Building Tailspin Spyworks

Nov 11th 2010

Beta Document

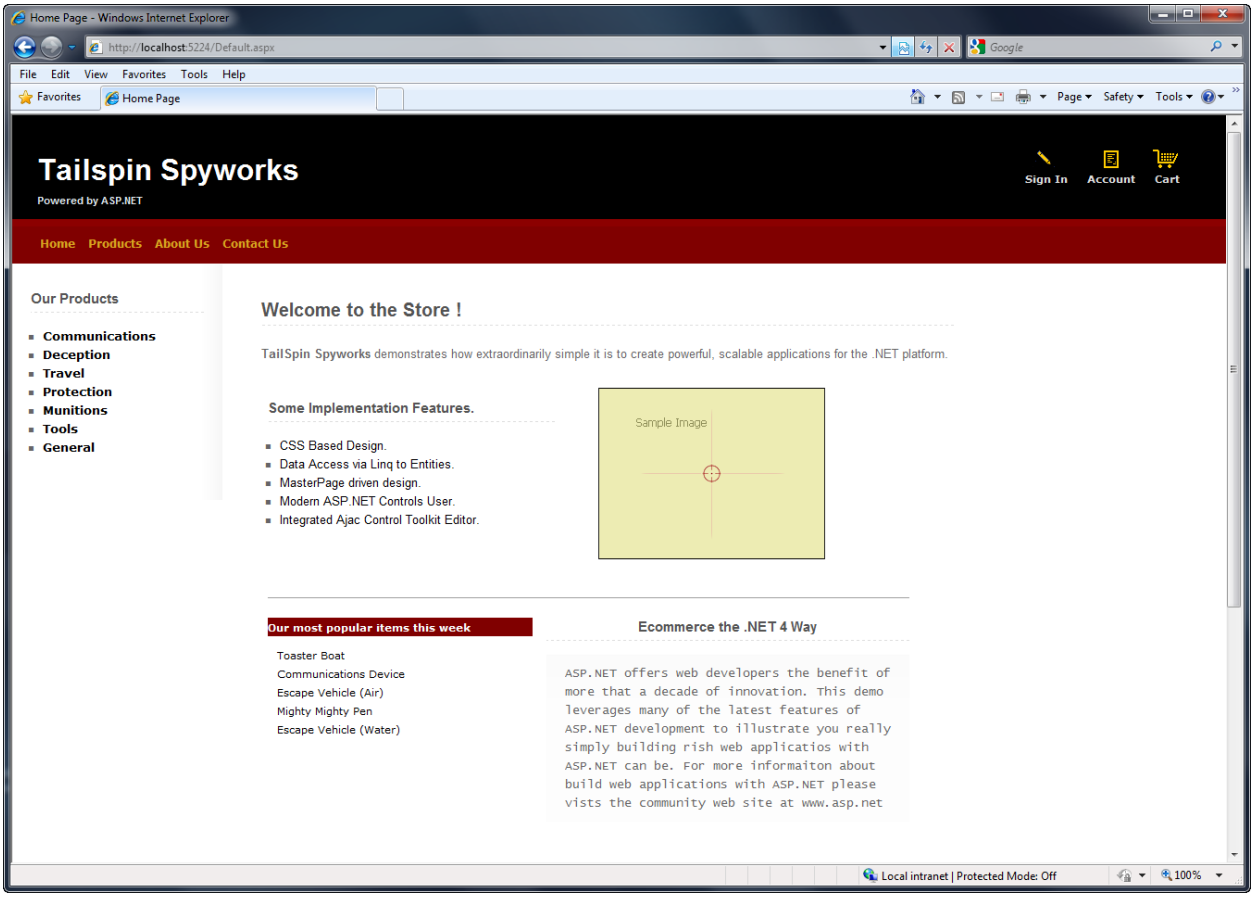
Contents

Building Tailspin Spyworks – an ASP.NET WebForms Sample

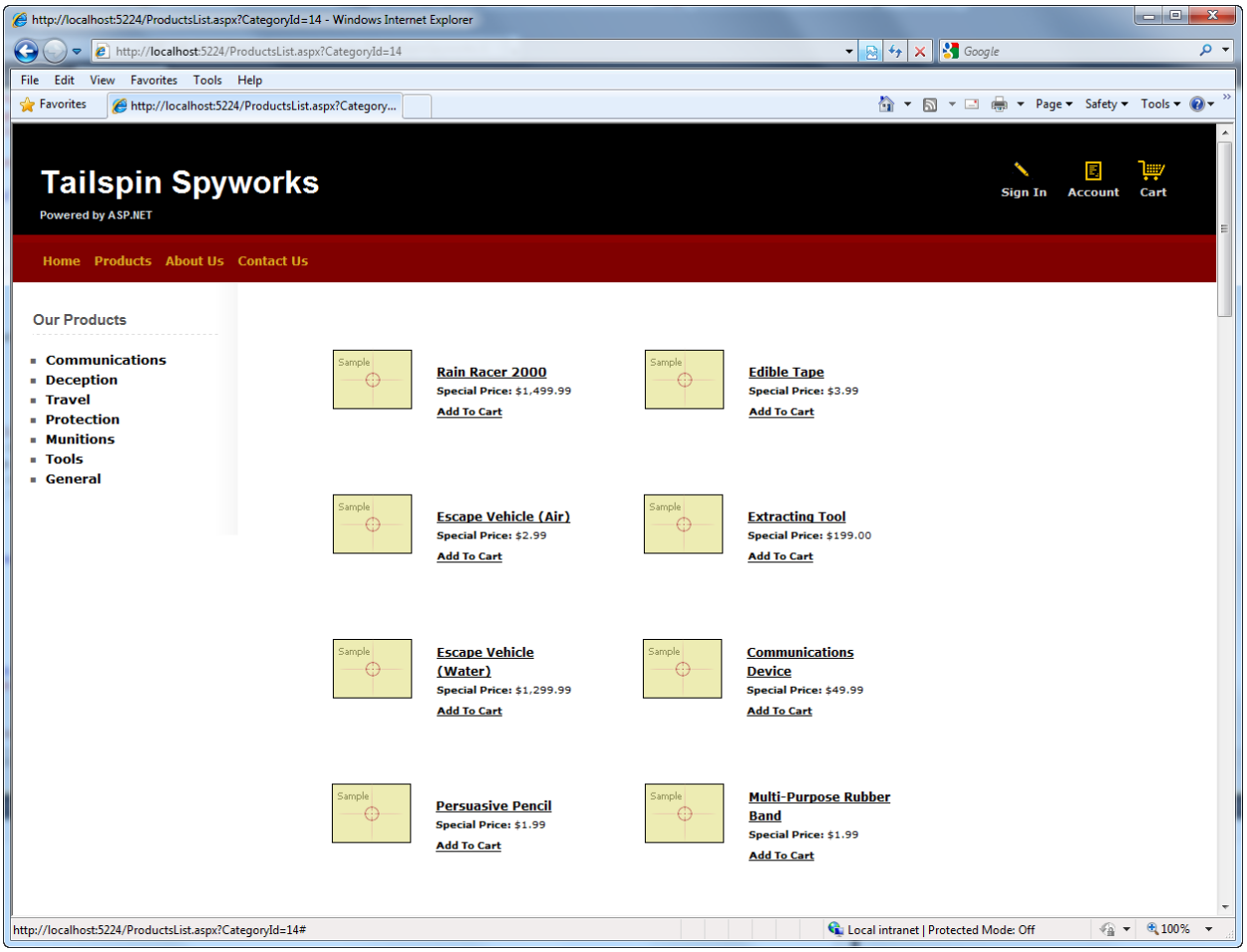
Overview

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

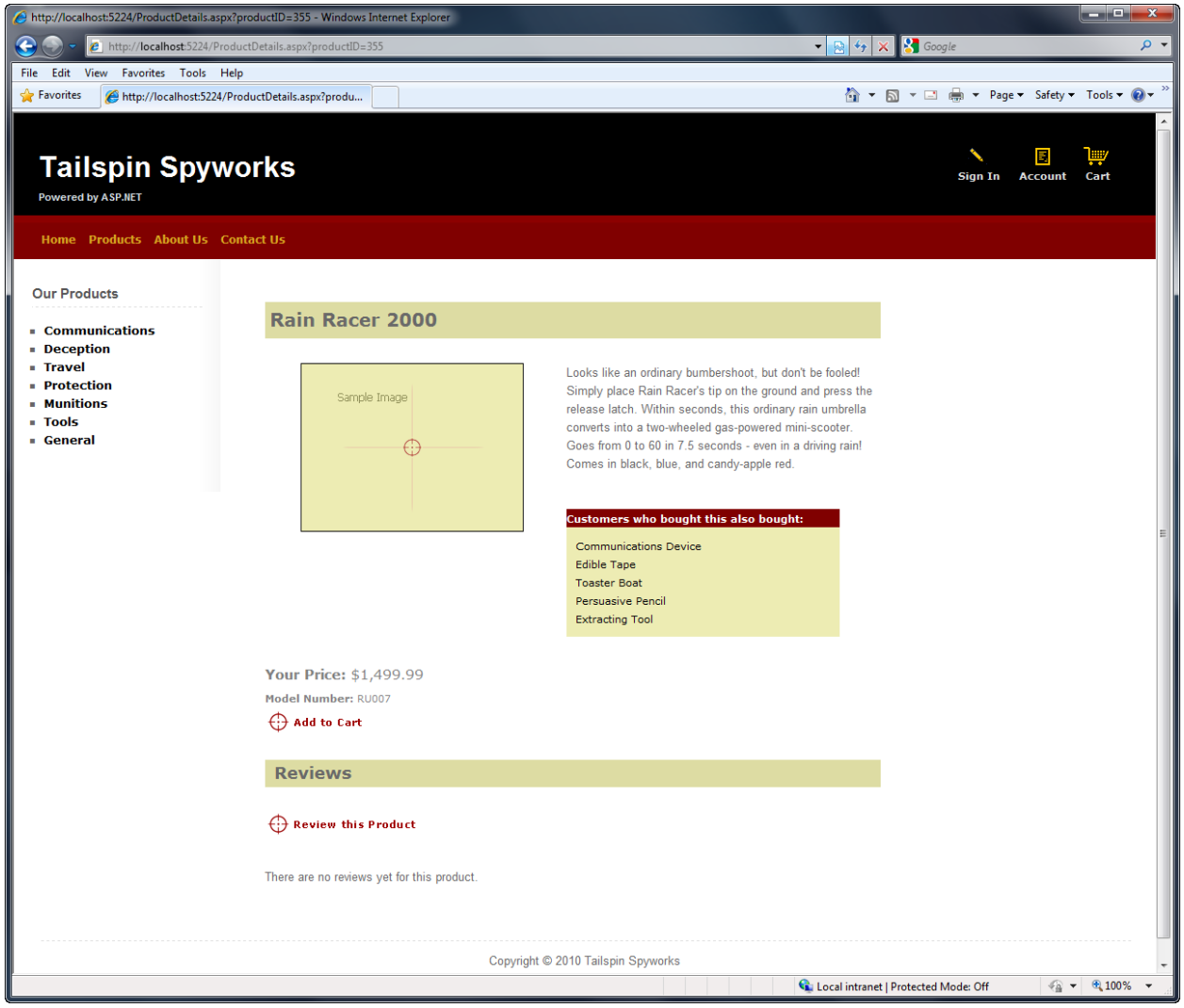
The application we'll be building is a simple on-line store.



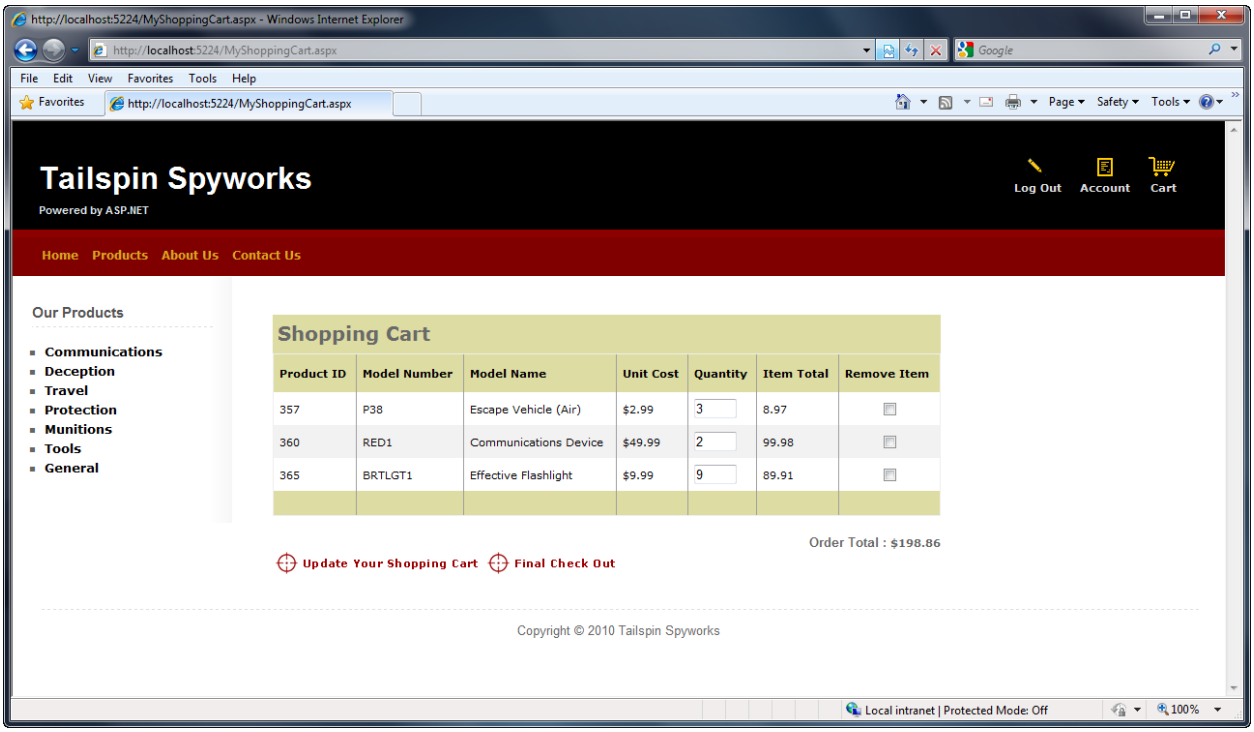
Visitors can browse Products by Category:



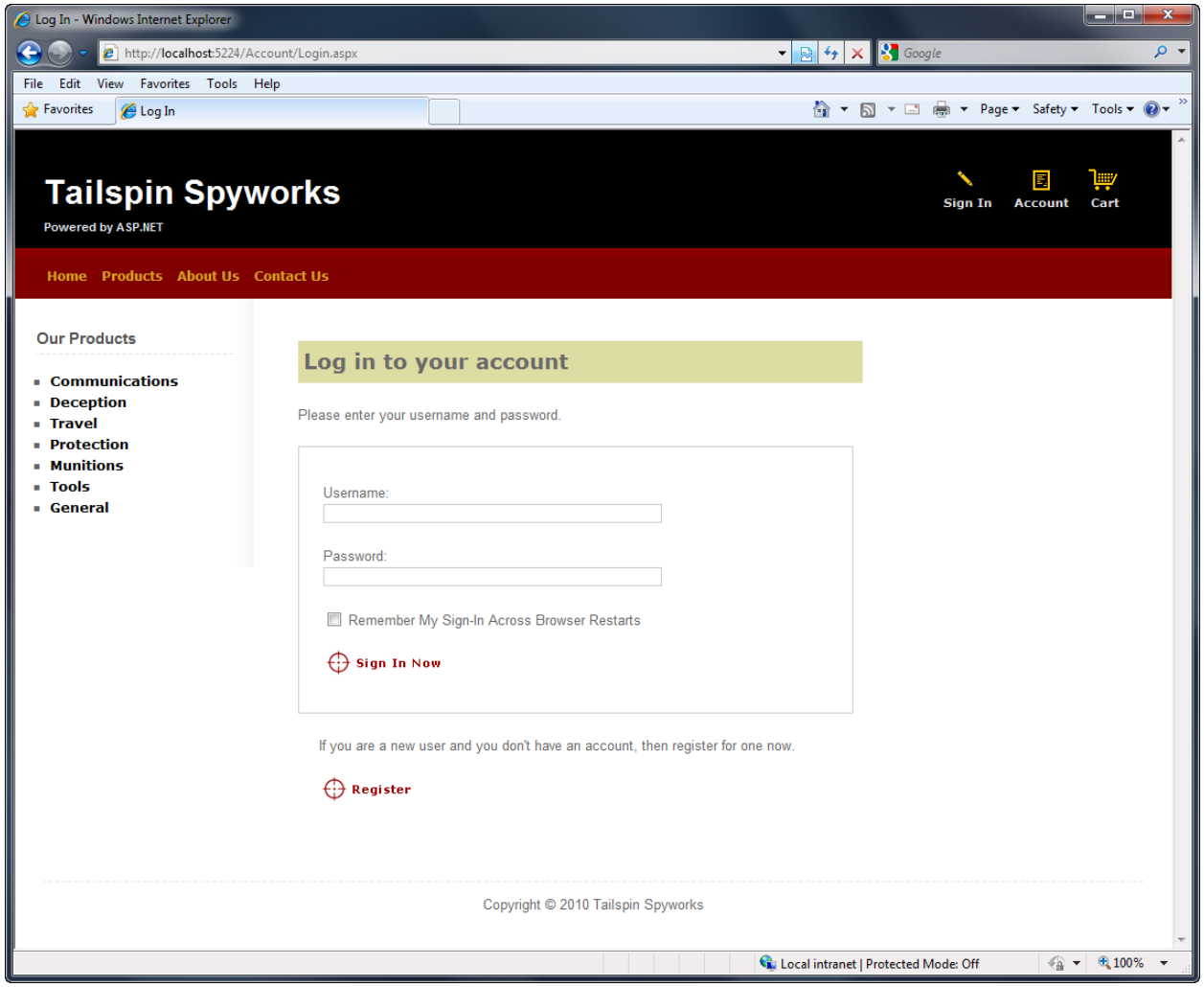
They can view a single product and add it to their cart:

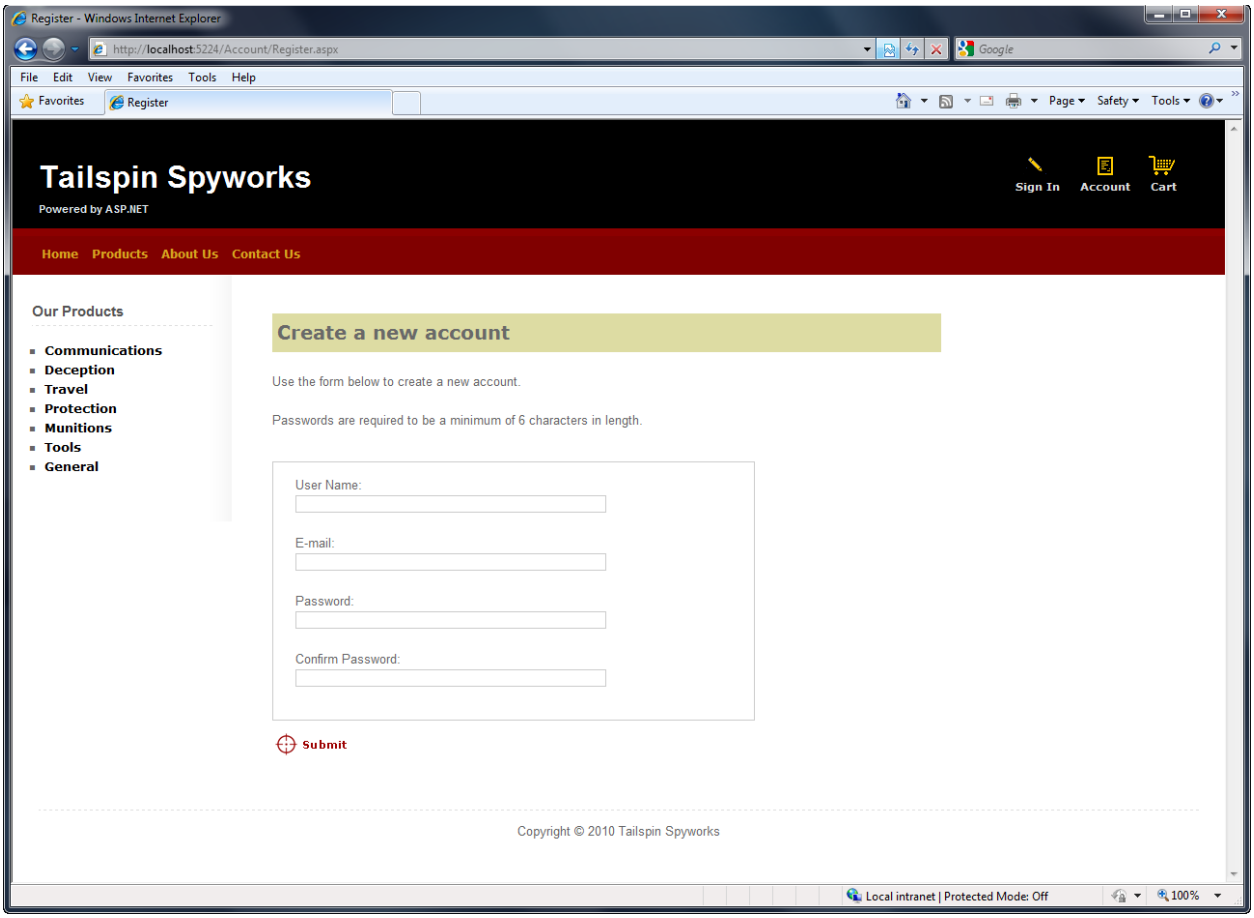


They can review their cart, removing any items they no longer want:

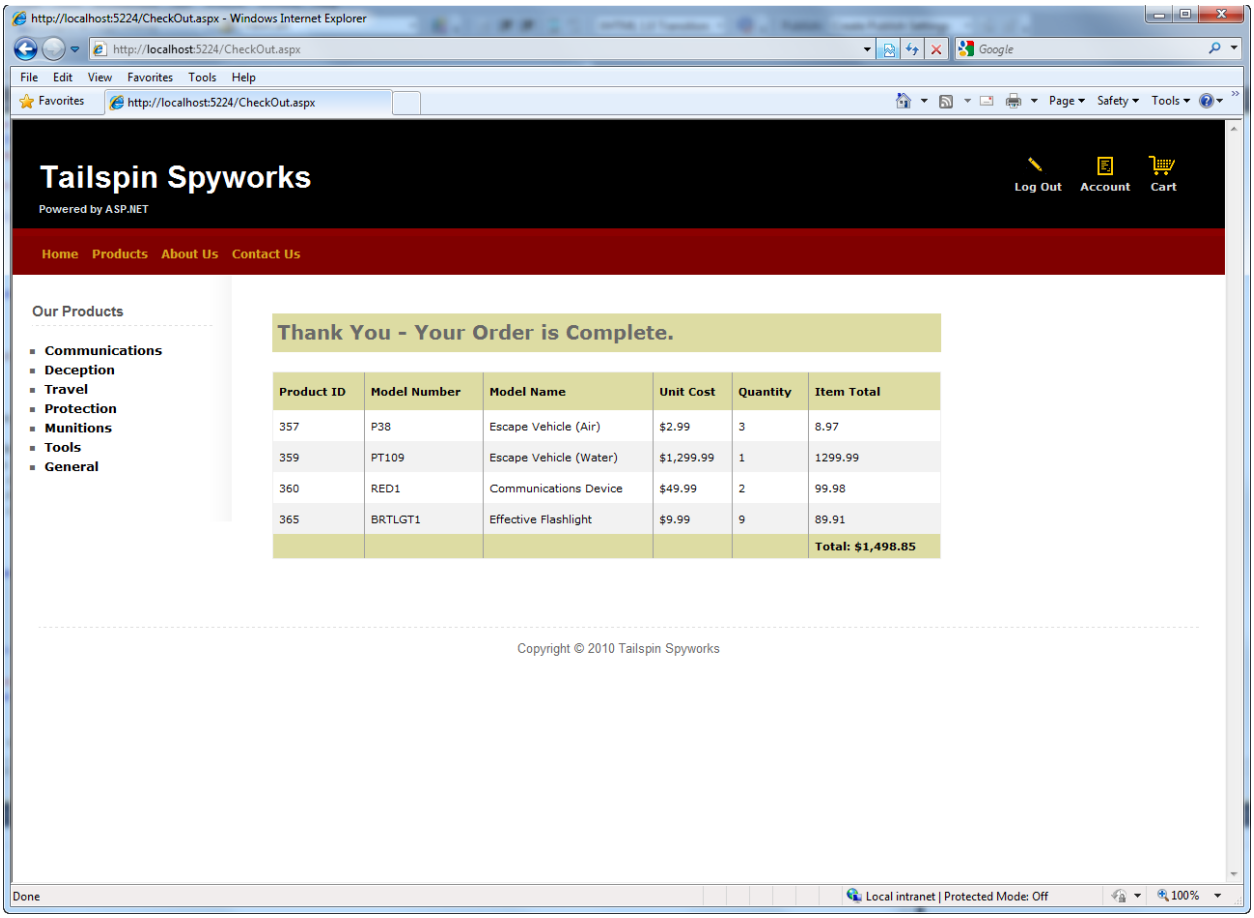


Proceeding to Checkout will prompt them to





After ordering, they see a simple confirmation screen:



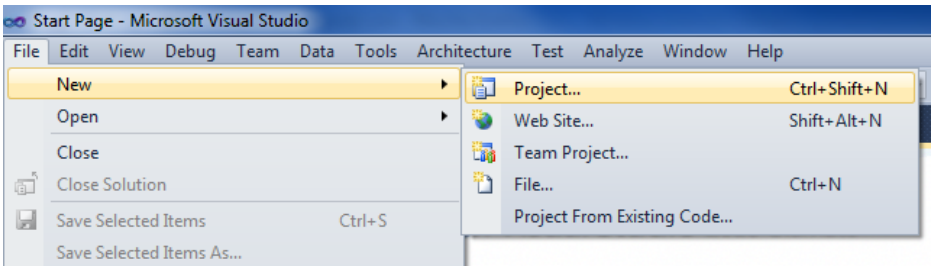
We'll begin by creating a new ASP.NET WebForms project in Visual Studio 2010, and we'll incrementally add features to create a complete functioning application. Along the way, we'll cover database access, list and grid views, data update pages, data validation, using master pages for consistent page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

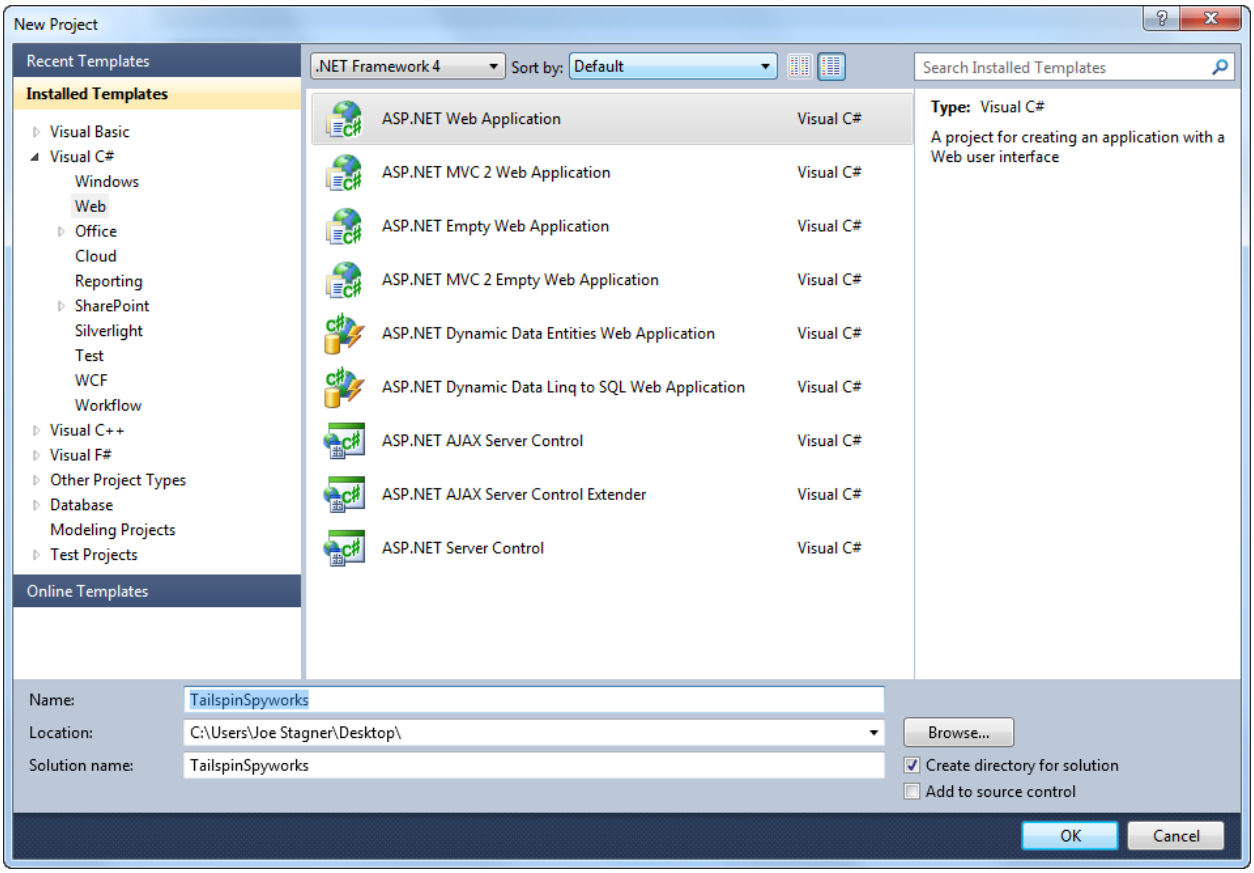
You can use either Visual Studio 2010 or the free Visual Web Developer 2010 from http://www.microsoft.com/express/Web/. To build the application, you can use either SQL Server or the free SQL Server Express to host the database.

