

Customization by Adding Your Own Code

1 CONTENTS

- 1 Introduction 2
 - 1.1 Read these tutorials in order 2
- 2 Adding New Files 2
 - 2.1 Where Can You Add Files? 2
 - 2.2 What Files Can You Add? 3
 - 2.3 Why Add New Files? 3
- 3 Adding Code to a Generated File 3
- 4 End-to-End Example on Adding Your Own File and Code 4
 - 4.1 Generate Code Using AspCoreGen 3.0 Razor Professional Plus 4
 - 4.2 The Tutorial..... 4

Customization by Adding Your Own Code

1 INTRODUCTION

This topic will show you how to add your own code to the ASP.NET Core 3.0 Razor's generated code.

1.1 READ THESE TUTORIALS IN ORDER

1. Database Settings Tab
2. Code Settings Tab
3. UI Settings Tab
4. App Settings Tab
5. Selected Tables Tab
6. Selected Views Tab
7. Generating Code
8. The Generated Code for Database Tables
9. The Generated Code for Database Views

Then follow these step-by-step instructions.

2 ADDING NEW FILES

Unlike the older versions, you can now add new files to any of the generated projects.

2.1 WHERE CAN YOU ADD FILES?

You can add files to the following generated projects:

1. Web Application Project (ASP.NET Core Razor Pages)
2. Business Layer and Data Layer Project (Class Library).
3. Web API Project (ASP.NET Core MVC API)

2.2 WHAT FILES CAN YOU ADD?

Any file that is permissible by the respective projects listed above (*see 2.1*). For example, for an ASP.NET Core Razor Page project you can add a/an:

1. Razor Page
2. Razor Page Model
3. Partial Razor Page
4. Partial Razor Page Model
5. Class Files
6. Images
7. CSS Files
8. JavaScript Files
9. And many, many more

2.3 WHY ADD NEW FILES?

You don't have to add new files, but, if you want to, you can.

Most of the time you may want to add functionality to a generated *Razor Page*. **You should not do this because it will just get overwritten when you regenerate code for the same project.** Instead add a new *Razor Page* to the project and you can name it distinct name, e.g. *MyNewPage.cshtml*.

3 ADDING CODE TO A GENERATED FILE

You can add your own customized code in some of the generated files. This is discussed in the *App Settings Tab* tutorial. Please read the *App Settings Tab* tutorial to see the list of generated files where you can add your own code to, **these files will not be overwritten even when you regenerate code for the same project.**

4 END-TO-END EXAMPLE ON ADDING YOUR OWN FILE AND CODE

In here we'll show you how to add files to the generated projects, and also add your own code to existing generated files.

4.1 GENERATE CODE USING ASPCOREGEN 3.0 RAZOR PROFESSIONAL PLUS

You can generate your own Web Application using AspCoreGen 3.0 Razor Professional Plus and just follow along with this tutorial. Make sure to:

1. Choose *Use Stored Procedures* under the *Generated SQL* in the *Database Settings* tab.
2. Choose *All Tables* or *Selected Tables Only* under the *Database Objects to Generate From* in the *Code Settings* tab.
3. Check the *Use Web API* under the *Web API* in the *Code Settings* tab.

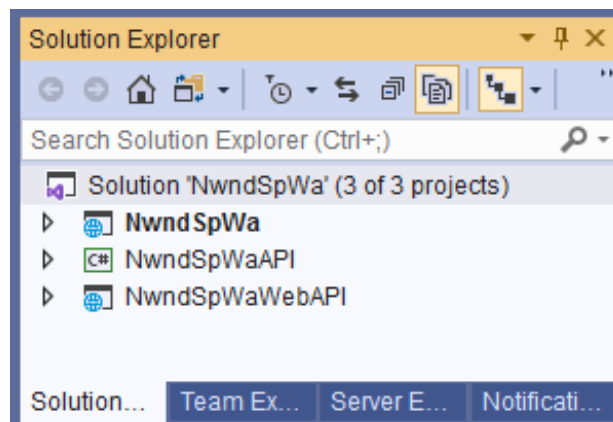
Alternatively, you can download the sample *Generated Web Project Example* from our website:

<https://junnark.com/Products/AspCoreGen3Razor/GeneratedProjects>. Download #4, the *Stored Procedures Using Web API Sample Project*. Unzip the downloaded project and make sure to follow the instructions in the *Readme.txt* file.

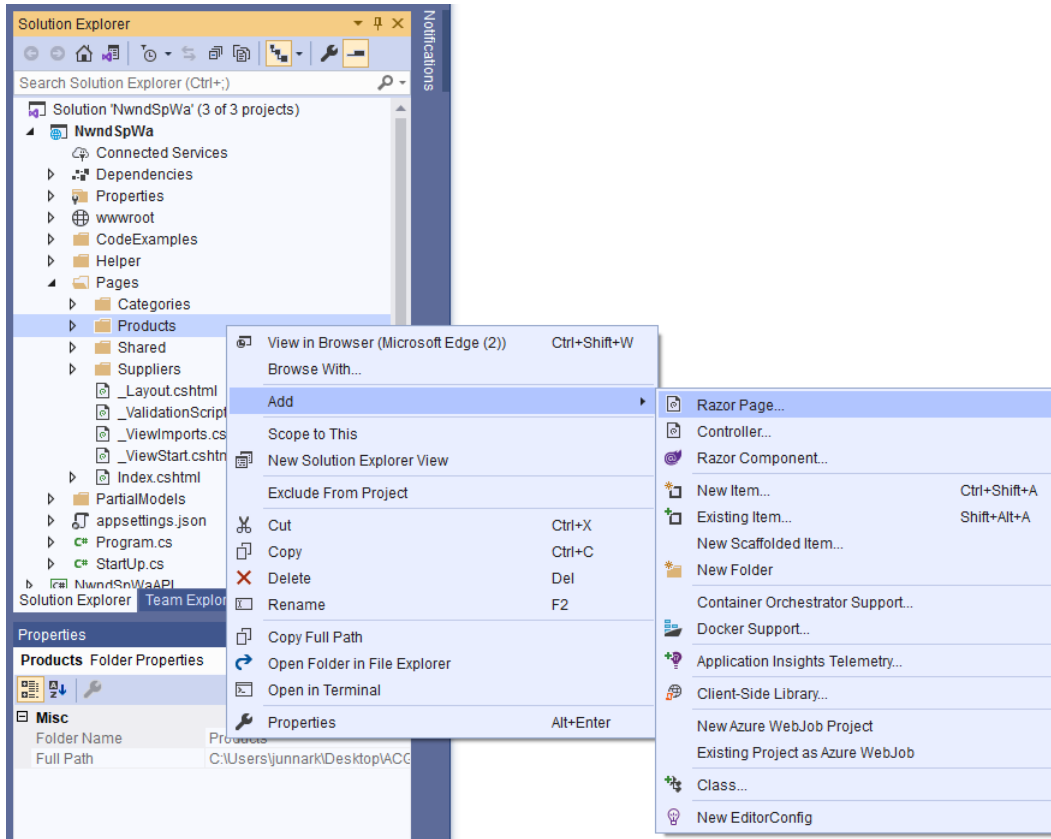
4.2 THE TUTORIAL

In this tutorial we're going to create a new *MVC View* that is similar to the *ListCrudRedirect.cshtml*, but we will add a functionality that shows the *Supplier Name* and *Category Name* instead of the *Supplier ID* and *Category ID* respectively. We will also remove the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* columns for display.

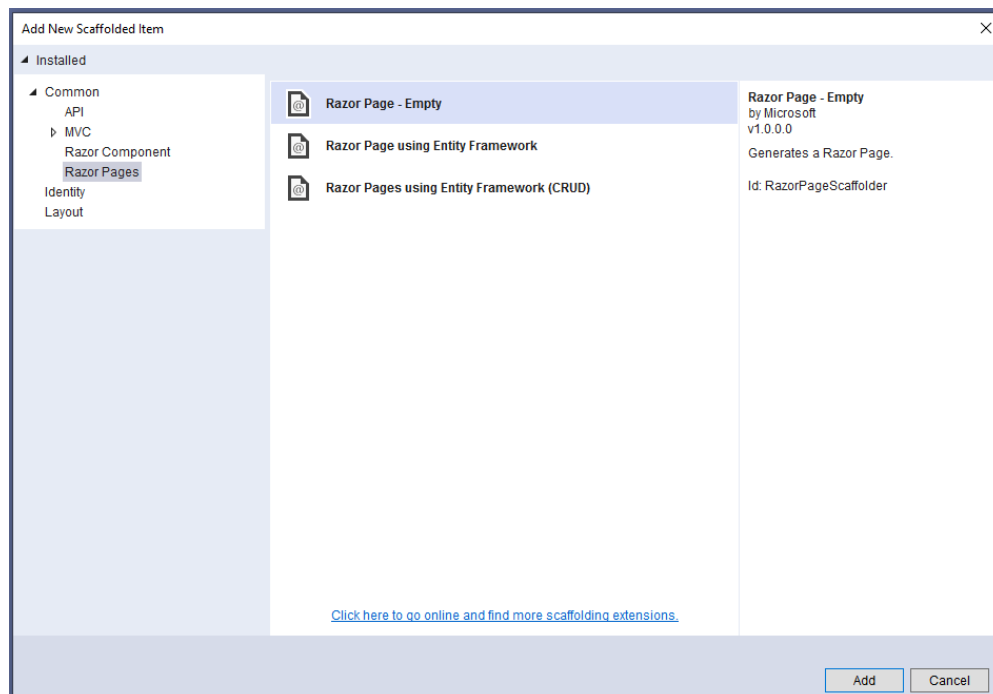
1. Open the *Generated Web Application (NwndSpWa.sln)* in *Visual Studio 2019*. This solution should have 3 projects: The *Web Application (NwndSpWa)*, the *Class Library (NwndSpWaAPI)*, and the *Web API (NwndSpWaWebAPI)* projects.



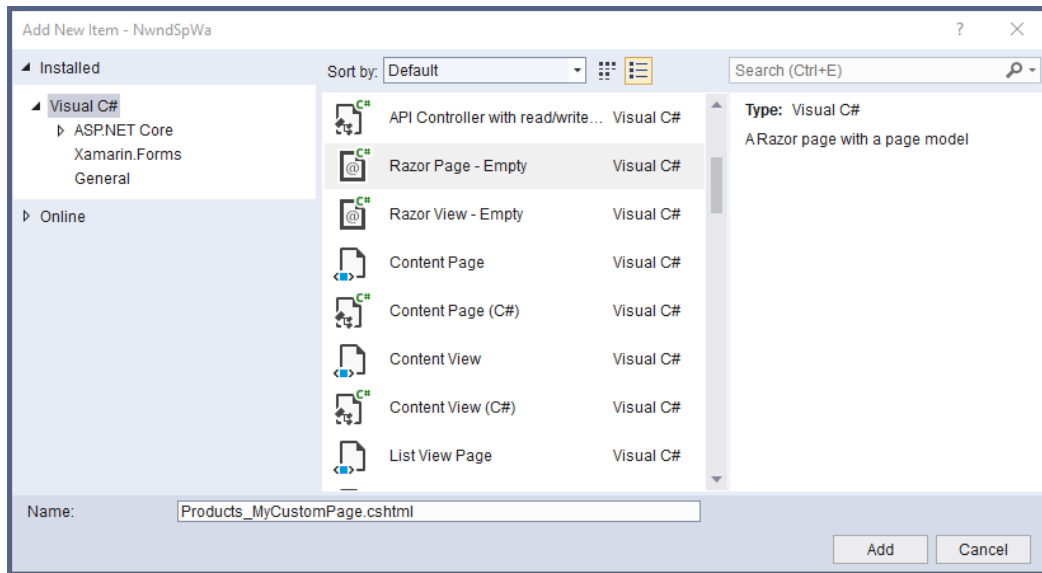
2. Add a new *Razor Page* under the *Products* folder.



