

## Lesson 18: Northwind Store

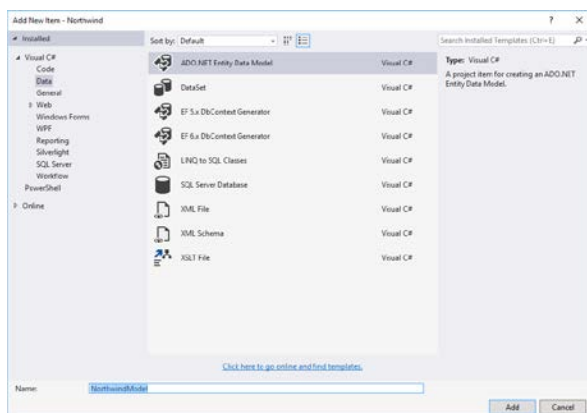
In this lesson, we will begin creating an enterprise level web application using the Northwnd database from Microsoft. We will utilize database-first entity design for this project.

### Agenda

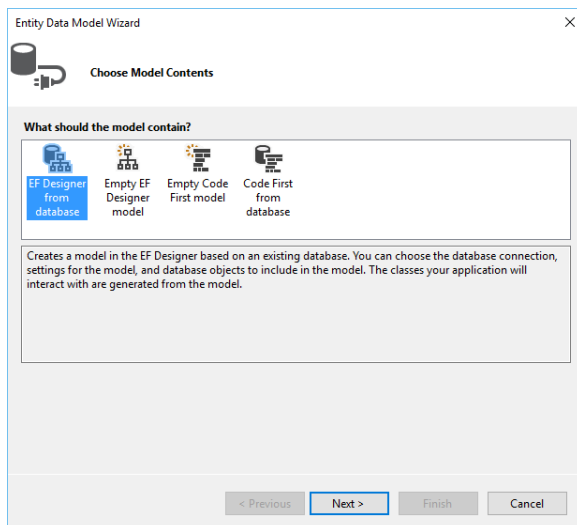
1. Demonstration
2. Lab/Homework

### Demonstration

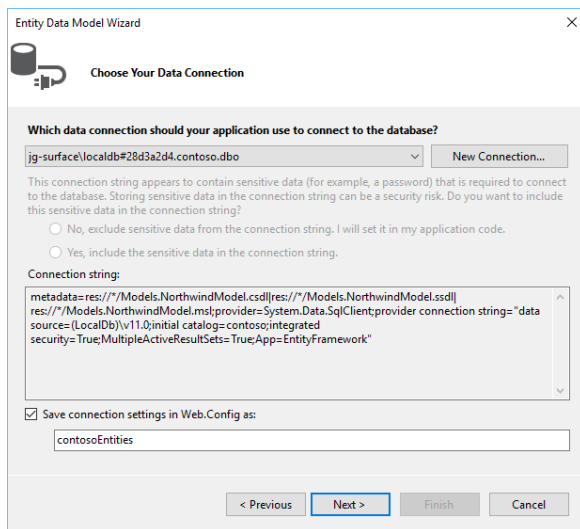
1. Open the **Northwind** project.
2. In Visual Studio, right-click the **Models** folder in the Solution Explorer. Select **Add – New Item**.
3. In the Add New Item dialog, choose the **Data – ADO.NET Entity Data Model**. Enter **NorthwindModel** as the name. Click **Add**.