File / New Project

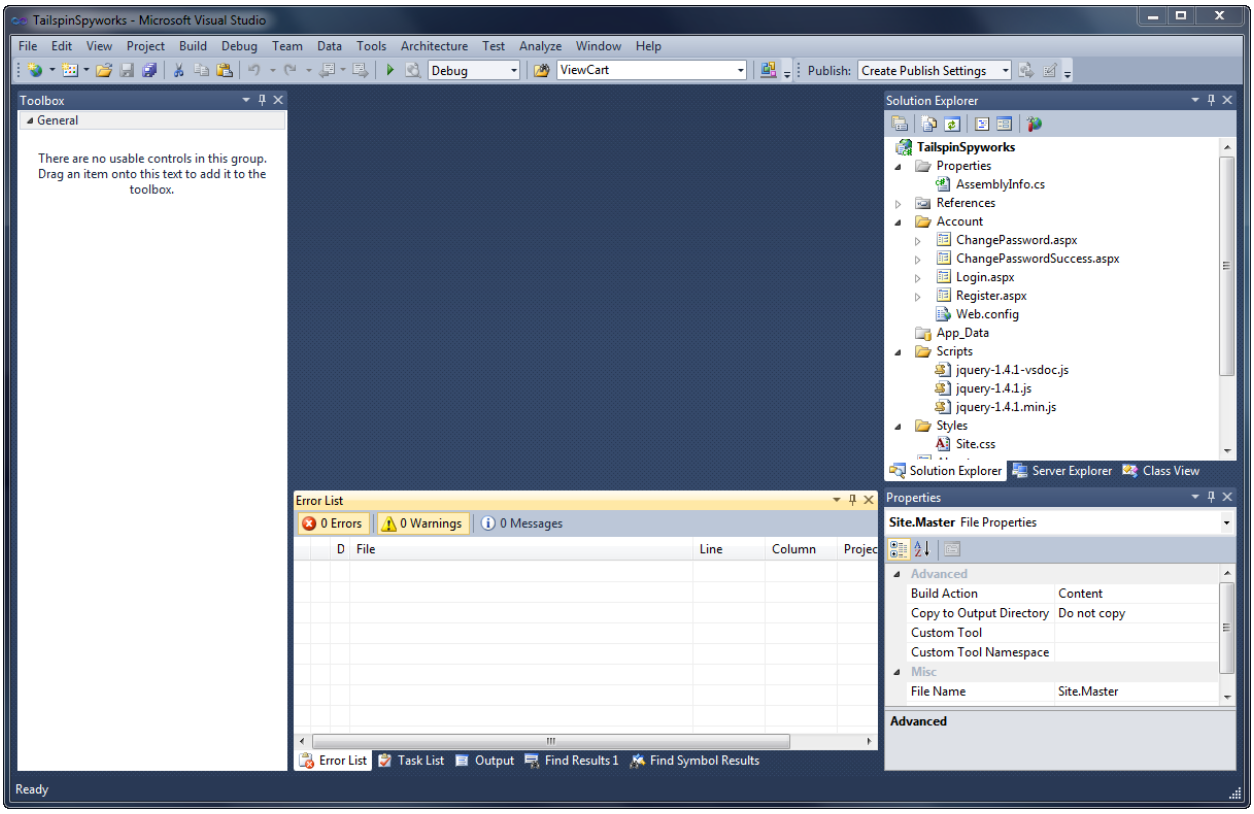
We'll start by selecting the New Project from the File menu in Visual Studio. This brings up the New Project dialog.



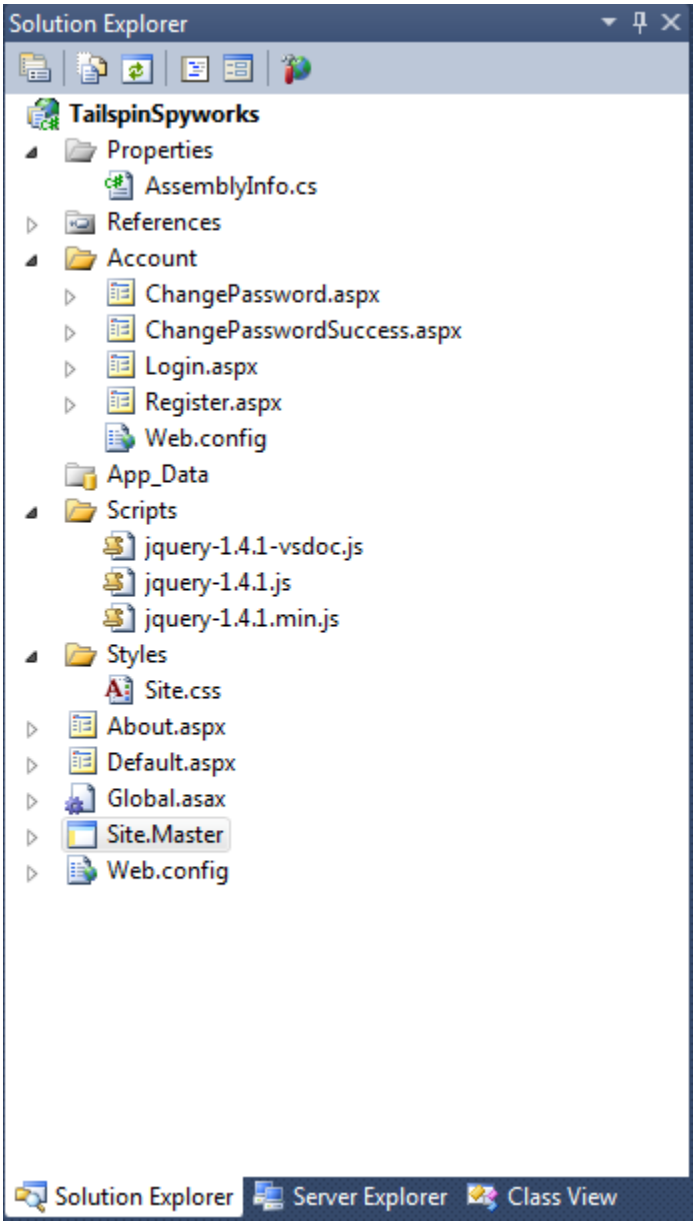
We'll select the Visual C# / Web Templates group on the left, and then choose the "ASP.NET Web Application" template in the center column. Name your project Tailspin Spyworks and press the OK button.



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.



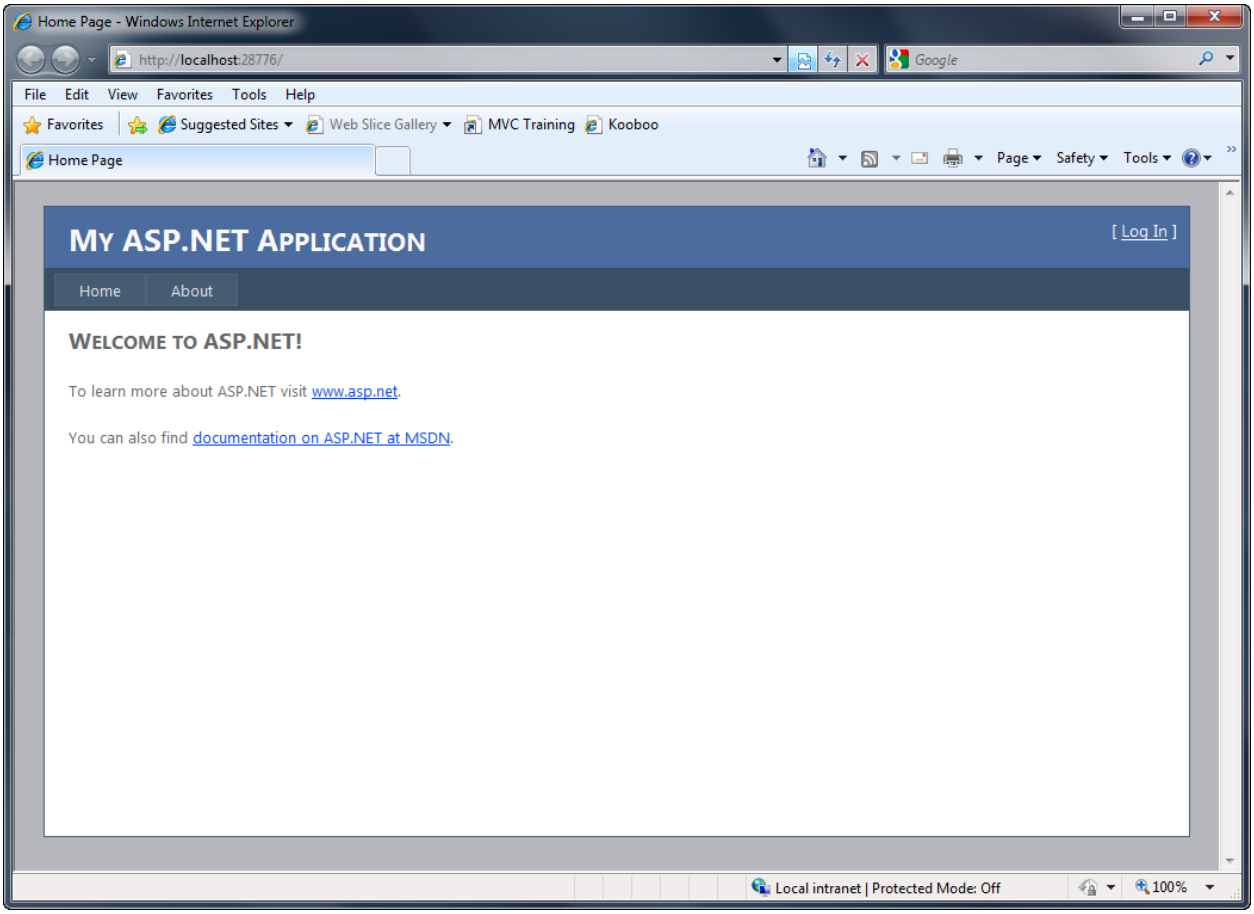
The Solution includes basic folder structure:



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.



Our first application enhancement will be to modify the default application template with the visual style that we desire for our application.

To accomplish this….

1. Replace the Site.css file from the default WebForms template with the Site.css file provided to us by “a designer”.

   After downloading the current Tailspin Spyworks source code from CodePlex.com and unzipping the code into a local directory, copy the \Styles\Site.css file from the completed project source into our working solution \Styles\Site.css.

2. Add to our solution the associated image files are used by the CSS classes in our new Site.css file.

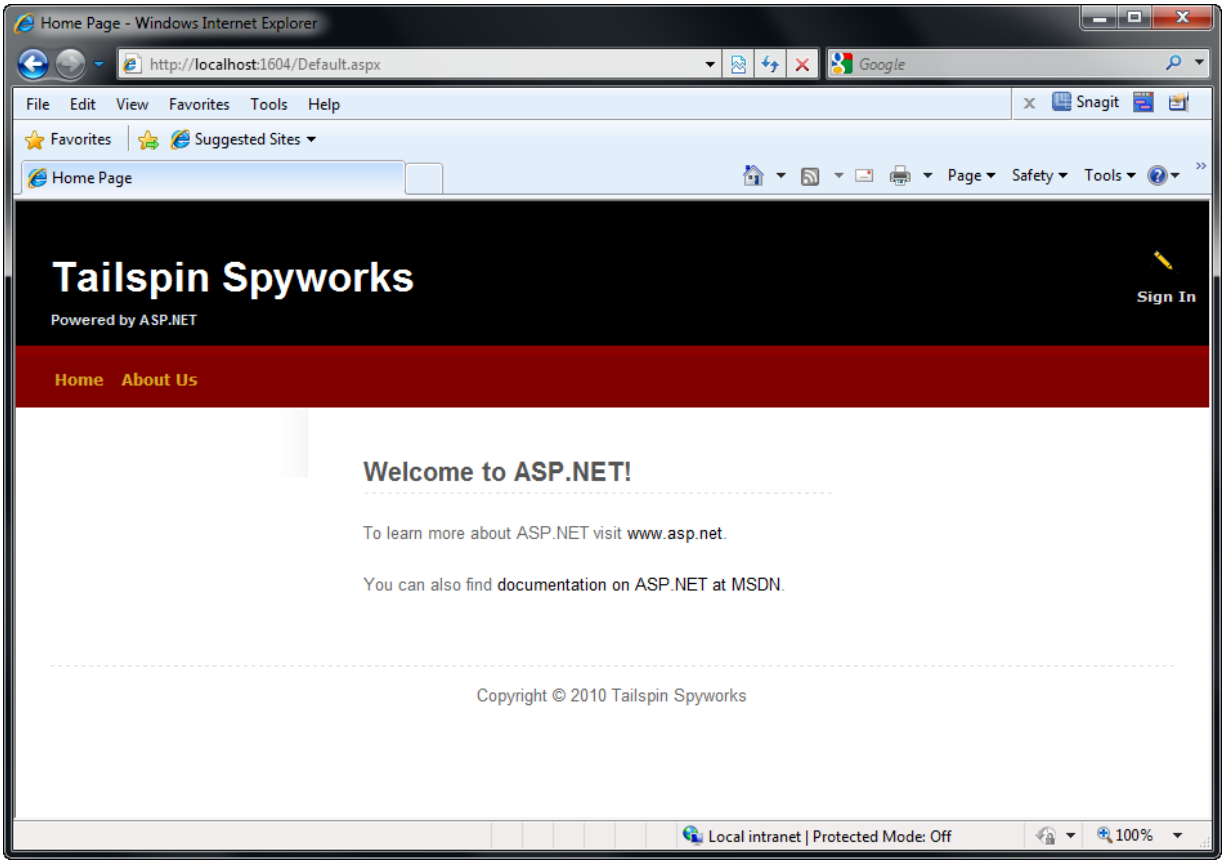
   Import the \Styles\Images folder from the completed project source into our working solution \Styles\

3. Copy some default html markup provided by “our designer” into the Site.Master page generated by Visual Studio from the default ASP.NET WebForms templates.

In our working solution, open the Site.Master page, select and delete all the markup. Using Windows NotePad (or a text editor of your choice) find and open the text file MasterPage-Template.txt. Then copy and paste the entire content of the text file into the Ste.Master file.

Though a detailed analysis of HTML and CSS is beyond the scope of this tutorial, you should examine the new CSS and HTML to understand the relationships between the CSS classes and their use in the MasterPage markup.

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



The content displayed has not changed but the page's style is quite different.

Before we proceed we'll 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.

Follow these steps…..

1. Move the Site.Master file into the \Styles folder.
2. Modify the first line of Default.aspx and About.aspx files as well as the .aspx files in the Account folder to resolve the MasterPage location.
   (`MasterPageFile="~/Styles/Site.master").`

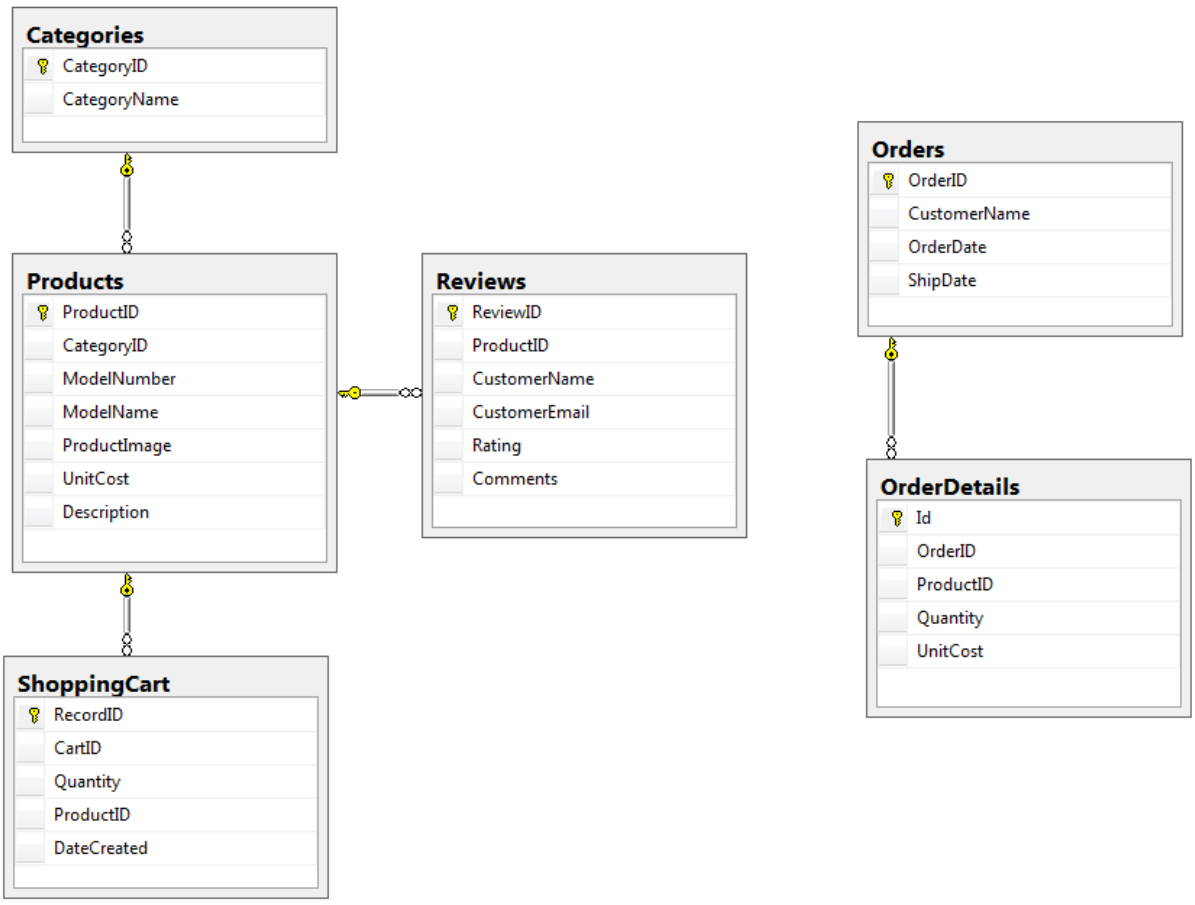
Adding the Data Access Layer

Our ecommerce application will depend on two databases.

For customer information we'll use the standard ASP.NET Membership database (which we'll generate later).

For our shopping cart and product catalog we'll implement a SQL Express database.

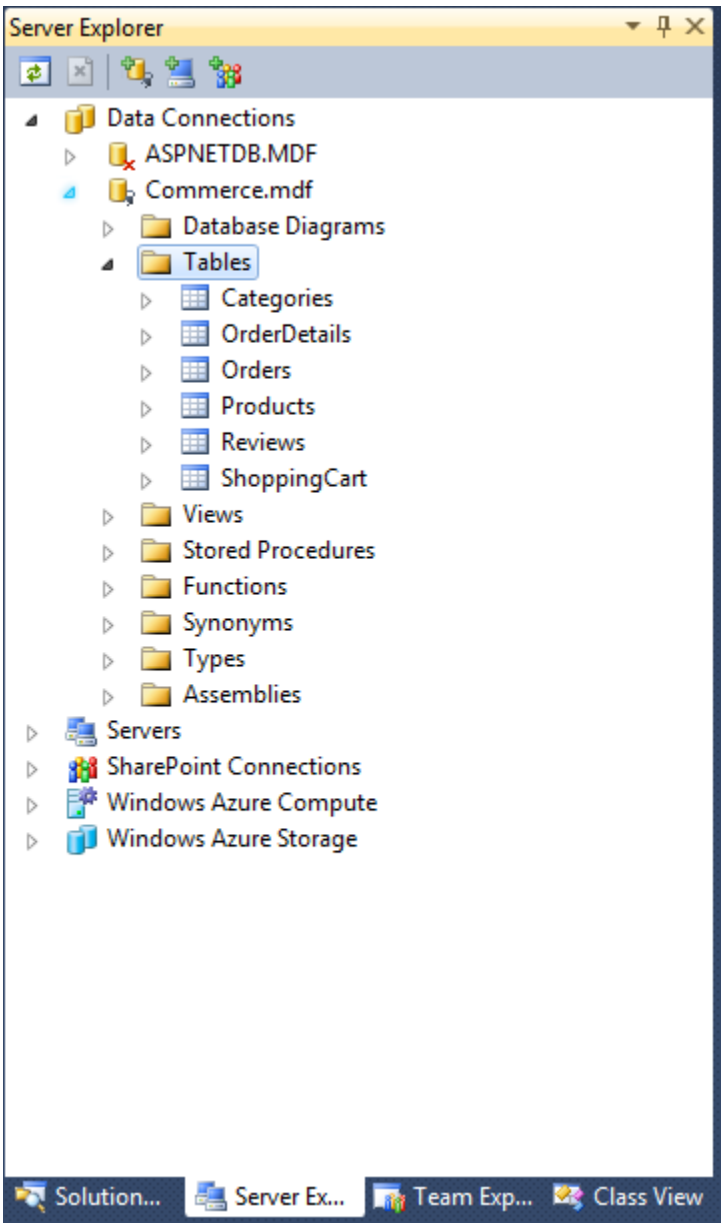
The overall schema is as follows. (See the Commerce Database Appendix for a detailed report of the Commerce Database Implementation.)



Database creation, organization, and Stored Procedure development in T-SQL is beyond the scope of this tutorial. As one will often do in the "real world" we will be programming against an existing database and using the Tables, Views, and Stored Procedures that exist in that database from our application code.

If you would like to try creating the database from scratch you can use the detailed definition report that describes the Commerce database (appendix below).

Visual Studio has a database manipulation tool that you can start using from the Solution Explorer.

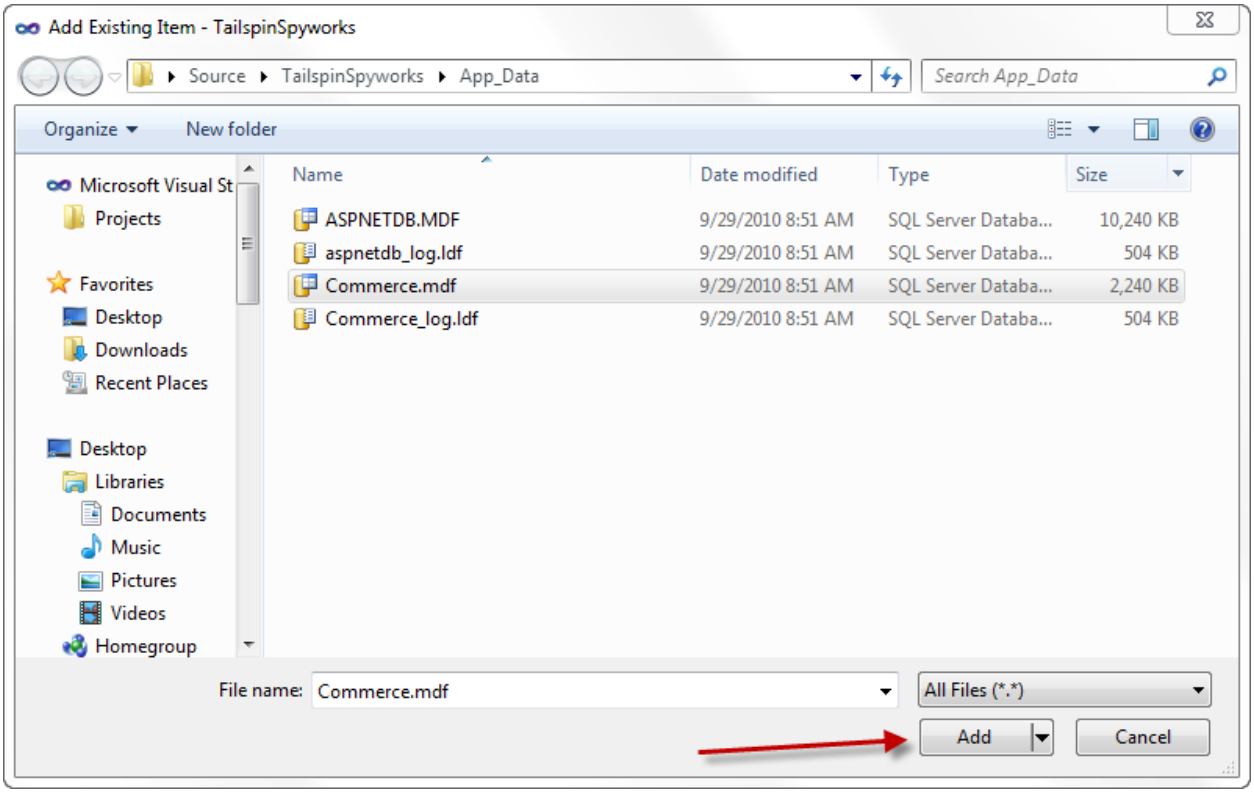


Another alternative is to download the SQL Server Management Studio Express from the following web site. http://www.microsoft.com/express/Database/

We could program against any SQL Server database that we had access to and permissions for, but in our case we'll be using SQL Server Express and so we will store the database's physical repository in our solution's App_Data older.

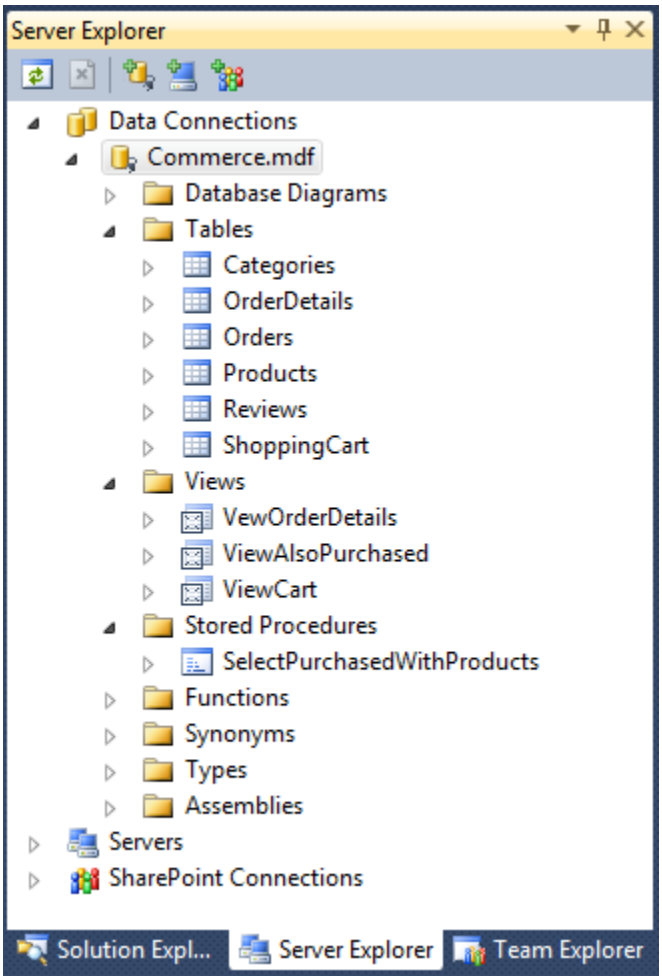
If the App_Data folder does not exist you can create it by Right-Clicking on the solution name in Visual Studio's Solution Explorer -> **Select Add -> Add ASP.NET Folder -> App_Data**

Next, Right-Click on the App_Data folder select **Add -> Existing Item**. In the Open File Dialog navigate to the Tailspin Spyworks source folder (expanded form the .zip file you downloaded) and highlight the Commerce.mdf file in the App_Data folder. Then Click **Add**.



Now double-click on the Commerce.mdf file in the Solution Explorer.

Visual Studio will create a Data Connection to the Commerce Database. You can now explore the Commerce Database in Visual Studio's Server Explorer Window.

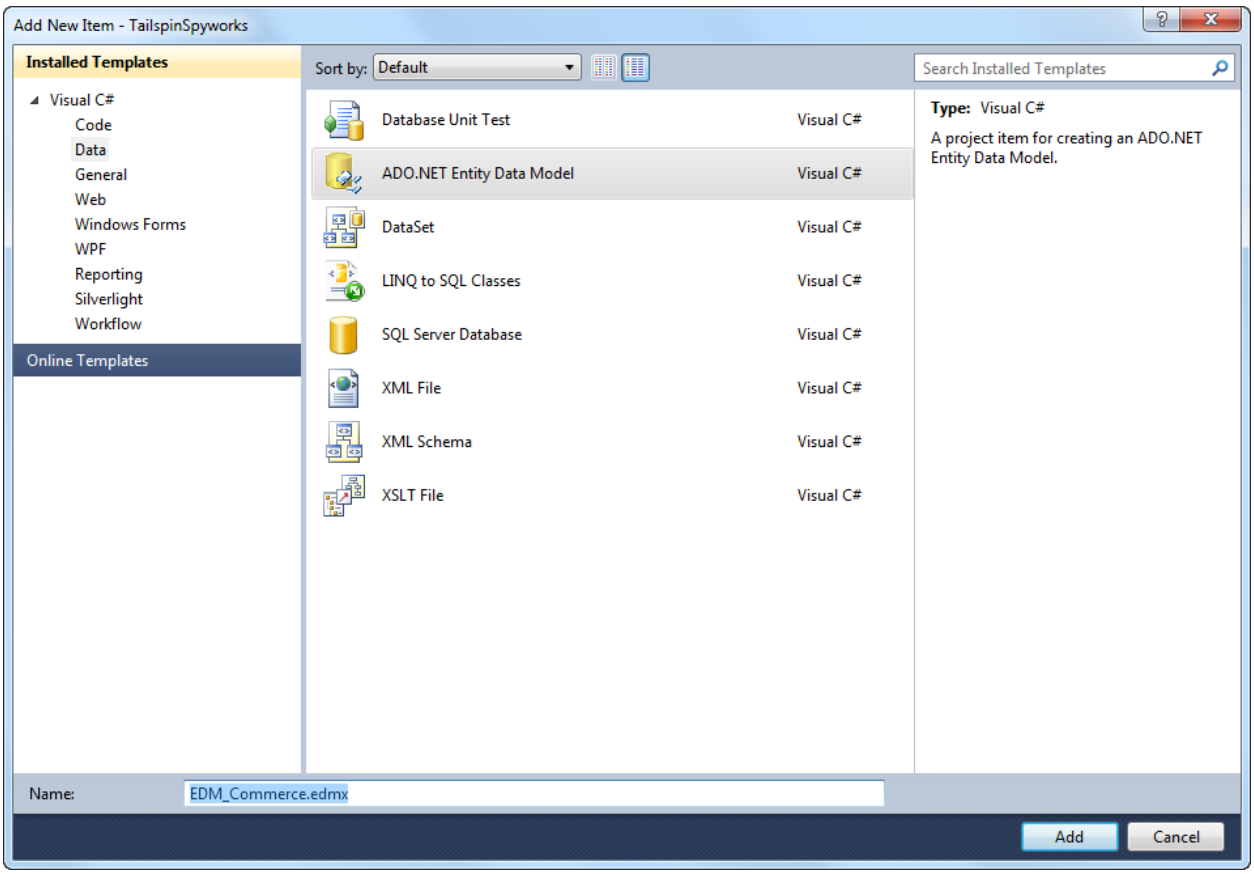


Having added the database (Commerce.mdf) in the application's App_Data folder we can proceed to create our Data Access Layer using the .NET Entity Framework.

