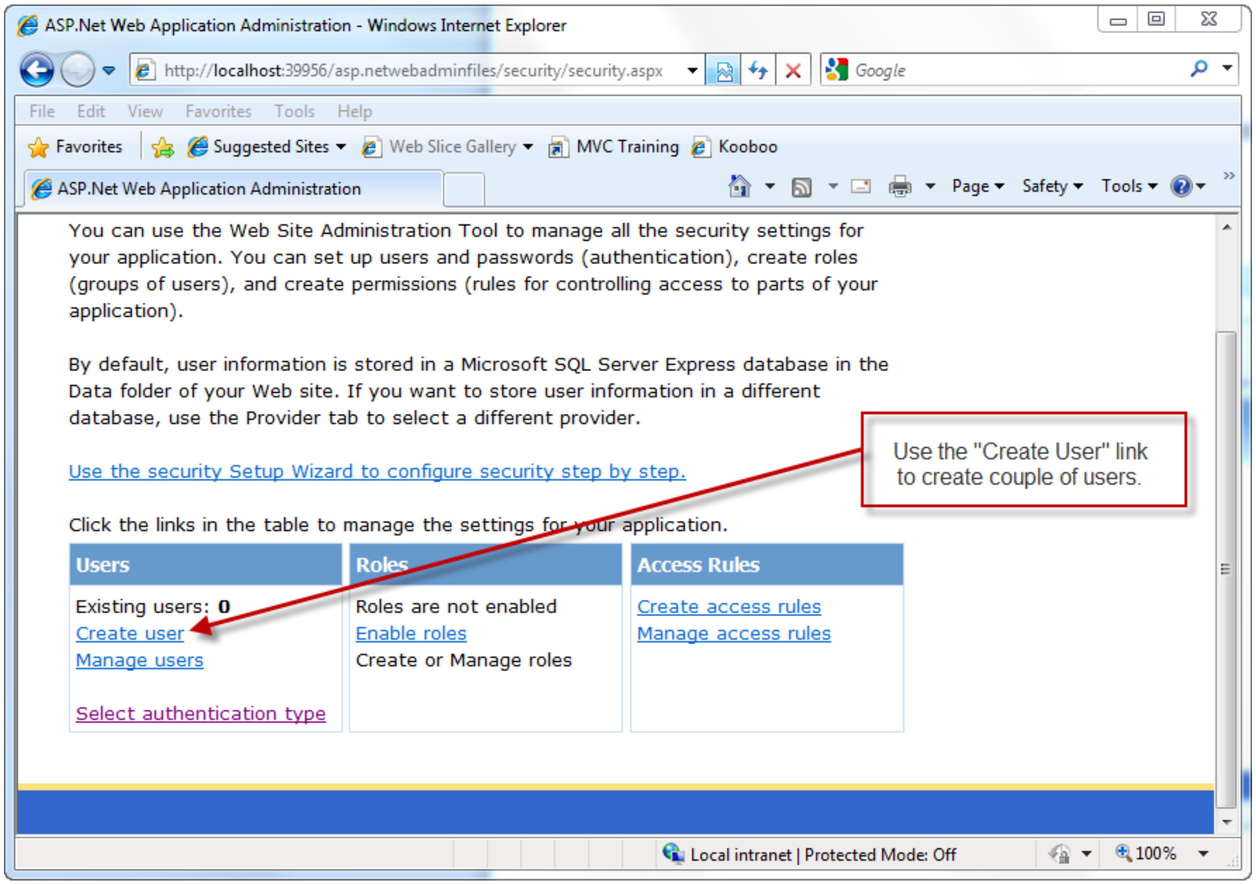
[✓ From the internet]

Select this option if users will access your web site from the public internet. Users will be required to log on using a web form. The site will use forms authentication to identify users according to user information that you store in a database.

From a local network

Select this option if users will access your web site only from a private local network. The site will use built-in Microsoft Windows authentication to identify users. Users with a valid Windows user name and password will be able to access your site.

Use the "Create User" link to create a couple of users.


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:39956/asp.netwebadminfiles/security/security.aspx
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Web Slice Gallery
MVC Training
Kooboo
ASP.Net Web Application Administration
Page
Safety
Tools
Local intranet | Protected Mode: Off
100%

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

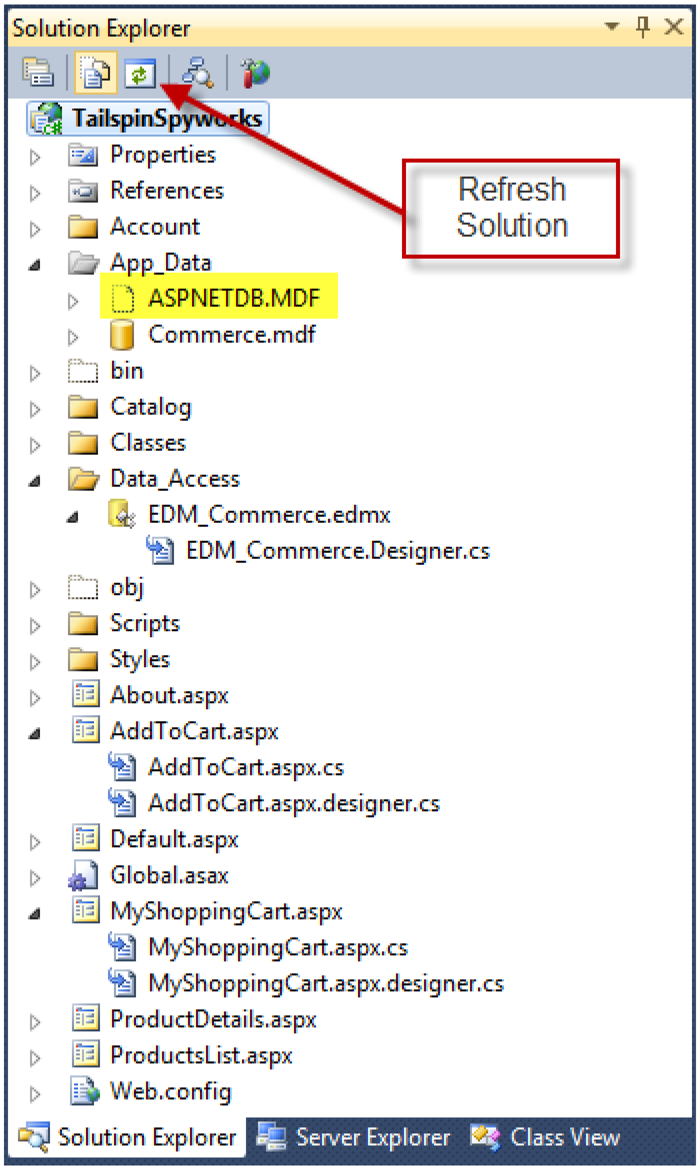
Use the security Setup Wizard to configure security step by step.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: **0**	Roles are not enabled	Create access rules
Create user	Enable roles	Manage access rules
Manage users	Create or Manage roles	
Select authentication type		

Use the "Create User" link to create couple of users.

When done, refer to the Solution Explorer window and refresh the view.



TailspinSpyworks
Properties
References
Account
App_Data
ASPNETDB.MDF
Commerce.mdf
bin
Catalog
Classes
Data_Access
EDM_Commerce.edmx
EDM_Commerce.Designer.cs
obj
Scripts
Styles
About.aspx
AddToCart.aspx
AddToCart.aspx.cs
AddToCart.aspx.designer.cs
Default.aspx
Global.asax
MyShoppingCart.aspx
MyShoppingCart.aspx.cs
MyShoppingCart.aspx.designer.cs
ProductDetails.aspx
ProductsList.aspx
Web.config

Note that the ASPNETDB.MDF fine has been created. This file contains the tables to support the core ASP.NET services like membership.

Now we can begin implementing the checkout process.

Begin by creating a CheckOut.aspx page.

The CheckOut.aspx page should only be available to users who are logged in so we will restrict access to logged in users and redirect users who are not logged in to the LogIn page.

To do this we'll add the following to the configuration section of our web.config file.

```
<location path="Checkout.aspx">
  <system.web>
```

```
      <authorization>
        <deny users="?" />
      </authorization>
    </system.web>
  </location>
```

The template for ASP.NET Web Forms applications automatically added an authentication section to our web.config file and established the default login page.

```
    <authentication mode="Forms">
      <forms loginUrl="~/Account/Login.aspx" timeout="2880" />
    </authentication>
```

We must modify the Login.aspx code behind file to migrate an anonymous shopping cart when the user logs in. Change the Page_Load event as follows.

```
using System.Web.Security;

protected void Page_Load(object sender, EventArgs e)
{
  // If the user is not submitting their credentials
  // save refferer
  if (!Page.IsPostBack)
     {
     if (Page.Request.UrlReferrer != null)
        {
        Session["LoginReferrer"] = Page.Request.UrlReferrer.ToString();
        }
      }

  // User is logged in so log them out.
  if (User.Identity.IsAuthenticated)
     {
     FormsAuthentication.SignOut();
     Response.Redirect("~/");
     }
}
```

Then add a “LoggedIn” event handler like this to set the session name to the newly logged in user and change the temporary session id in the shopping cart to that of the user by calling the MigrateCart method in our MyShoppingCart class. (Implemented in the .cs file)

```
protected void LoginUser_LoggedIn(object sender, EventArgs e)
{
  MyShoppingCart usersShoppingCart = new MyShoppingCart();
  String cartId = usersShoppingCart.GetShoppingCartId();
  usersShoppingCart.MigrateCart(cartId, LoginUser.UserName);

  if(Session["LoginReferrer"] != null)
```

```
    {
    Response.Redirect(Session["LoginReferrer"].ToString());
    }

  Session["UserName"] = LoginUser.UserName;
}
```

```
Implement the MigrateCart() method like this.
```

```
//--------------------------------------------------------------------------------------+
public void MigrateCart(String oldCartId, String UserName)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
      var myShoppingCart = from cart in db.ShoppingCarts
                           where cart.CartID == oldCartId
                           select cart;

      foreach (ShoppingCart item in myShoppingCart)
        {
        item.CartID = UserName;
        }
      db.SaveChanges();
      Session[CartId] = UserName;
      }
    catch (Exception exp)
      {
      throw new Exception("ERROR: Unable to Migrate Shopping Cart - " +
                           exp.Message.ToString(), exp);
      }
    }
}
```

In checkout.aspx we'll use an EntityDataSource and a GridView in our check out page much as we did in our shopping cart page.

```
<div id="CheckOutHeader" runat="server" class="ContentHead">
  Review and Submit Your Order
</div>
<span id="Message" runat="server"><br />
   <asp:Label ID="LabelCartHeader" runat="server"
              Text="Please check all the information below to be sure it&#39;s correct.">
   </asp:Label>
</span><br />
<asp:GridView ID="MyList" runat="server" AutoGenerateColumns="False"
              DataKeyNames="ProductID,UnitCost,Quantity"
              DataSourceID="EDS_Cart"
              CellPadding="4" GridLines="Vertical" CssClass="CartListItem"
              onrowdatabound="MyList_RowDataBound" ShowFooter="True">
  <AlternatingRowStyle CssClass="CartListItemAlt" />
  <Columns>
    <asp:BoundField DataField="ProductID" HeaderText="Product ID" ReadOnly="True"
                    SortExpression="ProductID"  />
    <asp:BoundField DataField="ModelNumber" HeaderText="Model Number"
                    SortExpression="ModelNumber" />
    <asp:BoundField DataField="ModelName" HeaderText="Model Name"
                    SortExpression="ModelName" />
    <asp:BoundField DataField="UnitCost" HeaderText="Unit Cost" ReadOnly="True"
                    SortExpression="UnitCost" DataFormatString="{0:c}" />
    <asp:BoundField DataField="Quantity" HeaderText="Quantity" ReadOnly="True"
                    SortExpression="Quantity" />
    <asp:TemplateField>
      <HeaderTemplate>Item&nbsp;Total</HeaderTemplate>
      <ItemTemplate>
        <%# (Convert.ToDouble(Eval("Quantity")) * Convert.ToDouble(Eval("UnitCost")))%>
      </ItemTemplate>
    </asp:TemplateField>
  </Columns>
  <FooterStyle CssClass="CartListFooter"/>
  <HeaderStyle  CssClass="CartListHead" />
</asp:GridView>

<br />
<asp:imagebutton id="CheckoutBtn" runat="server" ImageURL="Styles/Images/submit.gif"
                                  onclick="CheckoutBtn_Click">
</asp:imagebutton>
<asp:EntityDataSource ID="EDS_Cart" runat="server"
                      ConnectionString="name=CommerceEntities"
                      DefaultContainerName="CommerceEntities"
                      EnableFlattening="False"
                      EnableUpdate="True"
                      EntitySetName="ViewCarts"
                      AutoGenerateWhereClause="True"
                      EntityTypeFilter=""
                      Select="" Where="">
   <WhereParameters>
      <asp:SessionParameter Name="CartID" DefaultValue="0"
                                          SessionField="TailSpinSpyWorks_CartID" />
   </WhereParameters>
</asp:EntityDataSource>
```