4. Select **EF Designer from database**. Click **Next**.

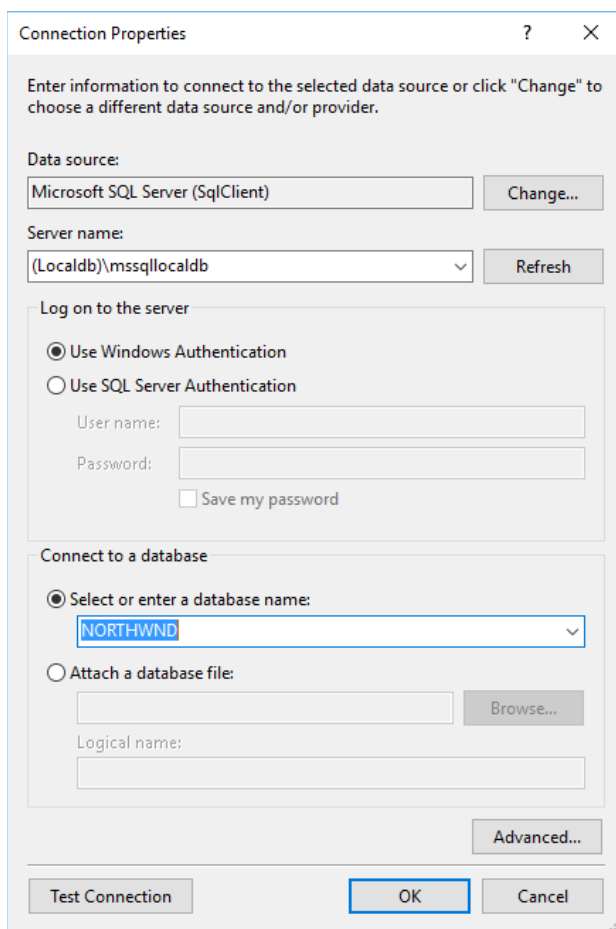


5. Click **New Connection**.



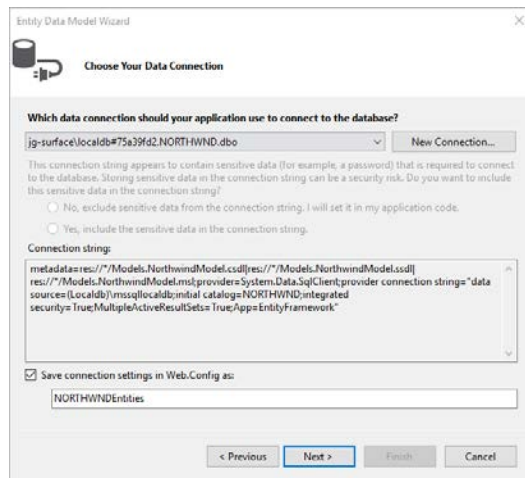
The dialog box is titled "Entity Data Model Wizard" and "Choose Your Data Connection". It asks "Which data connection should your application use to connect to the database?". A dropdown menu shows "ig-surface\localdb#28d3a2d4.contoso.dbo" with a "New Connection..." button next to it. Below this, a warning message states: "This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?". There are two radio buttons: "No, exclude sensitive data from the connection string. I will set it in my application code." (selected) and "Yes, include the sensitive data in the connection string.". A text area labeled "Connection string:" contains the following text: `metadata=res://*/Models.NorthwindModel.csdl|res://*/Models.NorthwindModel.ssdl|res://*/Models.NorthwindModel.msl;provider=System.Data.SqlClient;provider connection string="data source=(LocalDb)\v11.0;initial catalog=contoso;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"`. Below the text area is a checkbox "Save connection settings in Web.Config as:" which is checked, with a text box containing "contosoEntities". At the bottom are buttons: "< Previous", "Next >" (highlighted), "Finish", and "Cancel".

6. In the Connection Properties dialog, enter the **server name** and select the **database**. Click **OK**.



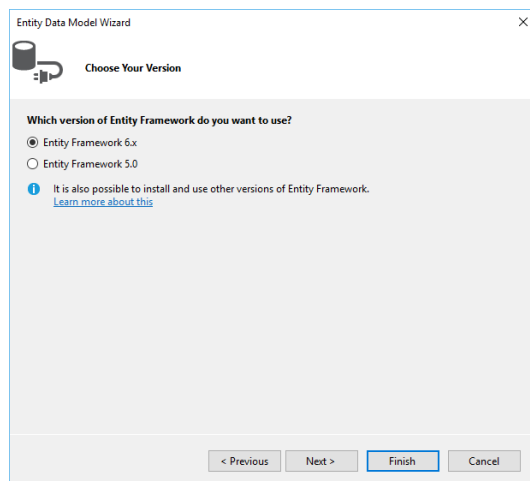
The dialog box is titled "Connection Properties". It contains the instruction: "Enter information to connect to the selected data source or click 'Change' to choose a different data source and/or provider." There are three sections: 1. "Data source:" with a dropdown showing "Microsoft SQL Server (SqlClient)" and a "Change..." button. 2. "Server name:" with a dropdown showing "(Localdb)\mssqllocaldb" and a "Refresh" button. 3. "Log on to the server" with two radio buttons: "Use Windows Authentication" (selected) and "Use SQL Server Authentication". Below these are fields for "User name:" and "Password:", and a checkbox "Save my password". There is also a "Connect to a database" section with two radio buttons: "Select or enter a database name:" (selected) and "Attach a database file:". The "Select or enter a database name:" option has a dropdown showing "NORTHWND". The "Attach a database file:" option has a "Browse..." button and a "Logical name:" field. At the bottom right is an "Advanced..." button. At the bottom are buttons: "Test Connection", "OK" (highlighted), and "Cancel".