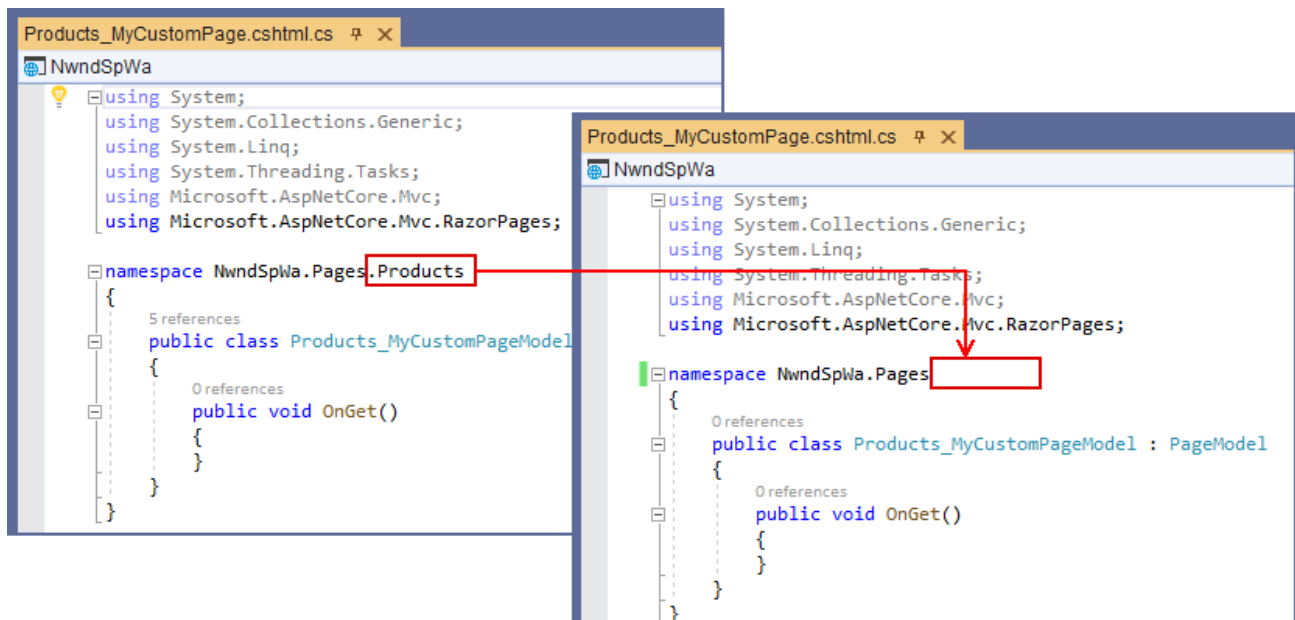
3. Choose *Razor Page - Empty* and click the *Add* button.



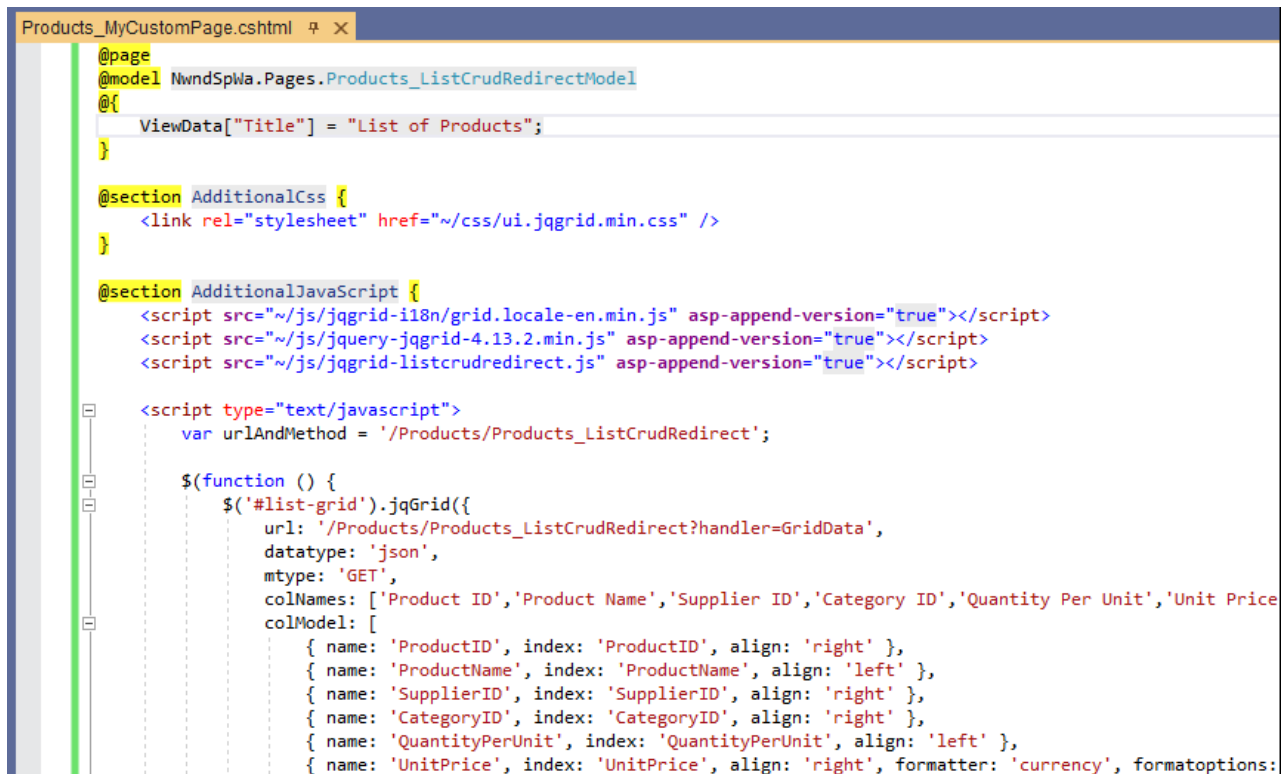
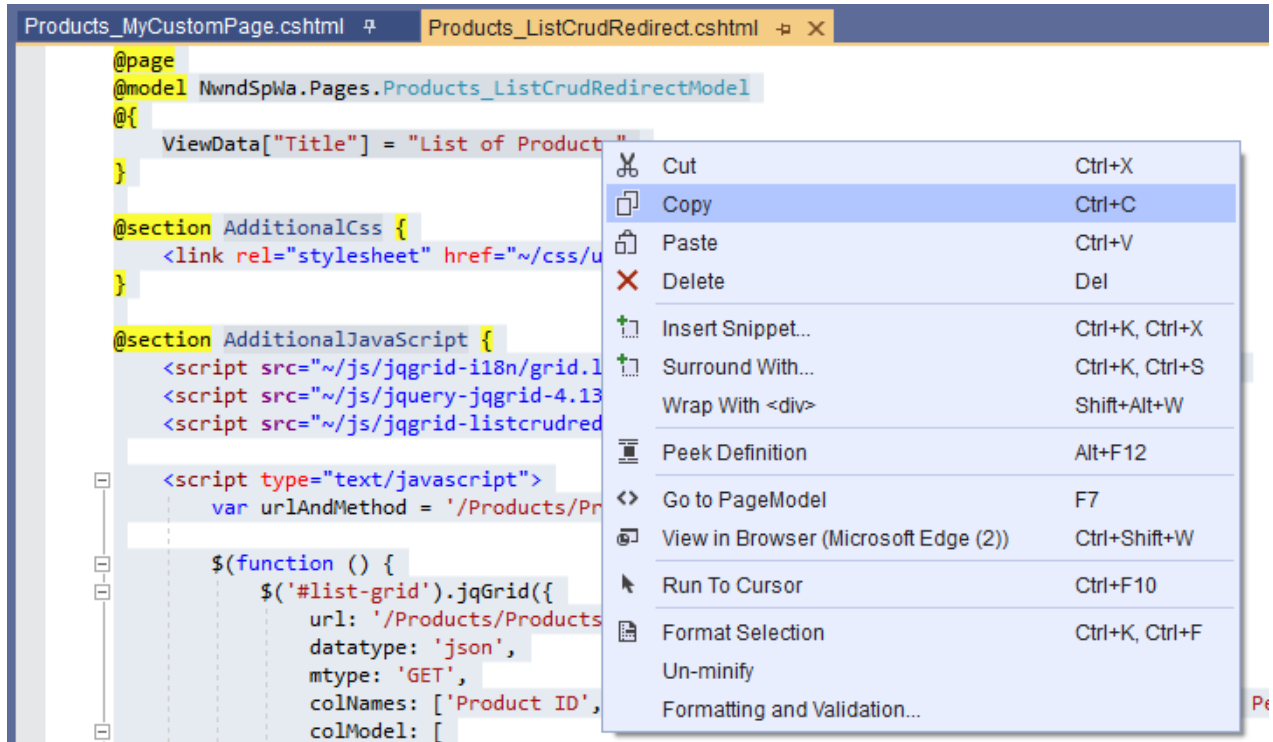
4. Name the new *Razor Page*: *Products_MyCustomPage.cshtml* and then click the *Add* button.