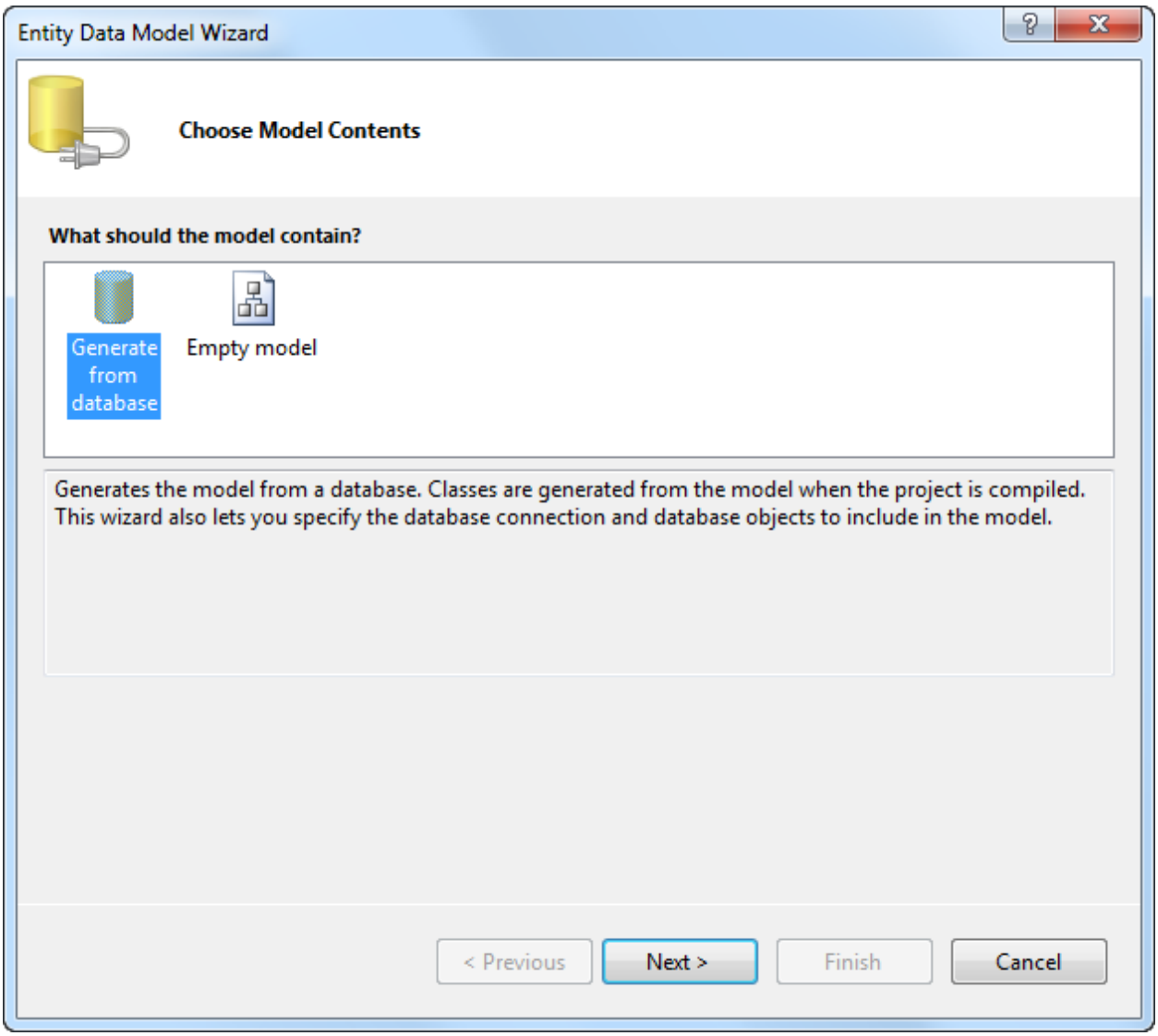
We'll create a folder named "Data_Access" in our solution. (Right-Click the Solution Name -> Select New Folder.)

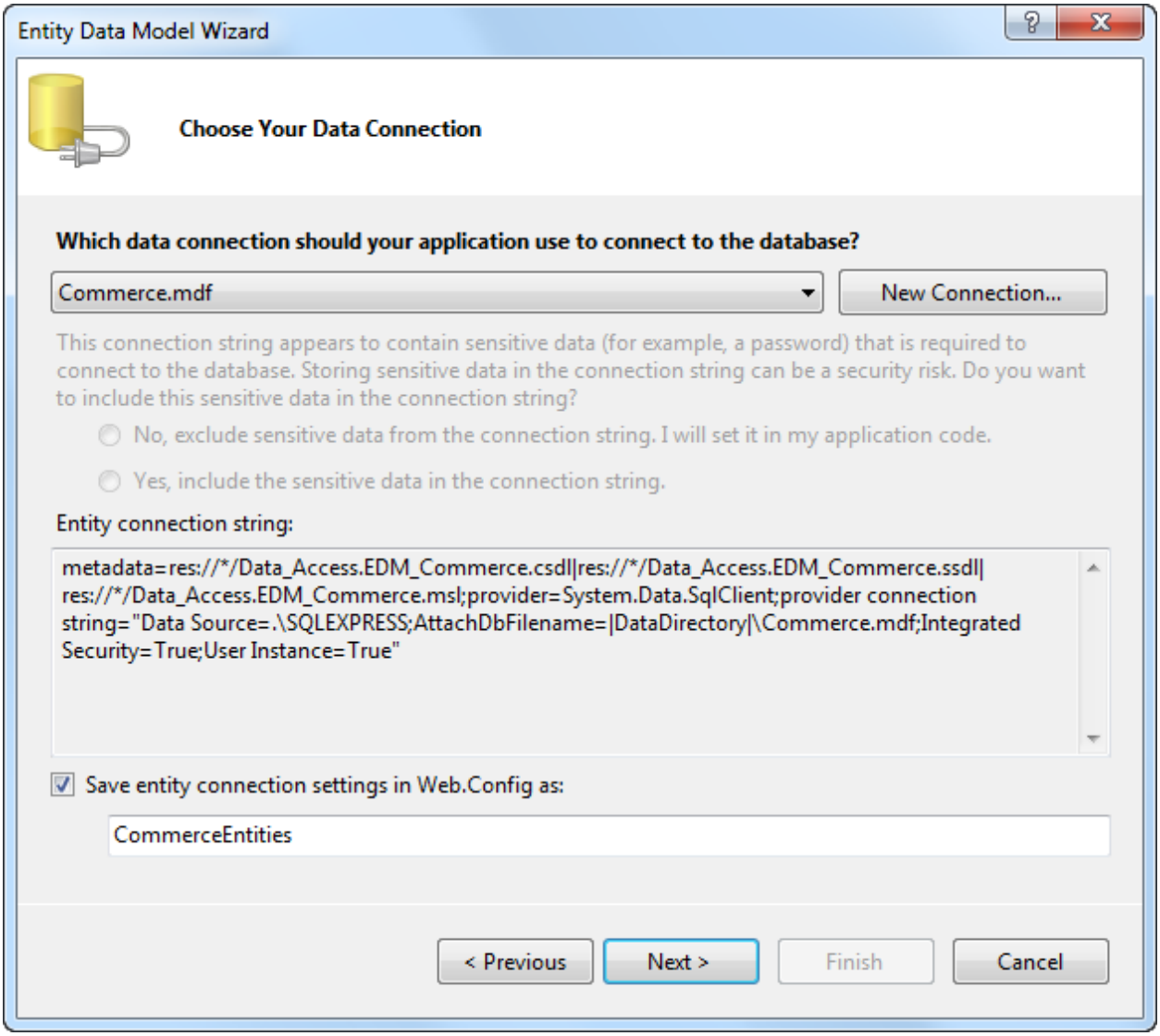
Then right click on that folder and select "Add New Item".

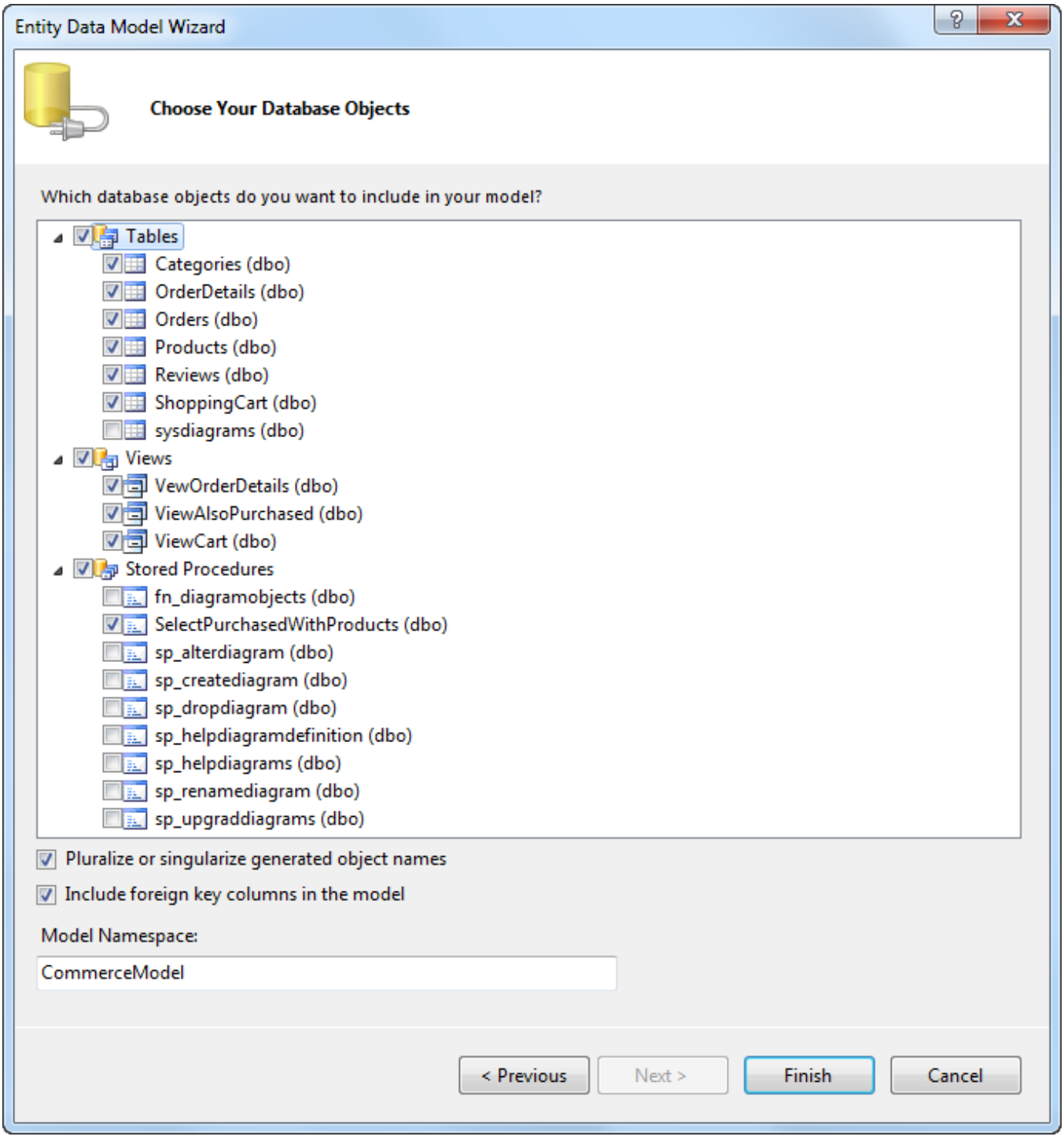
In the "Installed Templates" item and then select "ADO.NET Entity Data Model" enter EDM_Commerce.edmx as the name and click the "Add" button.

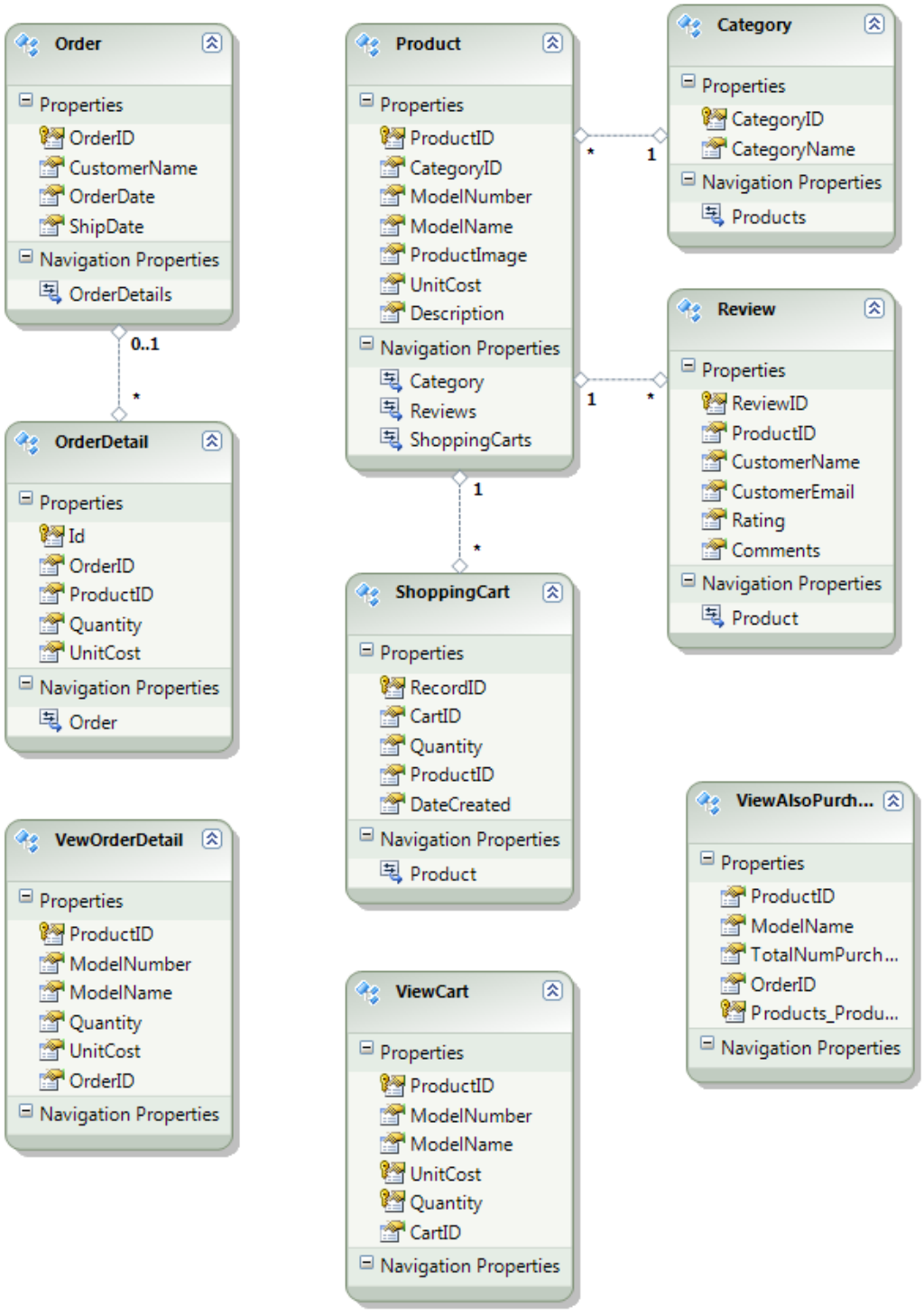


Choose “Generate from Database”.









Note that you may see a number of messages like this.

```
Message 1 The table/view 'C:\USERS\WILD WILLY\DESKTOP\TAILSPINSPYWORKS-V0.9\MY
    TAILSPIN\TAILSPINSPYWORKS\TAILSPINSPYWORKS\APP_DATA\COMMERCE.MDF.dbo.VewOrderDetails' does not
    have a primary key defined. The key has been inferred and the definition was created as a read-only table/view.
        C:\Users\Wild Willy\Desktop\TailspinSpyworks-v0.9\my
    !tailspin\TailspinSpyworks\TailspinSpyworks\Data_Access\EDM_Commerce.edmx 0 1
        TailspinSpyworks
```

This is normal as some of the views do not have a primary key.

Save and build your solution.

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.

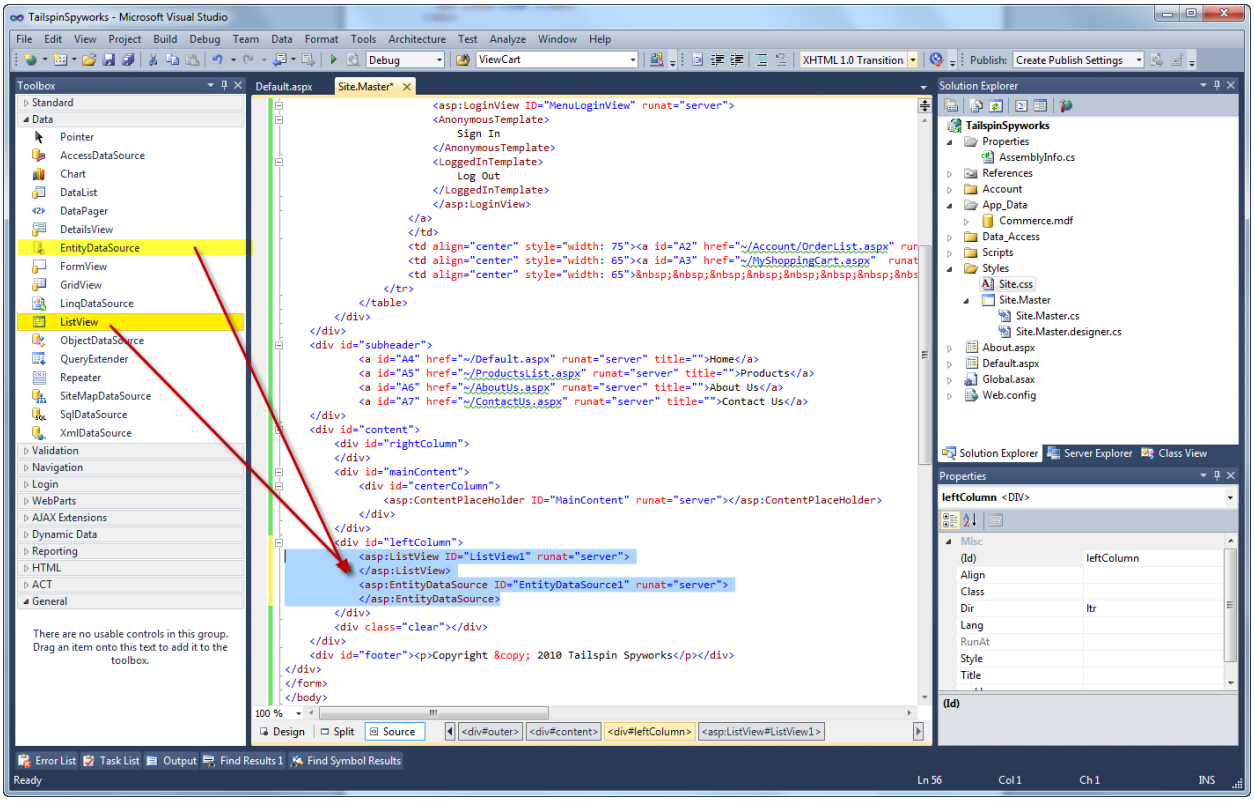
```
<div id="content">
  <div id="rightColumn"></div>
  <div id="mainContent">
    <div id="centerColumn">
       <asp:ContentPlaceHolder ID="MainContent" runat="server"></asp:ContentPlaceHolder>
    </div>
  </div>
  <div id="leftColumn">
  <!—Our menu will go here. →
  </div>
  <div class="clear">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

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

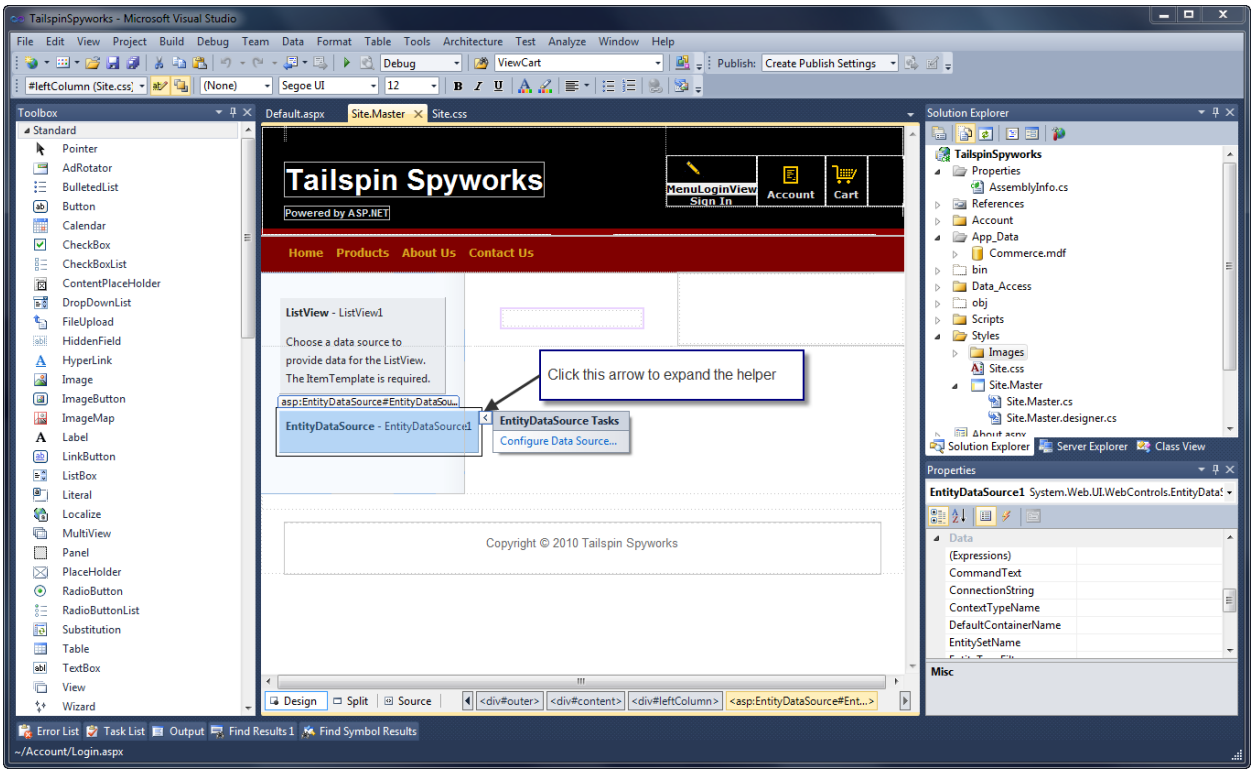
```
#leftColumn
{
       position: relative;
       float: left;
       width: 14em;
       padding: 2em 1.5em 2em;
       background: #fff url('images/a1.gif') repeat-y right top;
       top: 1px;
       left: 0px;
       height: 100%;
}
```

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.

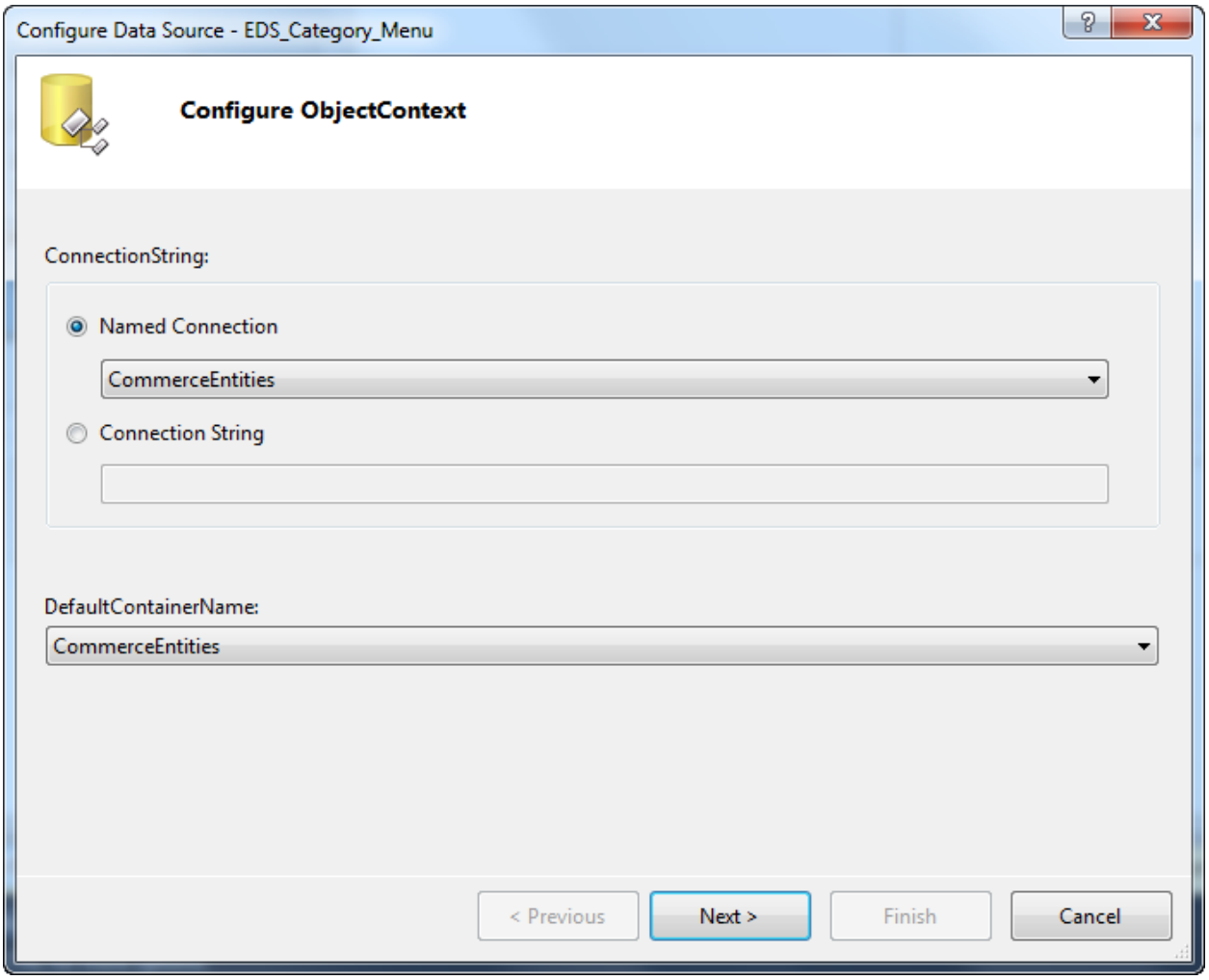
To accomplish this we will use two of ASP.NET's powerful data controls. The "Entity Data Source" control and the "ListView" control.



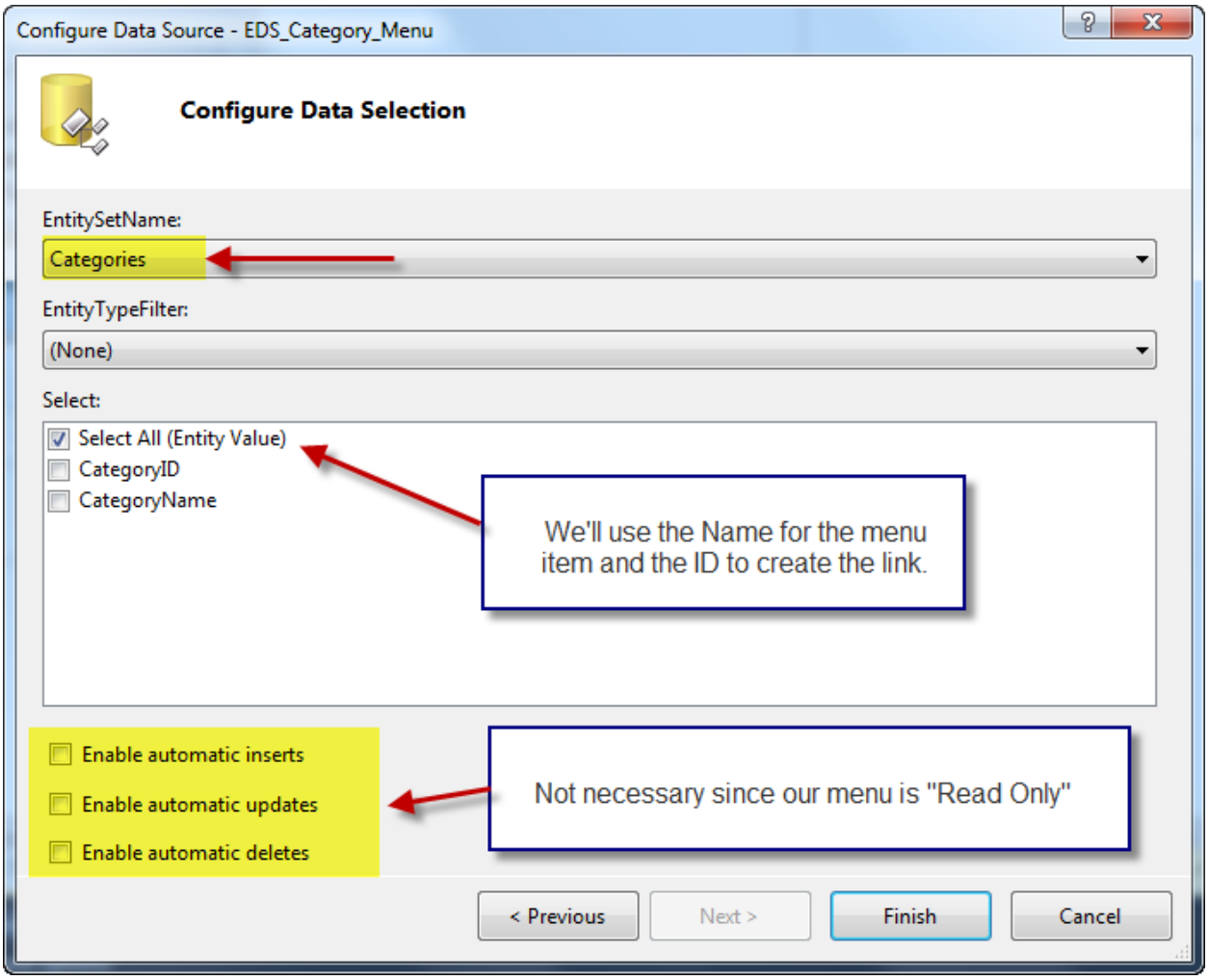
Let's switch to "Design View" and use the helpers to configure our controls.



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

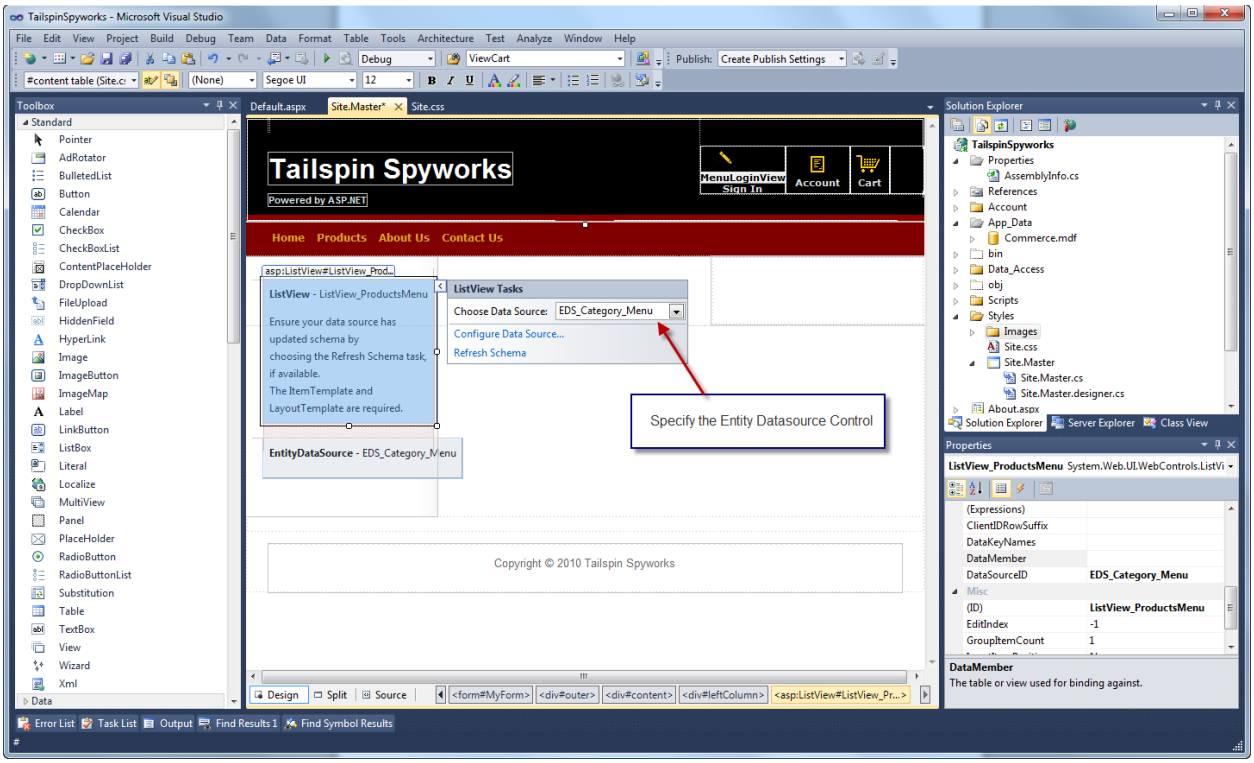


Select the CommerceEntities Connection that was created for us when we created the Entity Data Source Model for our Commerce Database and click “Next”.



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.



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.

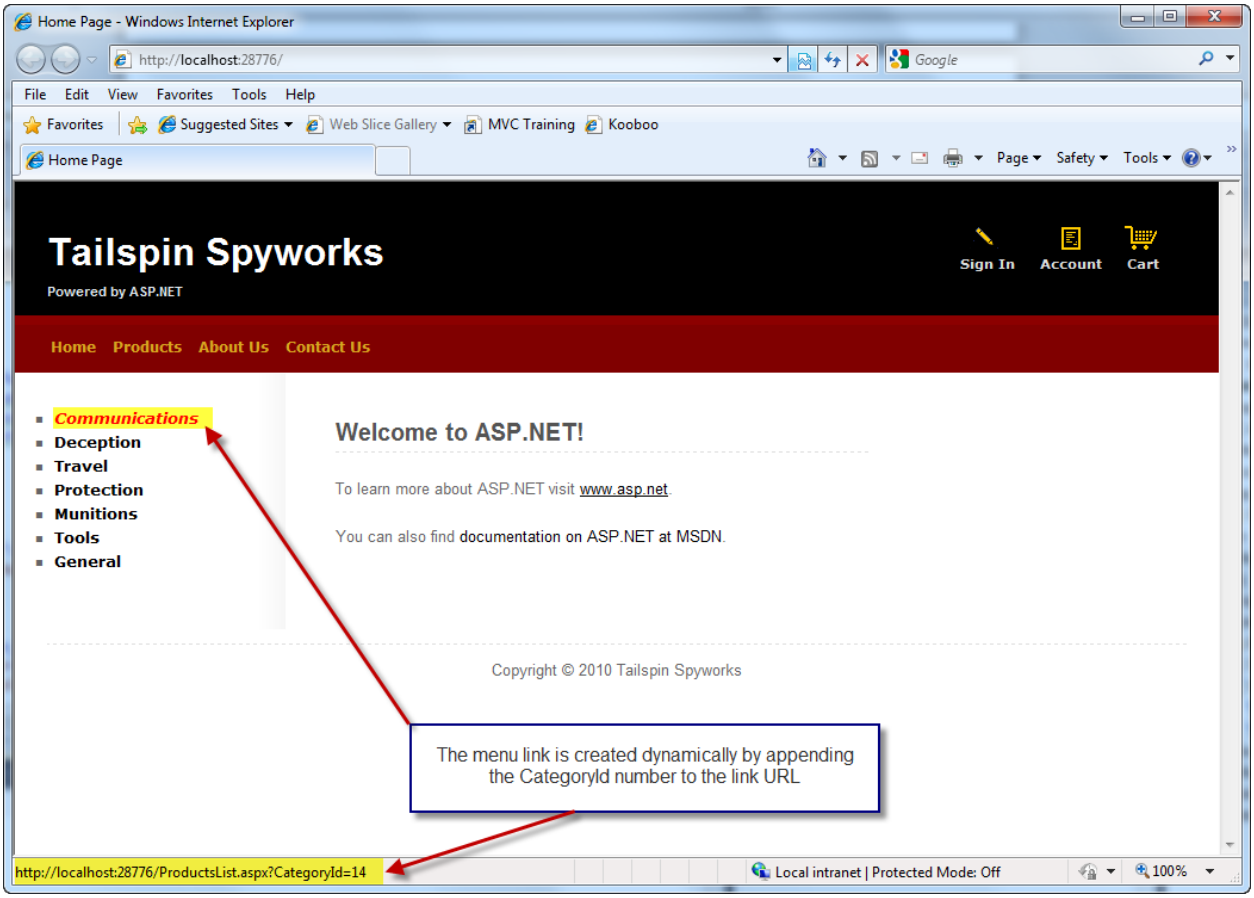
```
<asp:ListView ID="ListView_ProductsMenu" runat="server" DataKeyNames="CategoryID"
              DataSourceID="EDS_Category_Menu">
   <EmptyDataTemplate>No Menu Items.</EmptyDataTemplate>
   <ItemSeparatorTemplate></ItemSeparatorTemplate>
   <ItemTemplate>
      <li>
         <a href='<%# VirtualPathUtility.ToAbsolute("~/ProductsList.aspx?CategoryId=" +
                                 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.