Note that our GridView control specifies an "ondatabound" event handler named MyList_RowDataBound so let's implement that event handler like this.

```
decimal _CartTotal = 0;

//--------------------------------------------------------------------------------------+
protected void MyList_RowDataBound(object sender, GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
     {
     TailspinSpyworks.Data_Access.ViewCart myCart = new Data_Access.ViewCart();
     myCart = (TailspinSpyworks.Data_Access.ViewCart)e.Row.DataItem;
     _CartTotal += myCart.UnitCost * myCart.Quantity;
     }
   else if (e.Row.RowType == DataControlRowType.Footer)
     {
     if (_CartTotal > 0)
        {
```

```
        CheckOutHeader.InnerText = "Review and Submit Your Order";
        LabelCartHeader.Text = "Please check all the information below to be sure
                                                    it&#39;s correct.";
        CheckoutBtn.Visible = true;
        e.Row.Cells[5].Text = "Total: " + _CartTotal.ToString("C");
        }
     }
}
```

This method keeps a running total of the shopping cart as each row is bound and updates the bottom row of the GridView.

At this stage we have implemented a “review” presentation of the order to be placed.

Let’s handle an empty cart scenario by adding a few lines of code to our Page_Load event:

```
protected void Page_Load(object sender, EventArgs e)
{
   CheckOutHeader.InnerText = "Your Shopping Cart is Empty";
   LabelCartHeader.Text = "";
   CheckoutBtn.Visible = false;
}
```

When the user clicks on the “Submit” button we will execute the following code in the Submit Button Click Event handler.

```
protected void CheckoutBtn_Click(object sender, ImageClickEventArgs e)
{
  MyShoppingCart usersShoppingCart = new MyShoppingCart();
  if (usersShoppingCart.SubmitOrder(User.Identity.Name) == true)
    {
    CheckOutHeader.InnerText = "Thank You - Your Order is Complete.";
    Message.Visible = false;
    CheckoutBtn.Visible = false;
    }
  else
    {
    CheckOutHeader.InnerText = "Order Submission Failed - Please try again. ";
    }
}
```

The “meat” of the order submission process is to be implemented in the SubmitOrder() method of our MyShoppingCart class.

SubmitOrder will:

Take all the line items in the shopping cart and use them to create a new Order Record and the associated OrderDetails records.

Calculate Shipping Date.

Clear the shopping cart.

```
//------------------------------------------------------------------------------+

public bool SubmitOrder(string UserName)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
      //------------------------------------------------------------------------+
      //  Add New Order Record                                                  |
      //------------------------------------------------------------------------+
      Order newOrder = new Order();
      newOrder.CustomerName = UserName;
      newOrder.OrderDate = DateTime.Now;
      newOrder.ShipDate = CalculateShipDate();
      db.Orders.AddObject(newOrder);
      db.SaveChanges();

      //------------------------------------------------------------------------+
      //  Create a new OderDetail Record for each item in the Shopping Cart     |
      //------------------------------------------------------------------------+
      String cartId = GetShoppingCartId();
      var myCart = (from c in db.ViewCarts where c.CartID == cartId select c);
      foreach (ViewCart item in myCart)
        {
        int i = 0;
        if (i < 1)
          {
          OrderDetail od = new OrderDetail();
          od.OrderID = newOrder.OrderID;
          od.ProductID = item.ProductID;
          od.Quantity = item.Quantity;
          od.UnitCost = item.UnitCost;
          db.OrderDetails.AddObject(od);
          i++;
          }

        var myItem = (from c in db.ShoppingCarts where c.CartID == item.CartID &&
                                 c.ProductID == item.ProductID select c).FirstOrDefault();
        if (myItem != null)
          {
          db.DeleteObject(myItem);
          }
        }
      db.SaveChanges();
      }
    catch (Exception exp)
      {
      throw new Exception("ERROR: Unable to Submit Order - " + exp.Message.ToString(),
                                                                exp);
      }
    }
  return(true);
}
```

For the purposes of this sample application we'll calculate a ship date by simply adding two days to the current date.

```
//--------------------------------------------------------------------------------------+
DateTime CalculateShipDate()
{
   DateTime shipDate = DateTime.Now.AddDays(2);
   return (shipDate);
}
```

Running the application now will permit us to test the shopping process from start to finish.

Adding Features

Though users can browse our catalog, place items in their shopping cart, and complete the checkout process, there are a number of supporting features that we will include to improve our site.

1. Account Review (List orders placed and view details.)
2. Add some context specific content to the front page.
3. Add a feature to let users Review the products in the catalog.
4. Create a User Control to display Popular Items and Place that control on the front page.
5. Create an "Also Purchased" user control and add it to the product details page.
6. Add a Contact Page.
7. Add an About Page.
8. Global Error

Account Review

In the "Account" folder create two .aspx pages one named OrderList.aspx and the other named OrderDetails.aspx

OrderList.aspx will leverage the GridView and EntityDataSoure controls much as we have previously.

```
<div class="ContentHead">Order History</div><br />

<asp:GridView ID="GridView_OrderList" runat="server" AllowPaging="True"
              ForeColor="#333333" GridLines="None" CellPadding="4" Width="100%"
              AutoGenerateColumns="False" DataKeyNames="OrderID"
              DataSourceID="EDS_Orders" AllowSorting="True" ViewStateMode="Disabled" >
  <AlternatingRowStyle BackColor="White" />
  <Columns>
    <asp:BoundField DataField="OrderID" HeaderText="OrderID" ReadOnly="True"
                    SortExpression="OrderID" />
    <asp:BoundField DataField="CustomerName" HeaderText="Customer"
```

```
                SortExpression="CustomerName" />
    <asp:BoundField DataField="OrderDate" HeaderText="Order Date"
                SortExpression="OrderDate" />
    <asp:BoundField DataField="ShipDate" HeaderText="Ship Date"
                SortExpression="ShipDate" />
    <asp:HyperLinkField HeaderText="Show Details" Text="Show Details"
              DataNavigateUrlFields="OrderID"
              DataNavigateUrlFormatString="~/Account/OrderDetails.aspx?OrderID={0}" />
  </Columns>
  <FooterStyle BackColor="#990000" Font-Bold="True" ForeColor="White" />
  <HeaderStyle BackColor="#990000" Font-Bold="True" ForeColor="White" />
  <PagerStyle BackColor="#FFCC66" ForeColor="#333333" HorizontalAlign="Center" />
  <RowStyle BackColor="#FFFBD6" ForeColor="#333333" />
  <SelectedRowStyle BackColor="#FFCC66" Font-Bold="True" ForeColor="Navy" />
  <SortedAscendingCellStyle BackColor="#FDF5AC" />
  <SortedAscendingHeaderStyle BackColor="#4D0000" />
  <SortedDescendingCellStyle BackColor="#FCF6C0" />
  <SortedDescendingHeaderStyle BackColor="#820000" />
  <SortedAscendingCellStyle BackColor="#FDF5AC"></SortedAscendingCellStyle>
  <SortedAscendingHeaderStyle BackColor="#4D0000"></SortedAscendingHeaderStyle>
  <SortedDescendingCellStyle BackColor="#FCF6C0"></SortedDescendingCellStyle>
  <SortedDescendingHeaderStyle BackColor="#820000"></SortedDescendingHeaderStyle>
</asp:GridView>

<asp:EntityDataSource ID="EDS_Orders" runat="server" EnableFlattening="False"
                      AutoGenerateWhereClause="True"
                      Where=""
                      OrderBy="it.OrderDate DESC"
                      ConnectionString="name=CommerceEntities"
                      DefaultContainerName="CommerceEntities"
                      EntitySetName="Orders" >
   <WhereParameters>
      <asp:SessionParameter Name="CustomerName" SessionField="UserName" />
   </WhereParameters>
</asp:EntityDataSource>
```

The EntityDataSoure selects records from the Orders table filtered on the UserName (see the WhereParameter) which we set in a session variable when the user log's in.

Note also these parameters in the HyperlinkField of the GridView:

```
DataNavigateUrlFields="OrderID"
DataNavigateUrlFormatString="~/Account/OrderDetails.aspx?OrderID={0}"
```

These specify the link to the Order details view for each product specifying the OrderID field as a QueryString parameter to the OrderDetails.aspx page.

OrderDetsails.aspx

We will use an EntityDataSource control to access the Orders and a FormView to display the Order data and another EntityDataSource with a GridView to display all the Order's line items.

```
<asp:FormView ID="FormView1" runat="server" CellPadding="4"
                             DataKeyNames="OrderID"
                             DataSourceID="EDS_Order" ForeColor="#333333" Width="250px">
   <FooterStyle BackColor="#990000" Font-Bold="True" ForeColor="White" />
   <HeaderStyle BackColor="#990000" Font-Bold="True" ForeColor="White" />
   <ItemTemplate>
      OrderID : <%# Eval("OrderID") %><br />
      CustomerName : <%# Eval("CustomerName") %><br />
      Order Date : <%# Eval("OrderDate") %><br />
      Ship Date : <%# Eval("ShipDate") %><br />
   </ItemTemplate>
   <PagerStyle BackColor="#FFCC66" ForeColor="#333333" HorizontalAlign="Center" />
   <RowStyle BackColor="#FFFBD6" ForeColor="#333333" />
</asp:FormView>
<asp:EntityDataSource ID="EDS_Order" runat="server"  EnableFlattening="False"
                      ConnectionString="name=CommerceEntities"
                      DefaultContainerName="CommerceEntities"
                      EntitySetName="Orders"
                      AutoGenerateWhereClause="True"
                      Where=""
                      EntityTypeFilter="" Select="">
   <WhereParameters>
      <asp:QueryStringParameter Name="OrderID" QueryStringField="OrderID" Type="Int32" />
   </WhereParameters>
</asp:EntityDataSource>

<asp:GridView ID="GridView_OrderDetails" runat="server"
              AutoGenerateColumns="False"
              DataKeyNames="ProductID,UnitCost,Quantity"
              DataSourceID="EDS_OrderDetails"
              CellPadding="4" GridLines="Vertical" CssClass="CartListItem"
              onrowdatabound="MyList_RowDataBound" ShowFooter="True"
              ViewStateMode="Disabled">
   <AlternatingRowStyle CssClass="CartListItemAlt" />
   <Columns>
     <asp:BoundField DataField="ProductID" HeaderText="Product ID" ReadOnly="True"
                     SortExpression="ProductID"  />
     <asp:BoundField DataField="ModelNumber" HeaderText="Model Number"
                     SortExpression="ModelNumber" />
     <asp:BoundField DataField="ModelName" HeaderText="Model Name"
                     SortExpression="ModelName" />
     <asp:BoundField DataField="UnitCost" HeaderText="Unit Cost" ReadOnly="True"
                     SortExpression="UnitCost" DataFormatString="{0:c}" />
     <asp:BoundField DataField="Quantity" HeaderText="Quantity" ReadOnly="True"
                     SortExpression="Quantity" />
     <asp:TemplateField>
       <HeaderTemplate>Item&nbsp;Total</HeaderTemplate>
       <ItemTemplate>
         <%# (Convert.ToDouble(Eval("Quantity")) *  Convert.ToDouble(Eval("UnitCost")))%>
       </ItemTemplate>
     </asp:TemplateField>
   </Columns>
   <FooterStyle CssClass="CartListFooter"/>
   <HeaderStyle  CssClass="CartListHead" />
 </asp:GridView>
 <asp:EntityDataSource ID="EDS_OrderDetails" runat="server"
                       ConnectionString="name=CommerceEntities"
                       DefaultContainerName="CommerceEntities"
                       EnableFlattening="False"
                       EntitySetName="VewOrderDetails"
                       AutoGenerateWhereClause="True"
                       Where="">
   <WhereParameters>
     <asp:QueryStringParameter Name="OrderID" QueryStringField="OrderID" Type="Int32" />
   </WhereParameters>
</asp:EntityDataSource>
```

In the Code Behind file (OrdrDetails.aspx.cs) we have two little bits of housekeeping.