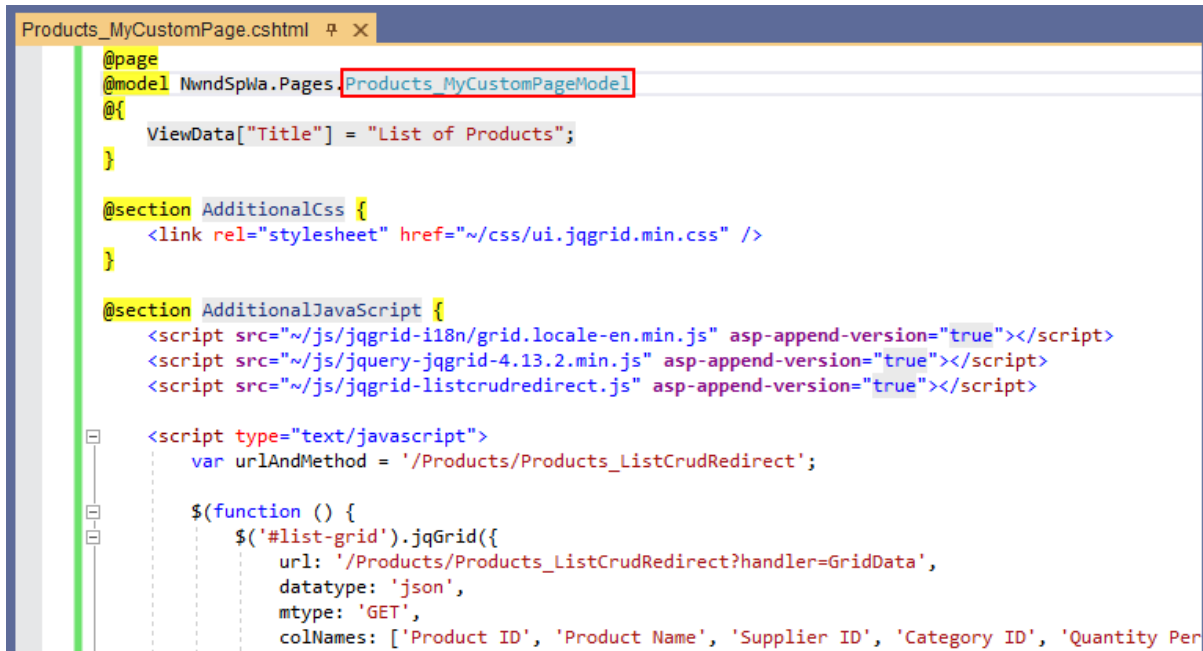
5. Open the *Razor Page Model* (*Products_MyCustomPage.cshtml*) and remove the *“.Products”* in the namespace.



6. Delete all the code in the *Products_MyCustomPage.cshtml*. And then Open the *Products_ListCrudRedirect.cshtml* under the *Products* folder and Copy all code to *Products_MyCustomPage.cshtml*.



7. Update the *Razor Page Model* “*Products_ListCrudRedirectModel*” and replace it with “*Products_MyCustomPageModel*”.



```

Products_MyCustomPage.cshtml
@page
@model NwndSpwa.Pages.Products_MyCustomPageModel
@{
    ViewData["Title"] = "List of Products";
}

@section AdditionalCss {
    <link rel="stylesheet" href="/css/ui.jqgrid.min.css" />
}

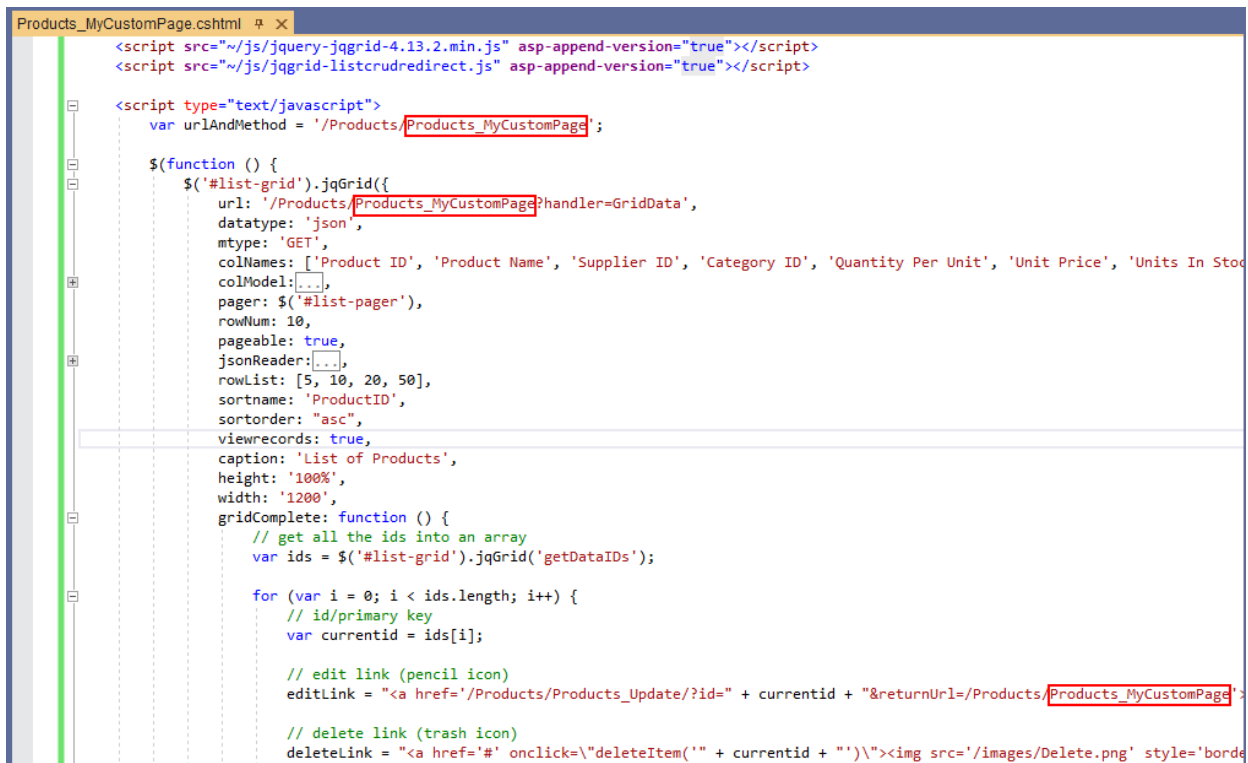
@section AdditionalJavaScript {
    <script src="/js/jqgrid-i18n/grid.locale-en.min.js" asp-append-version="true"></script>
    <script src="/js/jquery-jqgrid-4.13.2.min.js" asp-append-version="true"></script>
    <script src="/js/jqgrid-listcrudredirect.js" asp-append-version="true"></script>

    <script type="text/javascript">
        var urlAndMethod = '/Products/Products_ListCrudRedirect';

        $(function () {
            $('#list-grid').jqGrid({
                url: '/Products/Products_ListCrudRedirect?handler=GridData',
                datatype: 'json',
                mtype: 'GET',
                colNames: ['Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per

```

8. Update a total of 5 occurrences of the *Razor Page* “*Products_ListCrudRedirect*”, make sure to replace these with “*Products_MyCustomPage*”.



```

Products_MyCustomPage.cshtml
<script src="/js/jquery-jqgrid-4.13.2.min.js" asp-append-version="true"></script>
<script src="/js/jqgrid-listcrudredirect.js" asp-append-version="true"></script>

<script type="text/javascript">
    var urlAndMethod = '/Products/Products_MyCustomPage';

    $(function () {
        $('#list-grid').jqGrid({
            url: '/Products/Products_MyCustomPage?handler=GridData',
            datatype: 'json',
            mtype: 'GET',
            colNames: ['Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per Unit', 'Unit Price', 'Units In Stock', 'Units On Order', 'Reorder Level', 'Product Status'],
            colModel: [...],
            pager: $('#list-pager'),
            rowNum: 10,
            pageable: true,
            jsonReader: [...],
            rowList: [5, 10, 20, 50],
            sortname: 'ProductID',
            sortOrder: "asc",
            viewrecords: true,
            caption: 'List of Products',
            height: '100%',
            width: '1200',
            gridComplete: function () {
                // get all the ids into an array
                var ids = $('#list-grid').jqGrid('getDataIDs');

                for (var i = 0; i < ids.length; i++) {
                    // id/primary key
                    var currentid = ids[i];

                    // edit link (pencil icon)
                    editLink = "<a href='/Products/Products_Update/?id=" + currentid + "&returnUrl=/Products/Products_MyCustomPage'";

                    // delete link (trash icon)
                    deleteLink = "<a href='#' onclick='deleteItem('" + currentid + "')'><img src='/images/Delete.png' style='border: 1px solid black; width: 15px; height: 15px; vertical-align: middle;'/>";
                }
            }
        });
    });

```

9. Delete all the code in the Razor Page Model (*Products_MyCustomPage.cshtml.cs*). Then Open the *Products_ListCrudRedirect.cshtml.cs* Razor Page Mode under the *Products* folder and Copy all code to *Products_MyCustomPage.cshtml.cs*.

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using NwndSpWAPI.Domain;
using NwndSpWAPI.BusinessObject;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;

namespace NwndSpWAPI.Pages
{
    public class Products_ListCrudRedirectModel : PageModel
    {
        /// <summary>
        /// Handler, deletes a record.
        /// </summary>
        public async Task<IActionResult> OnGetRemoveAsync(int id)
        {
            // delete data via web api
            HttpResponseMessage response = await Functions.HttpClientDeleteAsync("ProductsApi/Delete/" + id);

            // check the status of the httpclient delete operation
            if (response != null)
            {
                if (response.IsSuccessStatusCode)
                {
                    return new JsonResult(true);
                }
                else
                {
                    return BadRequest(response.ReasonPhrase);
                }
            }

            return BadRequest();
        }

        /// <summary>
        /// GET: /Products/GridData
        /// Gets the json needed by the jqgrid for use by the client
        /// </summary>
    }
}

```

Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Remove and Sort Usings Ctrl+R, Ctrl+G
Go to Page F7
Peek Definition Alt+F12
Go to Definition F12
Go to Base Alt+Home
Go to Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Create Unit Tests
Breakpoint
Run To Cursor Ctrl+F10
Execute in Interactive Ctrl+E, Ctrl+E
Snippet
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using NwndSpWAPI.Domain;
using NwndSpWAPI.BusinessObject;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;

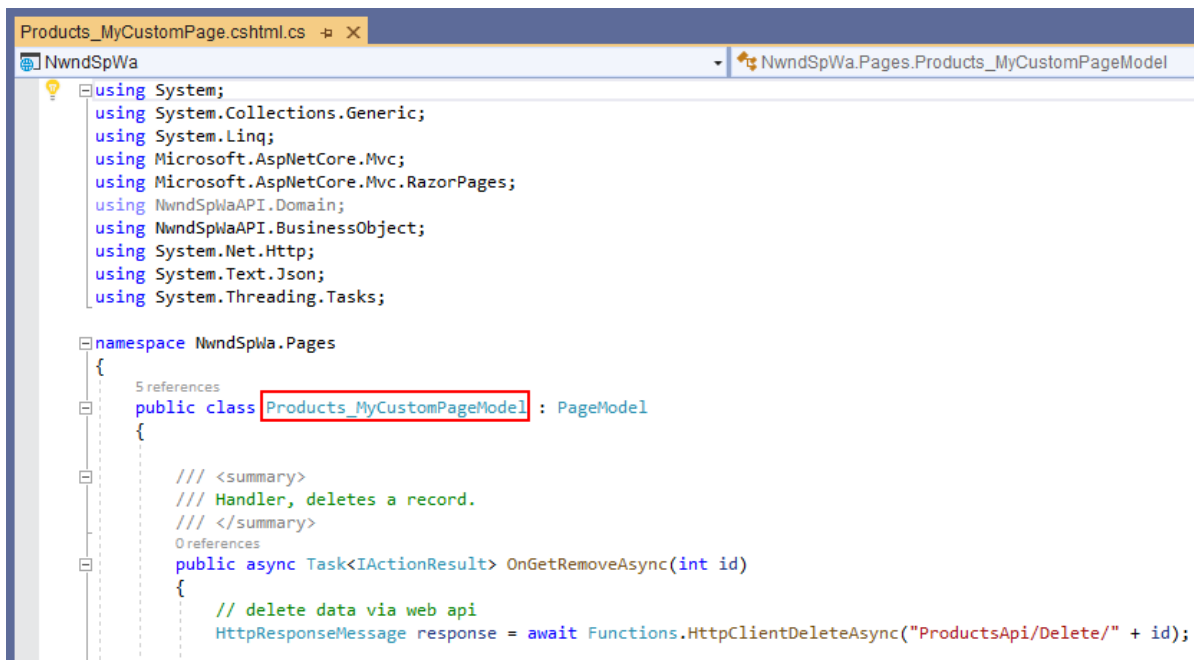
namespace NwndSpWAPI.Pages
{
    public class Products_ListCrudRedirectModel : PageModel
    {
        /// <summary>
        /// Handler, deletes a record.
        /// </summary>
        public async Task<IActionResult> OnGetRemoveAsync(int id)
        {
            // delete data via web api
            HttpResponseMessage response = await Functions.HttpClientDeleteAsync("ProductsApi/Delete/" + id);

            // check the status of the httpclient delete operation
            if (response != null)
            {
                if (response.IsSuccessStatusCode)
                {
                    return new JsonResult(true);
                }
                else
                {
                    return BadRequest(response.ReasonPhrase);
                }
            }

            return BadRequest();
        }
    }
}

