The Generated Code for Database Tables

The Generated Code for Database Tables

1 INTRODUCTION

This topic will walk you through AspCoreGen 3.0 MVC'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

Then follow these step-by-step instructions.

1.2 GENERATED CODE FOR DATABASE TABLES

In the *Generating Code Tutorial* under the *Database Objects to Generate From*, there are four (4) database objects where we can generate code from. This tutorial will discuss the generated code for database tables only:

1. All Tables
2. Selected Tables Only

1.3 GENERATED PROJECTS

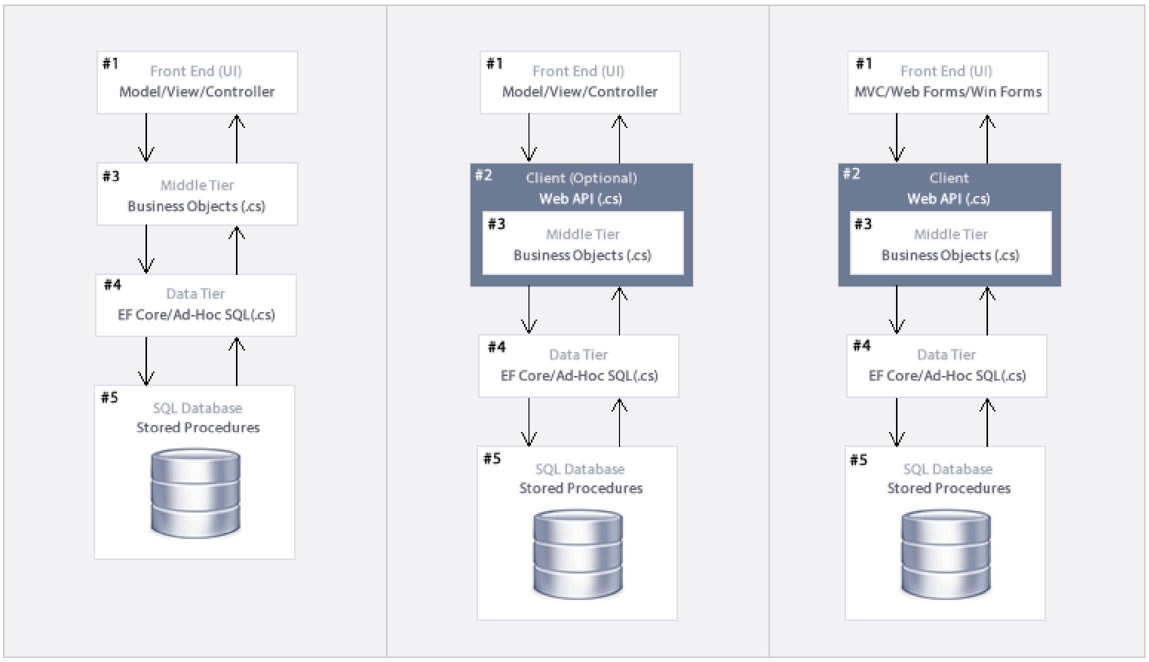
In the *App Settings Tutorial* there are 3 projects that can be generated in a solution:

- Web Application Project (Front End)
- Business Layer and Data Layer API Project (Class Library Project – Middle and Data Tier)
- Web API Project (Optional)

We will be discussing these generated projects including the *Web API Project*.

2 N-TIER LAYERING

AspCoreGen 3.0 MVC generates code in an *n-tier* architecture. A presentation tier (the client), middle tier (business objects), data tier (data access objects), and the database scripts such as stored procedures. Code is separated in different layers.



#1 Front End (UI) Model/View/Controller
#3 Middle Tier Business Objects (.cs)
#4 Data Tier EF Core/Ad-Hoc SQL(.cs)
#5 SQL Database Stored Procedures
#1 Front End (UI) Model/View/Controller
#2 Client (Optional) Web API (.cs)
#3 Middle Tier Business Objects (.cs)
#4 Data Tier EF Core/Ad-Hoc SQL(.cs)
#5 SQL Database Stored Procedures
#1 Front End (UI) MVC/Web Forms/Win Forms
#2 Client Web API (.cs)
#3 Middle Tier Business Objects (.cs)
#4 Data Tier EF Core/Ad-Hoc SQL(.cs)
#5 SQL Database Stored Procedures

2.1 FRONT END

User Interface or Presentation Layer. Views, Controllers, JavaScript, CSS, JQuery, and more.

2.2 MIDDLE-TIER/ MIDDLE LAYER

1. Business Logic Class files, Models, Views, View Models, etc. Or,
2. Web API (Optional). Optionally encapsulate calls to Business Objects when generating Web API code.

2.3 DATA-TIER/ DATA LAYER

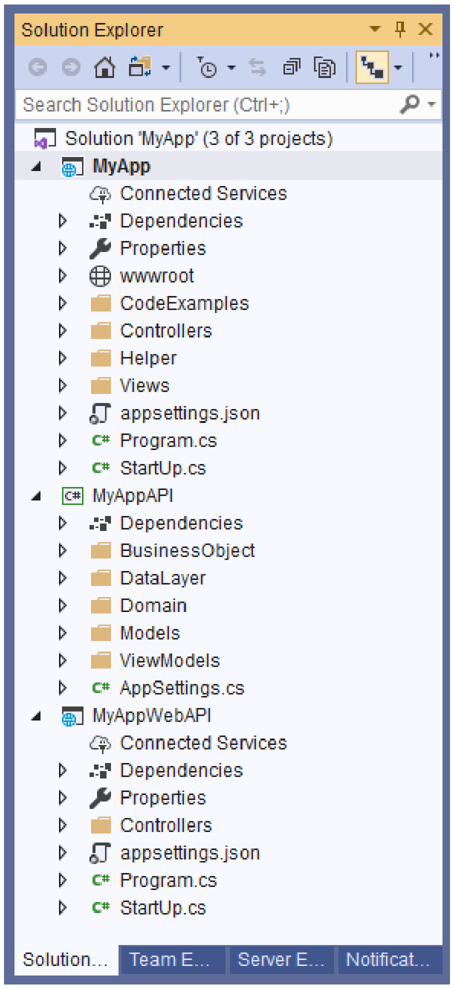
Class Files using Linq-to-Entities - Entity Framework Core or Ad-Hoc SQL.

2.4 SQL SCRIPTS

Stored Procedures.

3 Generated Projects

There are 3 projects that can be generated by AspCoreGen 3.0 MVC Professional Plus including the optional *Web API* project. And if you chose *Stored Procedures* under the *Generated SQL Script* in the *Database Settings Tab*, these SQL scripts will be generated straight in your MS SQL Server Database's *Stored Procedures* folder.