First we need to make sure that OrderDetails always gets an OrderId.

```
protected void Page_Load(object sender, EventArgs e)
{
  if (String.IsNullOrEmpty(Request.QueryString["OrderId"]))
     {
     Response.Redirect("~/Account/OrderList.aspx");
     }
}
```

We also need to calculate and display the order total from the line items.

```
decimal _CartTotal = 0;
```

```
protected void MyList_RowDataBound(object sender, GridViewRowEventArgs e)
{
  if (e.Row.RowType == DataControlRowType.DataRow)
     {
     TailspinSpyworks.Data_Access.VewOrderDetail myCart = new
                                                   Data_Access.VewOrderDetail();
     myCart = (TailspinSpyworks.Data_Access.VewOrderDetail)e.Row.DataItem;
     _CartTotal += Convert.ToDecimal(myCart.UnitCost * myCart.Quantity);
     }
   else if (e.Row.RowType == DataControlRowType.Footer)
     {
     e.Row.Cells[5].Text = "Total: " + _CartTotal.ToString("C");
   }
}
```

The Home Page

Let's add some static content to the Default.aspx page.

First I'll create a "Content" folder and within it an Images folder (and I'll include an image to be used on the home page.)

Into the bottom placeholder of the Default.aspx page, add the following markup.

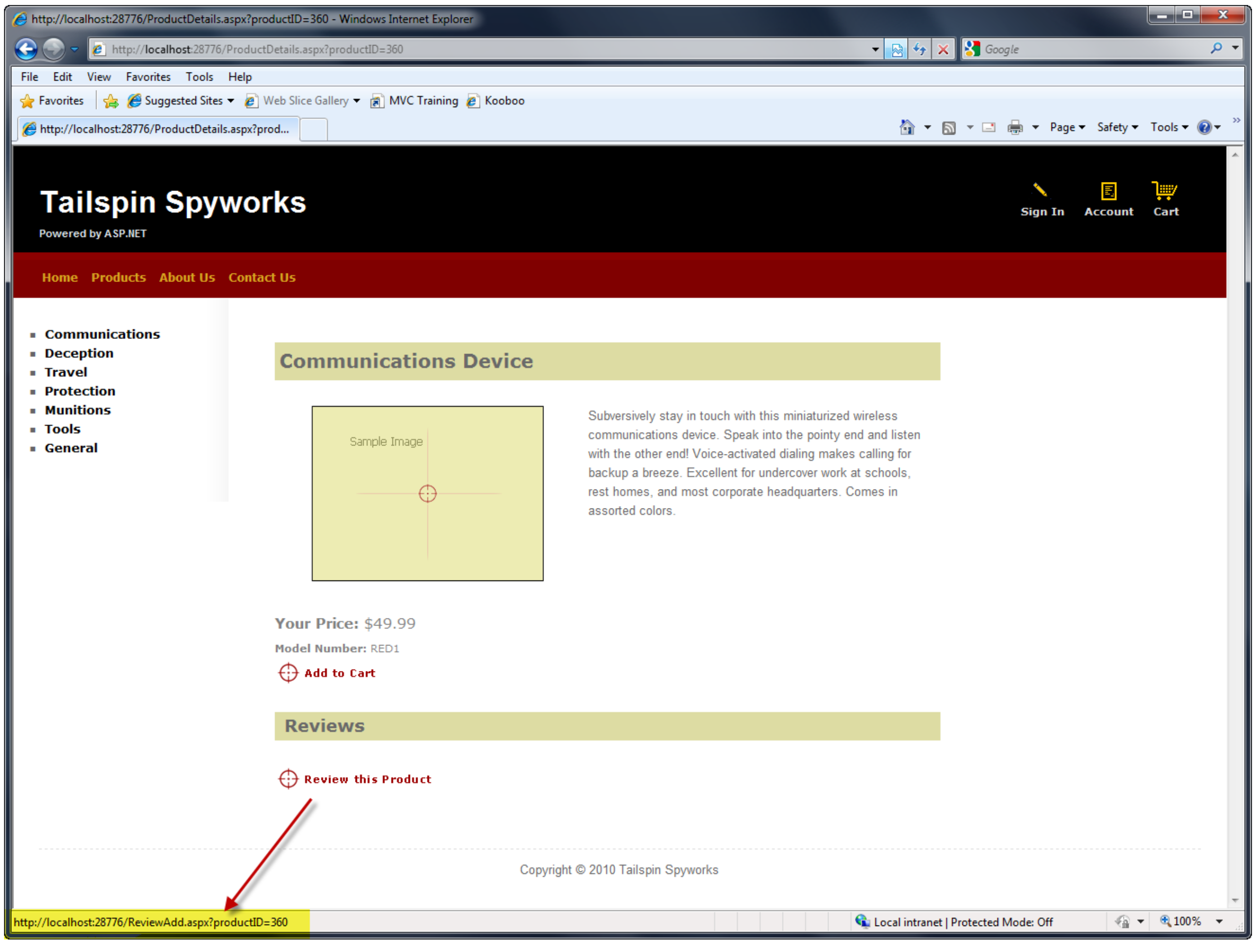
```
<h2>
  <asp:LoginView ID="LoginView_VisitorGreeting" runat="server">
    <AnonymousTemplate>
       Welcome to the Store !
    </AnonymousTemplate>
    <LoggedInTemplate>
      Hi <asp:LoginName ID="LoginName_Welcome" runat="server" />. Thanks for coming back.
    </LoggedInTemplate>
  </asp:LoginView>
</h2>

<p><strong>TailSpin Spyworks</strong> demonstrates how extraordinarily simple it is to
create powerful, scalable applications for the .NET platform. </p>
<table>
  <tr>
    <td style="width: 50%;">
      <h3>Some Implementation Features.</h3>
      <ul>
              <li><a href="#">CSS Based Design.</a></li>
              <li><a href="#">Data Access via Linq to Entities.</a></li>
              <li><a href="#">MasterPage driven design.</a></li>
              <li><a href="#">Modern ASP.NET Controls User.</a></li>
              <li><a href="#">Integrated Ajac Control Toolkit Editor.</a></li>
       </ul>
    </td>
    <td style="width: 50%;">
        <img src="Content/Images/SampleProductImage.gif" alt=""/>
    </td>
  </tr>
</table>

<table style="width: 600;">
  <tr>
    <td colspan="2"><hr /></td>
  </tr>
  <tr>
    <td style="width: 300px; vertical-align: top;">
        <!-- Popular Items -->
    </td>
    <td>
      <center><h3>Ecommerce the .NET 4 Way</h3></center>
      <blockquote>
        <p>
        ASP.NET offers web developers the benefit of more that a decade of innovation.
        This   demo leverages many of the latest features of ASP.NET development to
        illustrate really simply building rich web applications with ASP.NET can be.
        For more information about build web applications with ASP.NET please visit the
        community web site at www.asp.net
        </p>
      </blockquote>
    </td>
  </tr>
</table>

<h3>Spyworks Event Calendar</h3>
<table style="width: 740px;">
  <tr class="rowH">
```

```
    <th>Date</th>
    <th>Title</th>
    <th>Description</th>
  </tr>
  <tr class="rowA">
    <td style="width: 120px">June 01, 2011</td>
    <td style="width: 200px">Sed vestibulum blandit</td>
    <td>
    Come and check out demos of all the newest Tailspin Spyworks products and experience
    them hands on.
    </td>
  </tr>
  <tr class="rowB">
    <td>November 28, 2011</td>
    <td>Spyworks Product Demo</td>
    <td>
    Come and check out demos of all the newest Tailspin Spyworks products and experience
    them hands on.
    </td>
  </tr>
  <tr class="rowA">
    <td>November 23, 2011</td>
    <td>Spyworks Product Demo</td>
    <td>
     Come and check out demos of all the newest Tailspin Spyworks products and experience
     them hands on.
    </td>
  </tr>
  <tr class="rowB">
    <td>November 21, 2011</td>
    <td>Spyworks Product Demo</td>
    <td>
    Come and check out demos of all the newest Tailspin Spyworks products and experience
    them hands on.
    </td>
  </tr>
</table>
```

Product Reviews

First we'll add a button with a link to a form that we can use to enter a product review.

```
<div class="SubContentHead">Reviews</div><br />
<a id="ReviewList_AddReview" href="ReviewAdd.aspx?productID=<%# Eval("ProductID") %>">
   <img id="Img2" runat="server" style="vertical-align: bottom"
        src="~/Styles/Images/review_this_product.gif" alt="" />
</a>
```



Tailspin Spyworks

Powered by ASP.NET

Communications Device

Subversively stay in touch with this miniaturized wireless communications device. Speak into the pointy end and listen with the other end! Voice-activated dialing makes calling for backup a breeze. Excellent for undercover work at schools, rest homes, and most corporate headquarters. Comes in assorted colors.

Your Price: $49.99

Model Number: RED1

Reviews

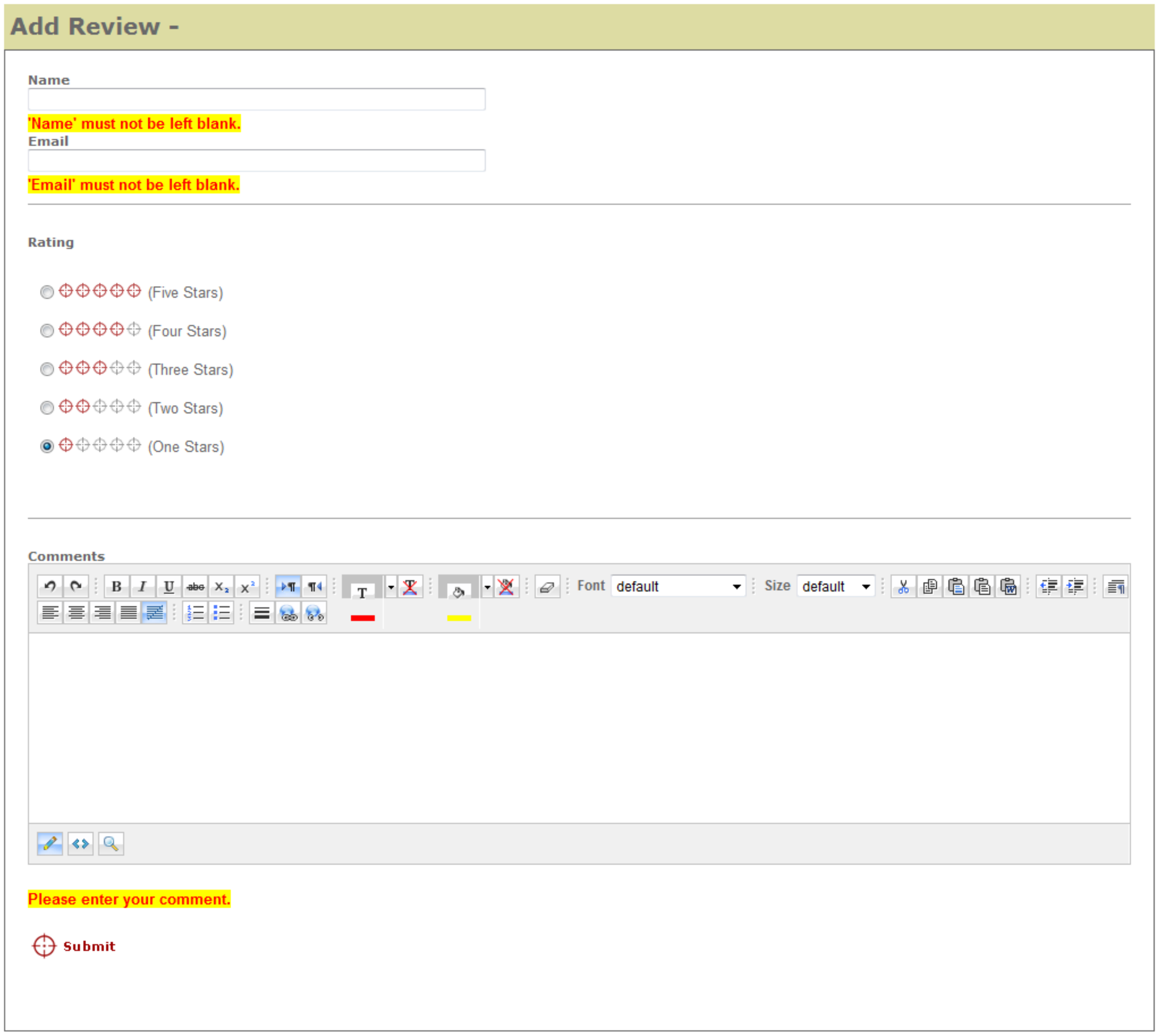
Copyright © 2010 Tailspin Spyworks

Note that we are passing the ProductID in the query string

Next let's add page named ReviewAdd.aspx

This page will use the ASP.NET AJAC Control Toolkit. If you have not already done so you can download it from here http://ajaxcontroltoolkit.codeplex.com/ and there is guidance on setting up the toolkit for use with Visual Studio here http://www.asp.net/learn/ajax-videos/video-76.aspx.