7. Note the connections string name. Click **Next**. We will use this later. This is the conduit between our controller class and our data using the Entity Framework.



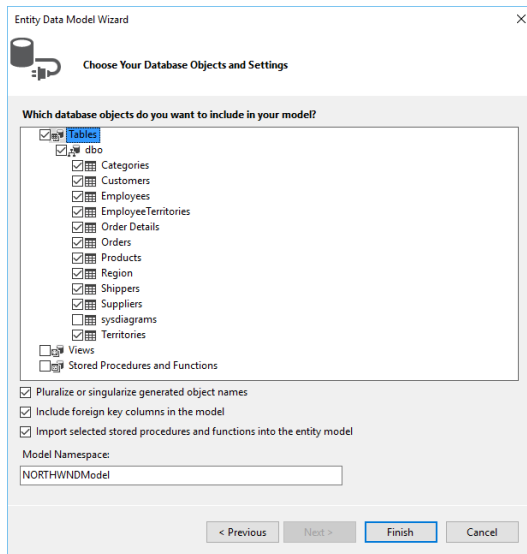
The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Data Connection' step. The window has a title bar with 'Entity Data Model Wizard' and a close button. Below the title bar is a header area with a database icon and the text 'Choose Your Data Connection'. The main content area is titled 'Which data connection should your application use to connect to the database?'. It features a dropdown menu with the selected connection string 'jg-surface\localdb#75a39fd2.NORTHWND.dbo' and a 'New Connection...' button. Below this, a warning message states: 'This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?'. There are two radio buttons: 'No, exclude sensitive data from the connection string. I will set it in my application code.' (which is selected) and 'Yes, include the sensitive data in the connection string.'. Below the radio buttons is a text box labeled 'Connection string:' containing the following text: 'metadata=res://\*/Models\NorthwindModel.csdl|res://\*/Models\NorthwindModel.ssdl|res://\*/Models\NorthwindModel.msl;provider=System.Data.SqlClient;provider connection string="data source=(localdb)\mssqllocaldb;initial catalog=NORTHWND;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"'. At the bottom, there is a checkbox 'Save connection settings in Web.Config as:' which is checked, and a text box containing 'NORTHWNDEntities'. Navigation buttons at the bottom include '< Previous', 'Next >', 'Finish', and 'Cancel'.