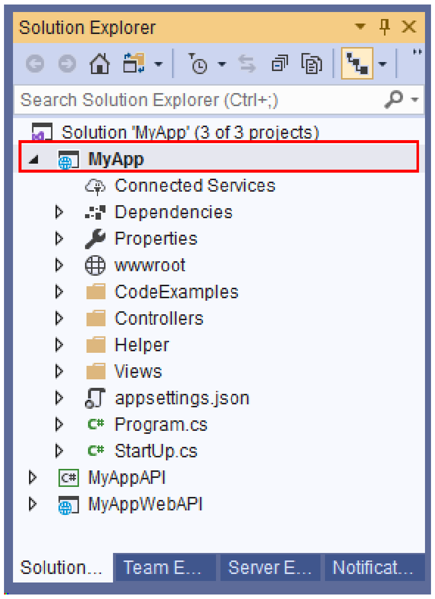


```
Solution 'MyApp' (3 of 3 projects)
  MyApp
    Connected Services
    Dependencies
    Properties
    wwwroot
    CodeExamples
    Controllers
    Helper
    Views
    appsettings.json
    Program.cs
    StartUp.cs
  MyAppAPI
    Dependencies
    BusinessObject
    DataLayer
    Domain
    Models
    ViewModels
    AppSettings.cs
  MyAppWebAPI
    Connected Services
    Dependencies
    Properties
    Controllers
    appsettings.json
    Program.cs
    StartUp.cs
```

Generated Projects in Visual Studio

3.1 Web Application Project

The generated *Web Application Project* is the *User Interface*, *Front End,* or *Presentation Layer* part of the N-tier layer generated code. This is an ASP.NET MVC core project. The application's main purpose is to serve as a client's user interface. The *Presentation Layer* is what the users see, use, and interact with.

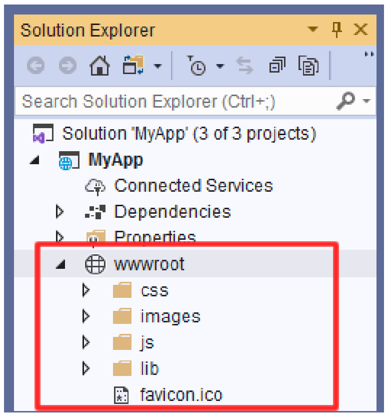


In this example, the *MyApp* project is the *Web Application Project* that was generated. Everything in this project are used to present users with an interface they can interact with, except the optional *CodeExamples* folder which contains *Class Files* for each of the database tables showing code examples on how to access the *Middle-Tier/Business Objects* to do CRUD* operations.

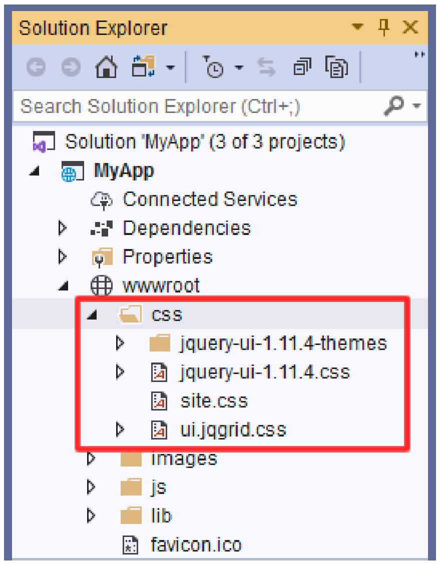
As shown in the *N-Tier Layering* above, the *Front End* (MVC view) accesses the *Middle Tier* (class) to do any kind of operation. Or it can also access the *Web API* instead of the *Middle Tier* (class).

3.1.1 wwwroot

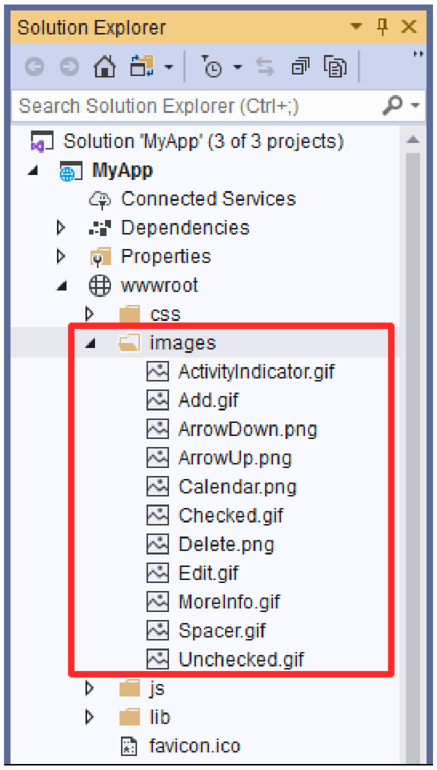
This folder is generally needed by ASP.NET Core MVC as the *Web Root* of the project by default.   You can place static files needed by the ASP.NET Core MVC project here. You can add folders and files and name them to whatever you like.



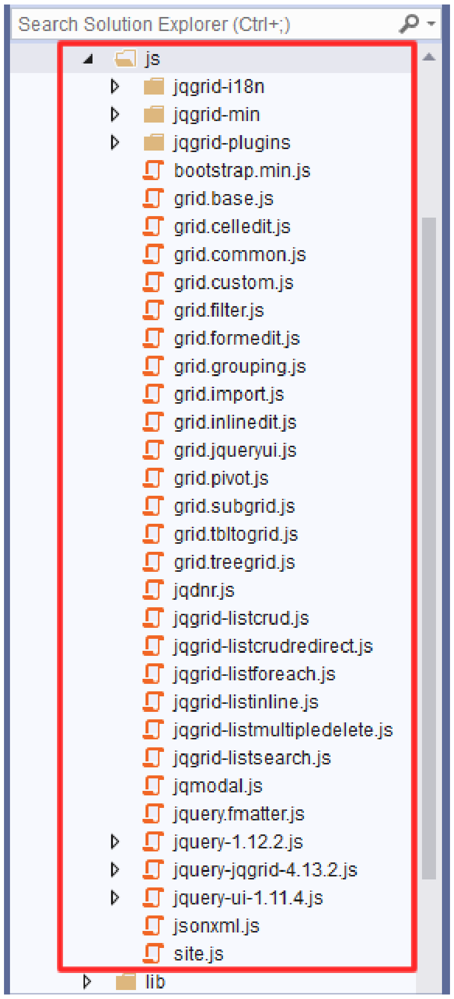
1. **css (folder):** Contains styles including 24 different JQuery-UI themes used by the project. You can add your own stylesheets here. You can also add and updates styles in the *site.css* stylesheet.



2. **images (folder):** Contains images used by the project. You can add your own images here.



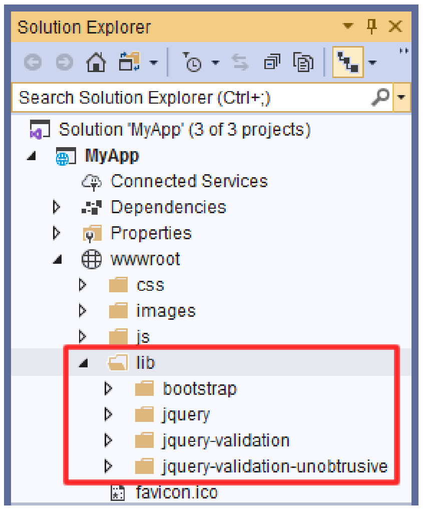
3. **js (folder):** Contains javascript files including JQGrid and JQuery plugins used by the project. You can add your own scripts here.