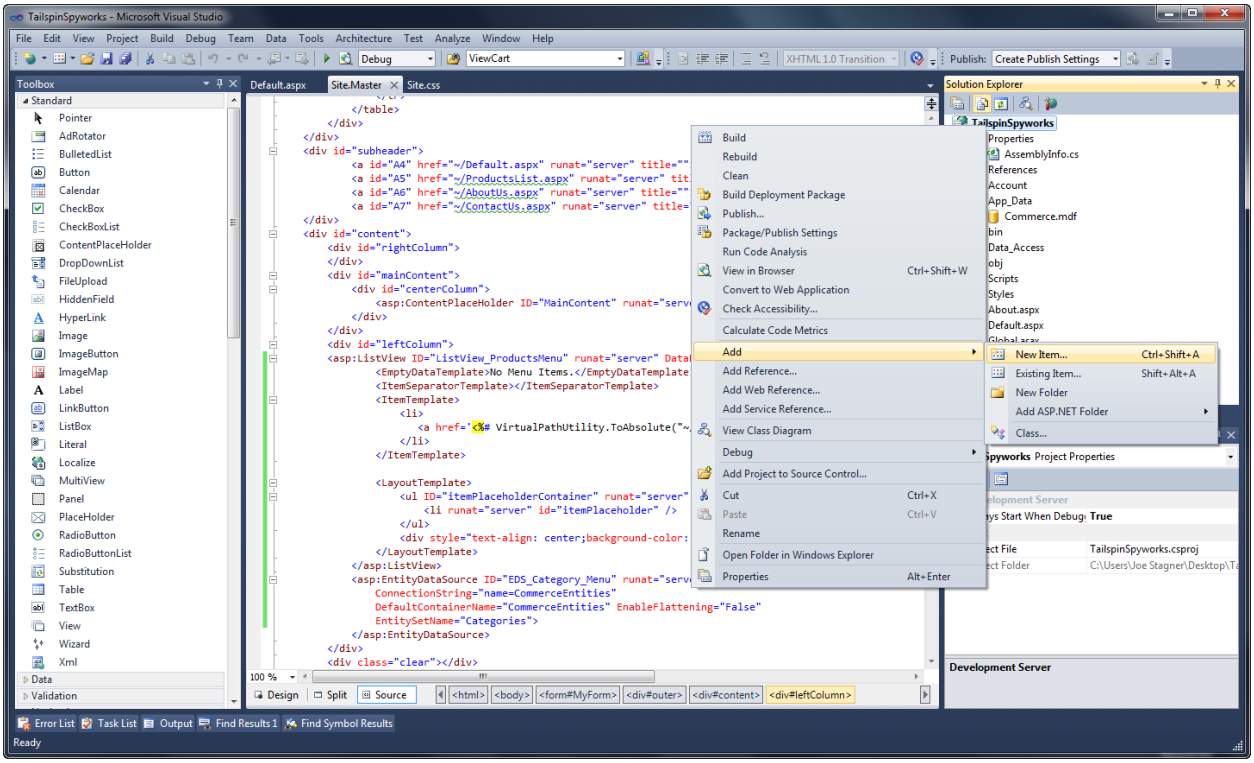
Let's run the application now.



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.

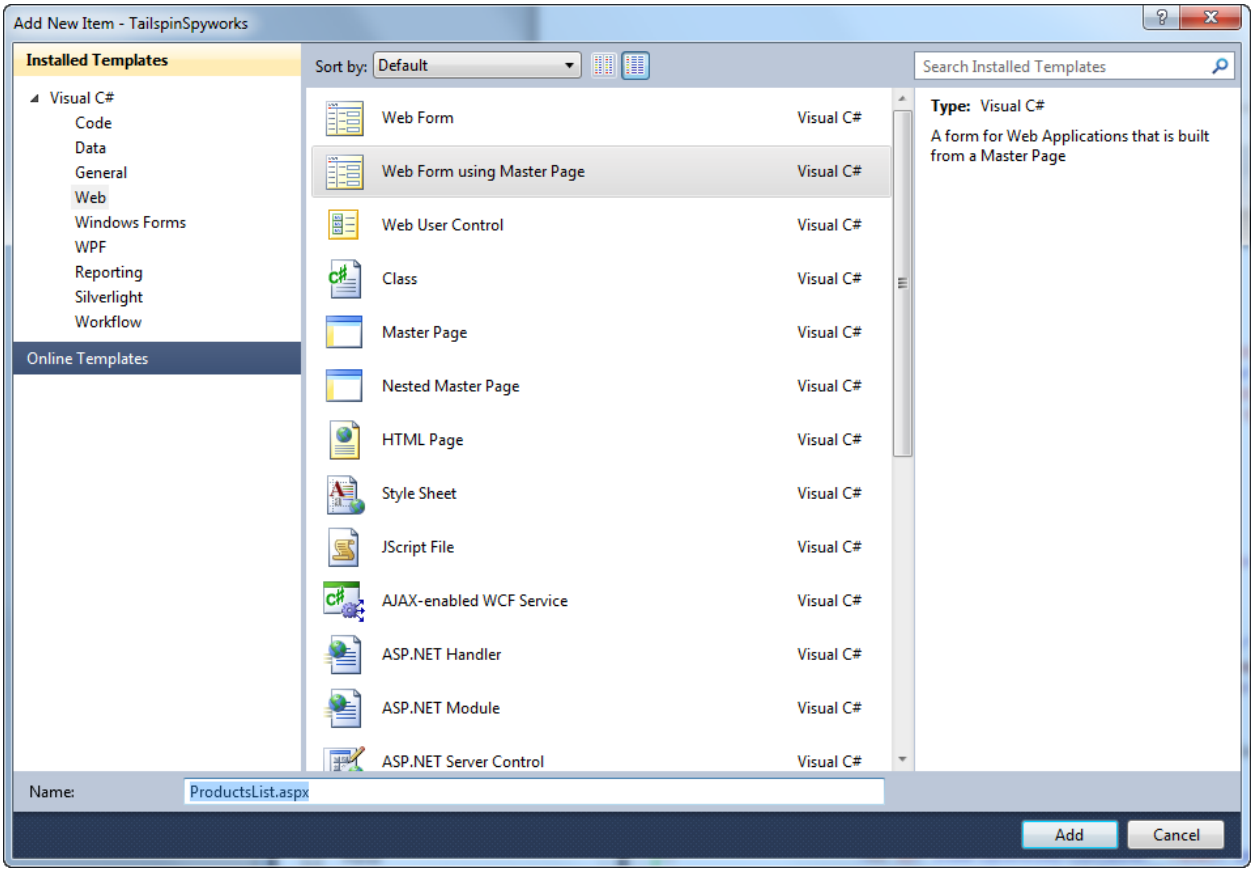
Listing Products with the GridView Control

Let's begin implementing our ProductsList.aspx page by "Right Clicking" on our solution and selecting "Add" and "New Item".

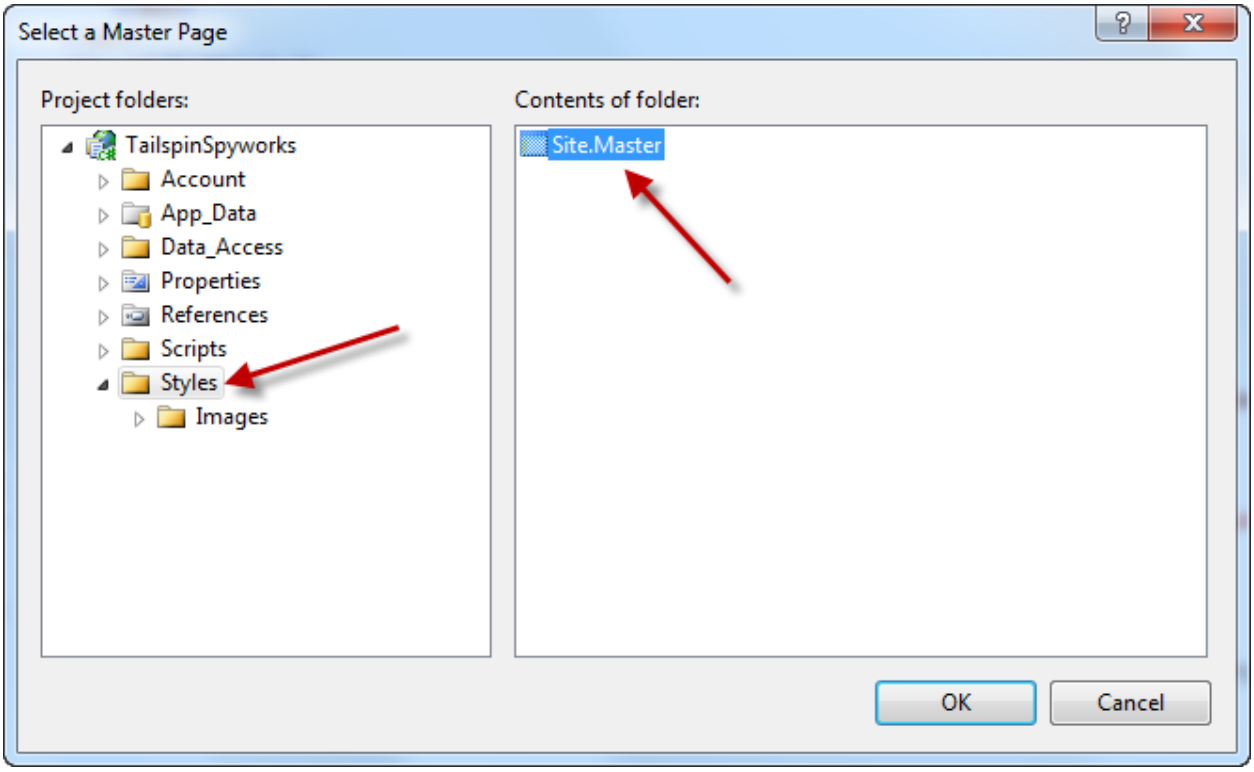


Choose “Web Form Using Master Page” and enter a page name of ProductsList.aspx”.

Click “Add”.

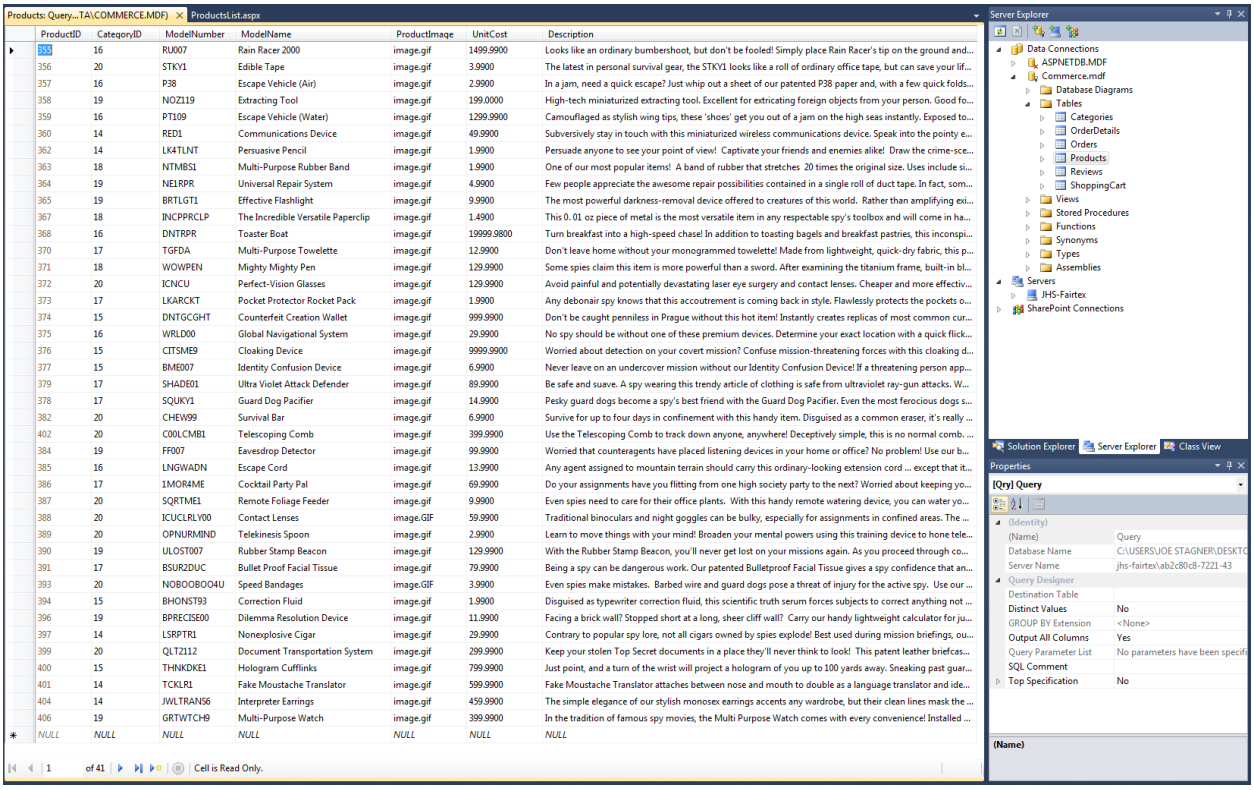


Next choose the “Styles” folder where we placed the Site.Master page and select it from the “Contents of folder” window.



Click “Ok” to create the page.

Our database is populated with product data as seen below.



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="CategoryId"
                                        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.

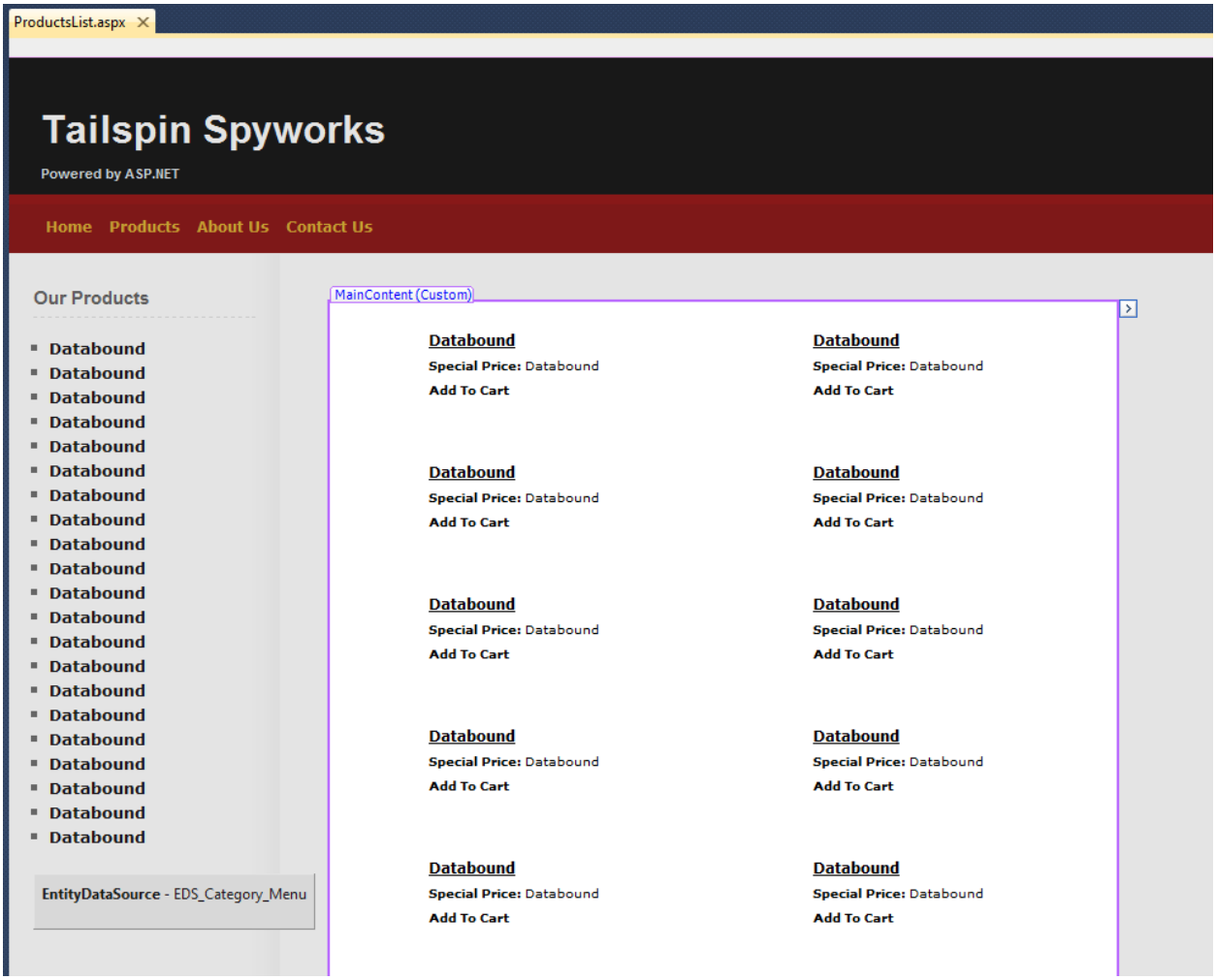
- The product name will be a link to the product's detail view.
- The product's price will be displayed.
- An image of the product will be displayed and we'll dynamically select the image from a catalog images directory in our application.
- We will include a link to immediately add the specific product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

Here is the markup for our ListView control instance which is inserted inside the "MainContent" asp:Content control.

```
<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;"> </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>  
          </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></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>
```

In design view our page will look like this.



We are dynamically building several links for each displayed product.

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

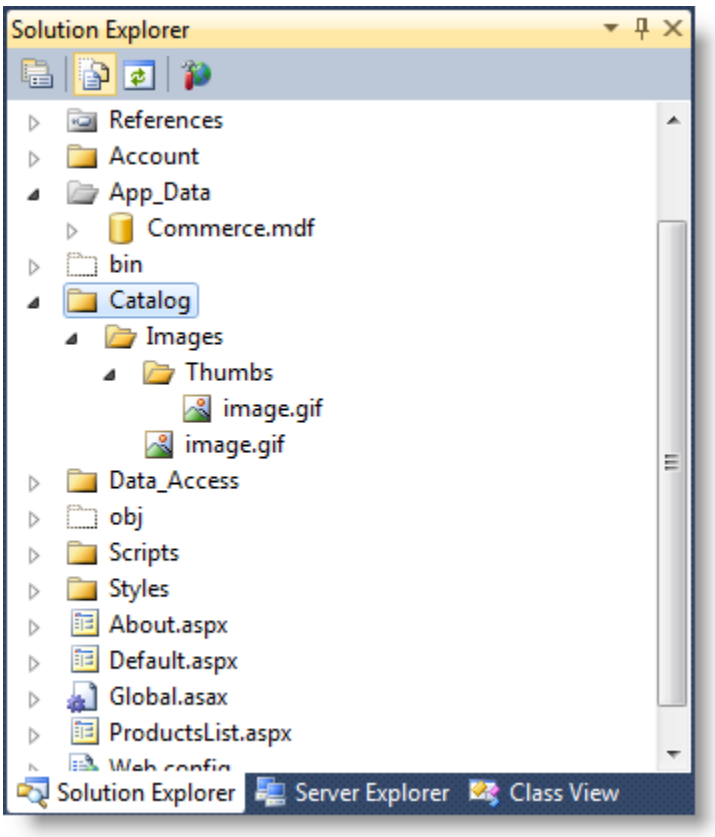
In your solution, create the following directories:

\Catalog

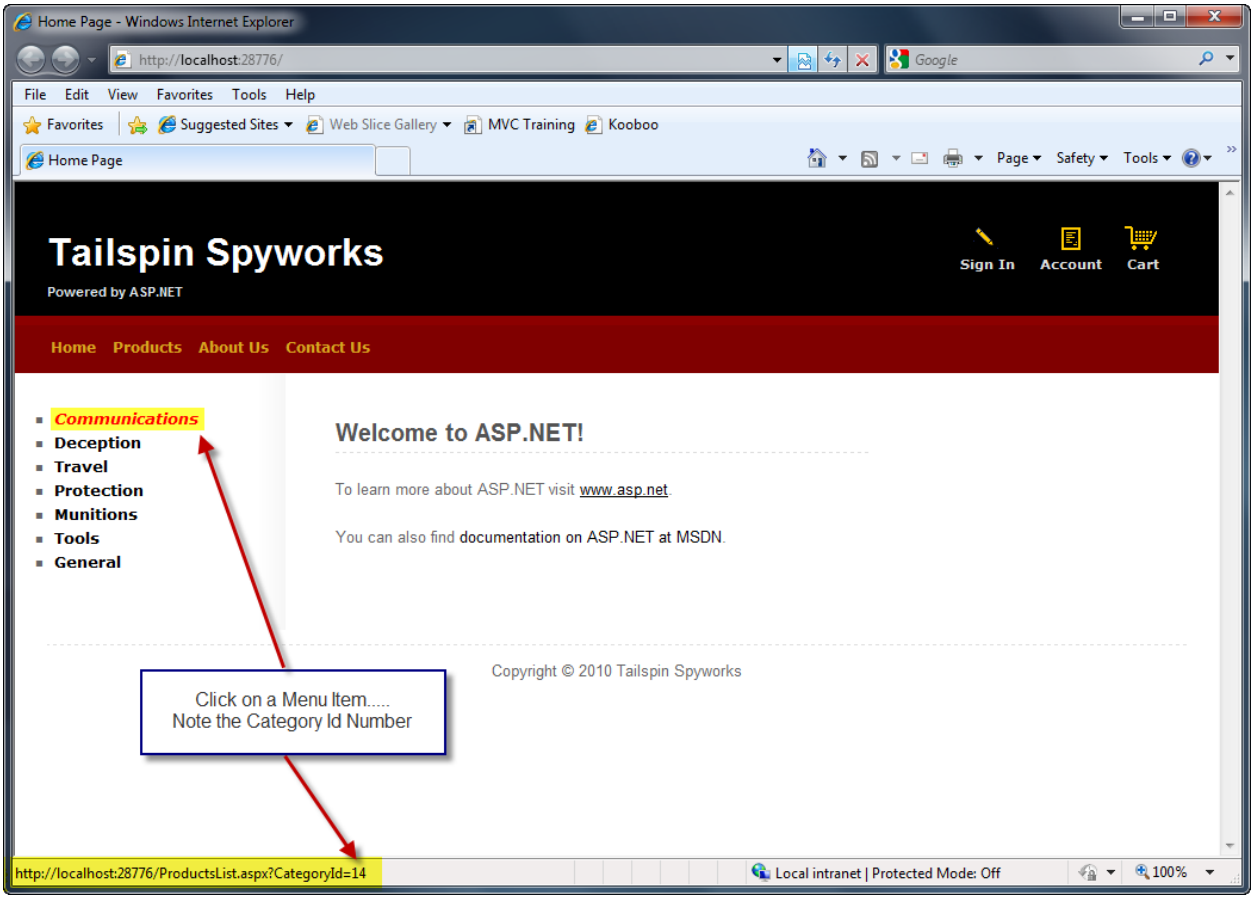
\Catalog\Images

\Catalog\Images\Thumbs

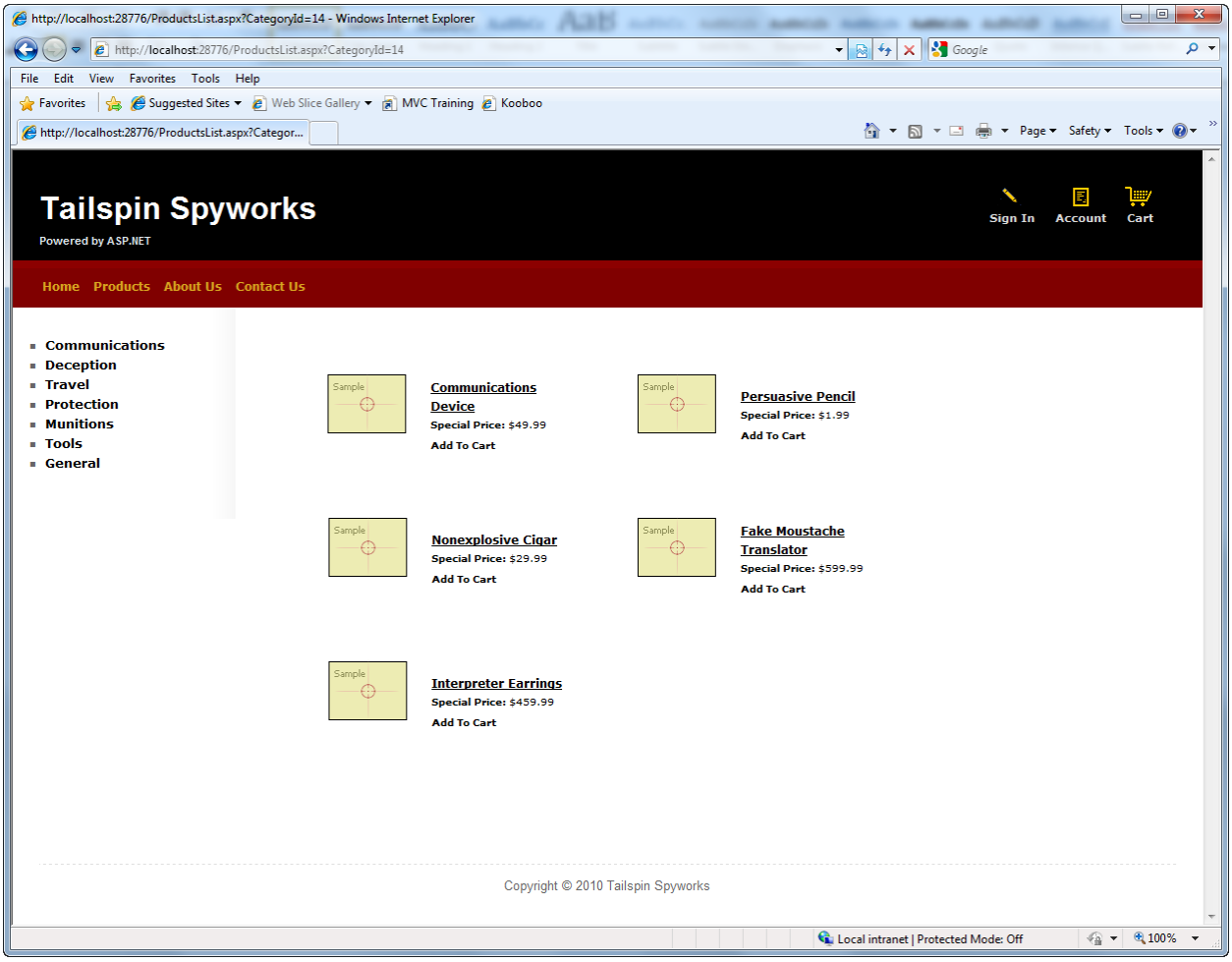
The structure will look like this in the Solution Explorer.