8. Select the Entity Framework version. Click **Next**.



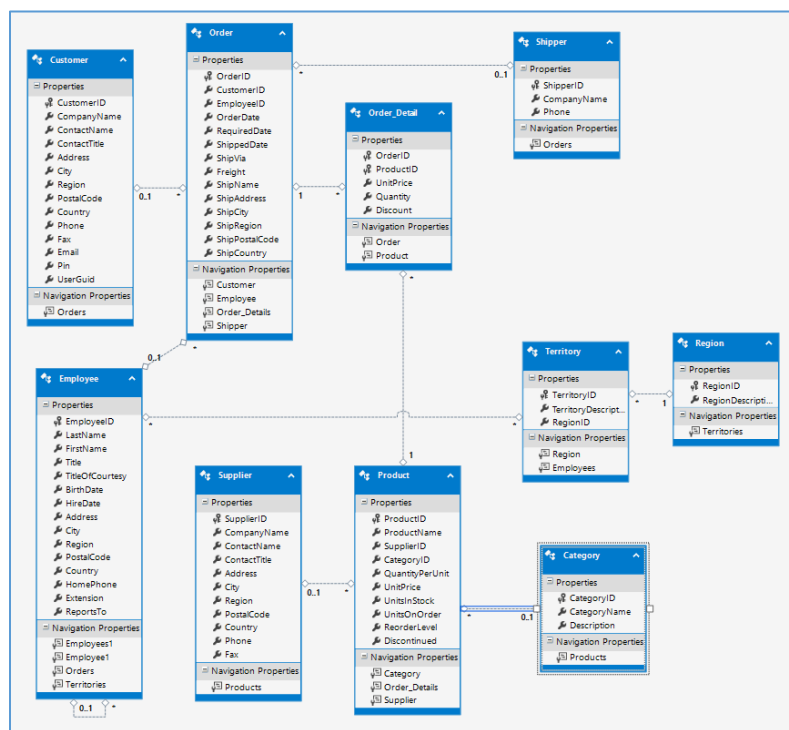
The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Version' step. The window has a title bar with 'Entity Data Model Wizard' and a close button. Below the title bar is a header area with a database icon and the text 'Choose Your Version'. The main content area is titled 'Which version of Entity Framework do you want to use?'. It features two radio buttons: 'Entity Framework 6.x' (which is selected) and 'Entity Framework 5.0'. Below the radio buttons is an information icon (i) followed by the text 'It is also possible to install and use other versions of Entity Framework.' and a link 'Learn more about this'. At the bottom, there are navigation buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'.

9. Select all Tables (except sysdiagrams). Click **Finish**.

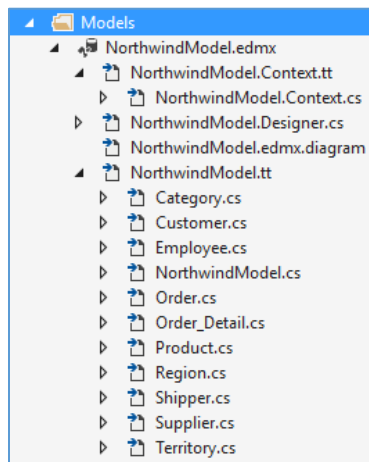


The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Database Objects and Settings' step. The 'Which database objects do you want to include in your model?' section has 'Tables' selected. Under 'dbo', the following tables are checked: Categories, Customers, Employees, EmployeeTerritories, Order Details, Orders, Products, Region, Shippers, Suppliers, and Territories. 'sysdiagrams' is unchecked. Under 'Views', 'Stored Procedures and Functions' are also unchecked. The 'Model Namespace' is set to 'NORTHWNDModel'. At the bottom, the 'Finish' button is highlighted.

10. Examine the database schema.



11. Examine the model classes created from the database. In today's lesson, we will be using the Product and Category tables. There is a 1-to-many relationship between the tables (a product can belong to 1 category, a category can contain many products).



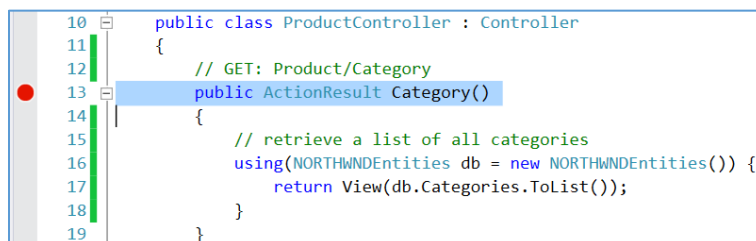
12. Open the Product controller. Add the Northwind.Models namespace to the controller.

```
using Northwind.Models;
```

13. Let's create a list all of the Product Categories. We utilize LINQ to access our data. Notice the NORTHWNDEntities class. This is the conduit between our controller class and our data. We now have access to all of the data model classes and the database using the Entity Framework.

```
// GET: Product/Category
public ActionResult Category()
{
    // retrieve a list of all categories
    using(NORTHWNDEntities db = new NORTHWNDEntities()) {
        return View(db.Categories.ToList());
    }
}
```

14. That's all we need. We are passing a list of Categories to the View. Set a breakpoint and debug.



15. The last thing we should do is sort the list by name. Modify the return statement. Test in browser.

```
return View(db.Categories.OrderBy(c => c.CategoryName).ToList());
```