Search Solution Explorer (Ctrl+;)

```
◢ js
    ▷ jqgrid-i18n
    ▷ jqgrid-min
    ▷ jqgrid-plugins
      bootstrap.min.js
      grid.base.js
      grid.celledit.js
      grid.common.js
      grid.custom.js
      grid.filter.js
      grid.formedit.js
      grid.grouping.js
      grid.import.js
      grid.inlinedit.js
      grid.jqueryui.js
      grid.pivot.js
      grid.subgrid.js
      grid.tbltogrid.js
      grid.treegrid.js
      jqdnr.js
      jqgrid-listcrud.js
      jqgrid-listcrudredirect.js
      jqgrid-listforeach.js
      jqgrid-listinline.js
      jqgrid-listmultipledelete.js
      jqgrid-listsearch.js
      jqmodal.js
      jquery.fmatter.js
    ▷ jquery-1.12.2.js
    ▷ jquery-jqgrid-4.13.2.js
    ▷ jquery-ui-1.11.4.js
      jsonxml.js
      site.js
▷ lib
```

4. **lib (folder):** Contains libraries, both styles and javascript used by the project. By default, these libraries are included even if you don't use AspCoreGen 3.0 MVC to generate the code. You can add your own libraries here, however, **we recommend that you don't**.

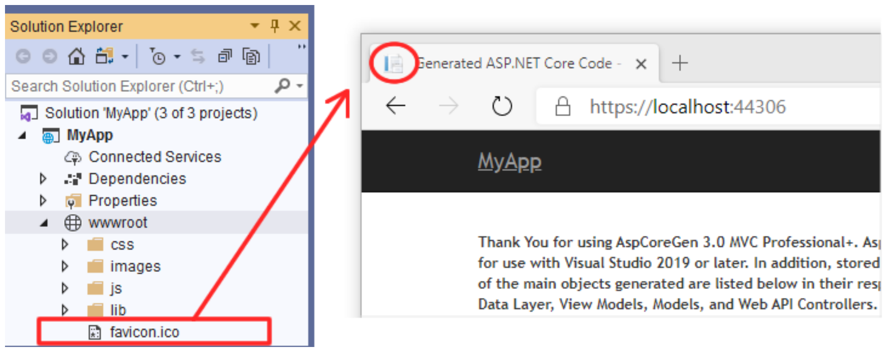


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'MyApp' (3 of 3 projects)
◢ MyApp
    Connected Services
  ▷ Dependencies
  ▷ Properties
  ◢ wwwroot
    ▷ css
    ▷ images
    ▷ js
    ◢ lib
      ▷ bootstrap
      ▷ jquery
      ▷ jquery-validation
      ▷ jquery-validation-unobtrusive
      favicon.ico
```

5. **favicon.ico:** An icon used by the browser as the default icon for your project. You can change this to your own icon (brand).



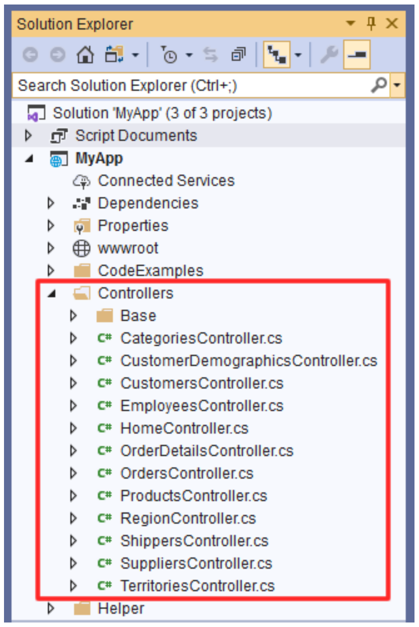
Thank You for using AspCoreGen 3.0 MVC Professional+. Asp
for use with Visual Studio 2019 or later. In addition, stored
of the main objects generated are listed below in their resp
Data Layer, View Models, Models, and Web API Controllers.

3.1.2 Controllers

This folder is generally needed by ASP.NET Core MVC by default. It houses *Controllers* used by the MVC *Views*. You can add your own *Controllers* here, and you don't need to copy the same layout such as that the *Controllers* generated by AspCoreGen 3.0 MVC inherits from a *Base (Parent) Controller*. Or, you could also create your own *Base (Parent) Controller* and a *Child Controller*, again, you don't need to, but you can, if you really, really, really want to.

You can also add your own *Methods* and or *Actions* in any existing *Child Controller* generated by AspCoreGen 3.0 MVC. *Child Controllers* are the *Controllers* directly beneath the *Controllers* folder. E.g. *CategoriesController.cs*, *CustomersController.cs*, *ProductsController.cs*, etc., these files **will not be overwritten** when you re-generated code for the same project (in this example - *MyApp*). Please see the *AppSettingsTab Tutorial*, page 4 (1.1.1 *Files That Will Be Written Once*) for more information.

Note: Do not add any code in any of the generated *Parent (Base) Controller*, these are the *Controllers* generated in the *Base* folder. Please see the *AppSettingsTab Tutorial*, page 4 (1.1.2 *Files That Will Always Be Overwritten*) for more information.

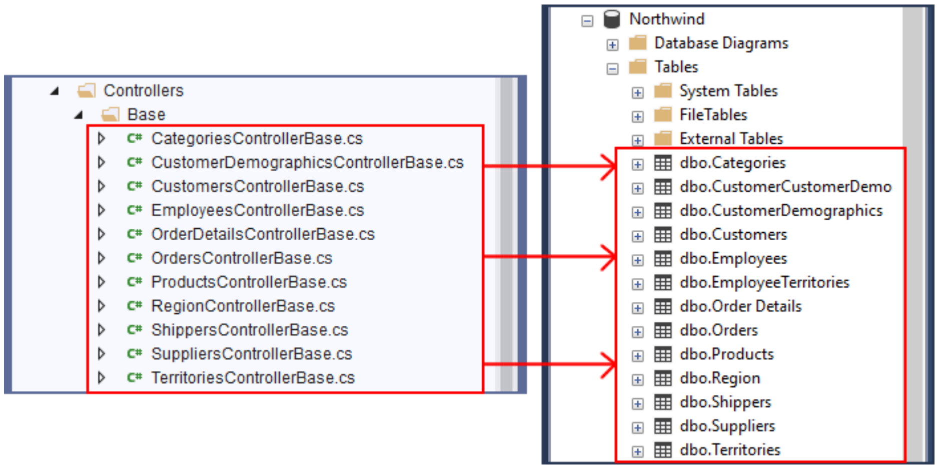


3.1.2.1 *Parent (Base) Class*

These are the class files generated in the *Base* folder. The naming convention used is: ***TableName****Controller****Base****.cs*. Because it's just a regular *Class* file, ASP.NET Core MVC does not really recognize it other than it being a *Class* file.

Do not add any code in these *Class* files.

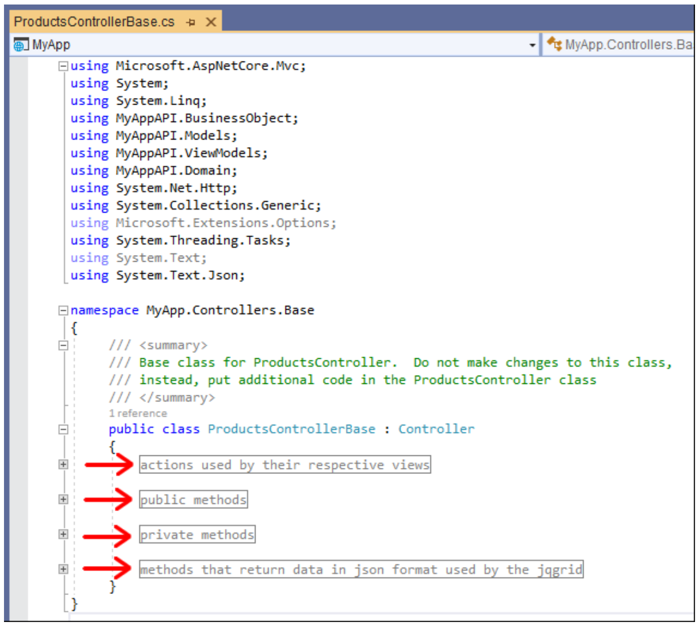
One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.



Base Controllers in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

3.1.2.1.1 Code Separated By Regions

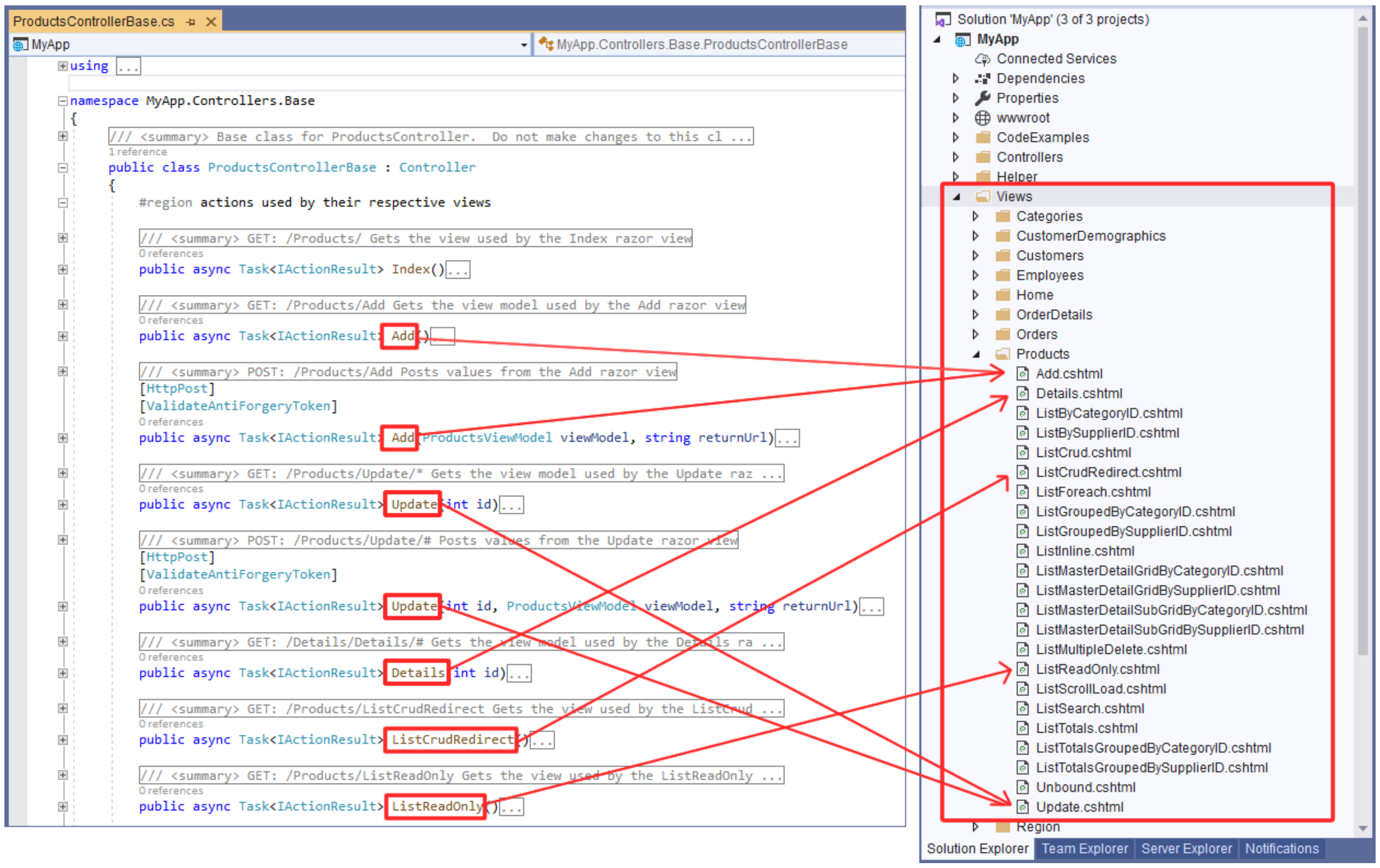
Because so much code is generated (depending on the number of *Database Tables* you have), the generated code is separated by *Regions* to classify the type of *Methods* that were generated.