```

10. In the *Products_MyCustomPage.cshtml.cs* Replace the *Razor Page Model's* name to "*Products_MyCustomPageModel*".



```

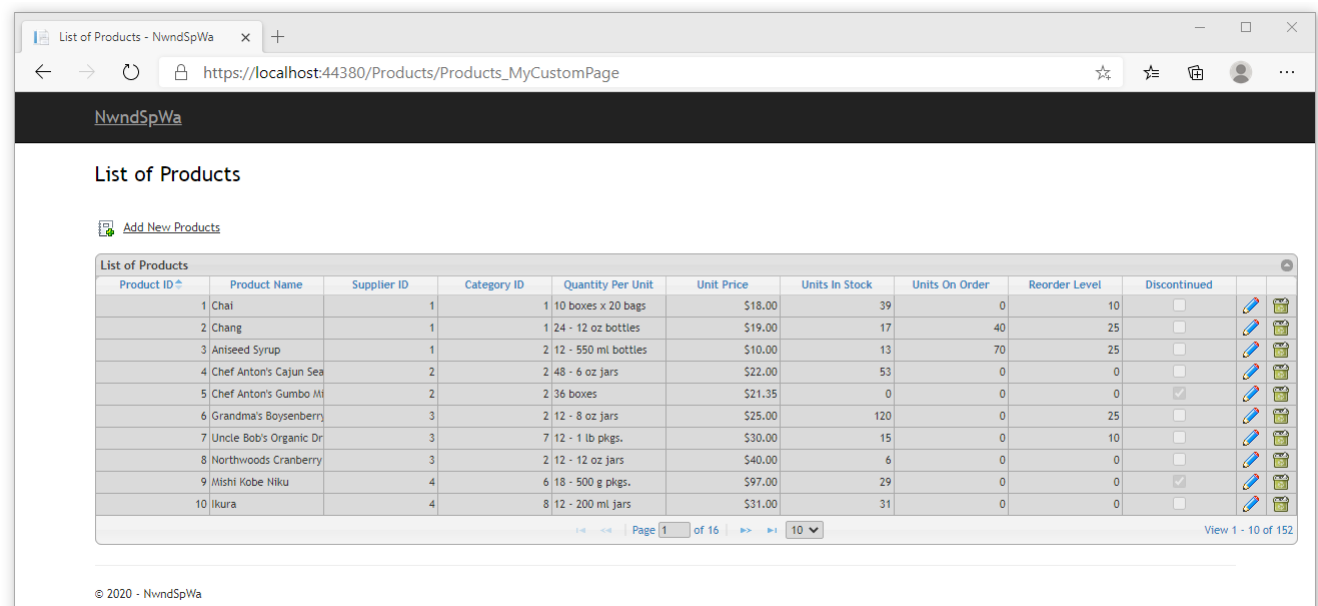
Products_MyCustomPage.cshtml.cs
NwndSpWa
NwndSpWa.Pages.Products_MyCustomPageModel

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using NwndSpWaAPI.Domain;
using NwndSpWaAPI.BusinessObject;
using System.Net.Http;
using System.Text.Json;
using System.Threading.Tasks;

namespace NwndSpWa.Pages
{
    5 references
    public class Products_MyCustomPageModel : PageModel
    {
        /// <summary>
        /// Handler, deletes a record.
        /// </summary>
        0 references
        public async Task<IActionResult> OnGetRemoveAsync(int id)
        {
            // delete data via web api
            HttpResponseMessage response = await Functions.HttpClientDeleteAsync("ProductsApi/Delete/" + id);
        }
    }
}

```

11. Run the Web Application by pressing *F5* while in Visual Studio 2019. And then go to the *Products_MyCustomPage* Razor Page. This page should look exactly like the *Products_ListCrudRedirect.cshtml* Razor Page.



© 2020 - NwndSpWa

12. Close the browser and go back to Visual Studio 2019.

13. Let's remove the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* from the grid. In the *Products_MyCustomPage* Razor Page, delete the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* in the *colNames* and *colModel* properties of the *JQGrid*. The code should look like the one shown below after deletion.

```
Products_MyCustomPage.cshtml  X
<script type="text/javascript">
    var urlAndMethod = '/Products/Products_MyCustomPage';

    $(function () {
        $('#list-grid').jqGrid({
            url: '/Products/Products_MyCustomPage?handler=GridData',
            datatype: 'json',
            mtype: 'GET',
            colNames: ['Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per Unit', 'Unit Price', 'Units In Stock', 'Units On Order', 'Reorder Level'],
            colModel: [
                { name: 'ProductID', index: 'ProductID', align: 'right' },
                { name: 'ProductName', index: 'ProductName', align: 'left' },
                { name: 'SupplierID', index: 'SupplierID', align: 'right' },
                { name: 'CategoryID', index: 'CategoryID', align: 'right' },
                { name: 'QuantityPerUnit', index: 'QuantityPerUnit', align: 'left' },
                { name: 'UnitPrice', index: 'UnitPrice', align: 'right', formatter: 'currency', formatoptions: { decimalPlaces: 2, prefix: "$" } },
                { name: 'UnitsInStock', index: 'UnitsInStock', align: 'right', formatter: 'integer' },
                { name: 'UnitsOnOrder', index: 'UnitsOnOrder', align: 'right', formatter: 'integer' },
                { name: 'ReorderLevel', index: 'ReorderLevel', align: 'right', formatter: 'integer' },
                { name: 'Discontinued', index: 'Discontinued', align: 'center', formatter: 'checkbox' },
                { name: 'editoperation', index: 'editoperation', align: 'center', width: 40, sortable: false, title: false },
                { name: 'deleteoperation', index: 'deleteoperation', align: 'center', width: 40, sortable: false, title: false }
            ],
            pager: $('#list-pager'),
        });
    });

```

```
Products_MyCustomPage.cshtml*  X
<script type="text/javascript">
    var urlAndMethod = '/Products/Products_MyCustomPage';

    $(function () {
        $('#list-grid').jqGrid({
            url: '/Products/Products_MyCustomPage?handler=GridData',
            datatype: 'json',
            mtype: 'GET',
            colNames: ['Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per Unit', 'Discontinued', '', ''],
            colModel: [
                { name: 'ProductID', index: 'ProductID', align: 'right' },
                { name: 'ProductName', index: 'ProductName', align: 'left' },
                { name: 'SupplierID', index: 'SupplierID', align: 'right' },
                { name: 'CategoryID', index: 'CategoryID', align: 'right' },
                { name: 'QuantityPerUnit', index: 'QuantityPerUnit', align: 'left' },
                { name: 'Discontinued', index: 'Discontinued', align: 'center', formatter: 'checkbox' },
                { name: 'editoperation', index: 'editoperation', align: 'center', width: 40, sortable: false, title: false },
                { name: 'deleteoperation', index: 'deleteoperation', align: 'center', width: 40, sortable: false, title: false }
            ],
            pager: $('#list-pager'),
        });
    });

```

14. In the *Products_MyCustomPage.cshtml.cs* Razor Page Model under the *OnGetGridDataAsync* method, delete the lines of code that pertains to the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level*. The code should look like the one shown below after deletion.

```
Products_MyCustomPage.cshtml.cs
NwndSpWa
References
public async Task<IActionResult> OnGetGridDataAsync(string sidx, string sord, int _page, int rows)
{
    // get the total number of records
    string responseBody1 = await Functions.HttpClientGetAsync("ProductsApi/GetRecordCount/");
    int totalRecords = JsonSerializer.Deserialize<int>(responseBody1);

    // get records
    List<Products> objProductsCol = null;
    string responseBody2 = await Functions.HttpClientGetAsync("ProductsApi/SelectSkipAndTake/?rows=" + rows);

    // make sure responseBody2 is not empty before deserialization
    if (!String.IsNullOrEmpty(responseBody2))
        objProductsCol = JsonSerializer.Deserialize<List<Products>>(responseBody2);

    // calculate the total number of pages
    int totalPages = (int)Math.Ceiling((float)totalRecords / (float)rows);

    // return a null in json for use by the jqgrid
    if (objProductsCol is null)
        return new JsonResult("{ total = 0, page = 0, records = 0, rows = null }");

    // create a serialized json object for use by the jqgrid
    var jsonData = new
    {
        total = totalPages,
        _page,
        records = totalRecords,
        rows = (
            from objProducts in objProductsCol
            select new
            {
                id = objProducts.ProductID,
                cell = new string[] {
                    objProducts.ProductID.ToString(),
                    objProducts.ProductName,
                    objProducts.SupplierID.HasValue ? objProducts.SupplierID.Value.ToString() : "",
                    objProducts.CategoryID.HasValue ? objProducts.CategoryID.Value.ToString() : "",
                    objProducts.QuantityPerUnit,
                    objProducts.UnitPrice.HasValue ? objProducts.UnitPrice.Value.ToString() : "",
                    objProducts.UnitsInStock.HasValue ? objProducts.UnitsInStock.Value.ToString() : "",
                    objProducts.UnitsOnOrder.HasValue ? objProducts.UnitsOnOrder.Value.ToString() : "",
                    objProducts.ReorderLevel.HasValue ? objProducts.ReorderLevel.Value.ToString() : "",
                    objProducts.Discontinued.ToString()
                }
            }
        ).ToArray()
    };
}
```

```
Products_MyCustomPage.cshtml.cs
NwndSpWa
// create a serialized json object for use by the jqgrid
var jsonData = new
{
    total = totalPages,
    _page,
    records = totalRecords,
    rows = (
        from objProducts in objProductsCol
        select new
        {
            id = objProducts.ProductID,
            cell = new string[] {
                objProducts.ProductID.ToString(),
                objProducts.ProductName,
                objProducts.SupplierID.HasValue ? objProducts.SupplierID.Value.ToString() : "",
                objProducts.CategoryID.HasValue ? objProducts.CategoryID.Value.ToString() : "",
                objProducts.QuantityPerUnit,
                objProducts.Discontinued.ToString()
            }
        }
    ).ToArray()
};
```