In design mode, drag controls and validators from the toolbox and build a form like the one below.


Add Review -

Name

'Name' must not be left blank.

Email

'Email' must not be left blank.

Rating

( ) (Five Stars)

( ) (Four Stars)

( ) (Three Stars)

( ) (Two Stars)

(●) (One Stars)

Comments

Font default Size default

Please enter your comment.

Submit

The markup will look something like this.

```
<asp:ToolkitScriptManager ID="ToolkitScriptManager1" runat="server">
</asp:ToolkitScriptManager>
<div class="ContentHead">Add Review - <asp:label id="ModelName" runat="server" /></div>
<div style="padding: 20px; border-style:solid; border-width: 1px">
  <span class="NormalBold">Name</span><br />
  <asp:TextBox id="Name" runat="server" Width="400px" /><br />
  <asp:RequiredFieldValidator runat="server" id="RequiredFieldValidator1"
                              ControlToValidate="Name"
                              Display="Dynamic"
                              CssClass="ValidationError"
                              ErrorMessage="'Name' must not be left blank."  /><br />
  <span class="NormalBold">Email</span><br />
  <asp:TextBox id="Email" runat="server" Width="400px" /><br />
  <asp:RequiredFieldValidator runat="server" id="RequiredFieldValidator2"
```

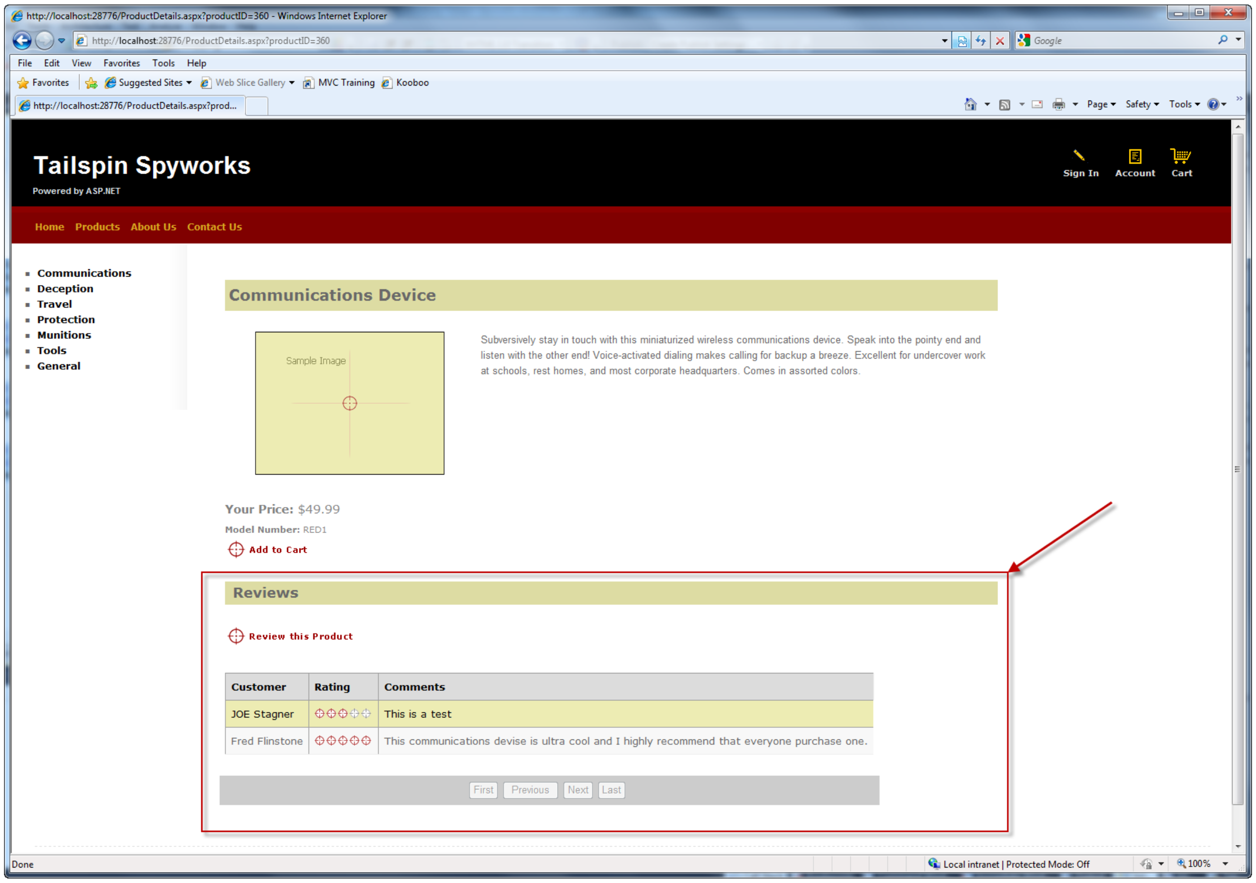
```
                              ControlToValidate="Email" Display="Dynamic"
                              CssClass="ValidationError"
                              ErrorMessage="'Email' must not be left blank." />
  <br /><hr /><br />
  <span class="NormalBold">Rating</span><br /><br />
  <asp:RadioButtonList ID="Rating" runat="server">
    <asp:ListItem value="5" selected="True"
             Text='<img src="Styles/Images/reviewrating5.gif" alt=""> (Five Stars) '  />
    <asp:ListItem value="4" selected="True"
             Text='<img src="Styles/Images/reviewrating4.gif" alt=""> (Four Stars) '  />
    <asp:ListItem value="3" selected="True"
             Text='<img src="Styles/Images/reviewrating3.gif" alt=""> (Three Stars) '  />
    <asp:ListItem value="2" selected="True"
             Text='<img src="Styles/Images/reviewrating2.gif" alt=""> (Two Stars) '  />
    <asp:ListItem value="1" selected="True"
             Text='<img src="Styles/Images/reviewrating1.gif" alt=""> (One Stars) '  />
  </asp:RadioButtonList>
  <br /><hr /><br />
  <span class="NormalBold">Comments</span><br />
  <cc1:Editor ID="UserComment" runat="server" />
  <asp:RequiredFieldValidator runat="server" id="RequiredFieldValidator3"
                              ControlToValidate="UserComment" Display="Dynamic"
                              CssClass="ValidationError"
                              ErrorMessage="Please enter your comment." /><br /><br />
  <asp:ImageButton ImageURL="Styles/Images/submit.gif" runat="server"
                   id="ReviewAddBtn" onclick="ReviewAddBtn_Click" />
  <br /><br /><br />
</div>
```

Now that we can enter reviews, lets display those reviews on the product page.

Add this markup to the ProductDetails.aspx page.

```
<asp:ListView ID="ListView_Comments" runat="server"
              DataKeyNames="ReviewID,ProductID,Rating" DataSourceID="EDS_CommentsList">
  <ItemTemplate>
    <tr style="background-color:#EDECB3;color: #000000;">
      <td><%# Eval("CustomerName") %></td>
      <td>
        <img src='Styles/Images/ReviewRating_d<%# Eval("Rating") %>.gif' alt="">
        <br />
      </td>
      <td>
        <%# Eval("Comments") %>
      </td>
    </tr>
  </ItemTemplate>
  <AlternatingItemTemplate>
    <tr style="background-color:#F8F8F8;">
      <td><%# Eval("CustomerName") %></td>
      <td>
        <img src='Styles/Images/ReviewRating_da<%# Eval("Rating") %>.gif' alt="">
        <br />
      </td>
      <td><%# Eval("Comments") %></td>
    </tr>
  </AlternatingItemTemplate>
   <EmptyDataTemplate>
     <table runat="server" style="background-color: #FFFFFF;
                                 border-collapse:  collapse;
                                 border-color: #999999;
                                 border-style:none;
                                 border-width:1px;">
       <tr><td>There are no reviews yet for this product.</td></tr>
     </table>
  </EmptyDataTemplate>
  <LayoutTemplate>
    <table runat="server">
      <tr runat="server">
        <td runat="server">
          <table ID="itemPlaceholderContainer" runat="server" border="1"
                 style="background-color: #FFFFFF;border-collapse: collapse;
                        border-color: #999999;border-style:none;border-width:1px;
                        font-family: Verdana, Arial, Helvetica, sans-serif;">
            <tr runat="server" style="background-color:#DCDCDC;color: #000000;">
              <th runat="server">Customer</th>
              <th runat="server">Rating</th>
              <th runat="server">Comments</th>
             </tr>
             <tr ID="itemPlaceholder" runat="server"></tr>
           </table>
         </td>
       </tr>
       <tr runat="server">
         <td runat="server" style="text-align: center;background-color: #CCCCCC;
                           font-family: Verdana, Arial, Helvetica, sans-serif;
                           color: #000000;">
           <asp:DataPager ID="DataPager1" runat="server">
             <Fields>
               <asp:NextPreviousPagerField ButtonType="Button"
                                           ShowFirstPageButton="True"
                                           ShowLastPageButton="True" />
             </Fields>
           </asp:DataPager>
         </td>
       </tr>
     </table>
  </LayoutTemplate>
</asp:ListView>
<asp:EntityDataSource ID="EDS_CommentsList" runat="server"  EnableFlattening="False"
                       AutoGenerateWhereClause="True"
                       EntityTypeFilter=""
                       Select="" Where=""
                       ConnectionString="name=CommerceEntities"
                       DefaultContainerName="CommerceEntities"
                       EntitySetName="Reviews">
   <WhereParameters>
    <asp:QueryStringParameter Name="ProductID" QueryStringField="productID"
                                               Type="Int32" />
  </WhereParameters>
</asp:EntityDataSource>
```

Running our application now and navigating to a product shows the product information including customer reviews.



Communications Device

Subversively stay in touch with this miniaturized wireless communications device. Speak into the pointy end and listen with the other end! Voice-activated dialing makes calling for backup a breeze. Excellent for undercover work at schools, rest homes, and most corporate headquarters. Comes in assorted colors.

Your Price: $49.99
Model Number: RED1

Reviews

Customer	Rating	Comments
JOE Stagner	●●●○○	This is a test
Fred Flinstone	●●●●●	This communications devise is ultra cool and I highly recommend that everyone purchase one.