```
using Microsoft.AspNetCore.Mvc;
using System;
using System.Linq;
using MyAppAPI.BusinessObject;
using MyAppAPI.Models;
using MyAppAPI.ViewModels;
using MyAppAPI.Domain;
using System.Net.Http;
using System.Collections.Generic;
using Microsoft.Extensions.Options;
using System.Threading.Tasks;
using System.Text;
using System.Text.Json;

namespace MyApp.Controllers.Base
{
     /// <summary>
     /// Base class for ProductsController.  Do not make changes to this class,
     /// instead, put additional code in the ProductsController class
     /// </summary>
     1 reference
     public class ProductsControllerBase : Controller
     {
         actions used by their respective views

         public methods

         private methods

         methods that return data in json format used by the jqgrid
     }
}
```

3.1.2.1.1.1 *Actions Used by Their Respective Views*

These are the *Action* methods used by their respective MVC *Views* under the *Views* folder. The example below is inherited by the *ProductsController (Child Class)* which makes every public method available to the *ProductsController*.



```
ProductsControllerBase.cs
MyApp                                   MyApp.Controllers.Base.ProductsControllerBase
using ...

namespace MyApp.Controllers.Base
{
    /// <summary> Base class for ProductsController.  Do not make changes to this cl ...
    1 reference
    public class ProductsControllerBase : Controller
    {
        #region actions used by their respective views

        /// <summary> GET: /Products/ Gets the view used by the Index razor view
        0 references
        public async Task<IActionResult> Index()...

        /// <summary> GET: /Products/Add Gets the view model used by the Add razor view
        0 references
        public async Task<IActionResult> Add()...

        /// <summary> POST: /Products/Add Posts values from the Add razor view
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Add(ProductsViewModel viewModel, string returnUrl)...

        /// <summary> GET: /Products/Update/* Gets the view model used by the Update raz ...
        0 references
        public async Task<IActionResult> Update(int id)...

        /// <summary> POST: /Products/Update/# Posts values from the Update razor view
        [HttpPost]
        [ValidateAntiForgeryToken]
        0 references
        public async Task<IActionResult> Update(int id, ProductsViewModel viewModel, string returnUrl)...

        /// <summary> GET: /Details/Details/# Gets the view model used by the Details ra ...
        0 references
        public async Task<IActionResult> Details(int id)...

        /// <summary> GET: /Products/ListCrudRedirect Gets the view used by the ListCrud ...
        0 references
        public async Task<IActionResult> ListCrudRedirect()...