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



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



Now we need to implement the ProductDetails.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 inside the "MainContent" asp:Content control of the page 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> <%# Eval("UnitCost", "{0:c}")%>
    </span><<br />
    <span class="ModelNumber">
      <b>Model Number:</b> <%# 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 slect a name that will avoid the conflict.

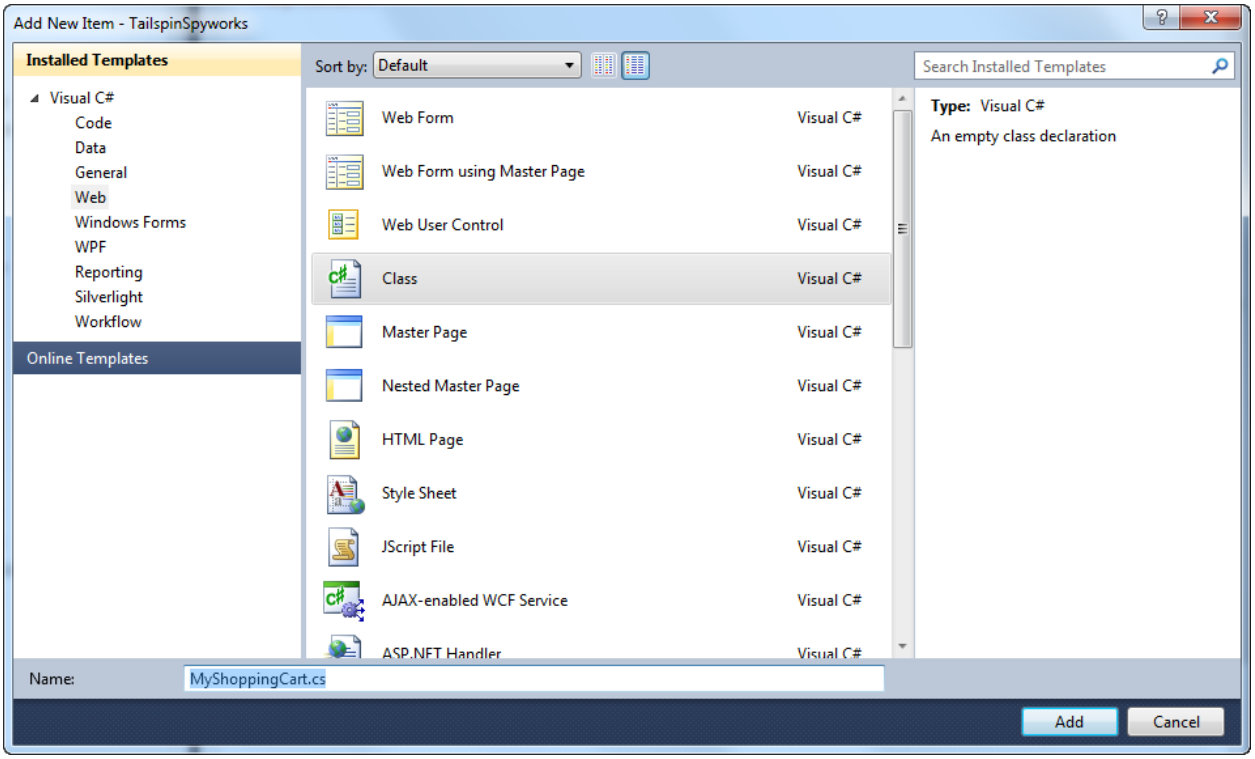
It's also worth noting that we will be creating a simple shopping cart and embedding the shopping cart logic with the shopping cart display. We might also choose to implement our shopping cart in a completely separate Business Layer.

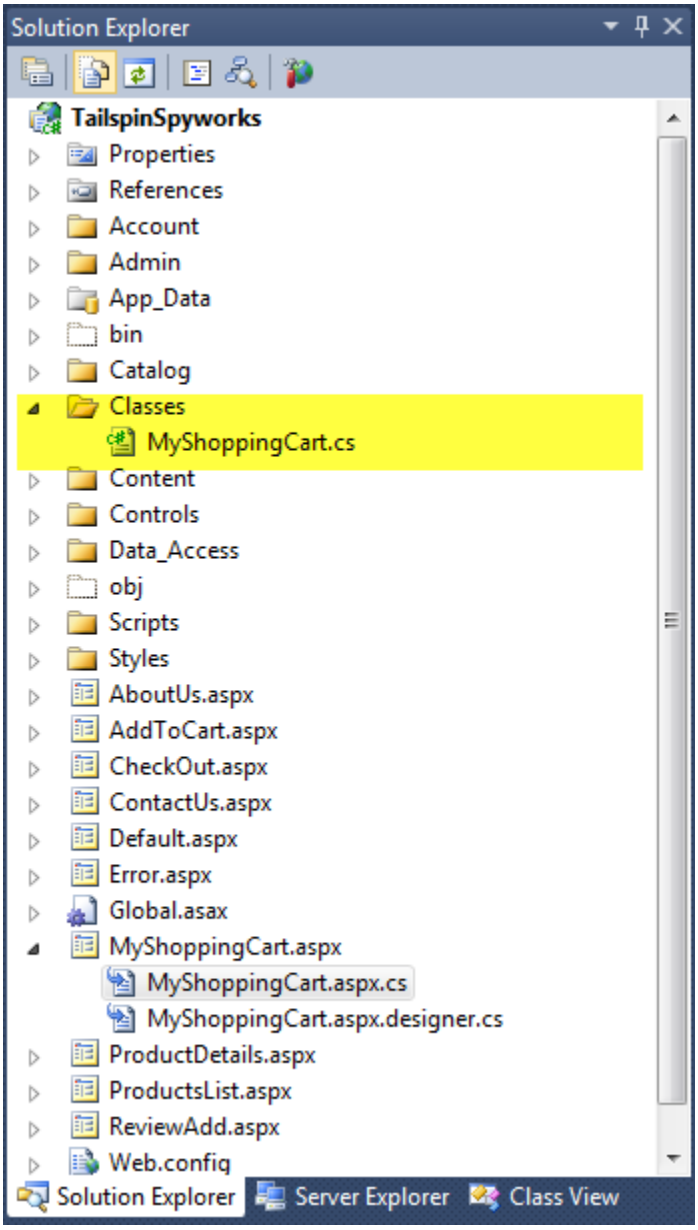
Adding Some Business Logic

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





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.cf 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 and deleting the .Classes expension to the namespace.

Our new class file now looks like this.

```
namespace TailspinSpyworks
{
    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 then 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;`)

Is 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 names "GetShoppingCartId".

In the MyShoppingCart class, 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.

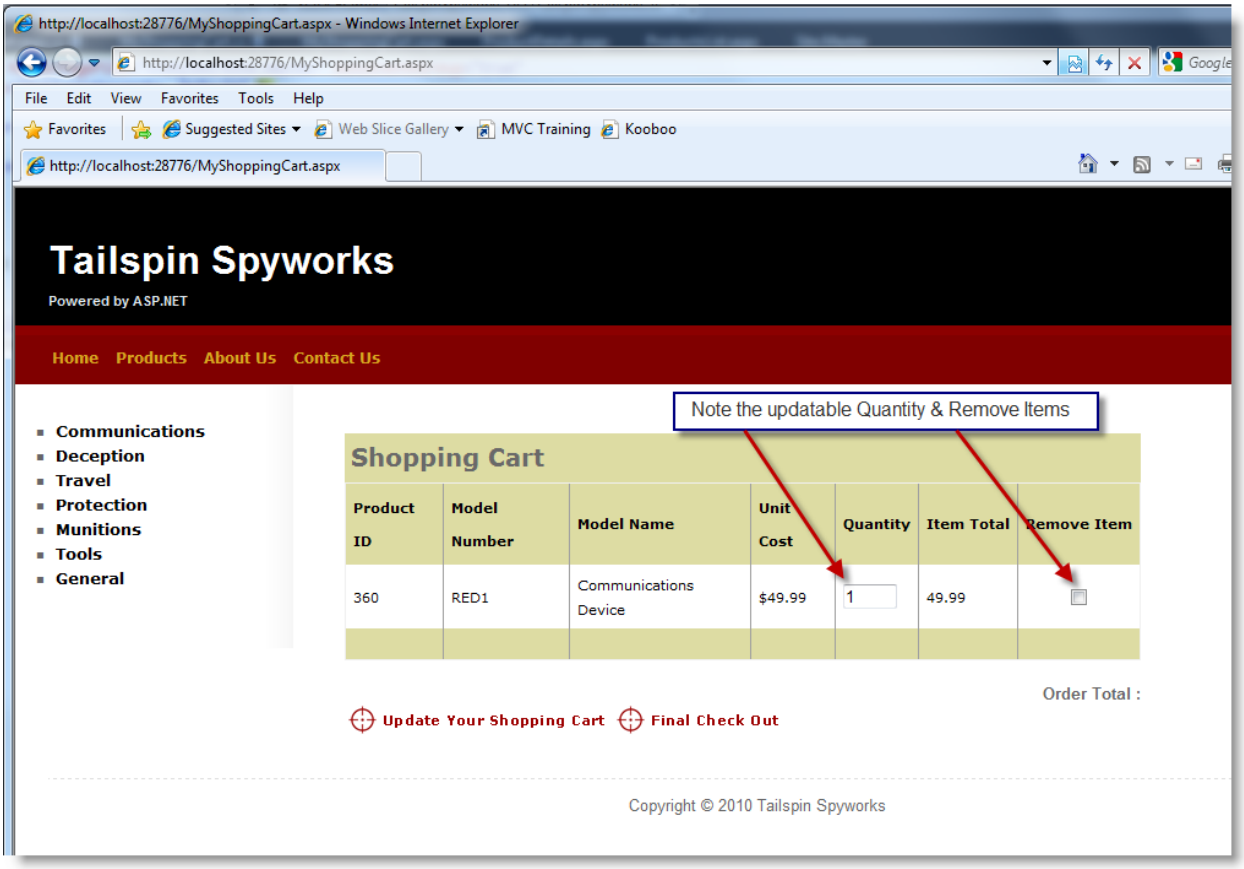
```
<div id="ShoppingCartTitle" runat="server" class="ContentHead">Shopping Cart</div>
<asp:GridView ID="MyList" runat="server" AutoGenerateColumns="False" ShowFooter="True"
                          GridLines="Vertical" CellPadding="4"
                          DataSourceID="EDS_Cart"
                          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 Total</HeaderTemplate>
      <ItemTemplate>
        <%# (Convert.ToDouble(Eval("Quantity")) *
             Convert.ToDouble(Eval("UnitCost")))%>
      </ItemTemplate>
    </asp:TemplateField>
    <asp:TemplateField>
    <HeaderTemplate>Remove 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 that we have deviated from the "default" grid display by implementing three custom columns.

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

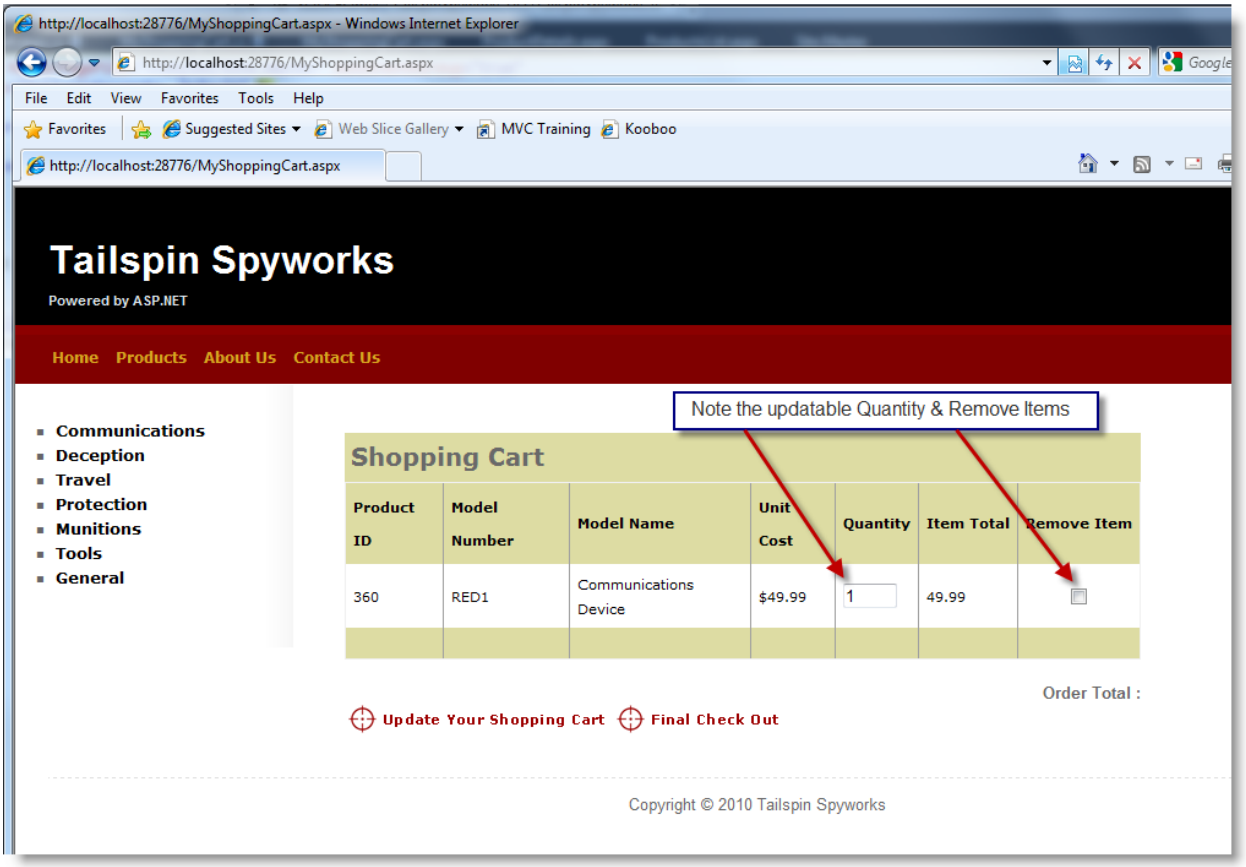
```
<asp:TemplateField>
  <HeaderTemplate>Quantity</HeaderTemplate>
  <ItemTemplate>
     <asp:TextBox ID="PurchaseQuantity" Width="40" runat="server"
                  Text='<%# Bind("Quantity") %>'></asp:TextBox>
  </ItemTemplate>
</asp:TemplateField>
```

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



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.

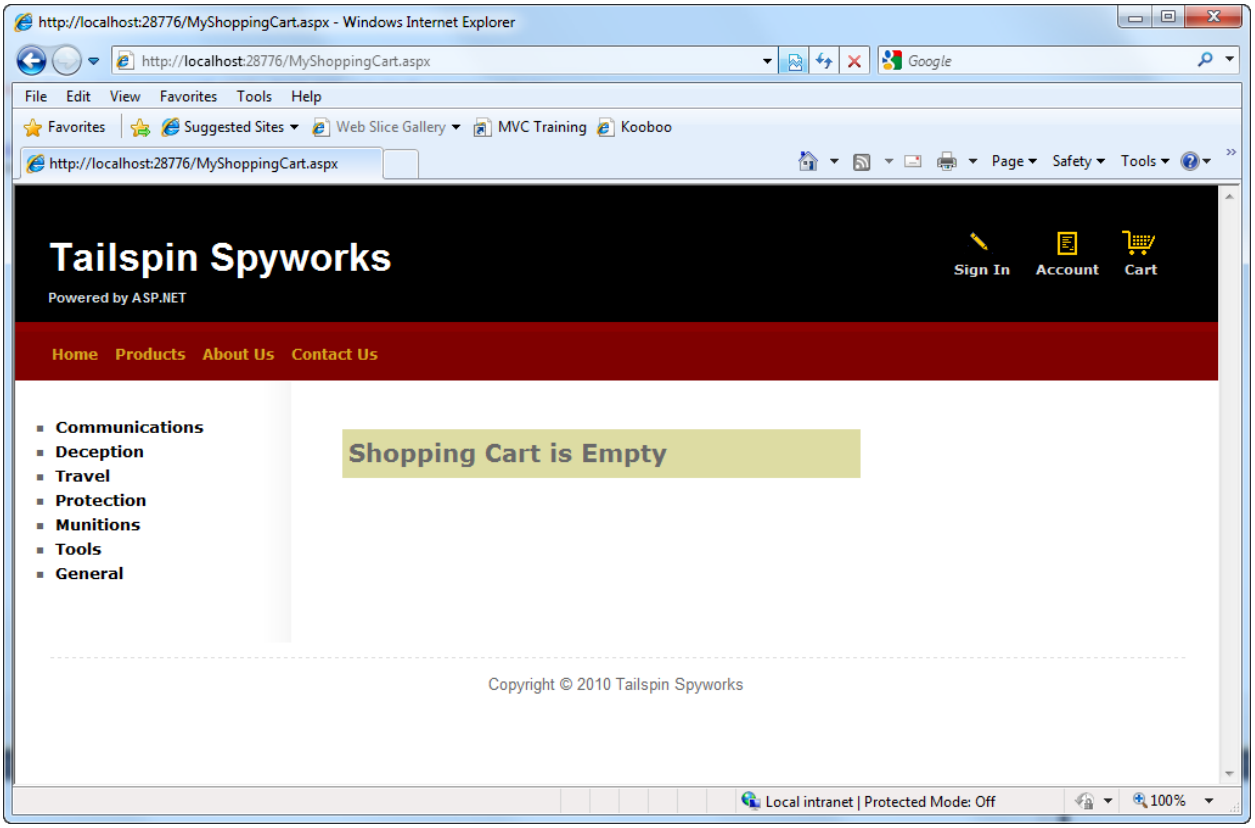
```
//---------------------------------------------------------------------------------+
public decimal GetTotal(string cartID)
{
  using (CommerceEntities db = new CommerceEntities())
    {
    decimal cartTotal = 0;
    try
```

```
      {
      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);
     }
   }
```

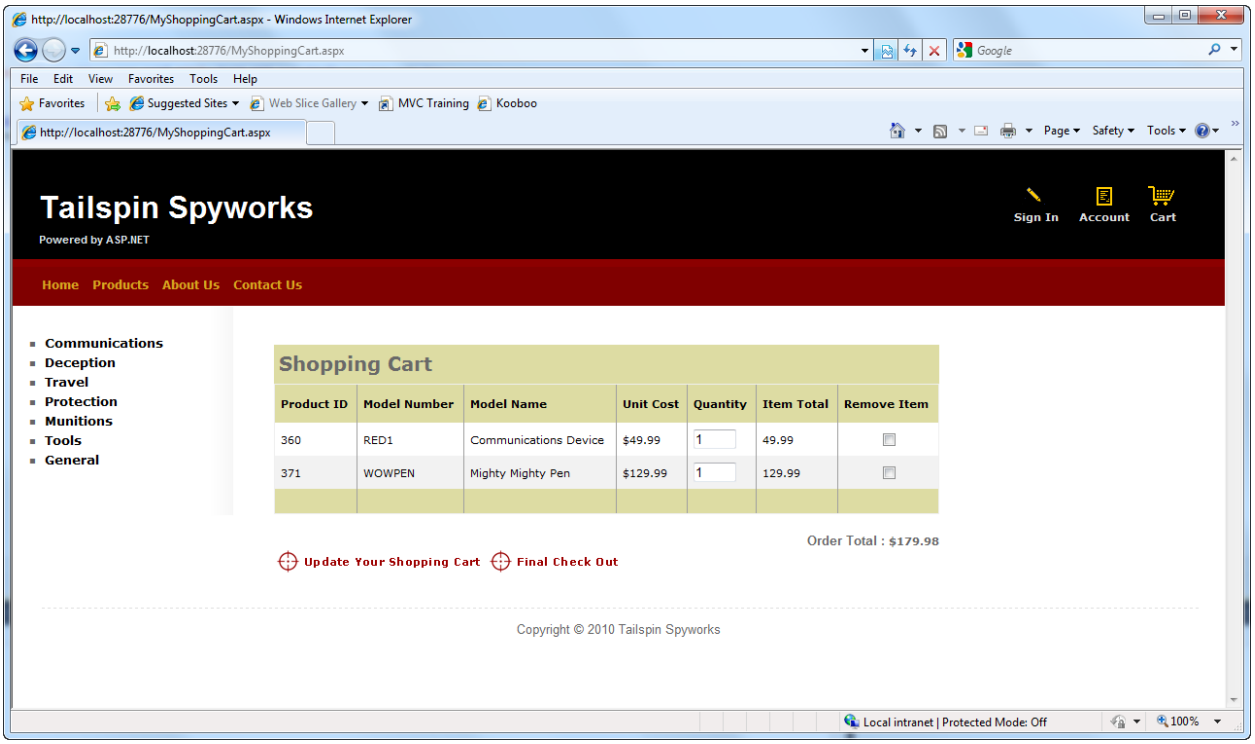
Then in the Page_Load event handler we'll can call our GetTotal method. At the same time we'll add a test to see if the shopping cart is empty and adjust the display accordingly if it is.

```
protected void Page_Load(object sender, EventArgs e)
{
   MyShoppingCart usersShoppingCart = new MyShoppingCart();
   String cartId = usersShoppingCart.GetShoppingCartId();
   decimal cartTotal = 0;
   cartTotal = usersShoppingCart.GetTotal(cartId);
   if (cartTotal > 0)
      {
      lblTotal.Text = String.Format("{0:c}", usersShoppingCart.GetTotal(cartId));
      }
   else
      {
      LabelTotalText.Text = "";
      lblTotal.Text = "";
      ShoppingCartTitle.InnerText = "Shopping Cart is Empty";
      UpdateBtn.Visible = false;
      CheckoutBtn.Visible = false;
      }
}
```

Now if the shopping cart is empty we get this:



And if not, we see our total.