Popular Items Control (Creating User Controls)

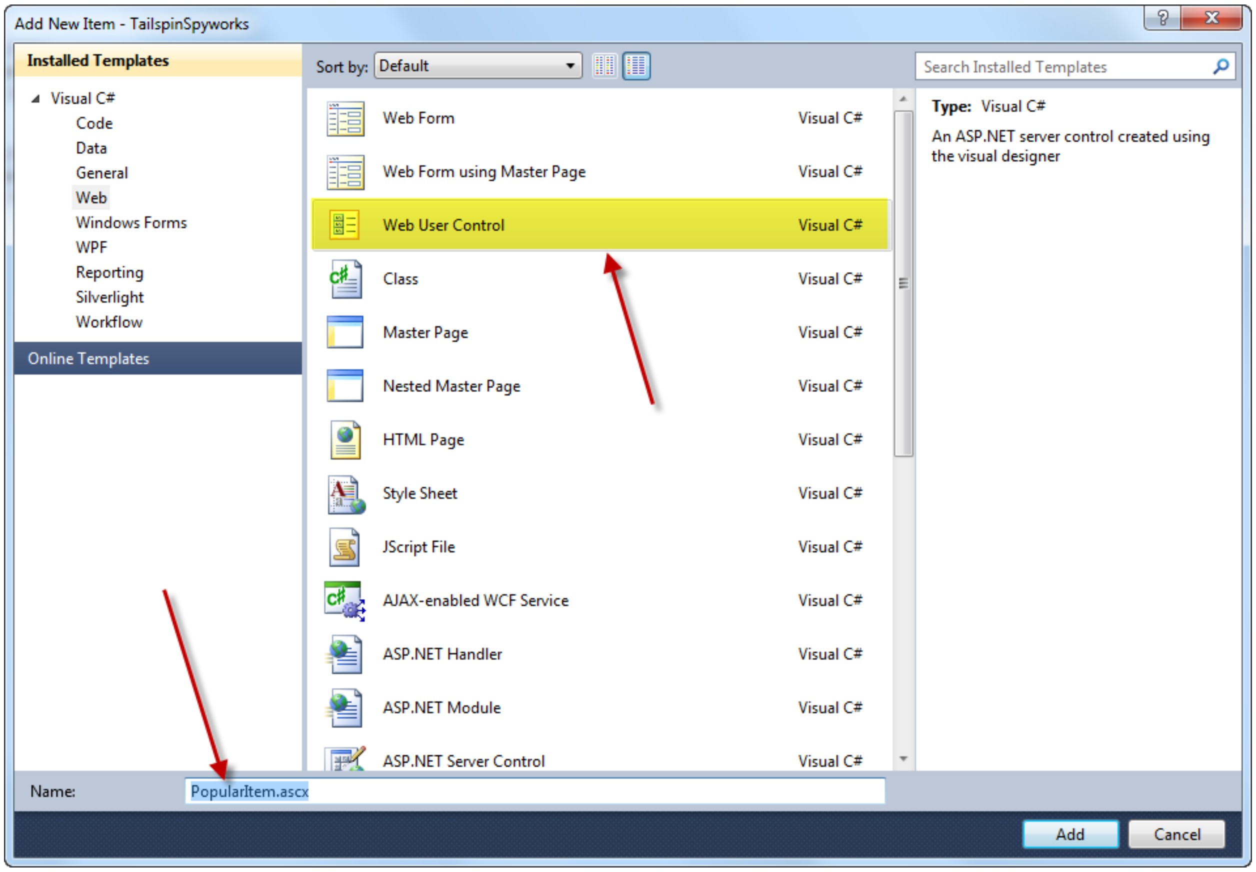
In order to increase sales on your web site we will add a couple of features to “suggestive sell” popular or related products.

The first of these features will be a list of the more popular product in our product catalog.

We will create a “User Control” to display the top selling items on the home page of our application. Since this will be a control, we can use it on any page by simply dragging and dropping the control in Visual Studio’s designer onto any page that we like.

In Visual Studio’s solutions explorer, right-click on the solution name and create a new directory named “Controls”. While it is not necessary to do so, we will help keep our project organized by creating all our user controls in the “Controls” directory.

Right-click on the controls folder and choose “New Item” :



Type: Visual C#

An ASP.NET server control created using the visual designer

Name: PopularItem.ascx

Specify a name for our control of “PopularItems”. Note that the file extension for user controls is .ascx not .aspx.

Our Popular Items User control will be defined as follows.

```
<%@ OutputCache Duration="3600" VaryByParam="None" %>
<div class="MostPopularHead">Our most popular items this week</div>
<div id="PanelPopularItems" style="padding: 10px;" runat="server">
  <asp:Repeater ID="RepeaterItemsList" runat="server">
    <HeaderTemplate></HeaderTemplate>
      <ItemTemplate>
        <a class='MostPopularItemText'
           href='ProductDetails.aspx?productID=<%# Eval("ProductId") %>'>
                                              <%# Eval("ModelName") %></a><br />
      </ItemTemplate>
    <FooterTemplate></FooterTemplate>
  </asp:Repeater>
</div>
```

Here we’re using a method we have not used yet in this application. We’re using the repeater control and instead of using a data source control we’re binding the Repeater Control to the results of a LINQ to Entities query.

In the code behind of our control we do that as follows.

```
using TailspinSpyworks.Data_Access;

protected void Page_Load(object sender, EventArgs e)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
      var query = (from ProductOrders in db.OrderDetails
                        join SelectedProducts in db.Products on ProductOrders.ProductID
                        equals SelectedProducts.ProductID
                        group ProductOrders by new
                            {
                            ProductId = SelectedProducts.ProductID,
                            ModelName = SelectedProducts.ModelName
                            } into grp
                        select new
                            {
                            ModelName = grp.Key.ModelName,
                            ProductId = grp.Key.ProductId,
                            Quantity = grp.Sum(o => o.Quantity)
                            } into orderdgrp where orderdgrp.Quantity > 0
                        orderby orderdgrp.Quantity descending select orderdgrp).Take(5);

                    RepeaterItemsList.DataSource = query;
                    RepeaterItemsList.DataBind();
      }
    catch (Exception exp)
      {
      throw new Exception("ERROR: Unable to Load Popular Items - " +
                              exp.Message.ToString(), exp);
      }
    }
}
```

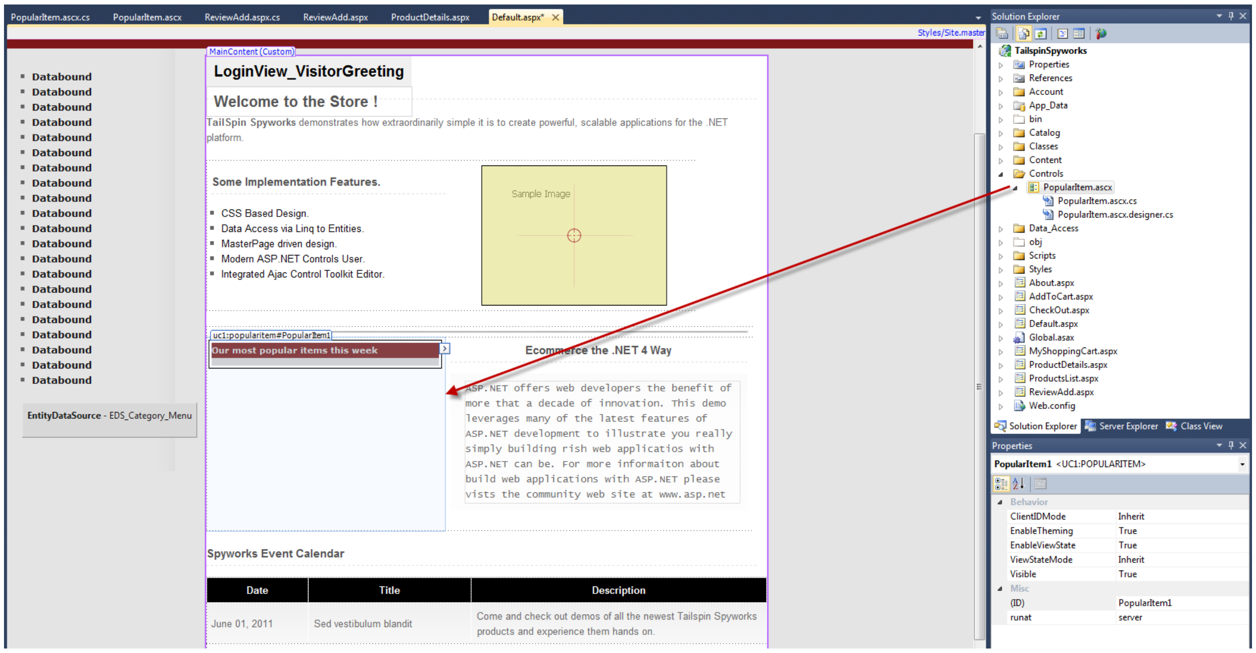
Note also this important line at the top of our control's markup.

```
<%@ OutputCache Duration="3600" VaryByParam="None" %>
```

Since the most popular items won't be changing on a minute to minute basis we can add a caching directive to improve the performance of our application. This directive will cause the controls code to only be executed when the cached output of the control expires. Otherwise, the cached version of the control's output will be used.

Now all we have to do is include our new control in our Default.aspx page.

Use drag and drop to place an instance of the control in the open column of our Default form.


PopularItem.ascx.cs | PopularItem.ascx | ReviewAdd.aspx.cs | ReviewAdd.aspx | ProductDetails.aspx | Default.aspx*

Styles/Site.master

MainContent (Custom)

LoginView_VisitorGreeting

Welcome to the Store !

TailSpin Spyworks demonstrates how extraordinarily simple it is to create powerful, scalable applications for the .NET platform.

Some Implementation Features.

- CSS Based Design.
- Data Access via Linq to Entities.
- MasterPage driven design.
- Modern ASP.NET Controls User.
- Integrated Ajac Control Toolkit Editor.

uc1:popularitem#PopularItem1

Our most popular items this week

Ecommerce the .NET 4 Way

```
ASP.NET offers web developers the benefit of
more that a decade of innovation. This demo
leverages many of the latest features of
ASP.NET development to illustrate you really
simply building rish web applicatios with
ASP.NET can be. For more informaiton about
build web applications with ASP.NET please
vists the community web site at www.asp.net
```

Spyworks Event Calendar

Date	Title	Description
June 01, 2011	Sed vestibulum blandit	Come and check out demos of all the newest Tailspin Spyworks products and experience them hands on.

- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound
- Databound

EntityDataSource - EDS_Category_Menu

Solution Explorer

- TailspinSpyworks
  - Properties
  - References
  - Account
  - App_Data
  - bin
  - Catalog
  - Classes
  - Content
  - Controls
    - PopularItem.ascx
      - PopularItem.ascx.cs
      - PopularItem.ascx.designer.cs
  - Data_Access
  - obj
  - Scripts
  - Styles
  - About.aspx
  - AddToCart.aspx
  - CheckOut.aspx
  - Default.aspx
  - Global.asax
  - MyShoppingCart.aspx
  - ProductDetails.aspx
  - ProductsList.aspx
  - ReviewAdd.aspx
  - Web.config