16. Open the Views/Product/Category view. Import the Northwind.Models namespace and add the model to the view.

```
@using Northwind.Models
@model IEnumerable<Category>
```

17. Modify the title, iterate thru the Category collection and add a link back to the home page. Test in browser.

```
@{
    ViewBag.Title = "Products";
}

<ul>
    @foreach (Category c in @Model)
    {
        <li>@c.CategoryName</li>
    }
</ul>
<div class="font-md">
    <a href="~/Home/Index"><i class="fa fa-home"></i> Home</a>
</div>
```

18. Let's spiff up the page a bit. Replace the ul with a Bootstrap list-group. Notice we are displaying the category name and the category description. Test in browser.

```
<div class="font-md">Product Categories</div>
<!-- List group -->
<div class="list-group">
    @foreach (Category c in Model)
    {
        <a href="#" class="list-group-item">
            <strong>@c.CategoryName</strong> - @c.Description
        </a>
    }
</div>

<div class="font-md">
    <a href="~/Home/Index"><i class="fa fa-home"></i> Home</a>
</div>
```

19. Ultimately, clicking a category in the list should open a view that displays the products from that category. Modify the href attribute. Notice we are passing the CategoryID to the Controller.

```
href="/~/Product/Product/@c.CategoryID">
```

20. Open the Product Controller. Import the System.Net namespace and add the Product method. Create the related Product view. Test in browser.

```
// GET: Product/Product/1
public ActionResult Product(int? id)
{
    // if there is no "category" id, return Http Bad Request
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    return View();
}
```

21. Pass a list of products to the View and save the category to the ViewBag. Replace the return statement.

```
using (NORTHWNDEntities db = new NORTHWNDEntities())
{
    // save the selected category name to the ViewBag
    ViewBag.Filter = db.Categories.Find(id).CategoryName;
    // retrieve list of products
    return View(db.Products.ToList());
}
```

22. Modify the Views\Product\Product view. Test in browser.

```
@using Northwind.Models
@model IEnumerable<Product>

@{
    ViewBag.Title = "Product";
}

<ul>
    @foreach (Product p in Model)
    {
        <li>@p.ProductName</li>
    }
</ul>
```

23. We have not yet accessed the ViewBag.CategoryName. We will use that later.

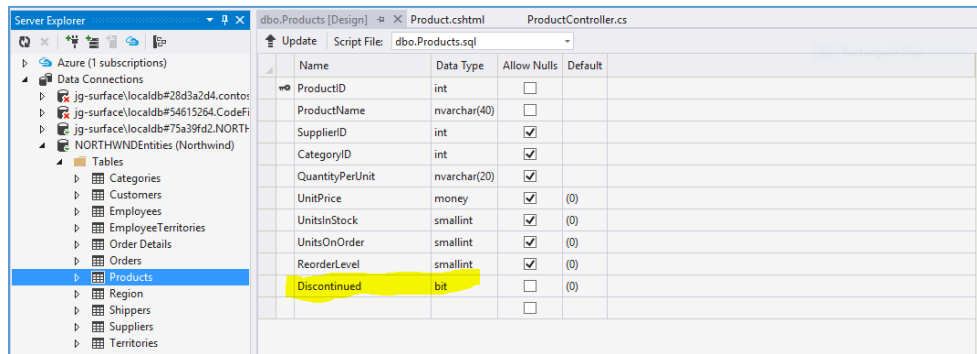
24. This is an unfiltered, unsorted list of all products. Let's sort the list by product name. Open the Product Controller and modify the return statement. Test in browser.

```
return View(db.Products.OrderBy(p => p.ProductName).ToList());
```

25. This is a sorted list of all products. Next we will filter the list using the selected category. Modify the return statement. Test in browser.

```
return View(db.Products.Where(p => p.CategoryID == id).OrderBy(p => p.ProductName).ToList());
```

26. This is almost perfect. If we examine the Products table closely, we see a Boolean field to indicate discontinued products. We don't want to display discontinued products here. We need to filter our list to exclude discontinued products.