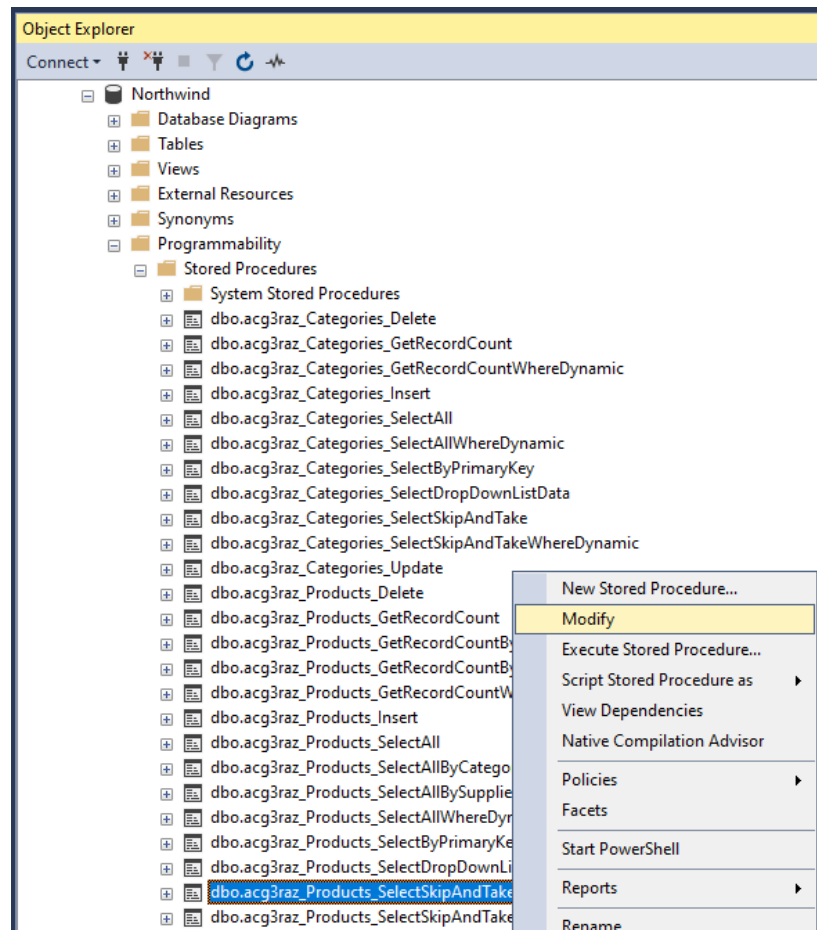
15. Run the *Web Application* by pressing *F5* while in Visual Studio 2019. Then go to the *Products_MyCustomPage.cshtml* Razor Page. The *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* should no longer be displayed on the grid.

Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit	Discontinued
1	Chai	1		10 boxes x 20 bags	<input type="checkbox"/>
2	Chang	1		1 24 - 12 oz bottles	<input type="checkbox"/>
3	Aniseed Syrup	1		2 12 - 550 ml bottles	<input type="checkbox"/>
4	Chef Anton's Cajun Seasoning	2		2 48 - 6 oz jars	<input type="checkbox"/>
5	Chef Anton's Gumbo Mix	2		2 36 boxes	<input checked="" type="checkbox"/>
6	Grandma's Boysenberry Spread	3		2 12 - 8 oz jars	<input type="checkbox"/>
7	Uncle Bob's Organic Dried Pears	3		7 12 - 1 lb pkgs.	<input type="checkbox"/>
8	Northwoods Cranberry Sauce	3		2 12 - 12 oz jars	<input type="checkbox"/>
9	Mishi Kobe Niku	4		6 18 - 500 g pkgs.	<input checked="" type="checkbox"/>
10	Ikura	4		8 12 - 200 ml jars	<input type="checkbox"/>

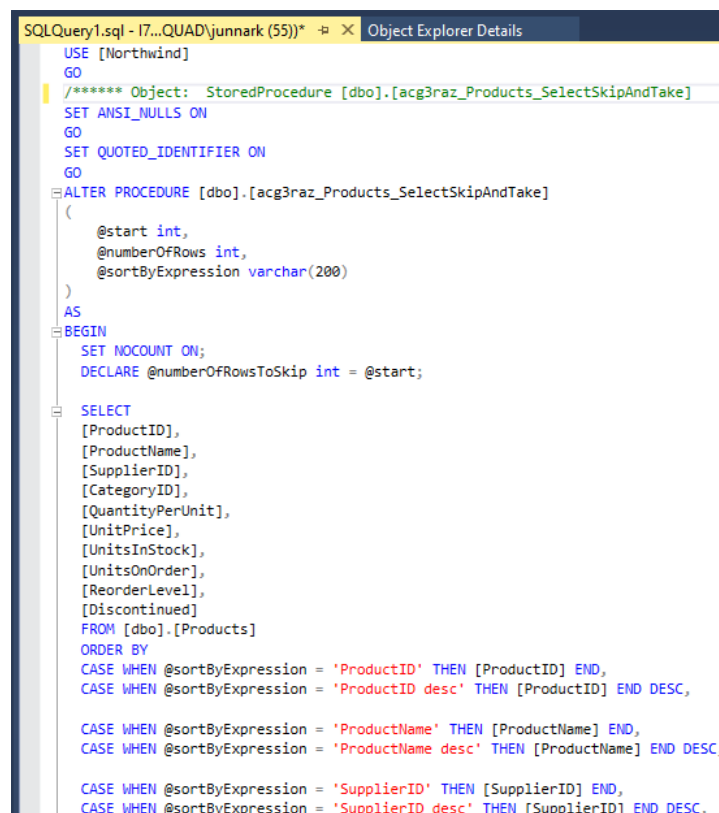
16. Close the browser and go back to Visual Studio 2019.
17. Now we will change the display on the *Supplier ID* and *Category ID*. Instead of showing just the IDs for these foreign keys, we will show the *Company Name (Supplier)* and *Category Name (Category)* respectively. To do this, we need to:
- Create a new *Stored Procedure*.
 - Create 2 new *Properties as Models* for *Company Name* and *Category Name*.
 - Create new *Data Layer* methods.
 - Create new *Business Layer* methods.
 - Create a new *Web API* method.

Note: There are many other ways to do this (since programming is also an art, not just science), but we'd like to walk you through the process of Adding New Code to the generated *Web Application* and Updating an Existing generated code.

18. **Create a new *Stored Procedure*** named *acg3raz_Products_MySelectSkipAndTake* in the *Northwind Database* using *Microsoft SQL Server Management Studio*. Go to the *Stored Procedures* folder under *Programmability* and Modify the *acg3raz_Products_SelectSkipAndTake* *Stored Procedure*.



19. This will open up the *acg3raz_Products_SelectSkipAndTake* Stored Procedure in a window.



20. Modify the *Stored Procedure*. Change the *ALTER* keyword to *CREATE*. Change the *Stored Procedure* name to *acg3raz_Products_MySelectSkipAndTake*. Add *INNER JOINS* to the *Suppliers* and *Categories* tables. Remove references to the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* columns.

```

CREATE PROCEDURE [dbo].[acg3raz_Products_MySelectSkipAndTake]
(
    @start int,
    @numberOfRows int,
    @sortByExpression varchar(200)
)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @numberOfRowsToSkip int = @start;

    SELECT
        prod.[ProductID],
        prod.[ProductName],
        prod.[SupplierID],
        prod.[CategoryID],
        prod.[QuantityPerUnit],
        prod.[Discontinued],
        cat.[CategoryName],
        sup.[CompanyName]
    FROM [dbo].[Products] prod
    INNER JOIN [dbo].[Suppliers] sup
    on prod.[SupplierID] = sup.[SupplierID]
    INNER JOIN [dbo].[Categories] cat
    on prod.[CategoryID] = cat.[CategoryID]
    ORDER BY
        CASE WHEN @sortByExpression = 'ProductID' THEN prod.[ProductID] END,
        CASE WHEN @sortByExpression = 'ProductID desc' THEN prod.[ProductID] END DESC,

        CASE WHEN @sortByExpression = 'ProductName' THEN prod.[ProductName] END,
        CASE WHEN @sortByExpression = 'ProductName desc' THEN prod.[ProductName] END DESC,

        CASE WHEN @sortByExpression = 'SupplierID' THEN prod.[SupplierID] END,
        CASE WHEN @sortByExpression = 'SupplierID desc' THEN prod.[SupplierID] END DESC,

        CASE WHEN @sortByExpression = 'CategoryID' THEN prod.[CategoryID] END,
        CASE WHEN @sortByExpression = 'CategoryID desc' THEN prod.[CategoryID] END DESC,

        CASE WHEN @sortByExpression = 'QuantityPerUnit' THEN prod.[QuantityPerUnit] END,
        CASE WHEN @sortByExpression = 'QuantityPerUnit desc' THEN prod.[QuantityPerUnit] END DESC,

        CASE WHEN @sortByExpression = 'CompanyName' THEN sup.[CompanyName] END,
        CASE WHEN @sortByExpression = 'CompanyName desc' THEN sup.[CompanyName] END DESC,

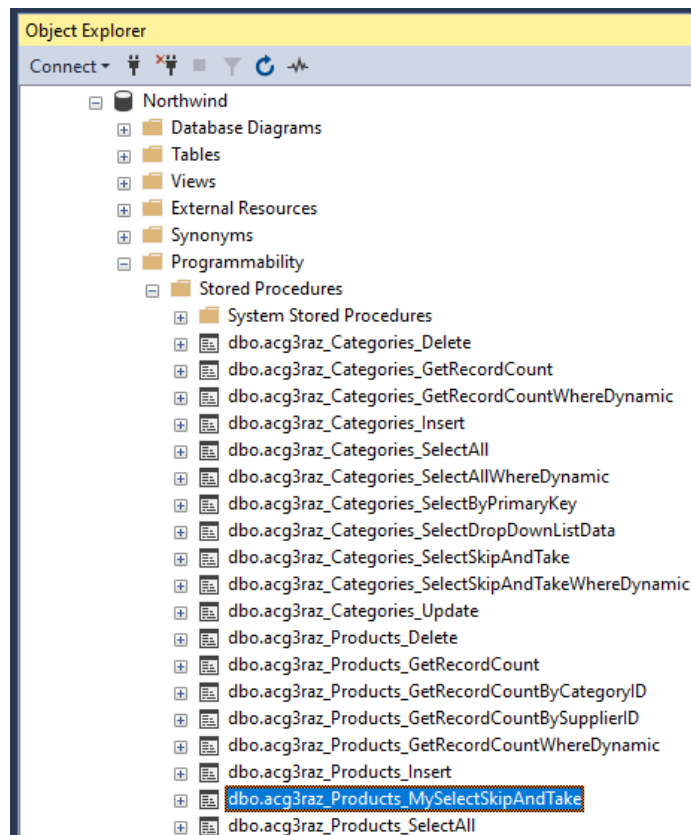
        CASE WHEN @sortByExpression = 'CategoryName' THEN cat.[CategoryName] END,
        CASE WHEN @sortByExpression = 'CategoryName desc' THEN cat.[CategoryName] END DESC,

        CASE WHEN @sortByExpression = 'Discontinued' THEN prod.[Discontinued] END,
        CASE WHEN @sortByExpression = 'Discontinued desc' THEN prod.[Discontinued] END DESC

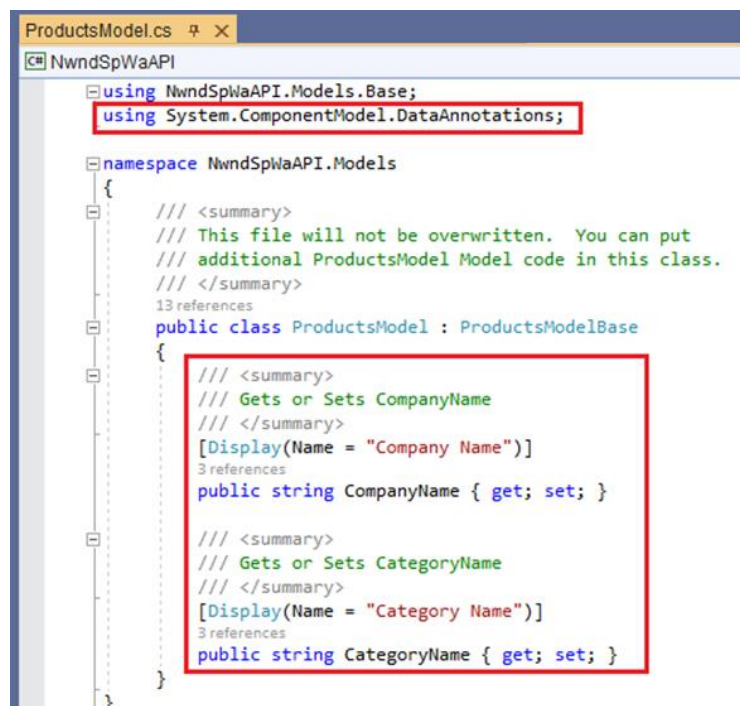
    OFFSET @numberOfRowsToSkip ROWS
    FETCH NEXT @numberOfRows ROWS ONLY
END