        /// <summary> GET: /Products/ListReadOnly Gets the view used by the ListReadOnly ...
        0 references
        public async Task<IActionResult> ListReadOnly()...
```

```
Solution 'MyApp' (3 of 3 projects)
  MyApp
    Connected Services
    Dependencies
    Properties
    wwwroot
    CodeExamples
    Controllers
    Helper
    Views
      Categories
      CustomerDemographics
      Customers
      Employees
      Home
      OrderDetails
      Orders
      Products
        Add.cshtml
        Details.cshtml
        ListByCategoryID.cshtml
        ListBySupplierID.cshtml
        ListCrud.cshtml
        ListCrudRedirect.cshtml
        ListForeach.cshtml
        ListGroupedByCategoryID.cshtml
        ListGroupedBySupplierID.cshtml
        ListInline.cshtml
        ListMasterDetailGridByCategoryID.cshtml
        ListMasterDetailGridBySupplierID.cshtml
        ListMasterDetailSubGridByCategoryID.cshtml
        ListMasterDetailSubGridBySupplierID.cshtml
        ListMultipleDelete.cshtml
        ListReadOnly.cshtml
        ListScrollLoad.cshtml
        ListSearch.cshtml
        ListTotals.cshtml
        ListTotalsGroupedByCategoryID.cshtml
        ListTotalsGroupedBySupplierID.cshtml
        Unbound.cshtml
        Update.cshtml
      Region
Solution Explorer  Team Explorer  Server Explorer  Notifications
```

The ***Products****Controller* looks for the ***Products*** folder under *Views* folder by default. The same goes for the MVC *Views* in the *Views* folder under the ***Products*** folder, it will look for the respective action in the ***Products****Controller.*

For example, the ***Add****.cshtml View* under the ***Products*** folder will look for an ***Add*** *Action* method in the ***Products****Controller*. In the same way, the ***ListCrudRedirect****.cshtml View* under the ***Products*** folder will look for a ***ListCrudRedirect*** *Action* method in the ***Products****Controller*. So you see the pattern here.

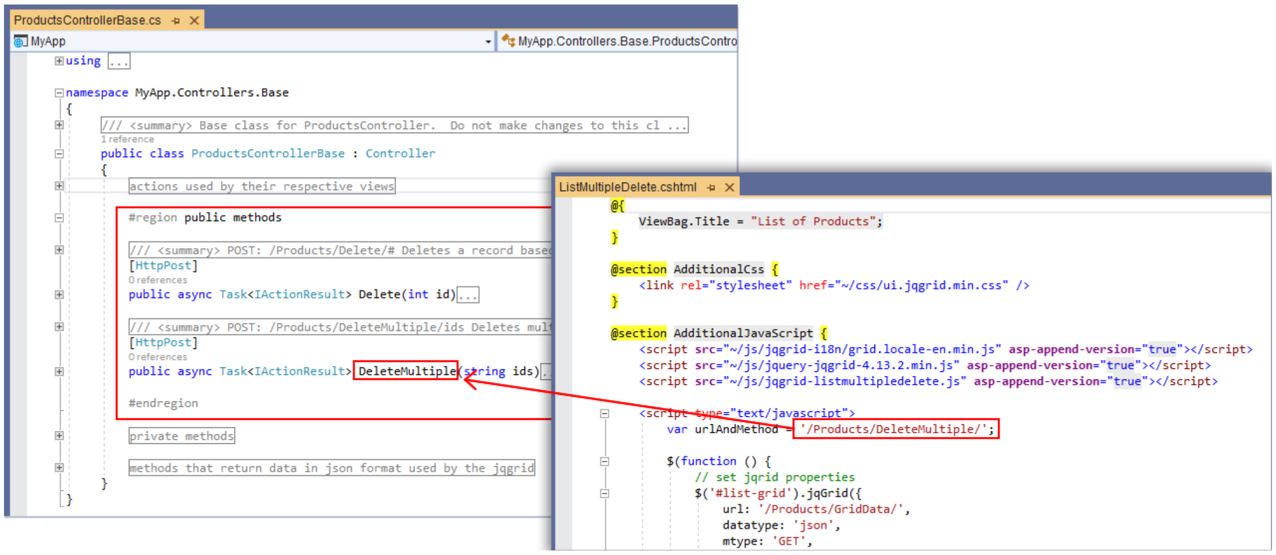
So why does the *Add* and *Update* have 2 *Action* methods each, while the *Details*, *ListCrudRedirect*, *ListReadOnly*, etc. only have 1 *Action* method each? The *Add* and *Update* MVC *Views* both required a *Get* and *Post Actions* methods, while the *Details*, *ListCrudRedirect*, *ListReadOnly* MVC Views only require a *Get Action* method.

Note: Each ASP.NET Core MVC *View* require a *Get Action* method minimum by default.

3.1.2.1.1.2 *Public Methods*

These are *Public **HttpPost** Web Methods* used by the generated MVC *Views*. These methods are called from a JavaScript client code. You can say that calls to these methods cross from a client (javascript) code to a server code (C#), some calls this AJAX functionality.

For example, the *Delete Multiple* functionality can be found in the generated MVC *View* and related *Controller (*technically the *Controller Base* Class*)* as shown below. When a user deletes multiple items, the generated *ListMultipleDelete.cshtml* MVC *View* looks for the ***Products**Controller* (remember this inherits from the *ProductsController**Base***) with a *Public DeleteMultiple Method* as highlighted in the MVC *View*'s code below: *'**Products**/DeleteMultiple'.* Code inside the *Controller*'s *DeleteMultiple* method is executed and the control flow is returned back to the calling *ListMultipleDelete.cshtml* MVC *View*.

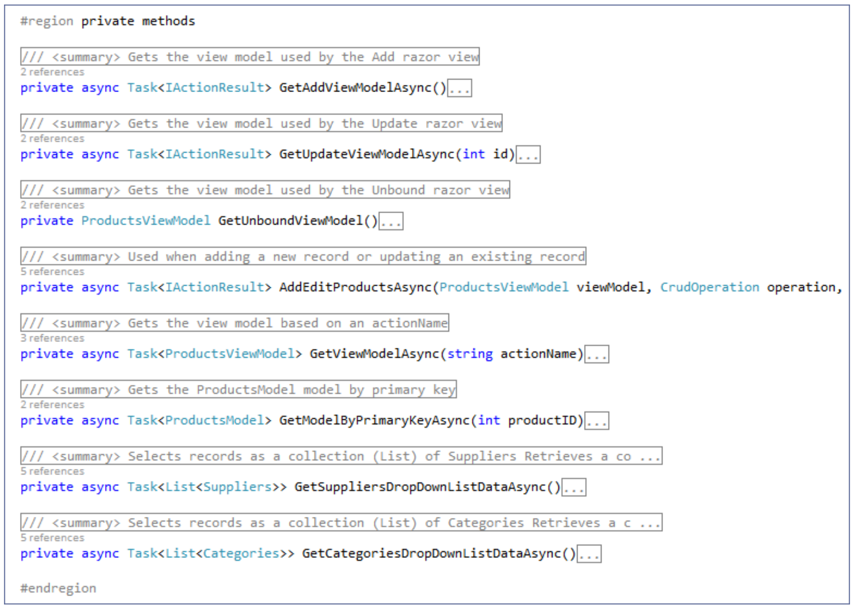

ProductsControllerBase.cs
MyApp
MyApp.Controllers.Base.ProductsContro
ListMultipleDelete.cshtml

```
using ...

namespace MyApp.Controllers.Base
{
    /// <summary> Base class for ProductsController.  Do not make changes to this cl ...
    1 reference
    public class ProductsControllerBase : Controller
    {
        actions used by their respective views

        #region public methods

        /// <summary> POST: /Products/Delete/# Deletes a record based
        [HttpPost]
        0 references
        public async Task<IActionResult> Delete(int id)...