Name	Data Type	Allow Nulls	Default
ProductID	int	<input type="checkbox"/>	
ProductName	nvarchar(40)	<input type="checkbox"/>	
SupplierID	int	<input checked="" type="checkbox"/>	
CategoryID	int	<input checked="" type="checkbox"/>	
QuantityPerUnit	nvarchar(20)	<input checked="" type="checkbox"/>	
UnitPrice	money	<input checked="" type="checkbox"/>	(0)
UnitsInStock	smallint	<input checked="" type="checkbox"/>	(0)
UnitsOnOrder	smallint	<input checked="" type="checkbox"/>	(0)
ReorderLevel	smallint	<input checked="" type="checkbox"/>	(0)
Discontinued	bit	<input type="checkbox"/>	(0)

27. Modify the return statement. Test in Browser.

```
return View(db.Products.Where(p => p.CategoryID == id && p.Discontinued == false).OrderBy(p => p.ProductName).ToList());
```

28. Let's spiff up the display a bit now. Open the Views\Product\Product view and replace the ul with a table and add a link back to the Category List.

```
<table class="table table-hover table-responsive">
  <thead>
    <tr class="font-md">
      <th>@ViewBag.Filter</th>
      <th class="text-right">
        Price
      </th>
      <th class="text-right">
        Stock
      </th>
    </tr>
  </thead>
  <tbody>
```



```

    @foreach (Product p in Model)
    {
        <tr class="product-row" id="@p.ProductID">
            <td>@p.ProductName (@p.QuantityPerUnit)</td>
            <td class="text-right">@string.Format("{0:c}", p.UnitPrice)</td>
            <td class="text-right">@p.UnitsInStock</td>
        </tr>
    }
</tbody>
</table>

<div class="font-md">
    <a href="~/Product/Category"><i class="fa fa-list"></i> Product Categories</a>
</div>

```

29. Ultimately, when a product is selected, we will add it to the customer's shopping basket. For now, let's display a javascript alert when a product is selected. **Notice the id attribute of each row in tbody.** Add the scripts section after the Product Categories link.

```

@section scripts
{
    <script>
        $(function () {
            $('<tr class="product-row">').click(function () {
                alert(this.id);
            });
        });
    </script>
}

```

30. Clicking a product's row will now display the ProductID of that product. Notice the cursor when you hover your mouse over a row. We want the cursor to be the same as if you were hovering over an anchor tag. Open the Content\Site.css. Add the product-row class selector. Test in browser.

```

.product-row{ cursor:pointer; }

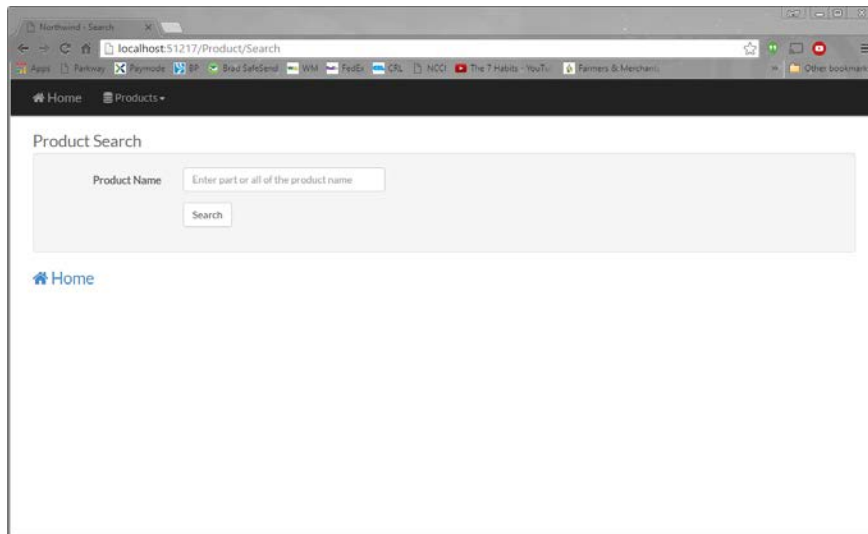
```

## Homework

### REMINDER: AZURE Account

Implement the product search feature. This will be similar to the demonstration from today. Instead of filtering the list of products by category, you will be filtering by product name using LINQ's Contains method.