```

21. Make sure to click *Execute* in the *Microsoft SQL Server Management Studio's* menu to create the *acg3raz_Products_MySelectSkipAndTake* *Stored Procedure*. When you refresh the *Stored Procedures*, the *acg3raz_Products_MySelectSkipAndTake* should now be displayed.



22. **Create 2 new Properties as Models for *CompanyName* and *CategoryName*.** Open the *ProductsModel.cs* located in the *NwndSpWaAPI (Class Library Project)* under the *Models* folder. Add the *CompanyName* (*Suppliers Database Table*) and *CategoryName* (*Categories Database Table*) properties. Also, add the using statement as shown below.



23. **Create new Data Layer methods.** Open the *ProductsDataLayerBase.cs* (Parent/Base Class) and the *ProductsDataLayer.cs* (Child Class) under the *DataLayer/Base* and *DataLayer* folders respectively. Copy the following methods to the *ProductsDataLayer.cs* from the *ProductsDataLayerBase.cs*:

- SelectSkipAndTakeAsync* method.
- SelectSharedAsync* method.
- CreateProductsFromDataRowShared* method.

```

ProductsDataLayerBase.cs ProductsDataLayer.cs
NwndSpWaAPI NwndSpWaAPI.DataLayer.Base.ProductsDataLayerBase ProductsData
/// <summary>
/// Selects Products records sorted by the sortByExpression and returns records from the startRowIndex with rows (# of rows)
/// </summary>
1 reference
internal static async Task<List<Products>> SelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows)
{
    return await SelectSharedAsync("[dbo].[acg3raz_Products_SelectSkipAndTake]", null, null, true, null, sortByExpression, startRowIndex, rows);
}
  
```

```

ProductsDataLayerBase.cs ProductsDataLayer.cs
NwndSpWaAPI NwndSpWaAPI.DataLayer.Base.ProductsDataLayerBase InsertUpdate
6 references
internal static async Task<List<Products>> SelectSharedAsync(string storedProcName, string param, object paramValue, bool isUseStoredProc = true,
{
    List<Products> objProductsCol = null;

    using (SqlConnection connection = new SqlConnection(AppSettings.GetConnectionString()))
    {
        connection.Open();

        using (SqlCommand command = new SqlCommand(storedProcName, connection))
        {
            // ...
        }
    }

    return objProductsCol;
}
  
```

```

ProductsDataLayerBase.cs ProductsDataLayer.cs
NwndSpWaAPI NwndSpWaAPI
/// <summary>
/// Creates a Products object from the passed data row
/// </summary>
4 references
private static Products CreateProductsFromDataRowShared(DataRow dr)
{
    Products objProducts = new Products();

    objProducts.ProductID = (int)dr["ProductID"];
    objProducts.ProductName = dr["ProductName"].ToString();

    if (dr["SupplierID"] != System.DBNull.Value)
        objProducts.SupplierID = (int)dr["SupplierID"];
    else
        objProducts.SupplierID = null;

    if (dr["CategoryID"] != System.DBNull.Value)
        objProducts.CategoryID = (int)dr["CategoryID"];
    else
        objProducts.CategoryID = null;

    if (dr["QuantityPerUnit"] != System.DBNull.Value)
        objProducts.QuantityPerUnit = dr["QuantityPerUnit"].ToString();
    else
        objProducts.QuantityPerUnit = null;
}
  
```

```

ProductsDataLayerBase.cs ProductsDataLayer.cs
NwndSpWAPI
using System;
using NwndSpWAPI.DataLayer.Base;

namespace NwndSpWAPI.DataLayer
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products DataLayer code in this class
    /// </summary>

    -references
    internal class ProductsDataLayer : ProductsDataLayerBase
    {
        // constructor
        -references
        internal ProductsDataLayer()
        {
        }

        /// <summary> Selects Products records sorted by the sortByExpression and return ...
        -references
        internal static async Task<List<Products>> SelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows)...

        -references
        internal static async Task<List<Products>> SelectSharedAsync(string storedProcName, string param, object paramValue, bool isUseSt

        /// <summary> Creates a Products object from the passed data row
        -references
        private static Products CreateProductsFromDataRowShared(DataRow dr)...
    }
}

```

24. Change the name of the following methods in the *ProductsDataLayer.cs*. Also add the using statements as shown below.

- SelectSkipAndTakeAsync* to **MySelectSkipAndTakeAsync**.
- SelectSharedAsync* to **MySelectSharedAsync**.
- CreateProductsFromDataRowShared* to **MyCreateProductsFromDataRowShared**.

```

ProductsDataLayerBase.cs ProductsDataLayer.cs
NwndSpWAPI
using System;
using NwndSpWAPI.BusinessObject;
using NwndSpWAPI.DataLayer.Base;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Threading.Tasks;

namespace NwndSpWAPI.DataLayer
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products DataLayer code in this class
    /// </summary>

    19 references
    internal class ProductsDataLayer : ProductsDataLayerBase
    {
        // constructor
        0 references
        internal ProductsDataLayer()
        {
        }

        /// <summary> Selects Products records sorted by the sortByExpression and return ...
        0 references
        internal static async Task<List<Products>> MySelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows)...

        1 reference
        internal static async Task<List<Products>> MySelectSharedAsync(string storedProcName, string param, object paramValue, bool isUseSt

        /// <summary> Creates a Products object from the passed data row
        1 reference
        private static Products MyCreateProductsFromDataRowShared(DataRow dr)...
    }
}

```

25. Change the *Stored Procedure* name to *acg3raz_Products_MySelectSkipAndTake* under the *MySelectSkipAndTakeAsync* method. This is the *Stored Procedure* that we created earlier.

```

ProductsDataLayerBase.cs  ProductsDataLayer.cs
NwndSpWaAPI
0 references
internal static async Task<List<Products>> MySelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows)
{
    return await MySelectSharedAsync("[dbo].[acg3raz_Products_MySelectSkipAndTake]", null, null, true, null, sortByExpression, startRowIndex, rows);
}

```

26. Change the *CreateProductsFromDataSource* method reference to *MyCreateProductsFromDataSource* under the *MySelectSharedAsync* method.

```

ProductsDataLayerBase.cs  ProductsDataLayer.cs
NwndSpWaAPI
using (SqlDataAdapter da = new SqlDataAdapter(command))
{
    DataTable dt = new DataTable();
    await Task.Run(() => da.Fill(dt));

    if (dt != null)
    {
        if (dt.Rows.Count > 0)
        {
            objProductsCol = new List<Products>();

            foreach (DataRow dr in dt.Rows)
            {
                Products objProducts = MyCreateProductsFromDataSource(dr);
                objProductsCol.Add(objProducts);
            }
        }
    }
}

```

27. Add code assignments for the *CompanyName* and *CategoryName* in the *MyCreateProductsFromDataSource* method. Also, remove references to the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel*.

```

ProductsDataLayerBase.cs  ProductsDataLayer.cs
NwndSpWaAPI
1 reference
private static Products MyCreateProductsFromDataSourceShared(DataRow dr)
{
    Products objProducts = new Products();

    objProducts.ProductID = (int)dr["ProductID"];
    objProducts.ProductName = dr["ProductName"].ToString();

    if (dr["SupplierID"] != System.DBNull.Value)
        objProducts.SupplierID = (int)dr["SupplierID"];
    else
        objProducts.SupplierID = null;

    if (dr["CategoryID"] != System.DBNull.Value)
        objProducts.CategoryID = (int)dr["CategoryID"];
    else
        objProducts.CategoryID = null;

    if (dr["QuantityPerUnit"] != System.DBNull.Value)
        objProducts.QuantityPerUnit = dr["QuantityPerUnit"].ToString();
    else
        objProducts.QuantityPerUnit = null;

    if (dr["CompanyName"] != System.DBNull.Value)
        objProducts.CompanyName = dr["CompanyName"].ToString();
    else
        objProducts.CompanyName = null;

    if (dr["CategoryName"] != System.DBNull.Value)
        objProducts.CategoryName = dr["CategoryName"].ToString();
    else
        objProducts.CategoryName = null;

    objProducts.Discontinued = (bool)dr["Discontinued"];

    return objProducts;
}

```

28. **Create new *Business Layer* methods.** Open the *ProductsBase.cs* (Parent/Base Class) and the *Products.cs* (Child Class) under the *BusinessObject /Base* and *BusinessObject* folders respectively. Copy the following methods to the *Products.cs* from the *ProductsBase.cs*:

- SelectSkipAndTakeAsync* method.
- GetSortExpression* method.