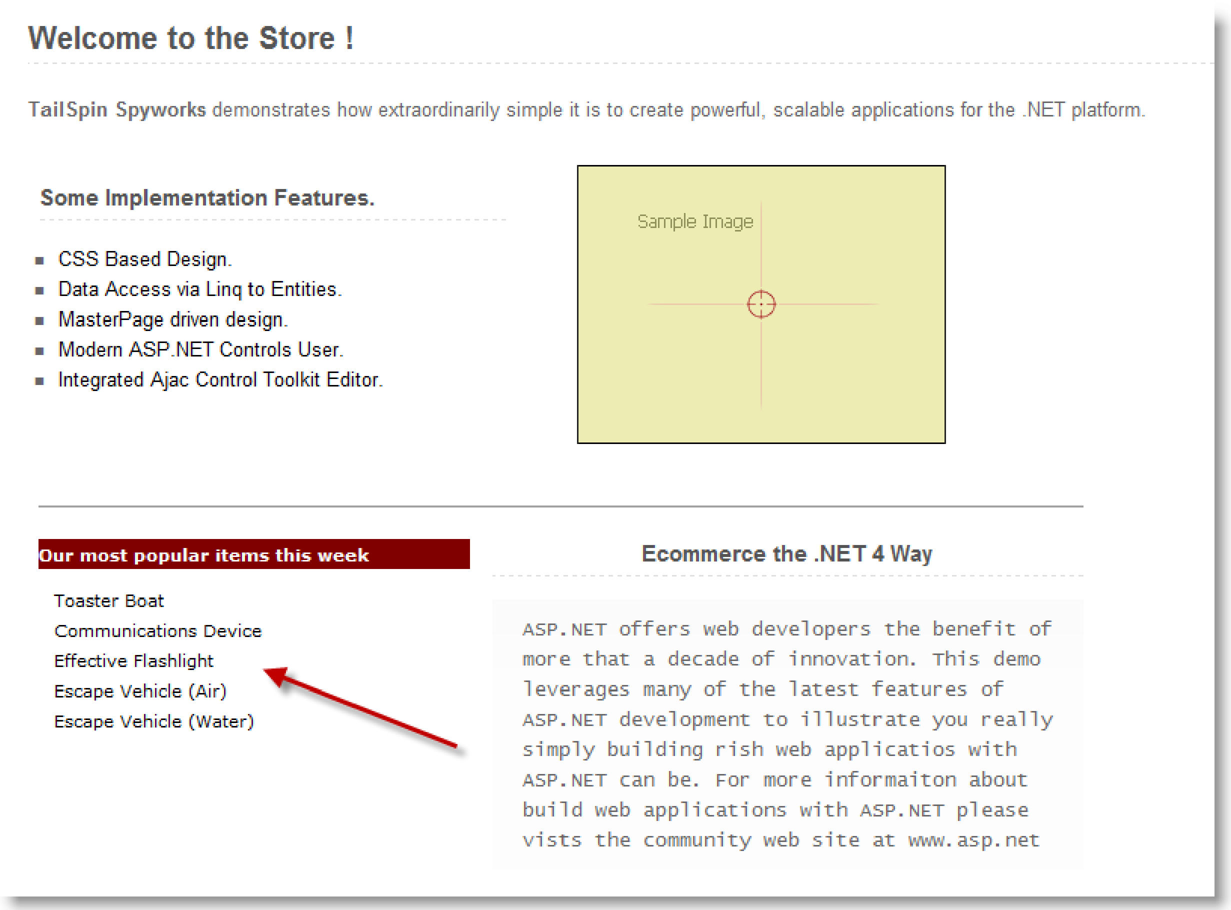
Solution Explorer | Server Explorer | Class View

Properties

PopularItem1 <UC1:POPULARITEM>

Behavior	
ClientIDMode	Inherit
EnableTheming	True
EnableViewState	True
ViewStateMode	Inherit
Visible	True
Misc	
(ID)	PopularItem1
runat	server

Now when we run our application the home page displays the most popular items.


Welcome to the Store !

TailSpin Spyworks demonstrates how extraordinarily simple it is to create powerful, scalable applications for the .NET platform.

Some Implementation Features.

- CSS Based Design.
- Data Access via Linq to Entities.
- MasterPage driven design.
- Modern ASP.NET Controls User.
- Integrated Ajac Control Toolkit Editor.

Sample Image

Our most popular items this week

Toaster Boat
Communications Device
Effective Flashlight
Escape Vehicle (Air)
Escape Vehicle (Water)

Ecommerce the .NET 4 Way

```
ASP.NET offers web developers the benefit of
more that a decade of innovation. This demo
leverages many of the latest features of
ASP.NET development to illustrate you really
simply building rish web applicatios with
ASP.NET can be. For more informaiton about
build web applications with ASP.NET please
vists the community web site at www.asp.net
```

“Also Purchased” Control (User Controls with Parameters)

The second User Control that we’ll create will take suggestive selling to the next level by adding context specificity.

The logic to calculate the top “Also Purchased” items is non-trivial.

Our “Also Purchased” control will select the OrderDetails records (previously purchased) for the currently selected ProductID and grab the OrderIDs for each unique order that is found.

Then we will select all the products from all those Orders and sum the quantities purchased. We’ll sort the products by that quantity sum and display the top five items.

Given the complexity of this logic, we will implement this algorithm as a stored procedure.

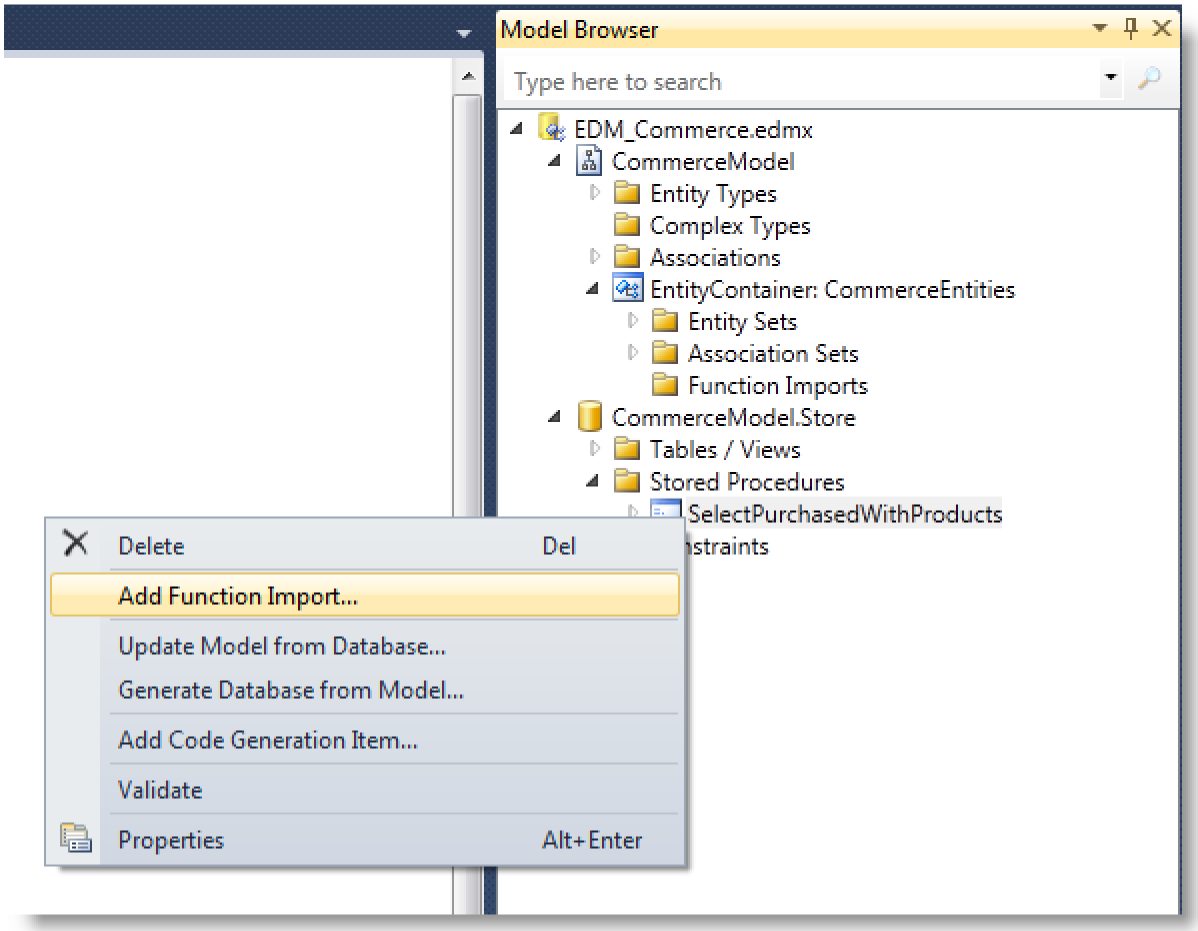
The T-SQL for the stored procedure is as follows.

```
ALTER PROCEDURE dbo.SelectPurchasedWithProducts
 @ProductID int
AS
       SELECT  TOP 5
    OrderDetails.ProductID,
    Products.ModelName,
    SUM(OrderDetails.Quantity) as TotalNum

FROM
    OrderDetails
  INNER JOIN Products ON OrderDetails.ProductID = Products.ProductID

WHERE   OrderID IN
(
    /* This inner query should retrieve all orders that have contained the productID */
    SELECT DISTINCT OrderID
    FROM OrderDetails
    WHERE ProductID = @ProductID
)
AND OrderDetails.ProductID != @ProductID

GROUP BY OrderDetails.ProductID, Products.ModelName

ORDER BY TotalNum DESC
RETURN
```

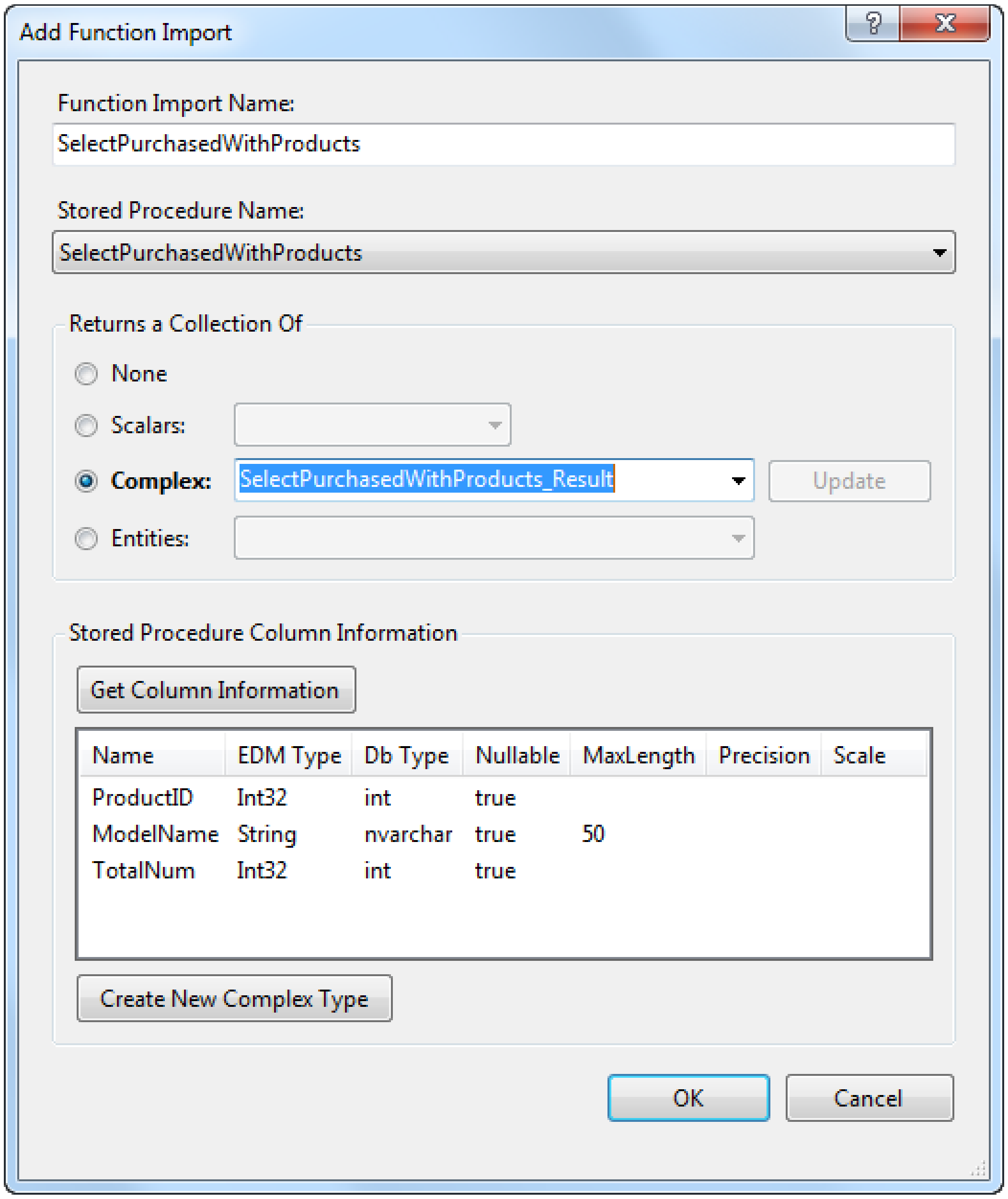
Note that this stored procedure (SelectPurchasedWithProducts) existed in the database when we included it in our application and when we generated the Entity Data Model we specified that, in addition to the Tables and Views that we needed, the Entity Data Model should include this stored procedure.

To access the stored procedure from the Entity Data Model we need to import the function.

Double Click on the Entity Data Model in the Solutions Explorer to open it in the designer and open the Model Browser, then right-click in the designer and select "Add Function Import".



Doing so will open this dialog.