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 ad a method to handle the circumstance when a user simply changes the quality 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].PurchaseQuantity < 1 ||
                 CartItemUpdates[i].RemoveItem == true)
                {
                RemoveItem(cartId, cartItem.ProductID);
                }
             else
                {
                UpdateItem(cartId, cartItem.ProductID,
                                   CartItemUpdates[i].PurchaseQuantity);
                }
              }
            }
          }
        }
      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 PurchaseQuantity;
   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].PurchaseQuantity = 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)
```

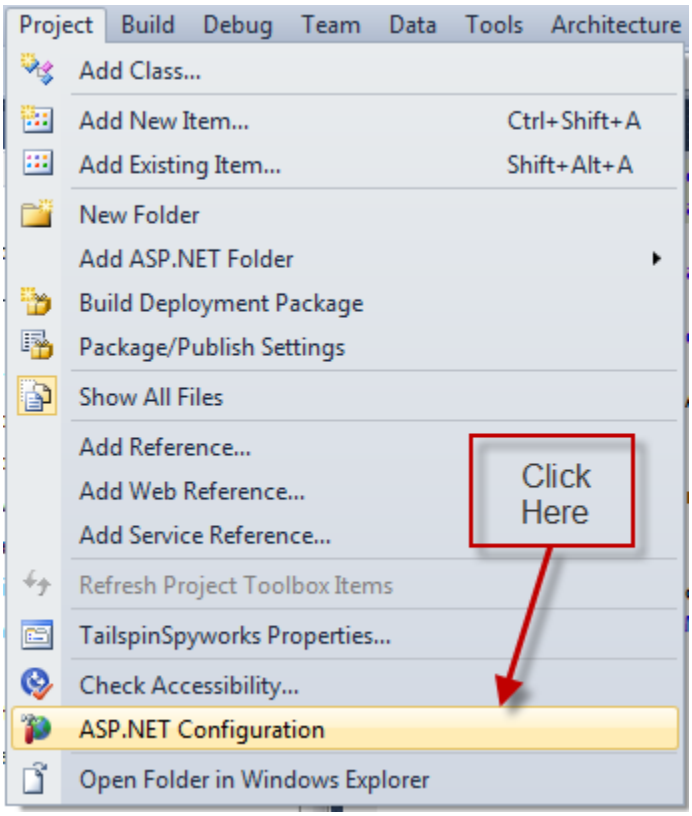
```
        {
        // Extract values from the cell
        cell.ContainingField.ExtractValuesFromCell(values, cell, row.RowState, true);
        }
    }
    return values;
}
```

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.

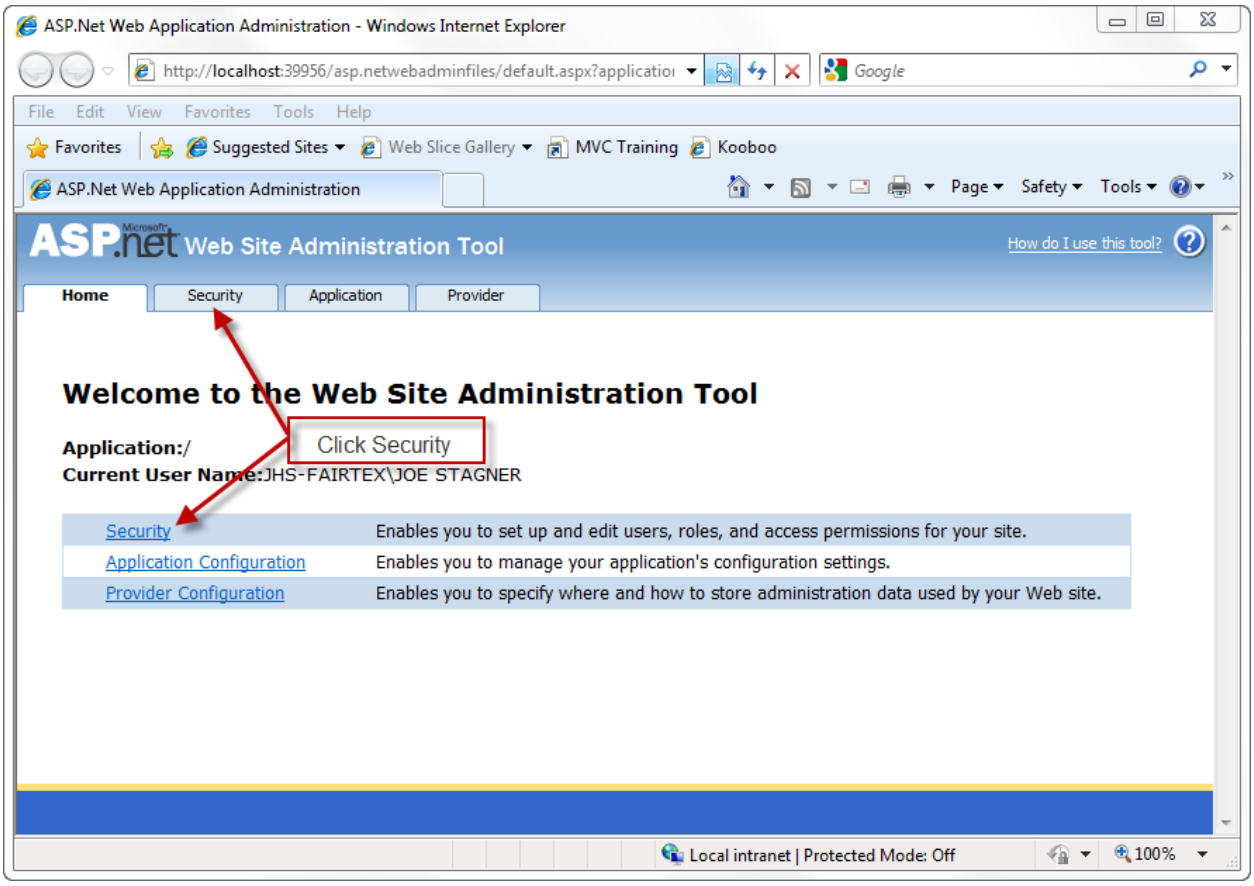
At this stage in your project's development we are getting ready to implement the checkout process.

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

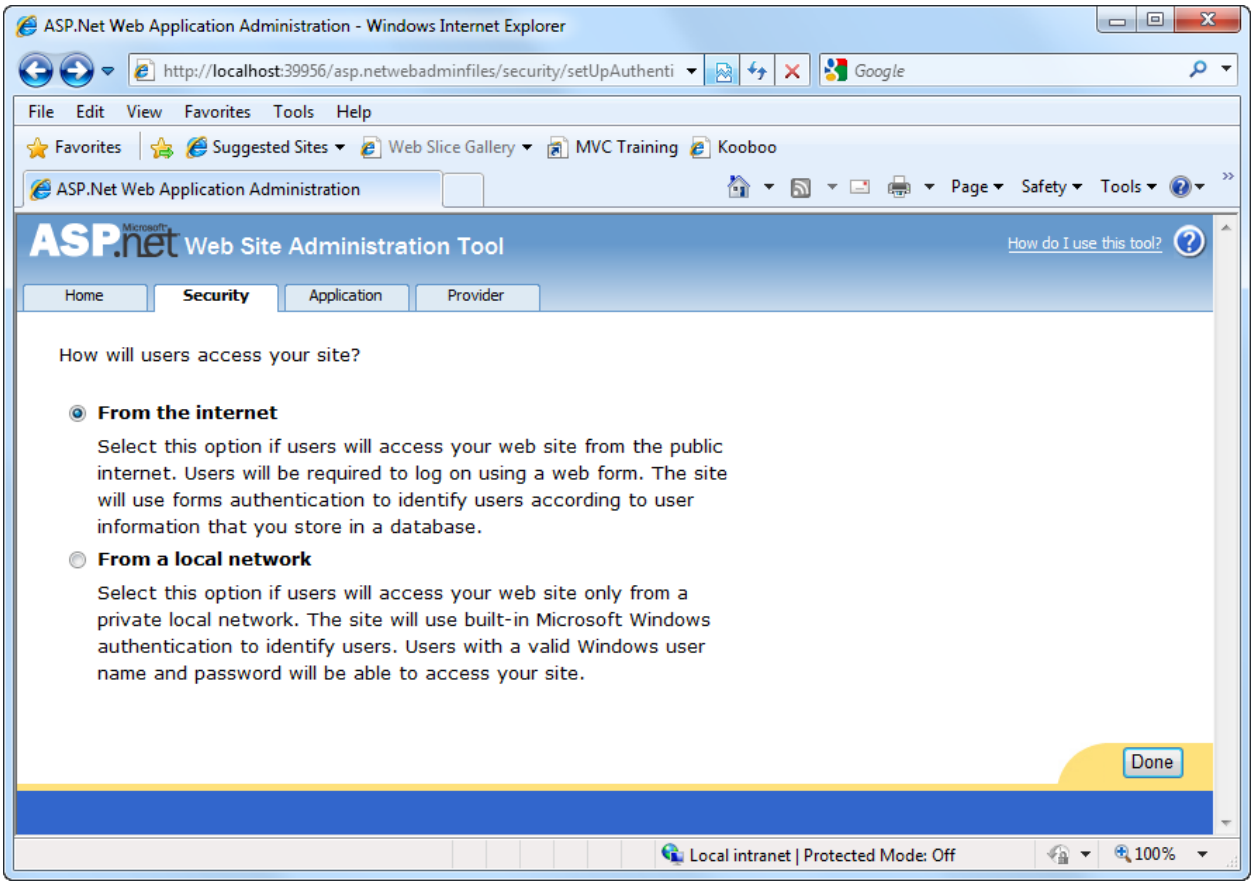
Working with ASP.NET Membership



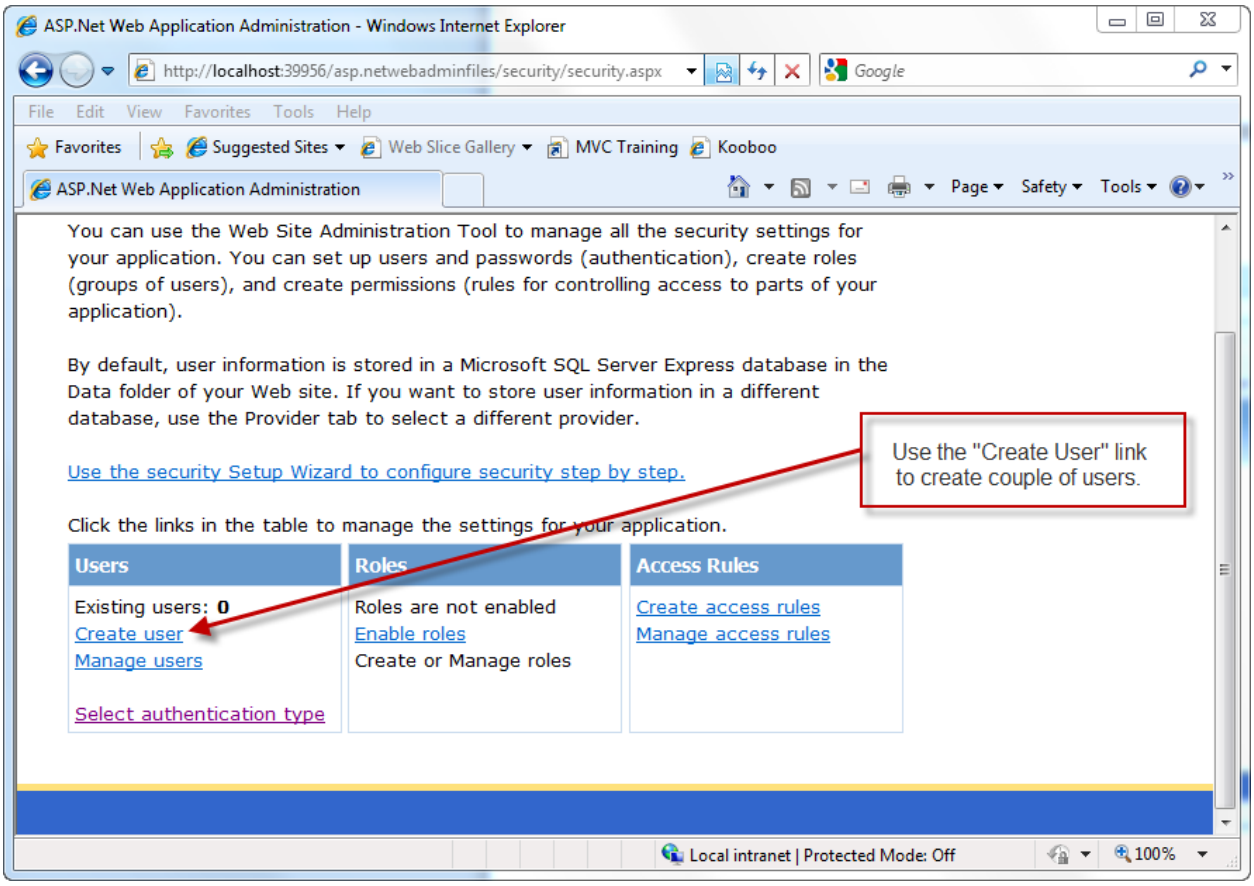
Click Security



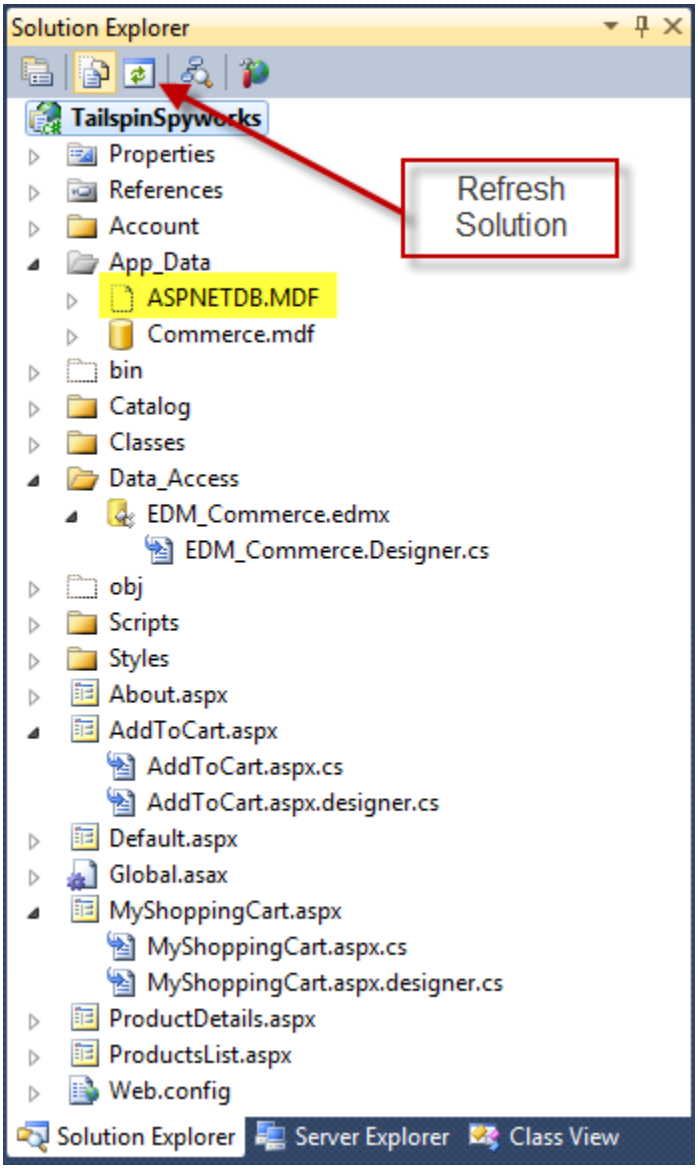
Make sure that we are using forms authentication.



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



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



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

You may need to select the “Show All Files” button in Solution Exdplorer to see it and manually Right-Click -> “Include in Project” to open it.

Now we can begin implementing the checkout process.

Begin by creating a CheckOut.aspx page.

The ChechOut.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 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" />
</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.

OrderDetails.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 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.

Copy images from the TailspinSpyworks downloaded Content Folder to the "Content" folder that we created earlier. (We'' use an image 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>
```

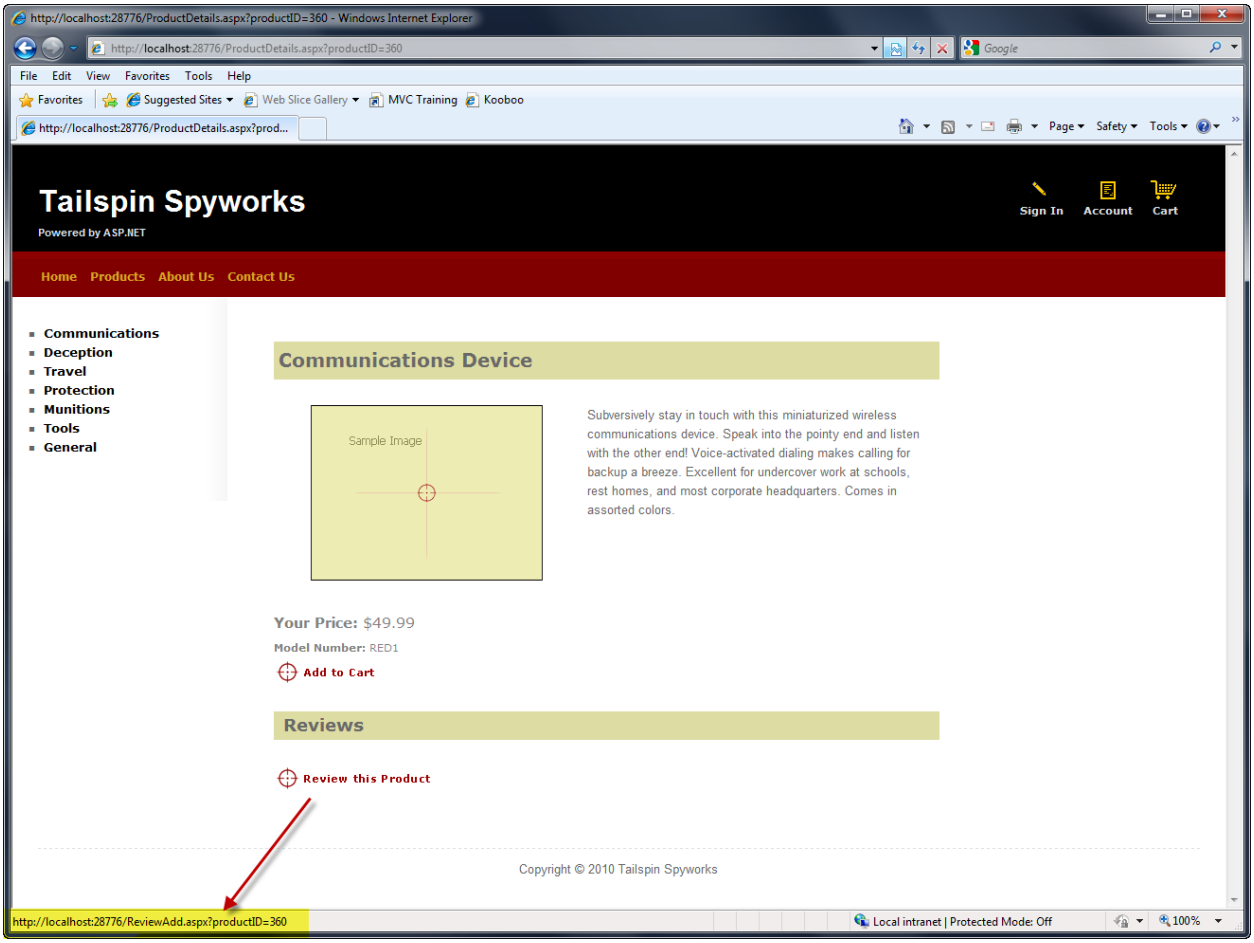
```
</h2>

<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 Ajax 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 than a decade of innovation.
        This   demo leverages many of the latest features of ASP.NET development to
        illustrate how really simply building rich web applications with ASP.NET can be.
        For more information about building 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

In the ProductDetails.aspx page, at the bottom of the formview control, 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>
```

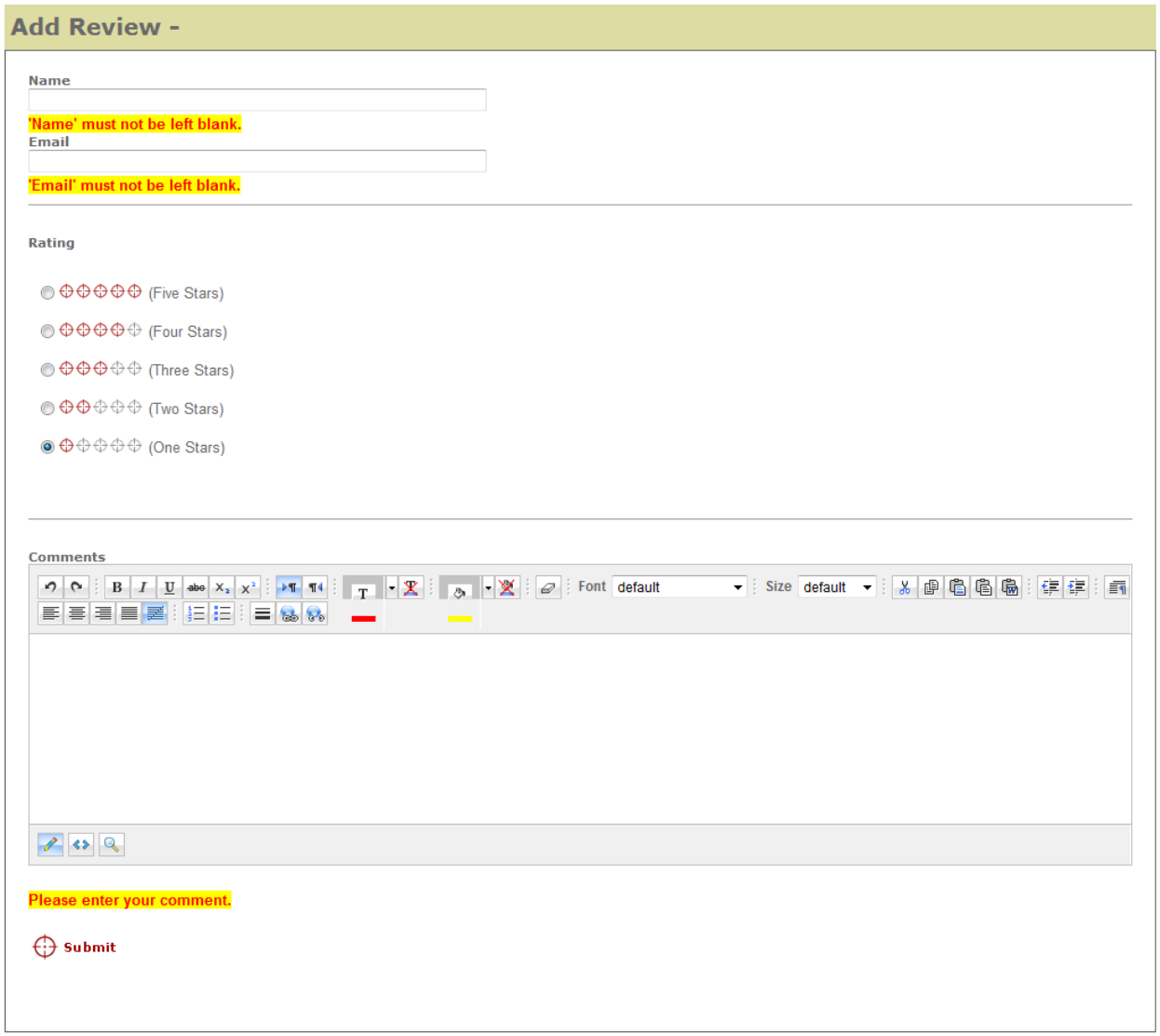


Note that we are passing the ProductID in the query string

Next let's add page named ReviewAdd.aspx

This page will use the ASP.NET AJAX 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.



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>
```

In the code-behind file we need to do two things.

1.) Make sure the ProductId sent to the page is valid.
2.) Save the user entered data in the database.

The ReviewAdd.aspx.cs looks like this.

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

namespace TailspinSpyworks
{
    public partial class ReviewAdd : System.Web.UI.Page
    {
        //--------------------------------------------------------------------------+
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!String.IsNullOrEmpty(Request.QueryString["ProductID"]))
                {
                int  productID = 0;
                Int32.TryParse(Request["productID"].ToString(), out productID);
                using (CommerceEntities db = new CommerceEntities())
```

```
            {
                try
                {
                    var thisProduct = (from p in db.Products
                                       where p.ProductID == productID
                                       select p).FirstOrDefault();
                    ModelName.Text = thisProduct.ModelName;
                }
                catch (Exception exp)
                {
                    throw new Exception("ERROR: Unable to Add Product Review - "
                                        + exp.Message.ToString(), exp);
                }
            }
        }
    else
        {
        Debug.Fail("ERROR : We should never get to ReviewAdd.aspx
                           without a ProductId.");
        throw new Exception("ERROR : It is illegal to load ReviewAdd.aspx
                                    without setting a ProductId.");
        }
}

//---------------------------------------------------------------------------+
protected void ReviewAddBtn_Click(object sender, ImageClickEventArgs e)
{
    if (Page.IsValid == true)
    {
        // Obtain ProductID from Page State
        int productID = Int32.Parse(Request["productID"]);

        // Obtain Rating number of RadioButtonList
        int rating = Int32.Parse(Rating.SelectedItem.Value);

        // Add Review to ReviewsDB.  HtmlEncode before entry
        using (CommerceEntities db = new CommerceEntities())
        {
            try
            {
                ShoppingCart cartadd = new ShoppingCart();
                Review newReview = new Review()
                {
                    ProductID = productID,
                    Rating = rating,
                    CustomerName = HttpUtility.HtmlEncode(Name.Text),
                    CustomerEmail = HttpUtility.HtmlEncode(Email.Text),
                    Comments = HttpUtility.HtmlEncode(UserComment.Content)
                };
                db.Reviews.AddObject(newReview);
                db.SaveChanges();
            }
            catch (Exception exp)
            {
                throw new Exception("ERROR: Unable to Update Cart Item - " +
                                    exp.Message.ToString(), exp);
            }
        }
```

```
            Response.Redirect("ProductDetails.aspx?ProductID=" + productID);
            }
        Response.Redirect("ProductList.aspx");
        }

        protected void LikeRating_Changed(object sender,
                                          AjaxControlToolkit.RatingEventArgs e)
        {

        }
    }
}
```

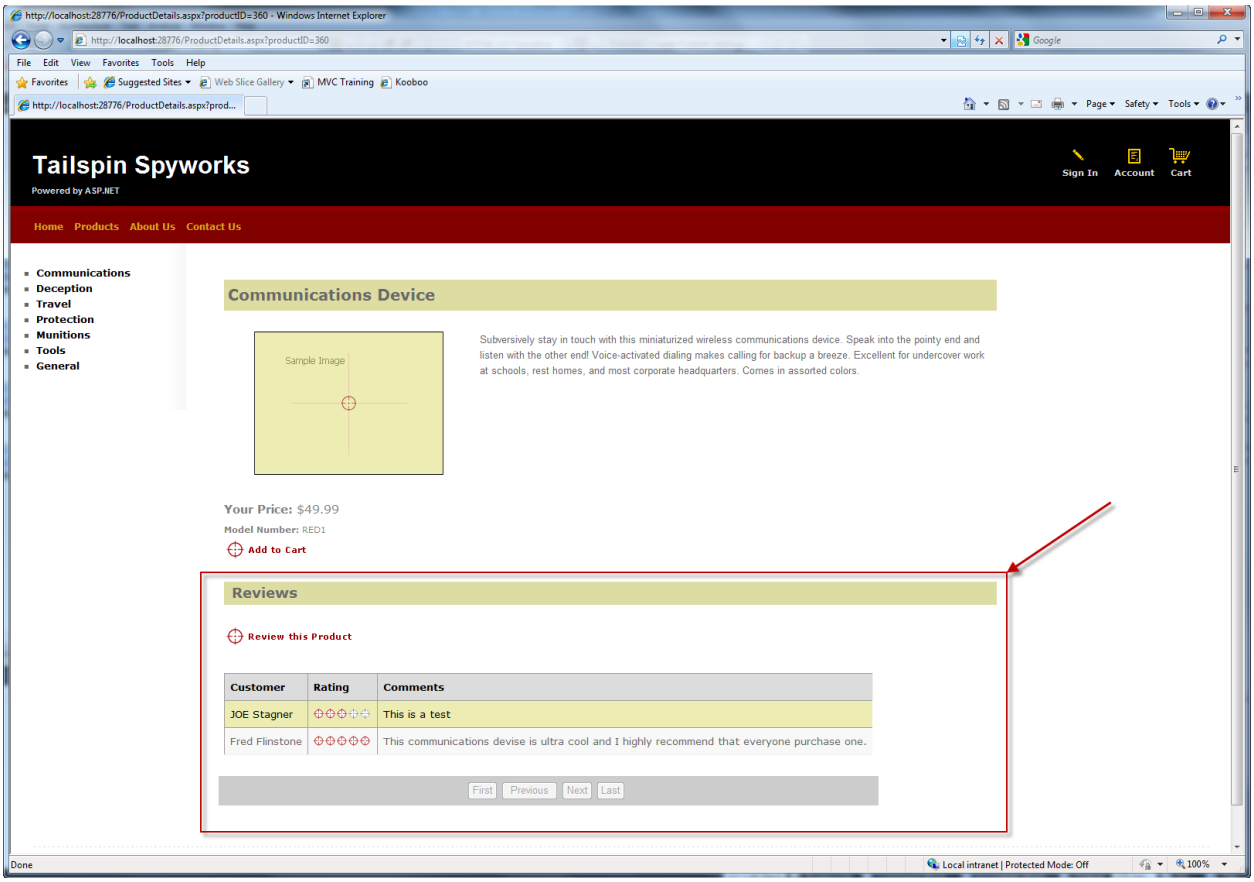
Now that we can enter reviews, let's display those reviews on the product page.

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.



Popular Items Control (Creating User Controls)

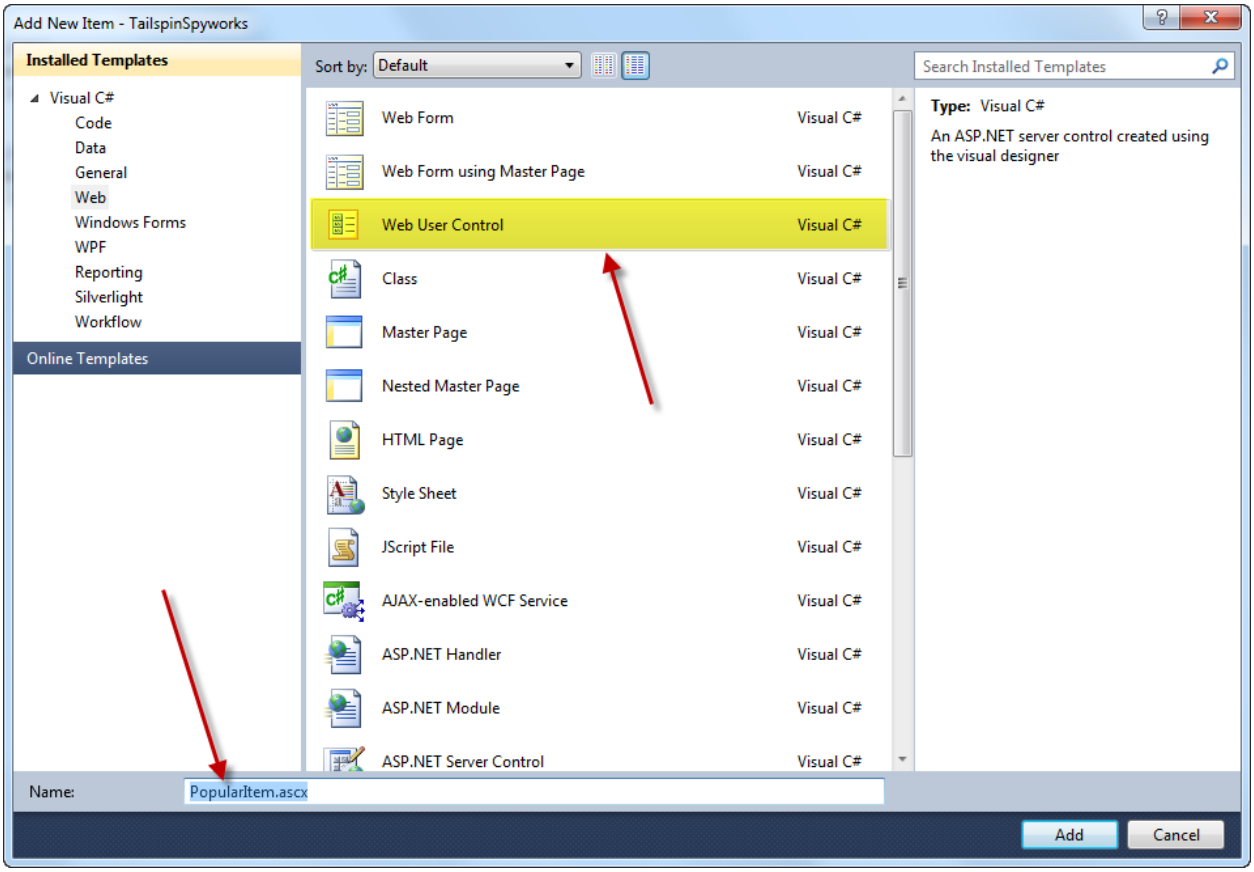
In order to increase sales on your web site we will add a couple of features to “suggestive sell” popular or related products.