- 1) First, create a new controller method in the Product controller.
- 2) Create the related search view (see below).



- 3) Prevent Cross Site Request Forgery (CSRF) attacks by implementing AntiForgeryToken() helper. Read the following article for an explanation. <http://blog.stevensanderson.com/2008/09/01/prevent-cross-site-request-forgery-csrf-using-aspnet-mvcs-antiforgerytoken-helper/>. The article does a good job of explaining CSRF and the AntiForgeryToken() helper, however, the sample code is NOT Razor. See below for a code example in Razor.

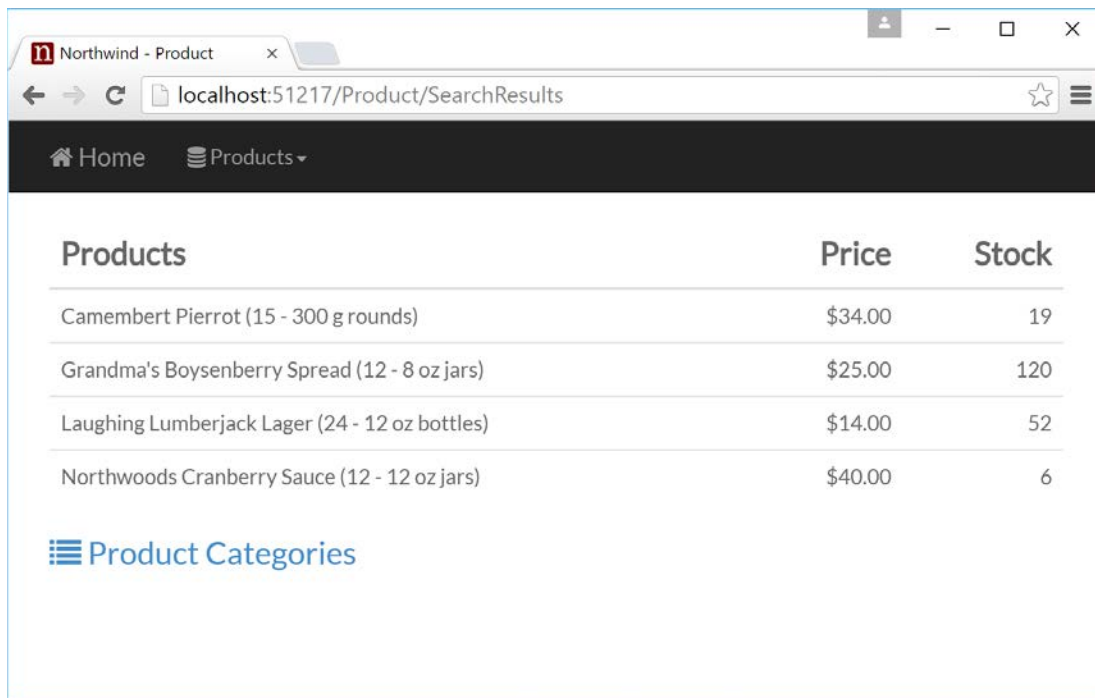
#### View:

```
<form method="post" action="~/Product/SearchResults">
  @Html.AntiForgeryToken()
  ...
</form>
```

#### Controller:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult SearchResults(FormCollection Form)
{
  ...
}
```

4) Results should be displayed in THE SAME VIEW as in the lesson (Product.cshtml).



The screenshot shows a web browser window with the title 'Northwind - Product'. The address bar displays 'localhost:51217/Product/SearchResults'. The browser has a dark navigation bar with 'Home' and 'Products' links. Below this, a table lists products with columns for 'Products', 'Price', and 'Stock'. The table contains four rows of product data. Below the table, there is a link for 'Product Categories'.

Products	Price	Stock
Camembert Pierrot (15 - 300 g rounds)	\$34.00	19
Grandma's Boysenberry Spread (12 - 8 oz jars)	\$25.00	120
Laughing Lumberjack Lager (24 - 12 oz bottles)	\$14.00	52
Northwoods Cranberry Sauce (12 - 12 oz jars)	\$40.00	6

[Product Categories](#)