This screenshot shows the `ProductsBase.cs` file in Visual Studio. The file is part of the `NwndSpWaAPI.BusinessObject.Base` namespace. A red box highlights the `SelectSkipAndTakeAsync` method, which is a public static async method that returns a `Task<List<Products>>`. It takes three parameters: `rows` (int), `startRowIndex` (int), and `sortByExpression` (string). The method calls `GetSortExpression` and then `ProductsDataLayer.SelectSkipAndTakeAsync`.

```

/// <summary>
/// Selects records as a collection (List) of Products sorted by the sortByExpression.
/// </summary>
- references
public static async Task<List<Products>> SelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
{
    sortByExpression = GetSortExpression(sortByExpression);
    return await ProductsDataLayer.SelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
}

```

This screenshot shows the `ProductsBase.cs` file in Visual Studio. The file is part of the `NwndSpWaAPI.BusinessObject.Base` namespace. A red box highlights the `GetSortExpression` method, which is a private static method that returns a string. It takes a `sortByExpression` (string) as a parameter. The method logic checks if the expression is null or empty, or if it ends with " asc". If so, it replaces " asc" with "ProductID". Otherwise, it returns the expression as is.

```

/// <summary>
/// Gets the default sort expression when no sort expression is provided
/// </summary>
- references
private static string GetSortExpression(string sortByExpression)
{
    // when no sort expression is provided, ProductID is set as the default in ascending order
    // for ascending order, "asc" is not needed, so it is removed
    if (String.IsNullOrEmpty(sortByExpression) || sortByExpression == " asc")
        sortByExpression = "ProductID";
    else if (sortByExpression.Contains(" asc"))
        sortByExpression = sortByExpression.Replace(" asc", "");

    return sortByExpression;
}

```

This screenshot shows the `Products.cs` file in Visual Studio. The file is part of the `NwndSpWaAPI.BusinessObject` namespace. It shows the `Products` class inheriting from `ProductsBase`. The `SelectSkipAndTakeAsync` and `GetSortExpression` methods are copied from the base class. The `Products` class is a partial class.

```

using System;
using NwndSpWaAPI.BusinessObject.Base;

namespace NwndSpWaAPI.BusinessObject
{
    /// <summary> This file will not be overwritten. You can put additional Product ...
    - references
    public partial class Products : ProductsBase
    {
        /// <summary> Selects records as a collection (List) of Products sorted by the s ...
        - references
        public static async Task<List<Products>> SelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
        {
            sortByExpression = GetSortExpression(sortByExpression);
            return await ProductsDataLayer.SelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
        }

        /// <summary> Gets the default sort expression when no sort expression is provid ...
        - references
        private static string GetSortExpression(string sortByExpression)
        {
            // when no sort expression is provided, ProductID is set as the default in ascending order
            // for ascending order, "asc" is not needed, so it is removed
            if (String.IsNullOrEmpty(sortByExpression) || sortByExpression == " asc")
                sortByExpression = "ProductID";
            else if (sortByExpression.Contains(" asc"))
                sortByExpression = sortByExpression.Replace(" asc", "");

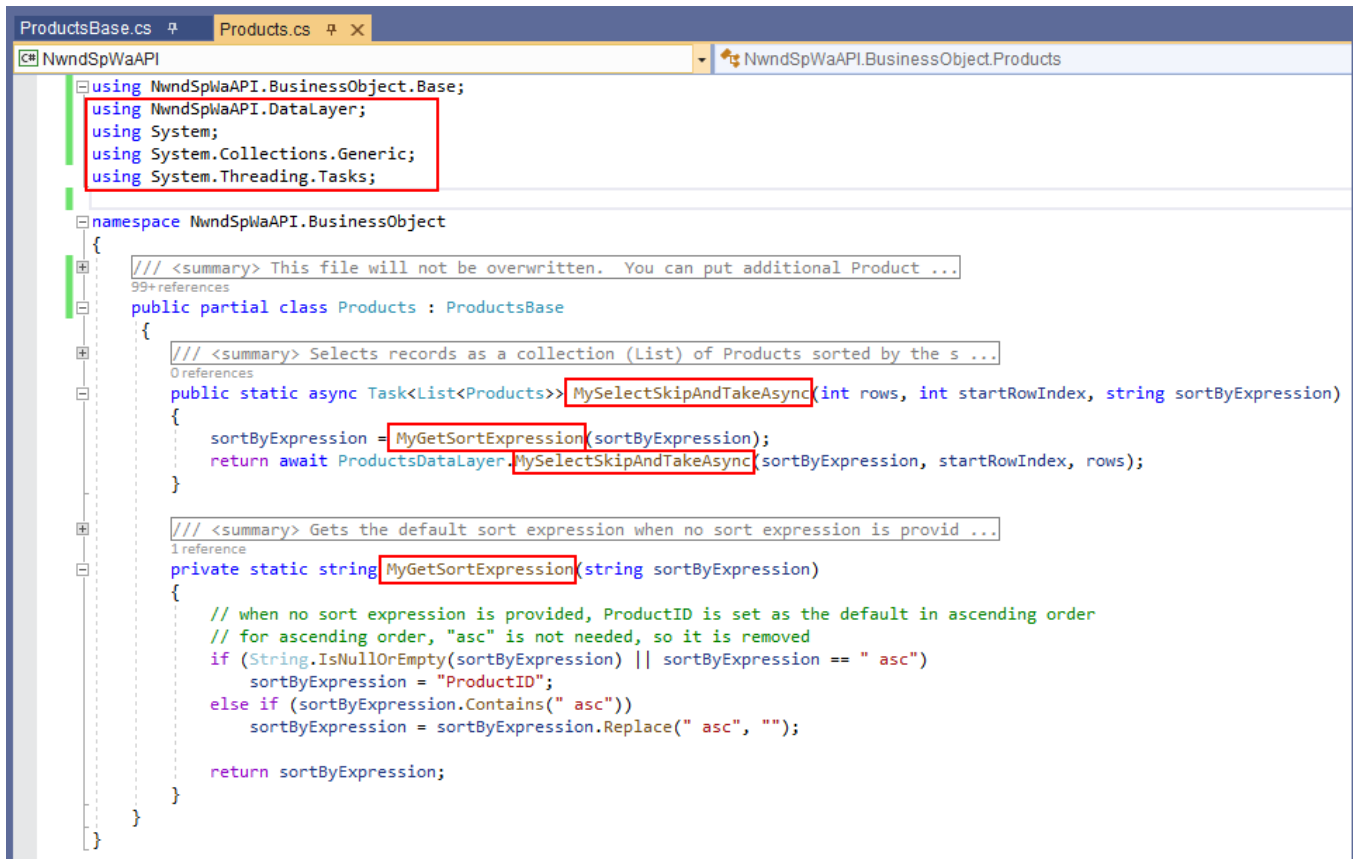
            return sortByExpression;
        }
    }
}

```

29. Change the name of the following methods in the *Products.cs*. Also, add the using statements as shown below.

- SelectSkipAndTakeAsync* to **MySelectSkipAndTakeAsync**.
- GetSortExpression* to **MyGetSortExpression**.

Also, under the *MySelectSkipAndTakeAsync* method, change the *GetSortExpression* reference to **MyGetSortExpression** and the *SelectSkipAndTakeAsync* reference to **MySelectSkipAndTakeAsync**.



```

ProductsBase.cs  Products.cs
C# NwndSpWaAPI  NwndSpWaAPI.BusinessObject.Products

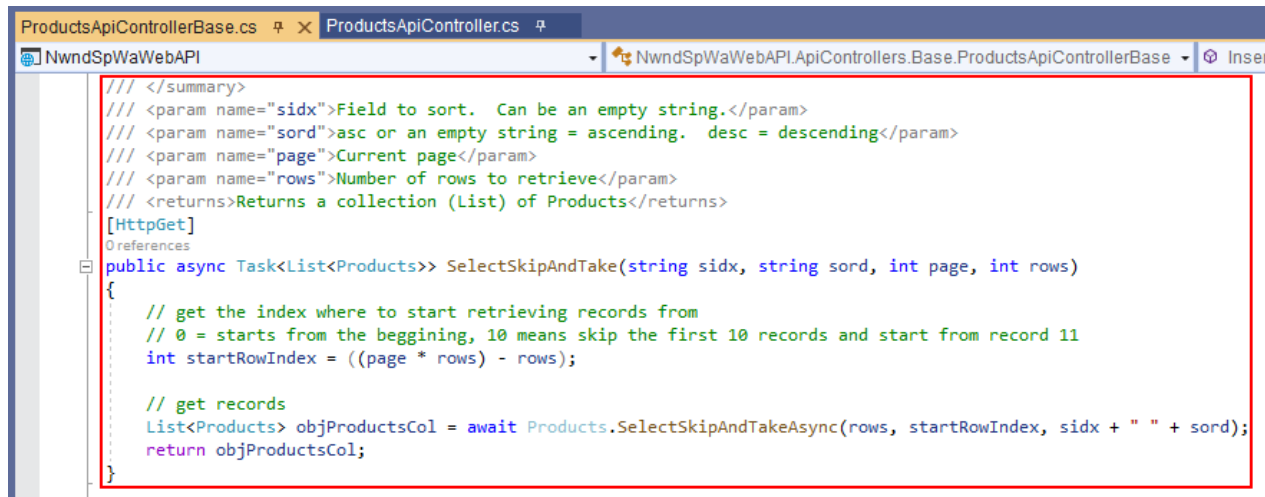
using NwndSpWaAPI.BusinessObject.Base;
using NwndSpWaAPI.DataLayer;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace NwndSpWaAPI.BusinessObject
{
    /// <summary> This file will not be overwritten. You can put additional Product ...
    99+ references
    public partial class Products : ProductsBase
    {
        /// <summary> Selects records as a collection (List) of Products sorted by the s ...
        0 references
        public static async Task<List<Products>> MySelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
        {
            sortByExpression = MyGetSortExpression(sortByExpression);
            return await ProductsDataLayer.MySelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
        }

        /// <summary> Gets the default sort expression when no sort expression is provid ...
        1 reference
        private static string MyGetSortExpression(string sortByExpression)
        {
            // when no sort expression is provided, ProductID is set as the default in ascending order
            // for ascending order, "asc" is not needed, so it is removed
            if (String.IsNullOrEmpty(sortByExpression) || sortByExpression == " asc")
                sortByExpression = "ProductID";
            else if (sortByExpression.Contains(" asc"))
                sortByExpression = sortByExpression.Replace(" asc", "");

            return sortByExpression;
        }
    }
}
  
```

30. **Create a new *Web API* method.** Copy the *SelectSkipAndTake* method from the *ProductsApiControllerBase.cs* (Parent/Base Class) to the *ProductsApiController.cs* (Child Class). Both the *ProductsApiControllerBase.cs* and *ProductsApiController.cs* are in the *Web API Project* (*NwndSpWaWebAPI*) under the *Controllers/Base* and *Controllers* folders respectively.



```

ProductsApiControllerBase.cs
ProductsApiController.cs

NwndSpWaWebAPI
NwndSpWaWebAPI.ApiControllers.Base.ProductsApiControllerBase
Insert

/// </summary>
/// <param name="sidx">Field to sort. Can be an empty string.</param>
/// <param name="sord">asc or an empty string = ascending. desc = descending</param>
/// <param name="page">Current page</param>
/// <param name="rows">Number of rows to retrieve</param>
/// <returns>Returns a collection (List) of Products</returns>
[HttpGet]
public async Task<List<Products>> SelectSkipAndTake(string sidx, string sord, int page, int rows)
{
    // get the index where to start retrieving records from
    // 0 = starts from the beginning, 10 means skip the first 10 records and start from record 11
    int startRowIndex = ((page * rows) - rows);

    // get records
    List<Products> objProductsCol = await Products.SelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);
    return objProductsCol;
}