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” :



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);
      }
    }
}
```

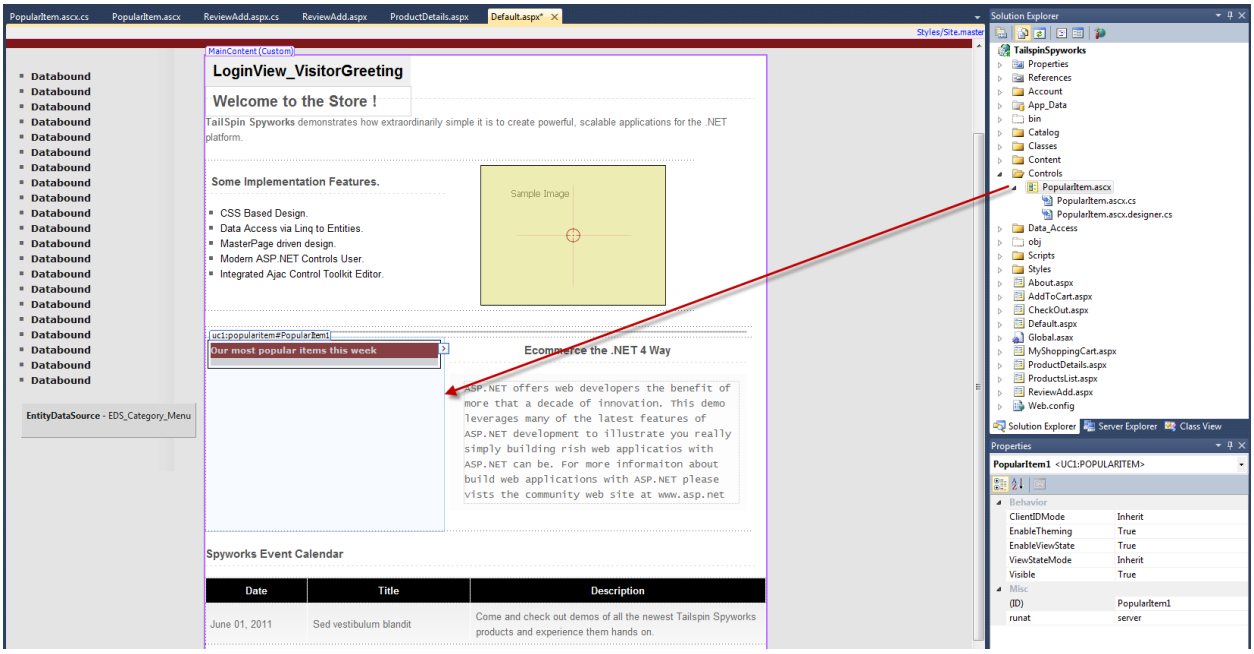
Note also this important line at the top of our control's markup.

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

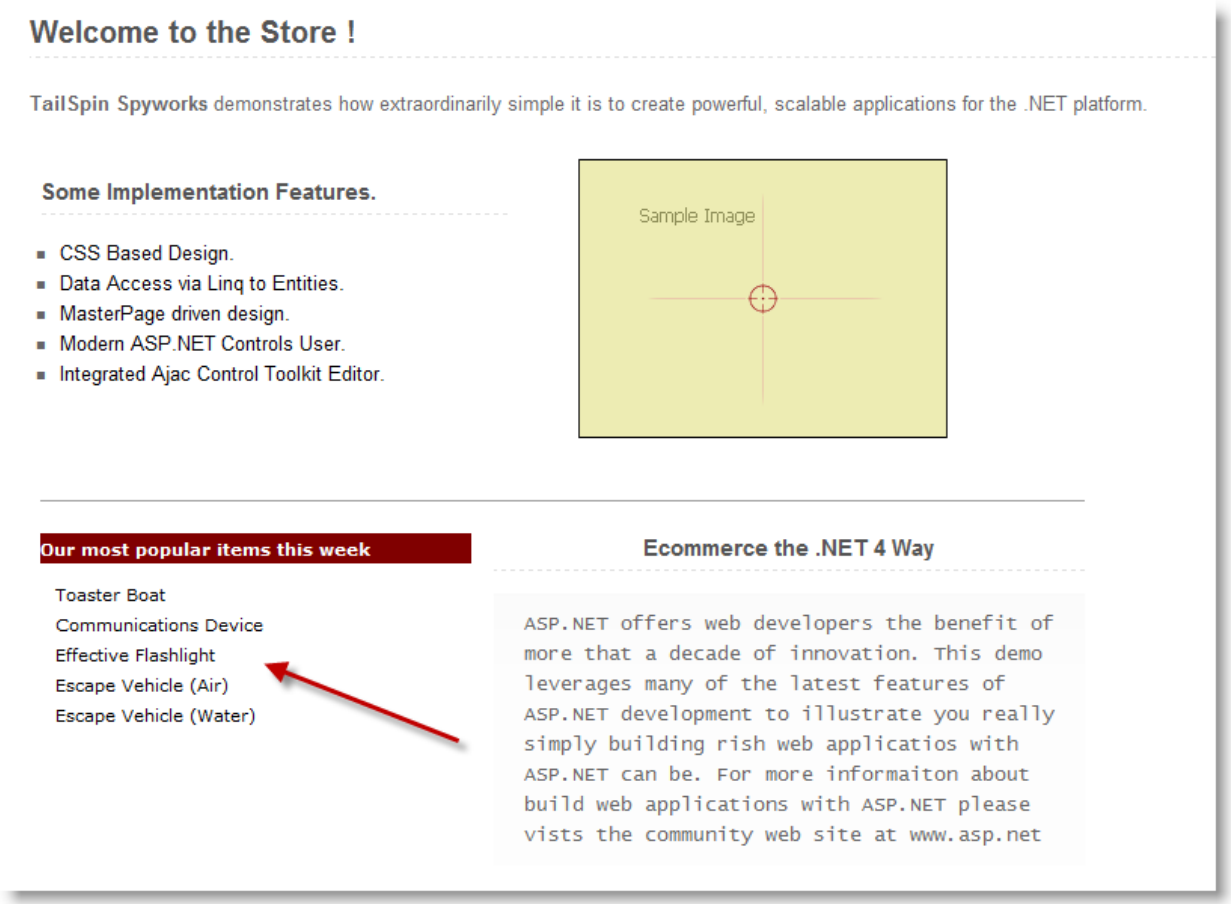
Since the most popular items won't be changing on a minute to minute basis we can add a aching directive to improve the performance of our application. This directive will cause the controls code to only be executed when the cached output of the control expires. Otherwise, the cached version of the control's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



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



"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.

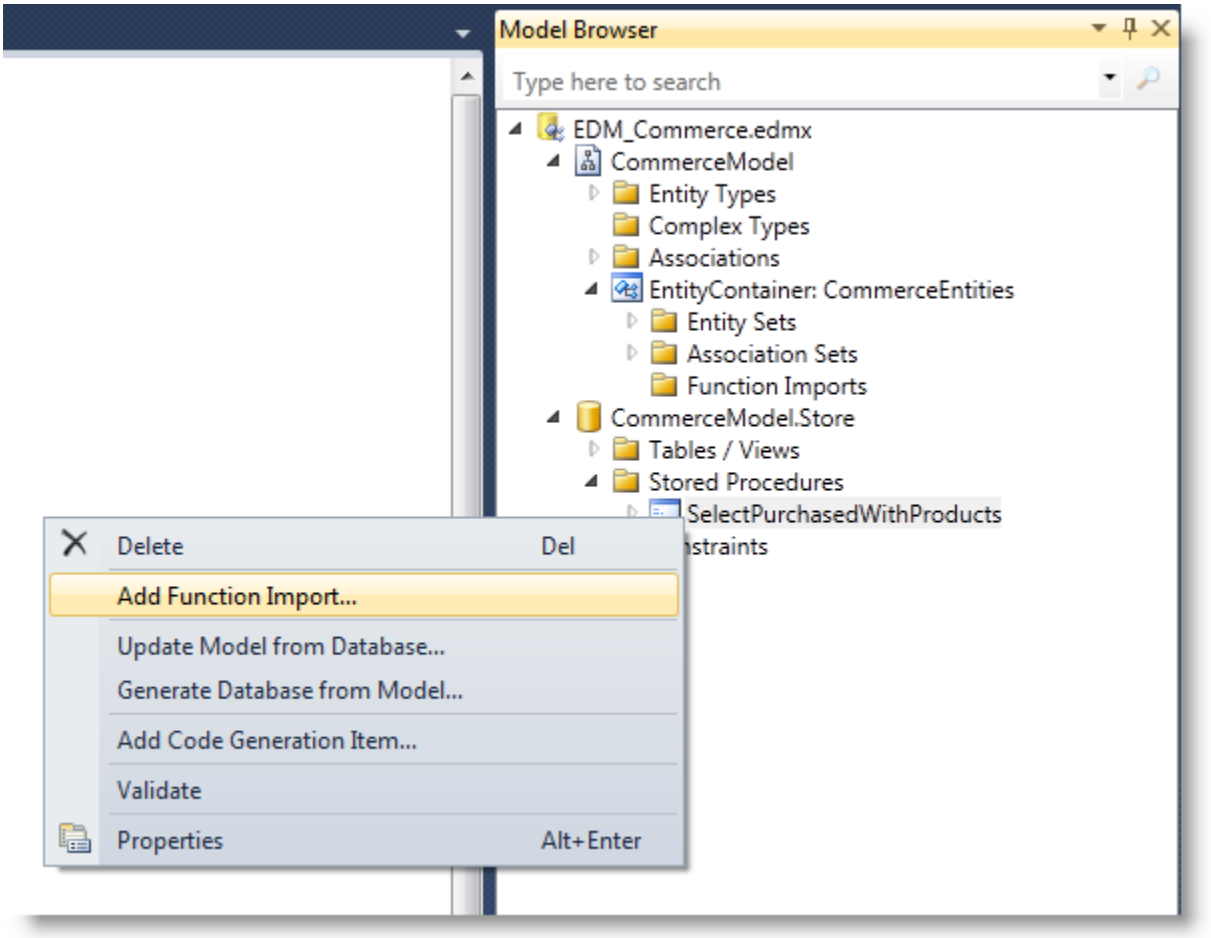
The T-SQL for the stored procedure is as follows.

```
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.

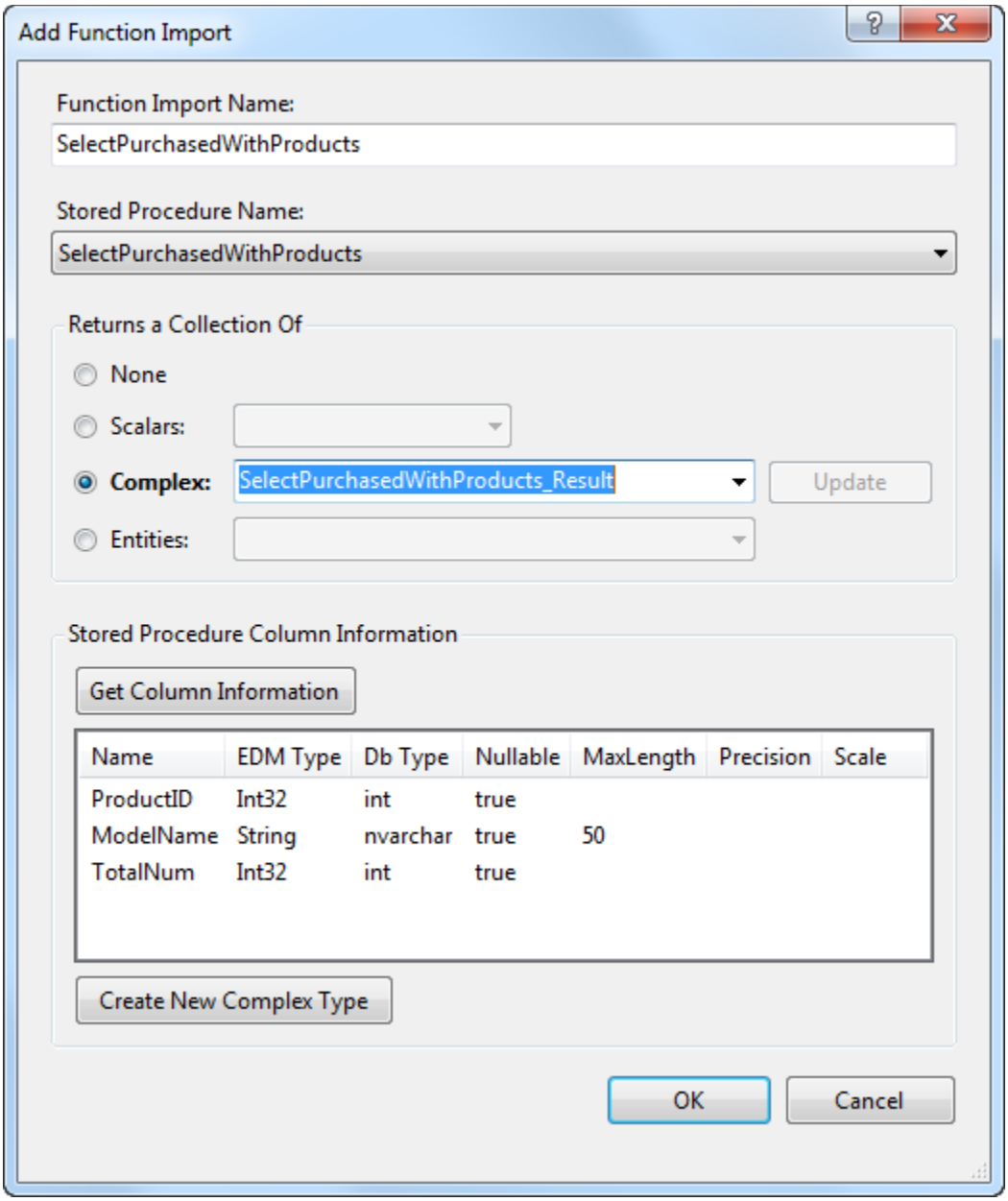
To access the stored procedure from the Entity Data Model we need to import the function.

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.

Click on “Add Function Import”



Click “Get Column Information” and “Create New Complex Type”.

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 need 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);
      }
    }
}
```

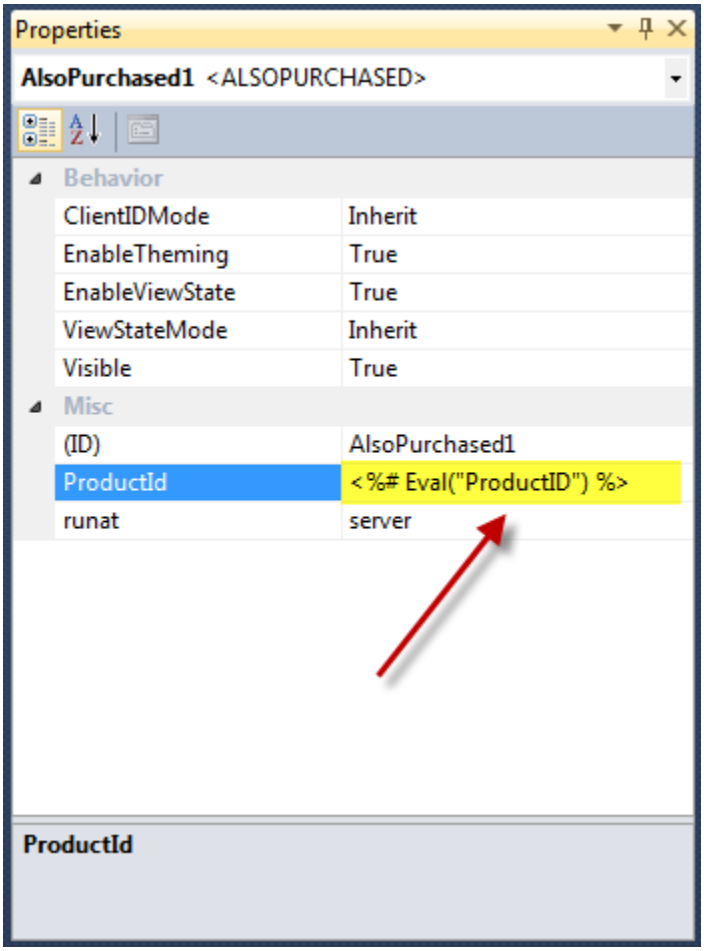
To view the “Also Purchased” items, open the ProductDetails.aspx page ***in design mode*** and drag the AlsoPurchased control from the Solutions Explorer so that it appears in this position in the markup.

```
<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>
```

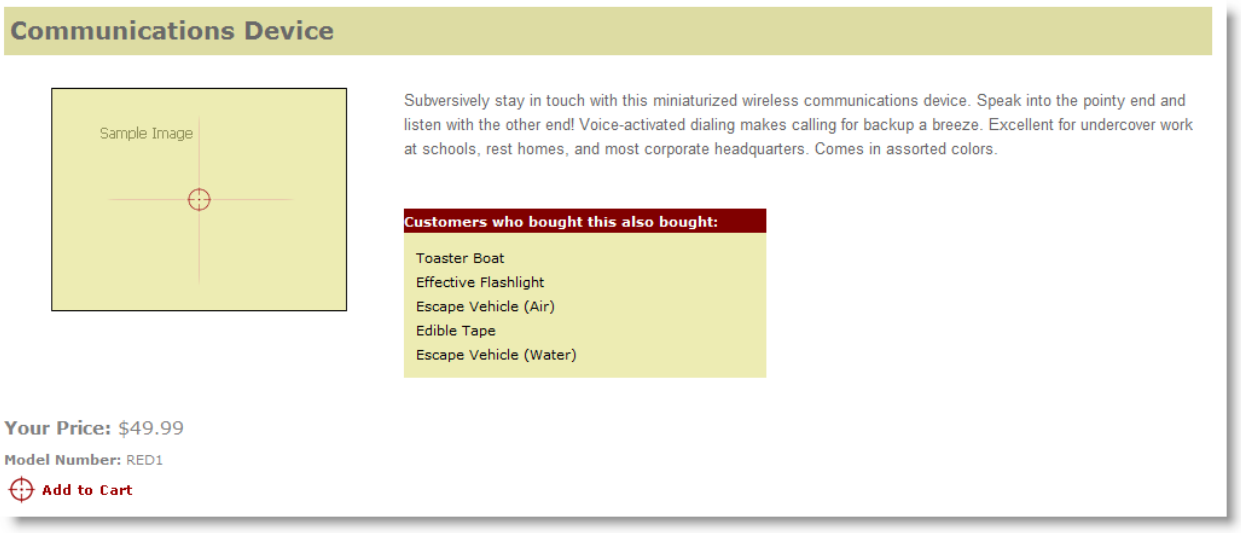
Doing so will create a reference to the control at the top of the ProductDetails page.

```
<%@ 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.



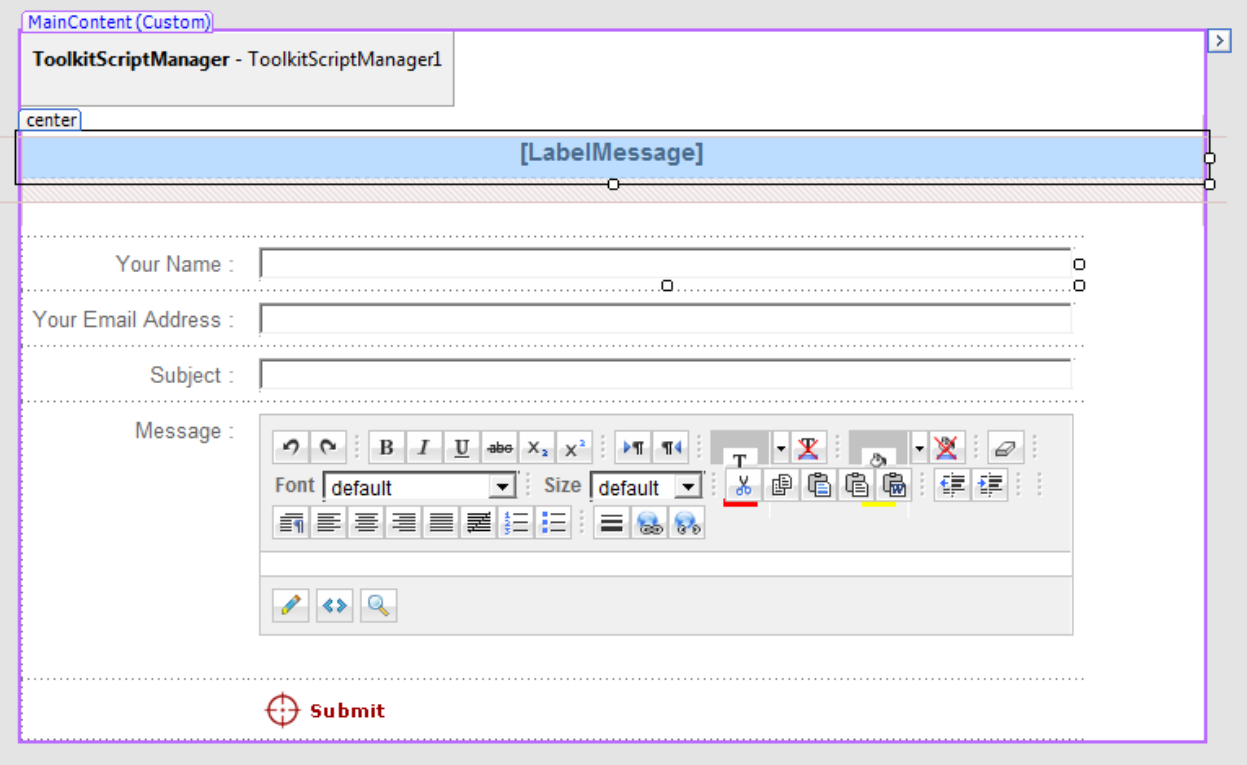
When we build and run now and browse to a product we see the “Also Purchased” items.



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.



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.

Add the using statement at the top of the file.

```
using Systen.Net.Mail;
```

And add code to the Click event handler as follows.

```
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>
```

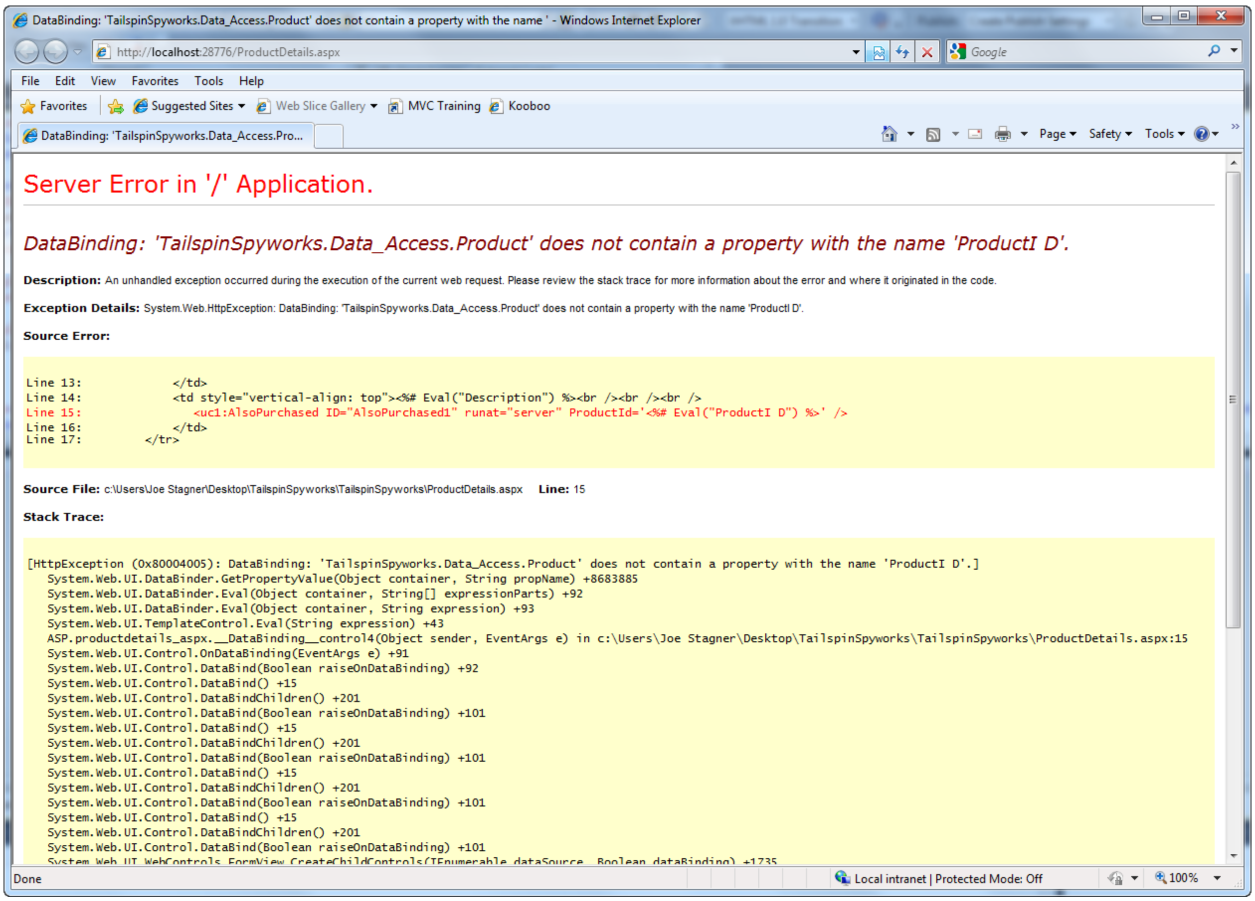
About Page

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.



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?";
           if (myEx.InnerException != null)
           {
               RedirectUrlString += "InnerErr=" + myEx.InnerException.Message.ToString()
                                 + "&";
           }
           RedirectUrlString  += "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.

```
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 Error : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (Request["Err"] != null)
            {
                Label_ErrorFrom.Text = Request["Err"].ToString();
            }
            else
            {
                Label_ErrorFrom.Text = "Undetermined Source.";
            }
            if (Request["InnerErr"] != null)
            {
                Label_ErrorMessage.Text = Request["InnerErr"].ToString();
            }
            else
            {
```

```
                Label_ErrorMessage.Text = "Undetermined Detail.";
            }
        }
    }
}
```

Conclusion

We've seen that 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!

Database Design Appendix

Tailspin Spyworks

09/30/2010

Database: COMMERCE.MD F

Database: COMMERCE.MDF

Properties

Name:	COMMERCE.MDF
Owner:	
Create for Attach:	09/28/2010
Compatibility Level:	100
Recovery Model:	Simple recovery model
Status:	Database is available for query
Full Text Search Enabled:	
Primary File:	App_Data\Commerce.mdf
Version:	
Database size, MB:	2.68 MB
Allocated Space, KB:	1760 KB
Data Space in Use and Reserved for Use, KB:	800 KB
Index Space Usage, KB:	864 KB
Allocated and Unused Space, KB:	96 KB
Description:	

Tables

Name	Owner	Data Size KB	Index Size KB	Creation Date	Cols	Rows	Description
Categories	dbo	24	8	02/16/2010	2	7	
OrderDetails	dbo	8	8	02/23/2010	5	5	
Orders	dbo	24	8	02/23/2010	4	1	
Products	dbo	24	32	02/16/2010	7	41	
Reviews	dbo	8	8	02/16/2010	6	0	

ShoppingCart	dbo	40	8	02/16/2010	5	2
sysdiagrams	dbo	24	32	02/23/2010	5	1

Views

Name	Owner	Schema bound	Encrypted	Creation Date	Modification Date	Description
VewOrderDetails	dbo	☐	☐	02/23/2010	02/23/2010	
ViewAlsoPurchased	dbo	☐	☐	02/24/2010	02/24/2010	
ViewCart	dbo	☐	☐	02/17/2010	02/24/2010	

Stored procedures

Name	Owner	Description
SelectPurchasedWithProducts	dbo	

Database: COMMERCE.MDF

Database Options

Settings: Auto Close:	☑
Settings: Auto Shrink:	☑
Settings: Auto Create Statistics:	☑
Settings: Auto Update Statistics:	☑
Settings: Use Quoted Identifiers:	☐
Settings: Torn Page Detection:	☐
Settings: Recursive Triggers:	☐
Settings: ANSI NULL default:	☐
Access: Read Only:	☐
Access: Single User:	☐
Access: Members of db_owner, dbcreator or sysadmin:	☐
Truncate Log On Checkpoint:	☑

Select Into/Bulkcopy:	☑
Cursor Close On Commit:	☐
Cursors in a Batch are with Local Scope:	☐

Tables

Table: dbo.Categories

Table: dbo.Categories **Properties**

Owner:	dbo
Creation Date:	02/16/2010
Located On:	PRIMARY
Data Size KB:	8
Index Size KB:	24
Rows:	7
Description:	

Columns

Name	Data Type	Length	NULL	Default	IsIdentity	IsGUID	Description
CategoryID	int	4	☐		☑	☐	
CategoryName	nvarchar	50	☑		☐	☐	

Identity column

Name	Seed	Increment	Not for replication
CategoryID	1	1	☐

Indexes

Index	Primary	Unique	Description
PK_Categories	☑	☑	

Referencing Tables

Table	Foreign Key	Primary Key or Unique Constraint
dbo.Products	FK_Products_Categories	PK_Categories

Objects that depend on [dbo].[Categories]

Object Name	Owner	Object Type	Dep Level
Products	dbo	Table	1
VewOrderDetails	dbo	View	2
ViewAlsoPurchased	dbo	View	2

Reviews	dbo	Table	2
ShoppingCart	dbo	Table	2
SelectPurchasedWithProducts	dbo	Procedure	2
ViewCart	dbo	View	3

SQL

```
CREATE TABLE [Categories] (
   [CategoryID] [int] IDENTITY (1, 1) NOT NULL ,
   [CategoryName] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
   CONSTRAINT [PK_Categories] PRIMARY KEY  NONCLUSTERED
   (
      [CategoryID]
   )  ON [PRIMARY]
) ON [PRIMARY]
GO
```

Indexes

Index: PK_Categories on table dbo.Categories

Index: PK_Categories on table dbo.Categories **Properties**

Clustered Index:	☐
Primary Key Index:	☑
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	16
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
CategoryID	☐	☐

Index: PK_Categories on table dbo.Categories

Statistics

Updated:	Feb 16 2010 1:21PM
Average key length:	4
Density:	1
Rows in the table:	7
Rows sampled for statistics data:	7
Distribution steps:	4

SQL

```
CREATE  UNIQUE  INDEX [PK_Categories] ON [dbo].[Categories]([CategoryID]) ON
[PRIMARY]
GO
```

Table: dbo.OrderDetails

Table: dbo.OrderDetails

Properties

Owner:	dbo
Creation Date:	02/23/2010
Located On:	PRIMARY
Data Size KB:	8
Index Size KB:	8
Rows:	5
Description:	

Columns

Name	Data Type	Length	NULL	Default	IsIdentity	IsGUID	Description
Id	int	4	☐		☑	☐	
OrderID	int	4	☑		☐	☐	
ProductID	int	4	☑		☐	☐	
Quantity	int	4	☑		☐	☐	
UnitCost	money	8	☑		☐	☐	

Identity column

Name	Seed	Increment	Not for replication
Id	1	1	☐

Indexes

Index	Primary	Unique	Description

PK_OrderDetails	☑	☑

Referenced Tables

Table	Foreign Key	Primary Key or Unique Constraint
dbo.Orders	FK_Order_OrderDetails	PK_Orders

Objects that [dbo].[OrderDetails] depends on

Object Name	Owner	Object Type	Dep Level
Orders	dbo	Table	1

Objects that depend on [dbo].[OrderDetails]

Object Name	Owner	Object Type	Dep Level
VewOrderDetails	dbo	View	1
ViewAlsoPurchased	dbo	View	1
SelectPurchasedWithProducts	dbo	Procedure	1

SQL

```
CREATE TABLE [OrderDetails] (
   [Id] [int] IDENTITY (1, 1) NOT NULL ,
   [OrderID] [int] NULL ,
   [ProductID] [int] NULL ,
   [Quantity] [int] NULL ,
   [UnitCost] [money] NULL ,
   CONSTRAINT [PK_OrderDetails] PRIMARY KEY  CLUSTERED
   (
      [Id]
   )  ON [PRIMARY] ,
   CONSTRAINT [FK_Order_OrderDetails] FOREIGN KEY
   (
      [OrderID]
   ) REFERENCES [Orders] (
      [OrderID]
   )
) ON [PRIMARY]
GO
```

Indexes

Index: PK_OrderDetails on table dbo.OrderDetails

Index: PK_OrderDetails on table dbo.OrderDetails **Properties**

Clustered Index:	☑
Primary Key Index:	☑
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	8
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
Id	☐	☐

Index: PK_OrderDetails on table dbo.OrderDetails

Statistics

Updated:	Feb 24 2010 2:29PM
Average key length:	4
Density:	1
Rows in the table:	14
Rows sampled for statistics data:	14
Distribution steps:	8

SQL

```
CREATE  UNIQUE  CLUSTERED  INDEX [PK_OrderDetails] ON [dbo].[OrderDetails]([Id]) ON
[PRIMARY]
GO
```

Relationships

Relationship: [FK_Order_OrderDetails] on [OrderDetails]

Relationship: [FK_Order_OrderDetails] on [OrderDetails]

Properties

Primary Table Owner:	dbo
Primary Table:	Orders
Primary Key or Unique Constraint:	PK_Orders
Delete Cascade:	☐
Update Cascade:	☐
Description:	

Relationship Columns

Column	Reference Column	Type
OrderID	OrderID	int

SQL

```
ALTER TABLE [dbo].[OrderDetails] ADD CONSTRAINT [FK_Order_OrderDetails] FOREIGN
KEY
   (
      [OrderID]
   ) REFERENCES [Orders] (
      [OrderID]
   )
GO
```

Table: dbo.Orders

Table: dbo.Orders

Properties

Owner:	dbo
Creation Date:	02/23/2010
Located On:	PRIMARY
Data Size KB:	8
Index Size KB:	24
Rows:	1
Description:	

Columns

Name	Data Type	Length	NULL	Default	IsIdentity	IsGUID	Description
OrderID	int	4	☐		☑	☐	
CustomerName	nvarchar	256	☐		☐	☐	
OrderDate	datetime	8	☐	(getdate())	☐	☐	
ShipDate	datetime	8	☐	(getdate())	☐	☐	