        /// <summary> POST: /Products/DeleteMultiple/ids Deletes mult
        [HttpPost]
        0 references
        public async Task<IActionResult> DeleteMultiple(string ids)...

        #endregion

        private methods

        methods that return data in json format used by the jqgrid
    }
}
```

```
@{
    ViewBag.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-listmultipledelete.js" asp-append-version="true"></script>

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

        $(function () {
            // set jqrid properties
            $('#list-grid').jqGrid({
                url: '/Products/GridData/',
                datatype: 'json',
                mtype: 'GET',
```

3.1.2.1.1.3 *Private Methods*

These are reusable *Private Methods* called by other methods in the *Controller*.



```
#region private methods

/// <summary> Gets the view model used by the Add razor view
2 references
private async Task<IActionResult> GetAddViewModelAsync()...

/// <summary> Gets the view model used by the Update razor view
2 references
private async Task<IActionResult> GetUpdateViewModelAsync(int id)...

/// <summary> Gets the view model used by the Unbound razor view
2 references
private ProductsViewModel GetUnboundViewModel()...

/// <summary> Used when adding a new record or updating an existing record
5 references
private async Task<IActionResult> AddEditProductsAsync(ProductsViewModel viewModel, CrudOperation operation,

/// <summary> Gets the view model based on an actionName
3 references
private async Task<ProductsViewModel> GetViewModelAsync(string actionName)...

/// <summary> Gets the ProductsModel model by primary key
2 references
private async Task<ProductsModel> GetModelByPrimaryKeyAsync(int productID)...

/// <summary> Selects records as a collection (List) of Suppliers Retrieves a co ...
5 references
private async Task<List<Suppliers>> GetSuppliersDropDownListDataAsync()...

/// <summary> Selects records as a collection (List) of Categories Retrieves a c ...
5 references
private async Task<List<Categories>> GetCategoriesDropDownListDataAsync()...

#endregion
```

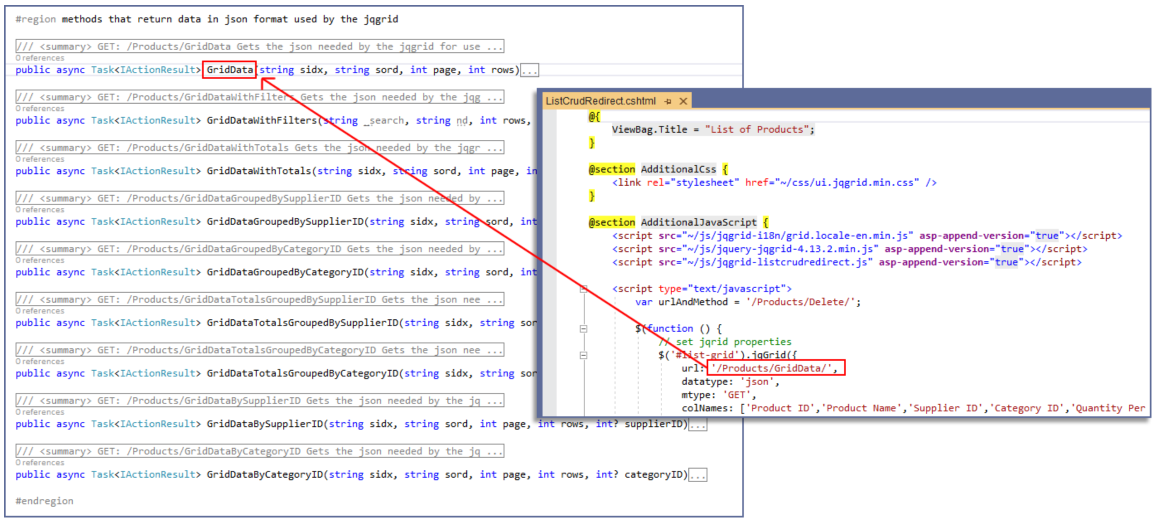
3.1.2.1.1.4 Methods that Return Data in JSON Format Use by the JQGrid

These are *Public **HttpGet** Web Methods* used by the generated MVC *Views*. These methods are called from a JavaScript client code. You can say that calls to these methods cross from a client (javascript) code to a server code (C#), some calls this AJAX functionality.

HttpGet Web Methods returns data to the calling client. In this instance, the client is a *JQGrid* plugin.

Note: These *Public HttpGet Web Methods* are not exclusively for use with a *JQGrid* client, any client can call them. So you can write your own custom code and call any of these *Public HttpGet Web Methods*.

For example, the *ListCrudRedirect.cshtml* MVC *View* uses the *JQGrid* plugin to pull data from the ***GridData***, a *Public Web Method* in the ***Products**Controller*.



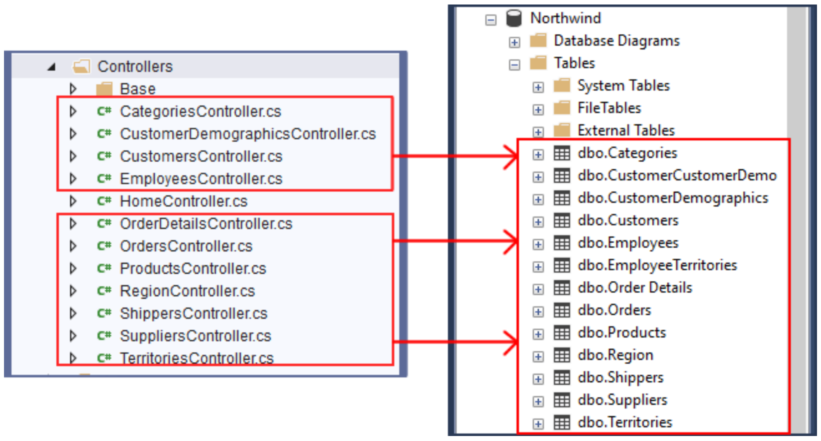
```
#region methods that return data in json format used by the jqgrid

/// <summary> GET: /Products/GridData Gets the json needed by the jqgrid for use ...
0 references
public async Task<IActionResult> GridData(string sidx, string sord, int page, int rows)...

/// <summary> GET: /Products/GridDataWithFilters Gets the json needed by the jqg ...
0 references
public async Task<IActionResult> GridDataWithFilters(string _search, string nd, int rows,

/// <summary> GET: /Products/GridDataWithTotals Gets the json needed by the jqgr ...
0 references
public async Task<IActionResult> GridDataWithTotals(string sidx, string sord, int page, in

/// <summary> GET: /Products/GridDataGroupedBySupplierID Gets the json needed by ...
0 references
public async Task<IActionResult> GridDataGroupedBySupplierID(string sidx, string sord, int

/// <summary> GET: /Products/GridDataGroupedByCategoryID Gets the json needed by ...
0 references
public async Task<IActionResult> GridDataGroupedByCategoryID(string sidx, string sord, int

/// <summary> GET: /Products/GridDataTotalsGroupedBySupplierID Gets the json nee ...
0 references
public async Task<IActionResult> GridDataTotalsGroupedBySupplierID(string sidx, string sor

/// <summary> GET: /Products/GridDataTotalsGroupedByCategoryID Gets the json nee ...
0 references
public async Task<IActionResult> GridDataTotalsGroupedByCategoryID(string sidx, string sor

/// <summary> GET: /Products/GridDataBySupplierID Gets the json needed by the jq ...
0 references
public async Task<IActionResult> GridDataBySupplierID(string sidx, string sord, int page, int rows, int? supplierID)...