Function Import Name: SelectPurchasedWithProducts

Stored Procedure Name: SelectPurchasedWithProducts

Returns a Collection Of

None [ ] Scalars: [ ] Complex: [✓] SelectPurchasedWithProducts_Result Entities: [ ]

Stored Procedure Column Information

Name	EDM Type	Db Type	Nullable	MaxLength	Precision	Scale
ProductID	Int32	int	true			
ModelName	String	nvarchar	true	50		
TotalNum	Int32	int	true			

Fill out the fields as you see above, selecting the “SelectPurchasedWithProducts” and use the procedure name for the name of our imported function.

Click “Ok”.

Having done this we can simply program against the stored procedure as we might any other item in the model.

So, in our “Controls” folder create a new user control named AlsoPurchased.ascx.

The markup for this control will look very familiar to the PopularItems control.

```
<div style="width: 300px;">
<div class="MostPopularHead">
<asp:Label ID="LabelTitle" runat="server" Text=" Customers who bought this also
bought:"></asp:Label></div>
<div id="PanelAlsoBoughtItems" style="padding: 10px; background-color:#EDECB3"
runat="server">
    <asp:Repeater ID="RepeaterItemsList" runat="server">
       <HeaderTemplate></HeaderTemplate>
          <ItemTemplate>
             <a class='MostPopularItemText' href='ProductDetails.aspx?productID=<%#
Eval("ProductId") %>'><%# Eval("ModelName") %></a><br />
          </ItemTemplate>
       <FooterTemplate></FooterTemplate>
    </asp:Repeater>
</div>
</div>
```

The notable difference is that are not caching the output since the item's to be rendered will differ by product.

The ProductId will be a "property" to the control.

```
private int _ProductId;

public int ProductId
{
get { return _ProductId ; }
set { _ProductId = Convert.ToInt32(value); }
}
```

In the control's PreRender event handler we eed to do three things.

1. Make sure the ProductID is set.
2. See if there are any products that have been purchased with the current one.
3. Output some items as determined in #2.

Note how easy it is to call the stored procedure through the model.

```
//--------------------------------------------------------------------------------------+
protected void Page_PreRender(object sender, EventArgs e)
{
  if (_ProductId < 1)
     {
     // This should never happen but we could expand the use of this control by reducing
     // the dependency on the query string by selecting a few RANDOME products here.
     Debug.Fail("ERROR : The Also Purchased Control Can not be used without
                         setting the ProductId.");
     throw new Exception("ERROR : It is illegal to load the AlsoPurchased COntrol
                                  without setting a ProductId.");
     }

  int ProductCount = 0;
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
      var v = db.SelectPurchasedWithProducts(_ProductId);
      ProductCount = v.Count();
      }
    catch (Exception exp)
      {
      throw new Exception("ERROR: Unable to Retrieve Also Purchased Items - " +
                                    exp.Message.ToString(), exp);
      }
    }

  if (ProductCount > 0)
     {
     WriteAlsoPurchased(_ProductId);
     }
  else
     {
     WritePopularItems();
     }
}
```

After determining that there ARE "also purchased" we can simply bind the repeater to the results returned by the query.

```
//------------------------------------------------------------------------------------+
private void WriteAlsoPurchased(int currentProduct)
{
  using (CommerceEntities db = new CommerceEntities())
        {
        try
          {
          var v = db.SelectPurchasedWithProducts(currentProduct);
          RepeaterItemsList.DataSource = v;
          RepeaterItemsList.DataBind();
          }
         catch (Exception exp)
          {
          throw new Exception("ERROR: Unable to Write Also Purchased - " +
                                                        exp.Message.ToString(), exp);
          }
        }
}
```

If there were not any "also purchased" items we'll simply display other popular items from our catalog.

```
//------------------------------------------------------------------------------------+
private void WritePopularItems()
{
  using (CommerceEntities db = new CommerceEntities())
    {
    try
      {
      var query = (from ProductOrders in db.OrderDetails
                   join SelectedProducts in db.Products on ProductOrders.ProductID
                   equals SelectedProducts.ProductID
                   group ProductOrders by new
                         {
                         ProductId = SelectedProducts.ProductID,
                         ModelName = SelectedProducts.ModelName
                         } into grp
                   select new
                         {
                         ModelName = grp.Key.ModelName,
                         ProductId = grp.Key.ProductId,
                         Quantity = grp.Sum(o => o.Quantity)
                         } into orderdgrp
                   where orderdgrp.Quantity > 0
                   orderby orderdgrp.Quantity descending
                   select orderdgrp).Take(5);

      LabelTitle.Text = "Other items you might be interested in: ";
      RepeaterItemsList.DataSource = query;
      RepeaterItemsList.DataBind();
      }
    catch (Exception exp)
      {
      throw new Exception("ERROR: Unable to Load Popular Items - " +
                                                   exp.Message.ToString(), exp);
      }
    }
}
```

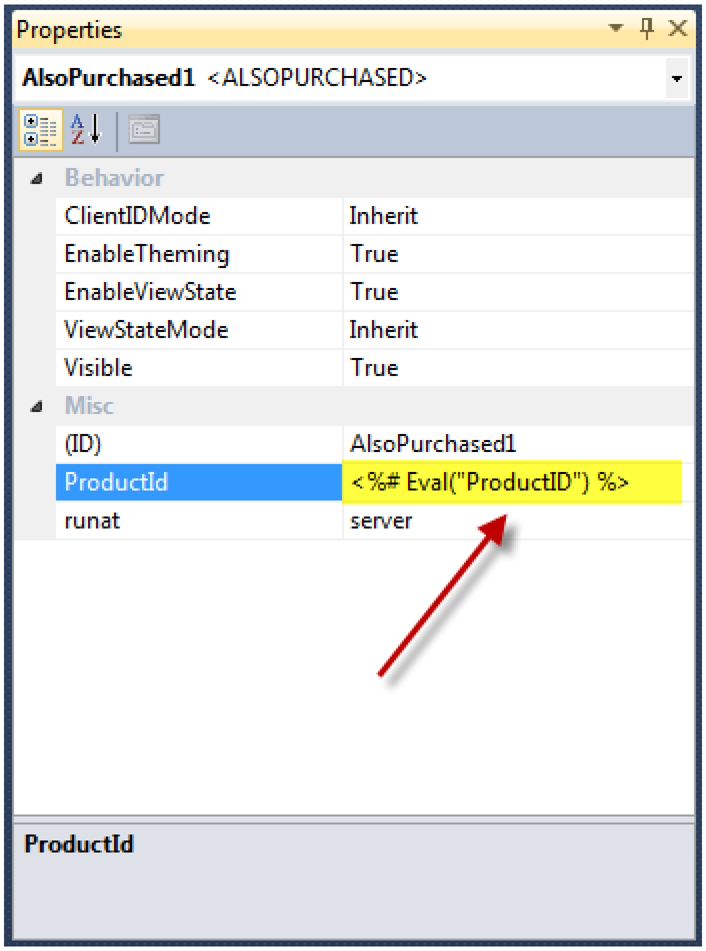
To view the “Also Purchased” items, open the ProductDetails.aspx page and drag the AlsoPurchased control from the Solutions Explorer so that it appears in this position in the markup.

```
<table  border="0">
  <tr>
     <td style="vertical-align: top;">
       <img src='Catalog/Images/<%# Eval("ProductImage") %>'  border="0"
                                                 alt='<%# Eval("ModelName") %>' />
     </td>
     <td style="vertical-align: top"><%# Eval("Description") %><br /><br /><br />
         <uc1:AlsoPurchased ID="AlsoPurchased1" runat="server" />
     </td>
   </tr>
</table>
```

Doing so will create a reference to the control at the top of the ProductDetails page.

```
<%@ Register src="Controls/AlsoPurchased.ascx" tagname="AlsoPurchased" tagprefix="uc1" %>
```

Since the AlsoPurchased user control requires a ProductId number we will set the ProductID property of our control by using an Eval statement against the current data model item of the page.

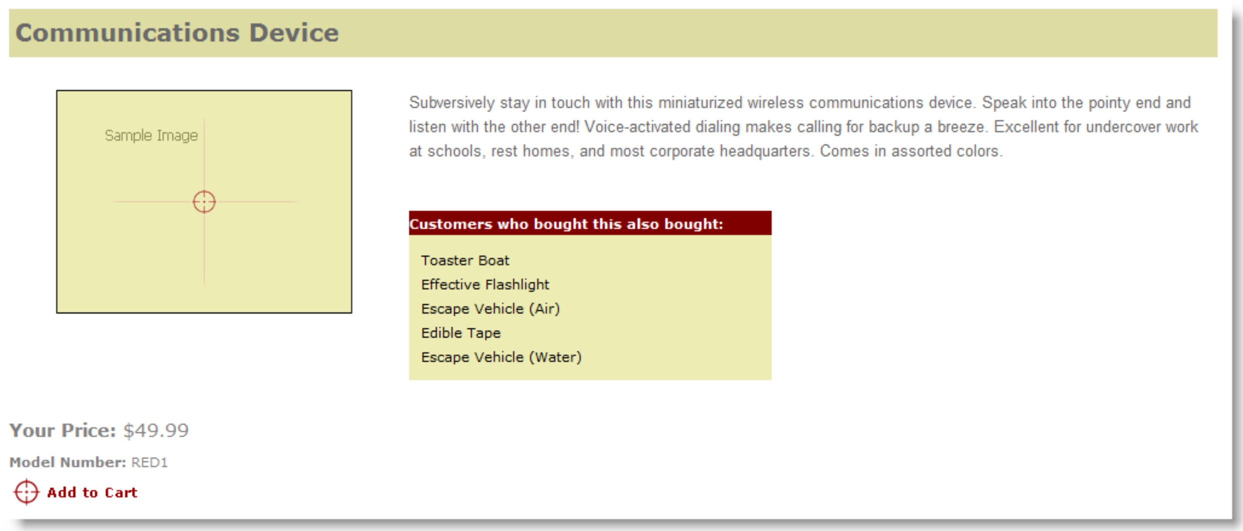


AlsoPurchased1 <ALSOPURCHASED>

Behavior	
ClientIDMode	Inherit
EnableTheming	True
EnableViewState	True
ViewStateMode	Inherit
Visible	True
Misc	
(ID)	AlsoPurchased1
ProductId	<%# Eval("ProductID") %>
runat	server

ProductId

When we build and run now and browse to a product we see the "Also Purchased" items.



Communications Device

Subversively stay in touch with this miniaturized wireless communications device. Speak into the pointy end and listen with the other end! Voice-activated dialing makes calling for backup a breeze. Excellent for undercover work at schools, rest homes, and most corporate headquarters. Comes in assorted colors.