Identity column

Name	Seed	Increment	Not for replication
OrderID	1	1	☐

Indexes

Index	Primary	Unique	Description
PK_Orders	☑	☑	

Referencing Tables

Table	Foreign Key	Primary Key or Unique Constraint
dbo.OrderDetails	FK_Order_OrderDetails	PK_Orders

Objects that depend on [dbo].[Orders]

Object Name	Owner	Object Type	Dep Level
OrderDetails	dbo	Table	1
VewOrderDetails	dbo	View	2
ViewAlsoPurchased	dbo	View	2
SelectPurchasedWithProducts	dbo	Procedure	2

SQL

```
CREATE TABLE [Orders] (
   [OrderID] [int] IDENTITY (1, 1) NOT NULL ,
   [CustomerName] [nvarchar] (256) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
   [OrderDate] [datetime] NOT NULL CONSTRAINT [DF_Orders_OrderDate] DEFAULT
```

```
(getdate()),
    [ShipDate] [datetime] NOT NULL CONSTRAINT [DF_Orders_ShipDate] DEFAULT (getdate()),
    CONSTRAINT [PK_Orders] PRIMARY KEY  NONCLUSTERED
    (
        [OrderID]
    )  ON [PRIMARY]
) ON [PRIMARY]
GO
```

Indexes

Index: PK_Orders on table dbo.Orders

Index: PK_Orders on table dbo.Orders **Properties**

Clustered Index:	☐
Primary Key Index:	☑
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	16
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
OrderID	☐	☐

Index: PK_Orders on table dbo.Orders

Statistics

Updated:	Feb 23 2010 9:16AM
Average key length:	4
Density:	1
Rows in the table:	12
Rows sampled for statistics data:	12
Distribution steps:	7

SQL

```
CREATE  UNIQUE  INDEX [PK_Orders] ON [dbo].[Orders]([OrderID]) ON [PRIMARY]
GO
```

Table: dbo.Products

Table: dbo.Products

Properties

Owner:	dbo
Creation Date:	02/16/2010
Located On:	PRIMARY
Data Size KB:	32
Index Size KB:	24
Rows:	41
Description:	

Columns

Name	Data Type	Length	NULL	Default	IsIdentity	IsGUID	Description
ProductID	int	4	☐		☑	☐	
CategoryID	int	4	☐		☐	☐	
ModelNumber	nvarchar	50	☑		☐	☐	
ModelName	nvarchar	50	☑		☐	☐	
ProductImage	nvarchar	50	☑		☐	☐	
UnitCost	money	8	☐		☐	☐	
Description	nvarchar	3800	☑		☐	☐	

Identity column

Name	Seed	Increment	Not for replication
ProductID	1	1	☐

Indexes

Index	Primary	Unique	Description
PK_Products	✓	✓	

Referencing Tables

Table	Foreign Key	Primary Key or Unique Constraint
dbo.Reviews	FK_Reviews_Products	PK_Products
dbo.ShoppingCart	FK_ShoppingCart_Products	PK_Products

Referenced Tables

Table	Foreign Key	Primary Key or Unique Constraint
dbo.Categories	FK_Products_Categories	PK_Categories

Objects that [dbo].[Products] depends on

Object Name	Owner	Object Type	Dep Level
Categories	dbo	Table	1

Objects that depend on [dbo].[Products]

Object Name	Owner	Object Type	Dep Level
VewOrderDetails	dbo	View	1
ViewAlsoPurchased	dbo	View	1
Reviews	dbo	Table	1
ShoppingCart	dbo	Table	1
SelectPurchasedWithProducts	dbo	Procedure	1
ViewCart	dbo	View	2

SQL

```
CREATE TABLE [Products] (
   [ProductID] [int] IDENTITY (1, 1) NOT NULL ,
   [CategoryID] [int] NOT NULL ,
   [ModelNumber] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
   [ModelName] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
   [ProductImage] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
   [UnitCost] [money] NOT NULL ,
```

```
    [Description] [nvarchar] (3800) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    CONSTRAINT [PK_Products] PRIMARY KEY  NONCLUSTERED
    (
        [ProductID]
    )  ON [PRIMARY] ,
    CONSTRAINT [FK_Products_Categories] FOREIGN KEY
    (
        [CategoryID]
    ) REFERENCES [Categories] (
        [CategoryID]
    )
) ON [PRIMARY]
GO
```

Indexes

Index: PK_Products on table dbo.Products

Index: PK_Products on table dbo.Products **Properties**

Clustered Index:	☐
Primary Key Index:	☑
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	16
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
ProductID	☐	☐

Index: PK_Products on table dbo.Products

Statistics

Updated:	Feb 16 2010 1:21PM
Average key length:	4
Density:	1
Rows in the table:	41
Rows sampled for statistics data:	41
Distribution steps:	25

SQL

```
CREATE  UNIQUE  INDEX [PK_Products] ON [dbo].[Products]([ProductID]) ON [PRIMARY]
GO
```

Relationships

Relationship: [FK_Products_Categories] on [Products]

Relationship: [FK_Products_Categories] on [Products] **Properties**

Primary Table Owner:	dbo
Primary Table:	Categories
Primary Key or Unique Constraint:	PK_Categories
Delete Cascade:	☐
Update Cascade:	☐
Description:	

Relationship Columns

Column	Reference Column	Type
CategoryID	CategoryID	int

SQL

```
ALTER TABLE [dbo].[Products] ADD CONSTRAINT [FK_Products_Categories] FOREIGN KEY
   (
      [CategoryID]
   ) REFERENCES [Categories] (
      [CategoryID]
   )
GO
```

Table: dbo.Reviews

Table: dbo.Reviews

Properties

Owner:	dbo
Creation Date:	02/16/2010
Located On:	PRIMARY
Data Size KB:	8
Index Size KB:	8
Rows:	0
Description:	

Columns

Name	Data Type	Length	NULL	Default	IsIdentity	IsGUID	Description
ReviewID	int	4	☐		☑	☐	
ProductID	int	4	☐		☐	☐	
CustomerName	nvarchar	50	☑		☐	☐	
CustomerEmail	nvarchar	50	☑		☐	☐	
Rating	int	4	☐		☐	☐	
Comments	nvarchar	3850	☑		☐	☐	

Identity column

Name	Seed	Increment	Not for replication
ReviewID	1	1	☐

Indexes

Index	Primary	Unique	Description
PK_Reviews	☑	☑	

Referenced Tables

Table	Foreign Key	Primary Key or Unique Constraint
dbo.Products	FK_Reviews_Products	PK_Products

Objects that [dbo].[Reviews] depends on

Object Name	Owner	Object Type	Dep Level
Categories	dbo	Table	1
Products	dbo	Table	2

SQL

```
CREATE TABLE [Reviews] (
   [ReviewID] [int] IDENTITY (1, 1) NOT NULL ,
   [ProductID] [int] NOT NULL ,
   [CustomerName] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
   [CustomerEmail] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
   [Rating] [int] NOT NULL ,
```

```
    [Comments] [nvarchar] (3850) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    CONSTRAINT [PK_Reviews] PRIMARY KEY  CLUSTERED
    (
        [ReviewID]
    )  ON [PRIMARY] ,
    CONSTRAINT [FK_Reviews_Products] FOREIGN KEY
    (
        [ProductID]
    ) REFERENCES [Products] (
        [ProductID]
    ) NOT FOR REPLICATION
) ON [PRIMARY]
GO
```

Indexes

Index: PK_Reviews on table dbo.Reviews

Index: PK_Reviews on table dbo.Reviews **Properties**

Clustered Index:	☑
Primary Key Index:	☑
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	8
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
ReviewID	☐	☐

Index: PK_Reviews on table dbo.Reviews

Statistics

Updated:	Feb 22 2010 12:56PM
Average key length:	4
Density:	0
Rows in the table:	2
Rows sampled for statistics data:	2
Distribution steps:	2

SQL

```
CREATE  UNIQUE  CLUSTERED  INDEX [PK_Reviews] ON [dbo].[Reviews]([ReviewID]) ON
[PRIMARY]
GO
```

Relationships

Relationship: [FK_Reviews_Products] on [Reviews]

Properties

Primary Table Owner:	dbo
Primary Table:	Products
Primary Key or Unique Constraint:	PK_Products
Delete Cascade:	☐
Update Cascade:	☐
Description:	

Relationship Columns

Column	Reference Column	Type
ProductID	ProductID	int

SQL

```
ALTER TABLE [dbo].[Reviews] ADD CONSTRAINT [FK_Reviews_Products] FOREIGN KEY
   (
      [ProductID]
   ) REFERENCES [Products] (
      [ProductID]
   ) NOT FOR REPLICATION
GO
```

Table: dbo.ShoppingCart

Table: dbo.ShoppingCart

Properties

Owner:	dbo
Creation Date:	02/16/2010
Located On:	PRIMARY
Data Size KB:	8
Index Size KB:	40
Rows:	2
Description:	

Columns

Name	Data Type	Length	NULL	Default	IsIdentity	IsGUID	Description
RecordID	int	4	☐		☑	☐	
CartID	nvarchar	50	☑		☐	☐	
Quantity	int	4	☐	((1))	☐	☐	
ProductID	int	4	☐		☐	☐	
DateCreated	datetime	8	☐	(getdate())	☐	☐	

Identity column

Name	Seed	Increment	Not for replication
RecordID	1	1	☐

Indexes

Index	Primary	Unique	Description
PK_ShoppingCart	☑	☑	
IX_ShoppingCart	☐	☐	

Referenced Tables

Table	Foreign Key	Primary Key or Unique Constraint
dbo.Products	FK_ShoppingCart_Products	PK_Products

Objects that [dbo].[ShoppingCart] depends on

Object Name	Owner	Object Type	Dep Level
Categories	dbo	Table	1
Products	dbo	Table	2

Objects that depend on [dbo].[ShoppingCart]

Object Name	Owner	Object Type	Dep Level
ViewCart	dbo	View	1

SQL

```
CREATE TABLE [ShoppingCart] (
   [RecordID] [int] IDENTITY (1, 1) NOT NULL ,
   [CartID] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
   [Quantity] [int] NOT NULL CONSTRAINT [DF_ShoppingCart_Quantity] DEFAULT ((1)),
   [ProductID] [int] NOT NULL ,
   [DateCreated] [datetime] NOT NULL CONSTRAINT [DF_ShoppingCart_DateCreated] DEFAULT
(getdate()),
   CONSTRAINT [PK_ShoppingCart] PRIMARY KEY  NONCLUSTERED
   (
      [RecordID]
   )  ON [PRIMARY] ,
   CONSTRAINT [FK_ShoppingCart_Products] FOREIGN KEY
   (
      [ProductID]
   ) REFERENCES [Products] (
      [ProductID]
   )
) ON [PRIMARY]
GO
```

Indexes

Index: PK_ShoppingCart on table dbo.ShoppingCart

Index: PK_ShoppingCart on table dbo.ShoppingCart **Properties**

Clustered Index:	☐
Primary Key Index:	☑
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	16
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
RecordID	☐	☐

Index: PK_ShoppingCart on table dbo.ShoppingCart

Statistics

Updated:	Feb 17 2010 9:25PM
Average key length:	4
Density:	0
Rows in the table:	2
Rows sampled for statistics data:	2
Distribution steps:	2

SQL

```
CREATE  UNIQUE  INDEX [PK_ShoppingCart] ON [dbo].[ShoppingCart]([RecordID]) ON
[PRIMARY]
GO
```

Index: IX_ShoppingCart on table dbo.ShoppingCart

Index: IX_ShoppingCart on table dbo.ShoppingCart — Properties

Clustered Index:	☐
Primary Key Index:	☐
Unique Index:	☐
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	16
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
CartID	☐	☐
ProductID	☐	☐

Index: IX_ShoppingCart on table dbo.ShoppingCart — Statistics

Updated:	Feb 16 2010 1:32PM
Average key length:	76
Density:	0
Rows in the table:	1
Rows sampled for statistics data:	1
Distribution steps:	1

SQL

```
CREATE  INDEX [IX_ShoppingCart] ON [dbo].[ShoppingCart]([CartID], [ProductID]) ON
[PRIMARY]
GO
```

Relationships

Relationship: [FK_ShoppingCart_Products] on [ShoppingCart]

Relationship: [FK_ShoppingCart_Products] on [ShoppingCart] **Properties**

Primary Table Owner:	dbo
Primary Table:	Products
Primary Key or Unique Constraint:	PK_Products
Delete Cascade:	☐
Update Cascade:	☐
Description:	

Relationship Columns

Column	Reference Column	Type
ProductID	ProductID	int

SQL

```
ALTER TABLE [dbo].[ShoppingCart] ADD CONSTRAINT [FK_ShoppingCart_Products]
FOREIGN KEY
   (
      [ProductID]
   ) REFERENCES [Products] (
      [ProductID]
   )
GO
```

Table: dbo.sysdiagrams

Table: dbo.sysdiagrams

Properties

Owner:	dbo
Creation Date:	02/23/2010
Located On:	PRIMARY
Data Size KB:	32
Index Size KB:	24
Rows:	1
Description:	

Columns

Name	Data Type	Length	NULL	Default	IsIdentity	IsGUID	Description
name	sysname	256	☐		☐	☐	
principal_id	int	4	☐		☐	☐	
diagram_id	int	4	☐		☑	☐	
version	int	4	☑		☐	☐	
definition	varbinary	-1	☑		☐	☐	

Identity column

Name	Seed	Increment	Not for replication
diagram_id	1	1	☐

Indexes

Index	Primary	Unique	Description
PK__sysdiagr__C2B05B6136B12243	☑	☑	
UK_principal_name	☐	☑	

Objects that depend on [dbo].[sysdiagrams]

Object Name	Owner	Object Type	Dep Level
sp_alterdiagram	dbo	Procedure	1
sp_creatediagram	dbo	Procedure	1
sp_dropdiagram	dbo	Procedure	1
sp_helpdiagramdefinition	dbo	Procedure	1
sp_helpdiagrams	dbo	Procedure	1
sp_renamediagram	dbo	Procedure	1

Extended Properties

Name	Value

SQL

```
CREATE TABLE [sysdiagrams] (
    [name] [sysname] NOT NULL ,
    [principal_id] [int] NOT NULL ,
    [diagram_id] [int] IDENTITY (1, 1) NOT NULL ,
    [version] [int] NULL ,
    [definition] [varbinary] (-1) NULL ,
     PRIMARY KEY  CLUSTERED
    (
        [diagram_id]
    )  ON [PRIMARY] ,
    CONSTRAINT [UK_principal_name] UNIQUE  NONCLUSTERED
    (
        [principal_id],
        [name]
    )  ON [PRIMARY]
) ON [PRIMARY]
GO
```

Indexes

Index: PK__sysdiagr__C2B05B6136B12243 on table dbo.sysdiagrams

Index: PK__sysdiagr__C2B05B6136B12243 on table **Properties**

dbo.sysdiagrams

Clustered Index:	☑
Primary Key Index:	☑
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	24
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
diagram_id	☐	☐

Index: PK__sysdiagr__C2B05B6136B12243 on table dbo.sysdiagrams

Statistics

Updated:	
Average key length:	
Density:	
Rows in the table:	
Rows sampled for statistics data:	
Distribution steps:	

SQL

```
CREATE  UNIQUE  CLUSTERED  INDEX [PK__sysdiagr__C2B05B6136B12243] ON
[dbo].[sysdiagrams]([diagram_id]) ON [PRIMARY]
GO
```

Index: UK_principal_name on table dbo.sysdiagrams

Index: UK_principal_name on table dbo.sysdiagrams

Properties

Clustered Index:	☐
Primary Key Index:	☐
Unique Index:	☑
Ignore Duplicates:	☐
Pad Index:	☐
Allow Row Locks:	☑
Allow Page Locks:	☑
Is computed column in index:	☐
Located On:	PRIMARY
Size KB:	16
Fill Factor:	0
Don't Recompute Statistics:	☐
Description:	

Index Columns

Name	Descending	Included
name	☐	☐
principal_id	☐	☐

Index: UK_principal_name on table dbo.sysdiagrams

Statistics

Updated:	Sep 30 2010 10:34AM
Average key length:	32
Density:	0
Rows in the table:	1
Rows sampled for statistics data:	1
Distribution steps:	1

SQL

```
CREATE  UNIQUE  INDEX [UK_principal_name] ON [dbo].[sysdiagrams]([principal_id], [name])
ON [PRIMARY]
GO
```

Views

View: dbo.VewOrderDetails

View: dbo.VewOrderDetails **Properties**

Owner:	dbo
Schema bound:	☐
Encrypted:	☐
Creation Date:	02/23/2010
Modification Date:	02/23/2010
Description:	

Columns

Name	Data Type	Length	NULL	IsGUID	Description
ProductID	int	4	☐	☐	
ModelNumber	nvarchar	50	☑	☐	
ModelName	nvarchar	50	☑	☐	
Quantity	int	4	☑	☐	
UnitCost	money	8	☑	☐	
OrderID	int	4	☑	☐	

Objects that [dbo].[VewOrderDetails] depends on

Object Name	Owner	Object Type	Dep Level
Categories	dbo	Table	1
Orders	dbo	Table	1
OrderDetails	dbo	Table	2
Products	dbo	Table	2

Extended Properties

Name	Value
MS_DiagramPane1	[0E232FF0-B466-11cf-A24F-00AA00A3EFFF, 1.00]Begin DesignProperties = Begin PaneConfigurations = Begin PaneConfiguration = 0 NumPanes = 4 Configuration = "(H (1[40] 4[20] 2[20] 3))" End Begin PaneConfiguration = 1 NumPanes = 3 Configuration = "(H (1 [50] 4 [25] 3))" End Begin PaneConfiguration = 2 NumPanes = 3 Configuration = "(H (1 [50] 2 [25] 3))" End Begin

PaneConfiguration = 3 NumPanes = 3 Configuration = "(H (4 [30] 2 [40] 3))" End Begin PaneConfiguration = 4 NumPanes = 2 Configuration = "(H (1 [56] 3))" End Begin PaneConfiguration = 5 NumPanes = 2 Configuration = "(H (2 [66] 3))" End Begin PaneConfiguration = 6 NumPanes = 2 Configuration = "(H (4 [50] 3))" End Begin PaneConfiguration = 7 NumPanes = 1 Configuration = "(V (3))" End Begin PaneConfiguration = 8 NumPanes = 3 Configuration = "(H (1[56] 4[18] 2))" End Begin PaneConfiguration = 9 NumPanes = 2 Configuration = "(H (1 [75] 4))" End Begin PaneConfiguration = 10 NumPanes = 2 Configuration = "(H (1[66] 2))" End Begin PaneConfiguration = 11 NumPanes = 2 Configuration = "(H (4 [60] 2))" End Begin PaneConfiguration = 12 NumPanes = 1 Configuration = "(H (1))" End Begin PaneConfiguration = 13 NumPanes = 1 Configuration = "(V (4))" End Begin PaneConfiguration = 14 NumPanes = 1 Configuration = "(V (2))" End ActivePaneConfig = 0 End Begin DiagramPane = Begin Origin = Top = 0 Left = 0 End Begin Tables = Begin Table = "OrderDetails" Begin Extent = Top = 6 Left = 38 Bottom = 219 Right = 198 End DisplayFlags = 280 TopColumn = 0 End Begin Table = "Products" Begin Extent = Top = 6 Left = 236 Bottom = 219 Right = 396 End DisplayFlags = 280 TopColumn = 0 End End End Begin SQLPane = End Begin DataPane = Begin ParameterDefaults = "" End Begin ColumnWidths = 9 Width = 284 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 End End Begin CriteriaPane = Begin ColumnWidths = 11 Column = 1440 Alias = 900 Table = 1170 Output = 720 Append = 1400 NewValue = 1170 SortType = 1350 SortOrder = 1410 GroupBy = 1350 Filter = 1350 Or = 1350 Or = 1350 Or = 1350 End EndEnd

MS_DiagramPaneCount	1

SQL

```
CREATE VIEW dbo.VewOrderDetails
AS
SELECT       dbo.Products.ProductID, dbo.Products.ModelNumber, dbo.Products.ModelName,
dbo.OrderDetails.Quantity, dbo.OrderDetails.UnitCost,
                  dbo.OrderDetails.OrderID
FROM          dbo.OrderDetails INNER JOIN
                  dbo.Products ON dbo.OrderDetails.ProductID = dbo.Products.ProductID
```

View: dbo.ViewAlsoPurchased

View: dbo.ViewAlsoPurchased

Properties

Owner:	dbo
Schema bound:	☐
Encrypted:	☐
Creation Date:	02/24/2010

Modification Date: 02/24/2010

Description:

Columns

Name	Data Type	Length	NULL	IsGUID	Description
ProductID	int	4	☑	☐	
ModelName	nvarchar	50	☑	☐	
TotalNumPurchased	int	4	☑	☐	
OrderID	int	4	☑	☐	
Products_ProductID	int	4	☐	☐	

Objects that [dbo].[ViewAlsoPurchased] depends on

Object Name	Owner	Object Type	Dep Level
Categories	dbo	Table	1
Orders	dbo	Table	1
OrderDetails	dbo	Table	2
Products	dbo	Table	2

Extended Properties

Name	Value
MS_DiagramPane1	[0E232FF0-B466-11cf-A24F-00AA00A3EFFF, 1.00]Begin DesignProperties = Begin PaneConfigurations = Begin PaneConfiguration = 0 NumPanes = 4 Configuration = "(H (1[30] 4[23] 2[28] 3))" End Begin PaneConfiguration = 1 NumPanes = 3 Configuration = "(H (1 [50] 4 [25] 3))" End Begin PaneConfiguration = 2 NumPanes = 3 Configuration = "(H (1 [50] 2 [25] 3))" End Begin PaneConfiguration = 3 NumPanes = 3 Configuration = "(H (4 [30] 2 [40] 3))" End Begin PaneConfiguration = 4 NumPanes = 2 Configuration = "(H (1 [56] 3))" End Begin PaneConfiguration = 5 NumPanes = 2 Configuration = "(H (2 [66] 3))" End Begin PaneConfiguration = 6 NumPanes = 2 Configuration = "(H (4 [50] 3))" End Begin PaneConfiguration = 7 NumPanes = 1 Configuration = "(V (3))" End Begin PaneConfiguration = 8 NumPanes = 3 Configuration = "(H (1[56] 4[18] 2))" End Begin PaneConfiguration = 9 NumPanes = 2 Configuration = "(H (1 [75] 4))" End Begin PaneConfiguration = 10 NumPanes = 2 Configuration = "(H (1[66] 2))" End Begin PaneConfiguration = 11 NumPanes = 2 Configuration = "(H (4 [60] 2))" End Begin PaneConfiguration = 12 NumPanes = 1 Configuration = "(H (1))" End Begin PaneConfiguration = 13 NumPanes = 1 Configuration = "(V (4))" End Begin

	PaneConfiguration = 14 NumPanes = 1 Configuration = "(V (2))" End ActivePaneConfig = 0 End Begin DiagramPane = Begin Origin = Top = 0 Left = 0 End Begin Tables = Begin Table = "OrderDetails" Begin Extent = Top = 6 Left = 38 Bottom = 135 Right = 208 End DisplayFlags = 280 TopColumn = 0 End Begin Table = "Products" Begin Extent = Top = 6 Left = 457 Bottom = 135 Right = 627 End DisplayFlags = 280 TopColumn = 0 End End End Begin SQLPane = End Begin DataPane = Begin ParameterDefaults = "" End Begin ColumnWidths = 9 Width = 284 Width = 1500 Width = 2820 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 End End Begin CriteriaPane = Begin ColumnWidths = 12 Column = 1440 Alias = 2010 Table = 1170 Output = 720 Append = 1400 NewValue = 1170 SortType = 1350 SortOrder = 1410 GroupBy = 1350 Filter = 4590 Or = 1350 Or = 1350 Or = 1350 End EndEnd
MS_DiagramPaneCount	1

SQL

```
CREATE VIEW dbo.ViewAlsoPurchased
AS
SELECT       TOP (5) dbo.OrderDetails.ProductID, dbo.Products.ModelName,
SUM(dbo.OrderDetails.Quantity) AS TotalNumPurchased, dbo.OrderDetails.OrderID,
                     dbo.Products.ProductID AS Products_ProductID
FROM          dbo.OrderDetails INNER JOIN
                     dbo.Products ON dbo.OrderDetails.ProductID = dbo.Products.ProductID
WHERE       (dbo.OrderDetails.OrderID IN
                         (SELECT DISTINCT OrderID
                           FROM          dbo.OrderDetails AS OrderDetailsSelected))
GROUP BY dbo.OrderDetails.ProductID, dbo.Products.ModelName, dbo.OrderDetails.OrderID,
dbo.Products.ProductID
ORDER BY TotalNumPurchased DESC
```

View: dbo.ViewCart

View: dbo.ViewCart

Properties

Owner:	dbo
Schema bound:	☐
Encrypted:	☐
Creation Date:	02/17/2010
Modification Date:	02/24/2010
Description:	

Columns

Name	Data Type	Length	NULL	IsGUID	Description
ProductID	int	4	☐	☐	
ModelNumber	nvarchar	50	☑	☐	
ModelName	nvarchar	50	☑	☐	
UnitCost	money	8	☐	☐	
Quantity	int	4	☐	☐	
CartID	nvarchar	50	☑	☐	

Objects that [dbo].[ViewCart] depends on

Object Name	Owner	Object Type	Dep Level
Categories	dbo	Table	1
Products	dbo	Table	2
ShoppingCart	dbo	Table	3

Extended Properties

Name	Value
MS_DiagramPane1	[0E232FF0-B466-11cf-A24F-00AA00A3EFFF, 1.00]Begin DesignProperties = Begin PaneConfigurations = Begin PaneConfiguration = 0 NumPanes = 4 Configuration = "(H (1[40] 4[20] 2[20] 3))" End Begin PaneConfiguration = 1 NumPanes = 3 Configuration = "(H (1 [50] 4 [25] 3))" End Begin PaneConfiguration = 2 NumPanes = 3 Configuration = "(H (1 [50] 2 [25] 3))" End Begin PaneConfiguration = 3 NumPanes = 3 Configuration = "(H (4 [30] 2 [40] 3))" End Begin PaneConfiguration = 4 NumPanes = 2 Configuration = "(H (1 [56] 3))" End Begin PaneConfiguration = 5 NumPanes = 2 Configuration = "(H (2 [66] 3))" End Begin PaneConfiguration = 6 NumPanes = 2 Configuration = "(H (4 [50] 3))" End Begin PaneConfiguration = 7 NumPanes = 1 Configuration = "(V (3))" End Begin PaneConfiguration = 8 NumPanes = 3 Configuration = "(H (1[56] 4[18] 2))" End Begin PaneConfiguration = 9 NumPanes = 2 Configuration = "(H (1 [75] 4))" End Begin PaneConfiguration = 10 NumPanes = 2 Configuration = "(H (1[66] 2))" End Begin PaneConfiguration = 11 NumPanes = 2 Configuration = "(H (4 [60] 2))" End Begin PaneConfiguration = 12 NumPanes = 1 Configuration = "(H (1))" End Begin PaneConfiguration = 13 NumPanes = 1 Configuration = "(V (4))" End Begin PaneConfiguration = 14 NumPanes = 1 Configuration = "(V (2))" End ActivePaneConfig = 0 End Begin DiagramPane = Begin Origin = Top = 0 Left = 0 End Begin Tables = Begin Table = "Products" Begin Extent = Top = 6 Left = 38 Bottom = 135 Right = 208 End DisplayFlags = 280 TopColumn = 0 End Begin Table = "ShoppingCart" Begin Extent = Top = 6 Left = 246 Bottom = 135 Right = 416 End DisplayFlags = 280 TopColumn = 0 End End End Begin SQLPane = End

	Begin DataPane = Begin ParameterDefaults = "" End Begin ColumnWidths = 9 Width = 284 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 Width = 1500 End End Begin CriteriaPane = Begin ColumnWidths = 11 Column = 5535 Alias = 900 Table = 2790 Output = 720 Append = 1400 NewValue = 1170 SortType = 1350 SortOrder = 1410 GroupBy = 1350 Filter = 1350 Or = 1350 Or = 1350 Or = 1350 End EndEnd
MS_DiagramPaneCount	1

SQL

```
CREATE VIEW dbo.ViewCart
AS
SELECT        TOP (100) PERCENT dbo.Products.ProductID, dbo.Products.ModelNumber,
dbo.Products.ModelName, dbo.Products.UnitCost, dbo.ShoppingCart.Quantity,
                         dbo.ShoppingCart.CartID
FROM          dbo.Products INNER JOIN
                         dbo.ShoppingCart ON dbo.Products.ProductID = dbo.ShoppingCart.ProductID AND
dbo.Products.ProductID = dbo.ShoppingCart.ProductID
ORDER BY dbo.Products.ModelName, dbo.Products.ModelNumber
```

Stored Procedures

Stored Procedure: dbo.SelectPurchasedWithPr oducts

Stored Procedure: dbo.SelectPurchasedWithPr oducts **Properties**

Owner: dbo

Encrypted: ☐

Creation Date: 02/25/2010

Modification Date:

Description:

Stored Procedure: dbo.SelectPurchasedWithPr oducts

Creation options

QUOTED_IDENTIFIER: ☑

ANSI_NULLS: ☑

Parameters

Name	Direction	DataType	Length	Default	Description
@ProductID	INPUT	int	4		

Objects that [dbo].[SelectPurchasedWithProducts] depends on

Object Name	Owner	Object Type	Dep Level
Categories	dbo	Table	1
Orders	dbo	Table	1
OrderDetails	dbo	Table	2
Products	dbo	Table	2

SQL

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
CREATE 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


GO
SET QUOTED_IDENTIFIER OFF
GO

GO
SET ANSI_NULLS OFF
GO
```

Server Info

Server Info

Properties

Network Name:	
Processor Count:	2
Physical Memory, MB:	2518
Clustered:	☐
Login Mode:	Mixed
CodePage:	iso_1
Language:	SQL_Latin1_General_CP1_CI_AS

Version

Microsoft SQL Server 2008 (SP1) - 10.0.2531.0 (Intel X86)
Mar 29 2009 10:27:29
Copyright (c) 1988-2008 Microsoft Corporation
Express Edition on Windows NT 6.1 <X86> (Build 7600:)

Databases

Database	Owner	System
COMMERCE.MDF		☐

Server Configuration

Name	Value
allow updates	0
clr enabled	0
cross db ownership chaining	0
default language	0
filestream access level	0
max text repl size (B)	65536
nested triggers	1
remote access	1
remote admin connections	0
remote login timeout (s)	20
remote proc trans	0
remote query timeout (s)	600
server trigger recursion	1
show advanced options	0
user instances enabled	1
user options	0

About

Documentation details

Customer:	Microsoft
Project:	Tailspin Spyworks
Created:	09/30/2010