/// <summary> GET: /Products/GridDataByCategoryID Gets the json needed by the jq ...
0 references
public async Task<IActionResult> GridDataByCategoryID(string sidx, string sord, int page, int rows, int? categoryID)...

#endregion
```

ListCrudRedirect.cshtml

```
@{
    ViewBag.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/Delete/';

        $(function () {
            // set jqrid properties
            $('#list-grid').jqGrid({
                url: '/Products/GridData/',
                datatype: 'json',
                mtype: 'GET',
                colNames: ['Product ID','Product Name','Supplier ID','Category ID','Quantity Per
```

3.1.2.2 *Child Class – The Controller*

These are the *Class* files generated directly under the *Controllers* folder (not including everything inside the *Base* folder). The naming convention used is: ***TableName**Controller.cs*. ASP.NET Core MVC recognizes this as a *Controller* by default because of the suffix "*Controller*" in the name. One *Controller* is generated per *Database Table*. **You can add code in these *Class* files.**



```
Controllers
    Base
    CategoriesController.cs
    CustomerDemographicsController.cs
    CustomersController.cs
    EmployeesController.cs
    HomeController.cs
    OrderDetailsController.cs
    OrdersController.cs
    ProductsController.cs
    RegionController.cs
    ShippersController.cs
    SuppliersController.cs
    TerritoriesController.cs
```

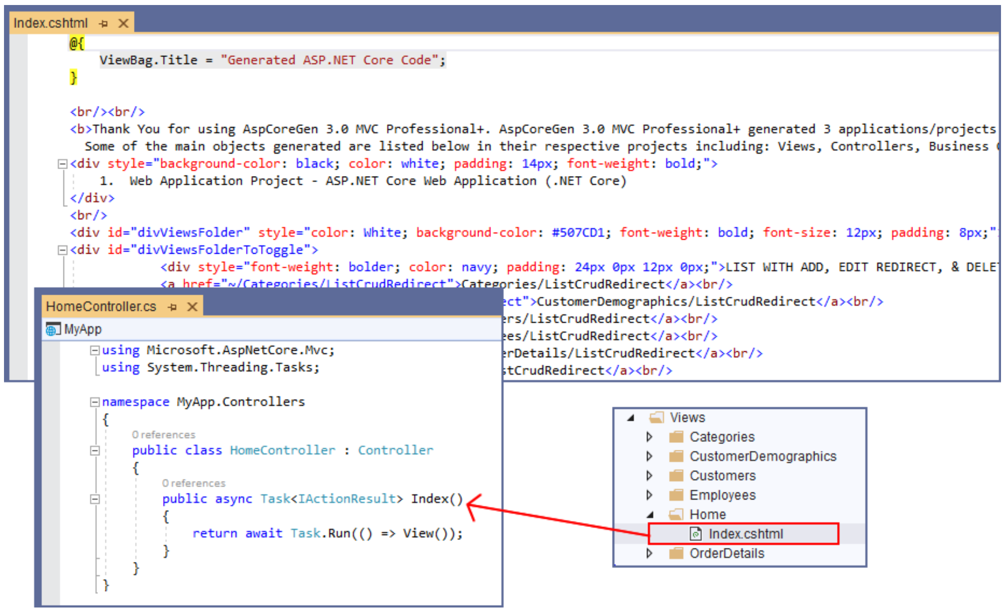
```
Northwind
    Database Diagrams
    Tables
        System Tables
        FileTables
        External Tables
        dbo.Categories
        dbo.CustomerCustomerDemo
        dbo.CustomerDemographics
        dbo.Customers
        dbo.Employees
        dbo.EmployeeTerritories
        dbo.Order Details
        dbo.Orders
        dbo.Products
        dbo.Region
        dbo.Shippers
        dbo.Suppliers
        dbo.Territories
```

Child Controllers in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

3.1.2.2.1.1 HomeController

The *HomeController.cs* is unlike the other *Controllers*. It's the only *Controller* that is generated without a *Parent (Base) Class*, so you will not see a *HomeController**Base*** in the *Base* folder. This is because it is not generated for a database table, instead, it is used to host the *Index Action Method*.

As shown in the example below, *Index.cshtml View* looks for the related *Index Action Method* in the *HomeController*.



Index.cshtml

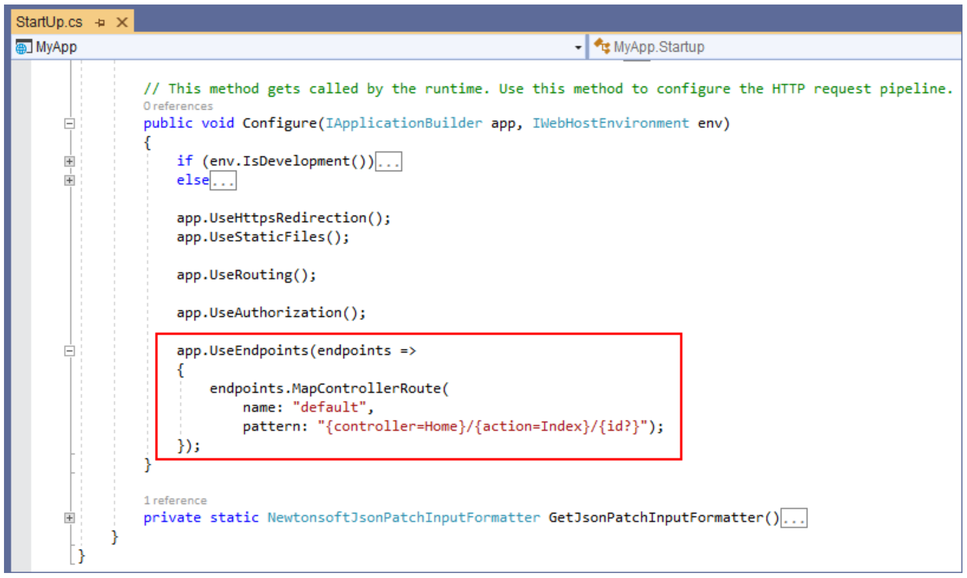
```
@{
    ViewBag.Title = "Generated ASP.NET Core Code";
}

<br/><br/>
<b>Thank You for using AspCoreGen 3.0 MVC Professional+. AspCoreGen 3.0 MVC Professional+ generated 3 applications/projects
  Some of the main objects generated are listed below in their respective projects including: Views, Controllers, Business
<div style="background-color: black; color: white; padding: 14px; font-weight: bold;">
    1.  Web Application Project - ASP.NET Core Web Application (.NET Core)
</div>
<br/>
<div id="divViewsFolder" style="color: White; background-color: #507CD1; font-weight: bold; font-size: 12px; padding: 8px;"
<div id="divViewsFolderToggle">
            <div style="font-weight: bolder; color: navy; padding: 24px 0px 12px 0px;">LIST WITH ADD, EDIT REDIRECT, & DELE
            <a href="~/Categories/ListCrudRedirect">Categories/ListCrudRedirect</a><br/>
ct">CustomerDemographics/ListCrudRedirect</a><br/>
rs/ListCrudRedirect</a><br/>
es/ListCrudRedirect</a><br/>
rDetails/ListCrudRedirect</a><br/>
tCrudRedirect</a><br/>
```

HomeController.cs

MyApp