Customers who bought this also bought:

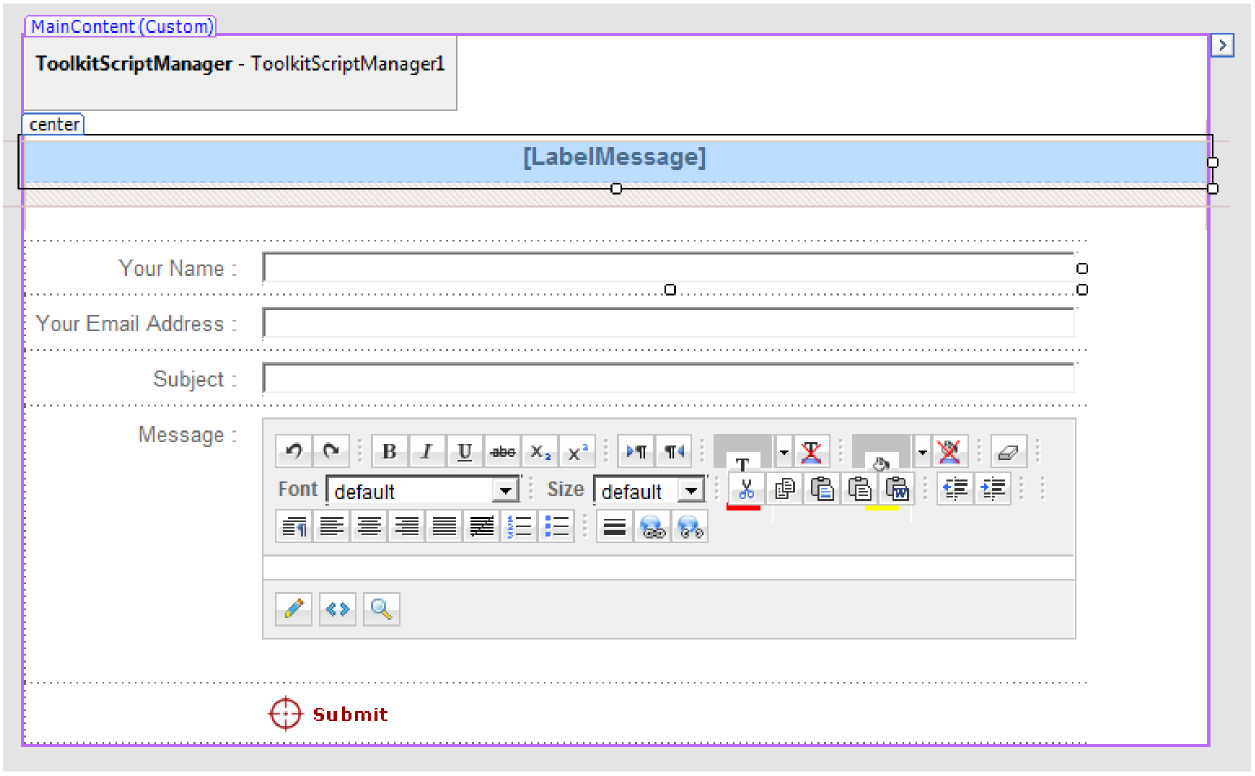
Toaster Boat
Effective Flashlight
Escape Vehicle (Air)
Edible Tape
Escape Vehicle (Water)

Your Price: $49.99
Model Number: RED1
Add to Cart

Contact Page (Sending email from ASP.NET)

Create a new page named ContactUs.aspx

Using the designer, create the following form taking special note to include the ToolkitScriptManager and the Editor control from the AjaxControlToolkit. .



MainContent (Custom)
ToolkitScriptManager - ToolkitScriptManager1
center
[LabelMessage]
Your Name :
Your Email Address :
Subject :
Message :
Font default
Size default
Submit

Double click on the “Submit” button to generate a click event handler in the code behind file and implement a method to send the contact information as an email.

```
protected void ImageButton_Submit_Click(object sender, ImageClickEventArgs e)
  {
  try
    {
    MailMessage mMailMessage = new MailMessage();
    mMailMessage.From = new MailAddress(HttpUtility.HtmlEncode(TextBoxEmail.Text));
    mMailMessage.To.Add(new MailAddress("Your Email Here"));

    // mMailMessage.Bcc.Add(new MailAddress(bcc));
    // mMailMessage.CC.Add(new MailAddress(cc));

   mMailMessage.Subject = "From:" + HttpUtility.HtmlEncode(TextBoxYourName.Text) + "-" +
                                    HttpUtility.HtmlEncode(TextBoxSubject.Text);
   mMailMessage.Body = HttpUtility.HtmlEncode(EditorEmailMessageBody.Content);
   mMailMessage.IsBodyHtml = true;
   mMailMessage.Priority = MailPriority.Normal;
   SmtpClient mSmtpClient = new SmtpClient();
   mSmtpClient.Send(mMailMessage);
   LabelMessage.Text = "Thank You - Your Message was sent.";
   }
 catch (Exception exp)
   {
   throw new Exception("ERROR: Unable to Send Contact - " + exp.Message.ToString(), exp);
   }
}
```

This code requires that your web.config file contain an entry in the configuration section that specifies the SMTP server to use for sending mail.

```
    <system.net>
        <mailSettings>
            <smtp>
                <network
                     host="mail..com"
                     port="25"
                     userName=""
                     password="" />
            </smtp>
        </mailSettings>
    </system.net>
```

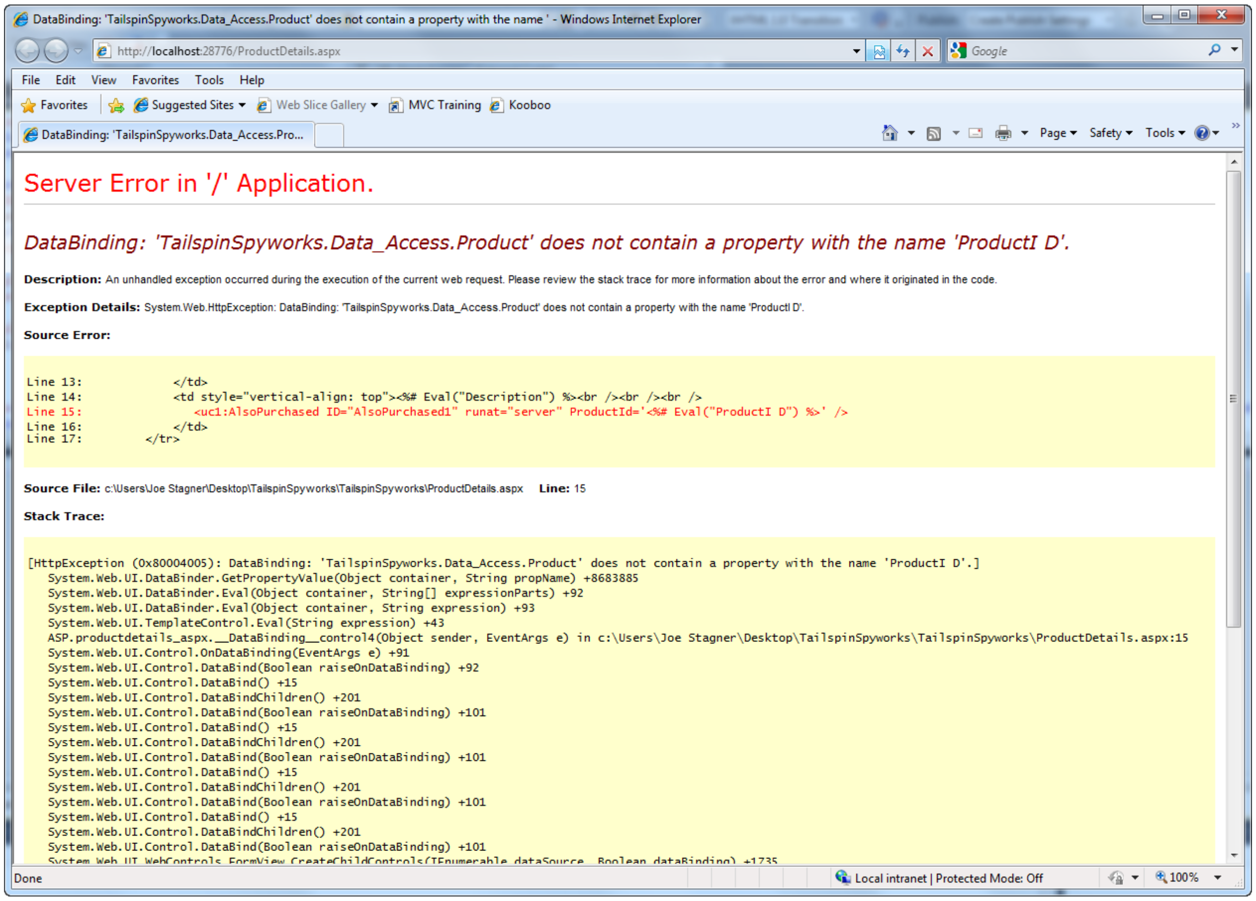
About Page

Create a page named AboutUs.aspx and add whatever content you like.

Global Exception Handler

Lastly, throughout the application we have thrown exceptions and there are unforeseen circumstances that cold also cause unhandled exceptions in our web application.

We never want an unhandled exception to be displayed to a web site visitor.


DataBinding: 'TailspinSpyworks.Data_Access.Product' does not contain a property with the name ' - Windows Internet Explorer
http://localhost:28776/ProductDetails.aspx
Google
File Edit View Favorites Tools Help
Favorites
Suggested Sites
Web Slice Gallery
MVC Training
Kooboo
DataBinding: 'TailspinSpyworks.Data_Access.Pro...
Page
Safety
Tools
Done
Local intranet | Protected Mode: Off
100%

Server Error in '/' Application.

DataBinding: 'TailspinSpyworks.Data_Access.Product' does not contain a property with the name 'ProductI D'.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Web.HttpException: DataBinding: 'TailspinSpyworks.Data_Access.Product' does not contain a property with the name 'ProductI D'.

Source Error:

```
Line 13:            </td>
Line 14:            <td style="vertical-align: top"><%# Eval("Description") %><br /><br /><br />
Line 15:               <uc1:AlsoPurchased ID="AlsoPurchased1" runat="server" ProductId='<%# Eval("ProductI D") %>' />
Line 16:            </td>
Line 17:        </tr>
```

Source File: c:\Users\Joe Stagner\Desktop\TailspinSpyworks\TailspinSpyworks\ProductDetails.aspx   **Line:** 15

Stack Trace:

```
[HttpException (0x80004005): DataBinding: 'TailspinSpyworks.Data_Access.Product' does not contain a property with the name 'ProductI D'.]
   System.Web.UI.DataBinder.GetPropertyValue(Object container, String propName) +8683885
   System.Web.UI.DataBinder.Eval(Object container, String[] expressionParts) +92
   System.Web.UI.DataBinder.Eval(Object container, String expression) +93
   System.Web.UI.TemplateControl.Eval(String expression) +43
   ASP.productdetails_aspx.__DataBinding__control4(Object sender, EventArgs e) in c:\Users\Joe Stagner\Desktop\TailspinSpyworks\TailspinSpyworks\ProductDetails.aspx:15
   System.Web.UI.Control.OnDataBinding(EventArgs e) +91
   System.Web.UI.Control.DataBind(Boolean raiseOnDataBinding) +92
   System.Web.UI.Control.DataBind() +15
   System.Web.UI.Control.DataBindChildren() +201
   System.Web.UI.Control.DataBind(Boolean raiseOnDataBinding) +101
   System.Web.UI.Control.DataBind() +15
   System.Web.UI.Control.DataBindChildren() +201
   System.Web.UI.Control.DataBind(Boolean raiseOnDataBinding) +101
   System.Web.UI.Control.DataBind() +15
   System.Web.UI.Control.DataBindChildren() +201
   System.Web.UI.Control.DataBind(Boolean raiseOnDataBinding) +101
   System.Web.UI.Control.DataBind() +15
   System.Web.UI.Control.DataBindChildren() +201
   System.Web.UI.Control.DataBind(Boolean raiseOnDataBinding) +101
   System.Web.UI.WebControls.FormView.CreateChildControls(IEnumerable dataSource, Boolean dataBinding) +1735
```

Apart from being a terrible user experience unhandled exceptions can also be a security problem.

To solve this problem we will implement a global exception handler.

To do this, open the Global.asax file and note the following pre-generated event handler.

```
void Application_Error(object sender, EventArgs e)
     {
     // Code that runs when an unhandled error occurs
     }
```

Add code to implement the Application_Error handler as follows.

```
void Application_Error(object sender, EventArgs e)
     {
```

```
     Exception myEx =  Server.GetLastError();
    String RedirectUrlString = "~/Error.aspx?InnerErr=" +
           myEx.InnerException.Message.ToString() + "&Err=" + myEx.Message.ToString();
     Response.Redirect(RedirectUrlString);
     }
```

Then add a page named Error.aspx to the solution and add this markup snippet.

```
<center>
  <div class="ContentHead">ERROR</div><br /><br />
  <asp:Label ID="Label_ErrorFrom" runat="server" Text="Label"></asp:Label><br /><br />
  <asp:Label ID="Label_ErrorMessage" runat="server" Text="Label"></asp:Label><br /><br />
</center>
```

Now in the Page_Load event handler extract the error messages from the Request Object.

```
protected void Page_Load(object sender, EventArgs e)
{
    Label_ErrorFrom.Text = Request["Err"].ToString();
    Label_ErrorMessage.Text = Request["InnerErr"].ToString();
}
```

Conclusion

We've seen that ASP.NET WebForms makes it easy to create a sophisticated website with database access, membership, AJAX, etc. pretty quickly.

Hopefully this tutorial has given you the tools you need to get started building your own ASP.NET WebForms applications!