```



```

ProductsApiControllerBase.cs
ProductsApiController.cs

NwndSpWaWebAPI
NwndSpWaWebAPI.Controllers.ProductsApiController
SelectSkipAndTake

using System;
using NwndSpWaWebAPI.ApiControllers.Base;

namespace NwndSpWaWebAPI.Controllers
{
    /// <summary> This file will not be overwritten. You can put additional Product ...
    public class ProductsApiController : ProductsApiControllerBase
    {
        /// <summary> Selects records as a collection (List) of Products sorted by the s ...
        [HttpGet]
        public async Task<List<Products>> SelectSkipAndTake(string sidx, string sord, int page, int rows)
        {
            // get the index where to start retrieving records from
            // 0 = starts from the beginning, 10 means skip the first 10 records and start from record 11
            int startRowIndex = ((page * rows) - rows);

            // get records
            List<Products> objProductsCol = await Products.SelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);
            return objProductsCol;
        }
    }
}

```

31. In the *ProductsApiController*, change the *SelectSkipAndTake* Web API method name to ***MySelectSkipAndTake***. Also, change the *SelectSkipAndTakeAsync* (Business Object) name reference to ***MySelectSkipAndTakeAsync***. Also, add the using statements as shown below.

```

ProductsApiControllerBase.cs ProductsApiController.cs
NwndSpWaWebAPI NwndSpWaWebAPI.Controllers.ProductsApiController MySelectSkipAndTake
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NwndSpWaAPI.BusinessObject;
using NwndSpWaWebAPI.ApiControllers.Base;

namespace NwndSpWaWebAPI.Controllers
{
    /// <summary> This file will not be overwritten. You can put additional Product ...
    References
    public class ProductsApiController : ProductsApiControllerBase
    {
        /// <summary> Selects records as a collection (List) of Products sorted by the s ...
        [HttpGet]
        References
        public async Task<List<Products>> MySelectSkipAndTake(string sidx, string sord, int page, int rows)
        {
            // get the index where to start retrieving records from
            // 0 = starts from the beginning, 10 means skip the first 10 records and start from record 11
            int startRowIndex = ((page * rows) - rows);

            // get records
            List<Products> objProductsCol = await Products.MySelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);
            return objProductsCol;
        }
    }
}

```

32. Now that all the code plumbing is done, we just need to display the *CompanyName* and *CategoryName* as we originally intended. First, we need to reference the *MySelectSkipAndTake* Web API, and then add code that will display the *CompanyName* and *CategoryName* in the *Products_MyCustomPage.cshtml.cs* Razor Page Model as shown below.

```

Products_MyCustomPage.cshtml.cs Products_MyCustomPageModel
NwndSpWa NwndSpWa.Pages.Products_MyCustomPageModel OnGetGridDataAsync
public class Products_MyCustomPageModel : PageModel
{
    /// <summary>
    /// Handler, deletes a record.
    /// </summary>
    References
    public async Task<ActionResult> OnGetRemoveAsync(int id)[...]

    /// <summary> GET: /Products/GridData Gets the json needed by the jqgrid for use ...
    References
    public async Task<ActionResult> OnGetGridDataAsync(string sidx, string sord, int _page, int rows)
    {
        // get the total number of records
        string responseBody1 = await Functions.HttpClientGetAsync("ProductsApi/GetRecordCount/");
        int totalRecords = JsonSerializer.Deserialize<int>(responseBody1);

        // get records
        List<Products> objProductsCol = null;
        string responseBody2 = await Functions.HttpClientGetAsync("ProductsApi/MySelectSkipAndTake/?rows=" + rows + "&page=" + _page + "&sidx=" + sidx + "&sord=" + sord);

        // make sure responseBody2 is not empty before deserialization
        if (!String.IsNullOrEmpty(responseBody2))
            objProductsCol = JsonSerializer.Deserialize<List<Products>>(responseBody2);

        // calculate the total number of pages
        int totalPages = (int)Math.Ceiling((float)totalRecords / (float)rows);

        // return a null in json for use by the jqgrid
        if (objProductsCol is null)
            return new JsonResult("{ total = 0, page = 0, records = 0, rows = null }");

        // create a serialized json object for use by the jqgrid
        var jsonData = new
        {
            total = totalPages,
            _page,
            records = totalRecords,
            rows = (
                from objProducts in objProductsCol
                select new
                {
                    id = objProducts.ProductID,
                    cell = new string[] {
                        objProducts.ProductID.ToString(),
                        objProducts.ProductName,
                        objProducts.SupplierID.HasValue ? objProducts.CompanyName + " (" + objProducts.SupplierID.Value.ToString() + ")": "",
                        objProducts.CategoryID.HasValue ? objProducts.CategoryName + " (" + objProducts.CategoryID.Value.ToString() + ")": "",
                        objProducts.QuantityPerUnit,
                        objProducts.Discontinued.ToString()
                    }
                }
            )
        };
    }
}

```

33. In the *Products_MyCustomPage.cshtml* Razor Page, change the *colNames* to *Supplier* and *Category* respectively. Also, remove the *SupplierID* and *CategoryID* and replace with *CompanyName* and *CategoryName* respectively as shown below.

```
Products_MyCustomPage.cshtml
@page
@model NwndSpwa.Pages.Products_MyCustomPageModel
@{
    ViewData["Title"] = "List of Products";
}

@section AdditionalCss {
    <link rel="stylesheet" href="~/css/ui.jqgrid.min.css" />
}

@section AdditionalJavaScript {
    <script src="~/js/jqgrid-i18n/grid.locale-en.min.js" asp-append-version="true"></script>
    <script src="~/js/jquery-jqgrid-4.13.2.min.js" asp-append-version="true"></script>
    <script src="~/js/jqgrid-listcrudredirect.js" asp-append-version="true"></script>
}

<script type="text/javascript">
    var urlAndMethod = '/Products/Products_MyCustomPage';

    $(function () {
        $('#list-grid').jqGrid({
            url: '/Products/Products_MyCustomPage?handler=GridData',
            datatype: 'json',
            mtype: 'GET',
            colNames: ['Product ID', 'Product Name', 'Supplier', 'Category', 'Quantity Per Unit', 'Discontinued', '', ''],
            colModel: [
                { name: 'ProductID', index: 'ProductID', align: 'right' },
                { name: 'ProductName', index: 'ProductName', align: 'left' },
                { name: 'CompanyName', index: 'CompanyName', align: 'right' },
                { name: 'CategoryName', index: 'CategoryName', align: 'right' },
                { name: 'QuantityPerUnit', index: 'QuantityPerUnit', align: 'left' },
                { name: 'Discontinued', index: 'Discontinued', align: 'center', formatter: 'checkbox' },
                { name: 'editoperation', index: 'editoperation', align: 'center', width: 40, sortable: false, title: false },
                { name: 'deleteoperation', index: 'deleteoperation', align: 'center', width: 40, sortable: false, title: false }
            ],
            pager: $('#list-pager'),
            rowNum: 10,
        });
    });
}
```

34. Run the *Web Application* by pressing *F5* while in Visual Studio 2019. Then go to the *Products_MyCustomPage* Razor Page.

This finished custom Razor Page no longer shows the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* columns. It also shows the *CompanyName* (*Supplier*) and *CategoryName* (*Category*) with the *SupplierID* and *CategoryID* in parenthesis instead of just showing the *SupplierID* and *CategoryID* respectively. The *CompanyName* (*Supplier*) and *CategoryName* (*Category*) are also sortable.

https://localhost:44380/Products/Products_MyCustomPage

NwndSpWa

List of Products

Add New Products

Product ID	Product Name	Supplier	Category	Quantity Per Unit	Discontinued		
1	Chai	Exotic Liquids (1)	Beverages(1)	10 boxes x 20 bags	<input type="checkbox"/>		
2	Chang	Exotic Liquids (1)	Beverages(1)	24 - 12 oz bottles	<input type="checkbox"/>		
3	Aniseed Syrup	Exotic Liquids (1)	Condiments(2)	12 - 550 ml bottles	<input type="checkbox"/>		
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights (2)	Condiments(2)	48 - 6 oz jars	<input type="checkbox"/>		
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights (2)	Condiments(2)	36 boxes	<input checked="" type="checkbox"/>		
6	Grandma's Boysenberry Spread	Grandma Kelly's Homestead (3)	Condiments(2)	12 - 8 oz jars	<input type="checkbox"/>		
7	Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead (3)	Produce(7)	12 - 1 lb pkgs.	<input type="checkbox"/>		
8	Northwoods Cranberry Sauce	Grandma Kelly's Homestead (3)	Condiments(2)	12 - 12 oz jars	<input type="checkbox"/>		
9	Mishi Kobe Niku	Tokyo Traders (4)	Meat/Poultry(6)	18 - 500 g pkgs.	<input checked="" type="checkbox"/>		
10	Ikura	Tokyo Traders (4)	Seafood(8)	12 - 200 ml jars	<input type="checkbox"/>		

Page 1 of 16 View 1 - 10 of 152

35. In summary: We added a new *Razor Page* to show that you can add your own code to the generated projects. The way we added code is by copying the existing generated code and then replacing bits and pieces in the way we want the new *Razor Page* to behave. We added code to the rest of the n-tier layers to show where you should add code and how easy it is to copy the existing generated code.

- We have created a custom *Razor Page* with the respective *Razor Page Model* and added it to the generated *Web Application Project* by copying from the existing generated code.
- We created a new *Stored Procedure* by copying from the existing generated stored procedure.
- We created 2 new *Properties as Models* for *Company Name* and *Category Name* and added them to the *ProductsModels* (child/derived) class.
- We created 3 new *Data Layer* methods in the *ProductsDataLayer* (child/derive) class. The *MySelectSkipAndTakeAsync*, *MySelectSharedAsync*, and *MyCreateProductsFromDataRowShared* methods by copying from the existing generated methods.
- We created 2 new *Business Layer* methods in the *Products* (child/derived) class. The *MySelectSkipAndTakeAsync* and *MyGetSortExpression* methods by copying from the existing generated methods.
- We created a new *Web API* method in the *ProductsApiController* (child/derived) class. The *MySelectSkipAndTake* method by copying from the existing generated *Web API*.

You can read end-to-end tutorials on more subjects on using AspCoreGen 3.0 Razor Professional Plus that came with your purchase. These tutorials are available to customers and are included in a link on your invoice when you purchase AspCoreGen 3.0 Razor Professional.

Note: Some features shown here are not available in the Express Edition. The code in this tutorial is available for download for paying customers only, please email us at Software Support for more information.

End of tutorial.