```
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace MyApp.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public async Task<IActionResult> Index()
        {
            return await Task.Run(() => View());
        }
    }
}
```

The *Index.cshtml* MVC View is the *Default* page for the generated *Web Application Project*. It is the first page that is launched when you run the generated *Web Application Project* in Visual Studio. It lists all the main objects generated by AspCoreGen 3.0 MVC. ASP.NET Core MVC looks for an *Index.cshtml View* in the *HomeController* to run as the default page as set up by the generated code in the *StartUp.cs* class as shown below.



StartUp.cs

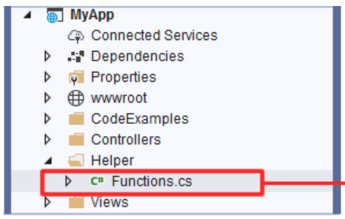
MyApp | MyApp.Startup

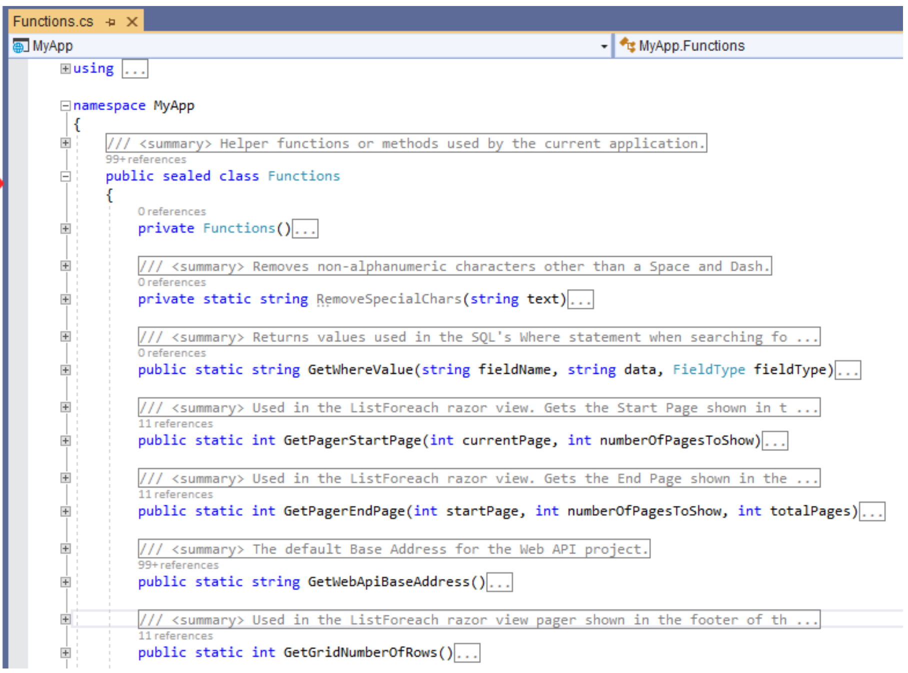
```
        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())...
            else...

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Home}/{action=Index}/{id?}");
            });
        }

        1 reference
        private static NewtonsoftJsonPatchInputFormatter GetJsonPatchInputFormatter()...
    }
}
```

3.1.3 Helper

This folder houses helper *Class*(es).

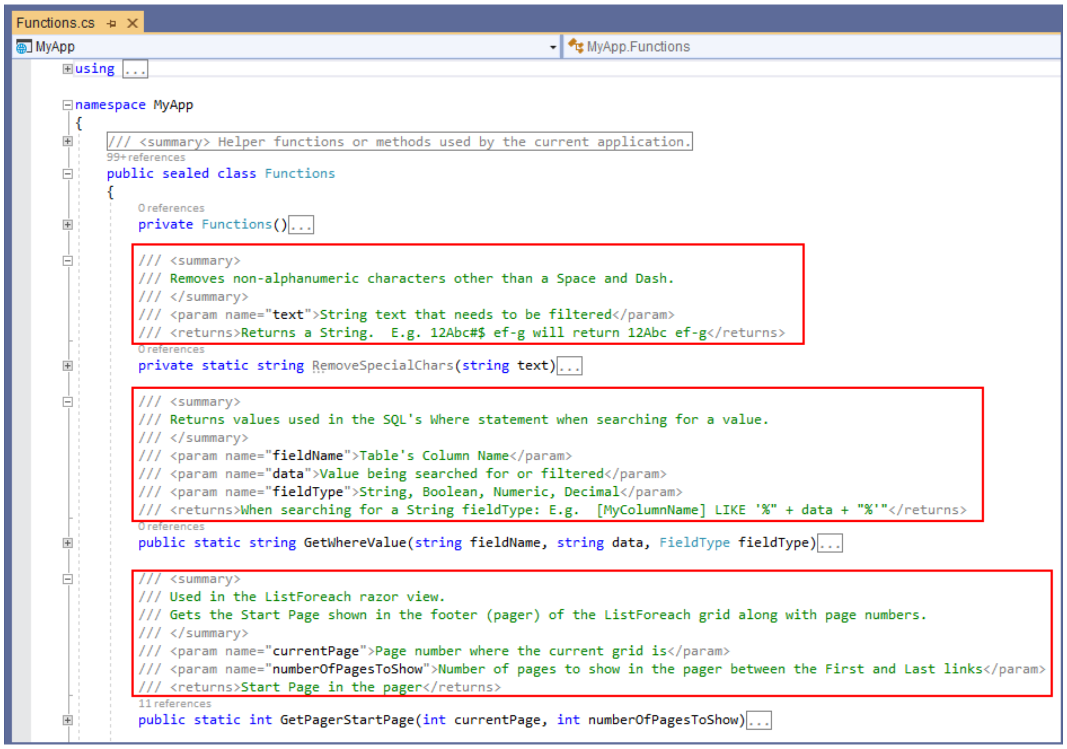
1. **Functions.cs:** Reusable *Functions* or *Methods* used by the *Front-End* application. **You can add your own code here.**





```
using ...

namespace MyApp
{
    /// <summary> Helper functions or methods used by the current application.
    99+ references
    public sealed class Functions
    {
        0 references
        private Functions()...

        /// <summary> Removes non-alphanumeric characters other than a Space and Dash.
        0 references
        private static string RemoveSpecialChars(string text)...

        /// <summary> Returns values used in the SQL's Where statement when searching fo ...
        0 references
        public static string GetWhereValue(string fieldName, string data, FieldType fieldType)...

        /// <summary> Used in the ListForeach razor view. Gets the Start Page shown in t ...
        11 references
        public static int GetPagerStartPage(int currentPage, int numberOfPagesToShow)...

        /// <summary> Used in the ListForeach razor view. Gets the End Page shown in the ...
        11 references
        public static int GetPagerEndPage(int startPage, int numberOfPagesToShow, int totalPages)...

        /// <summary> The default Base Address for the Web API project.
        99+ references
        public static string GetWebApiBaseAddress()...

        /// <summary> Used in the ListForeach razor view pager shown in the footer of th ...
        11 references
        public static int GetGridNumberOfRows()...
```

Read the documentation comments on each one of the methods to learn about their respective functionalities.



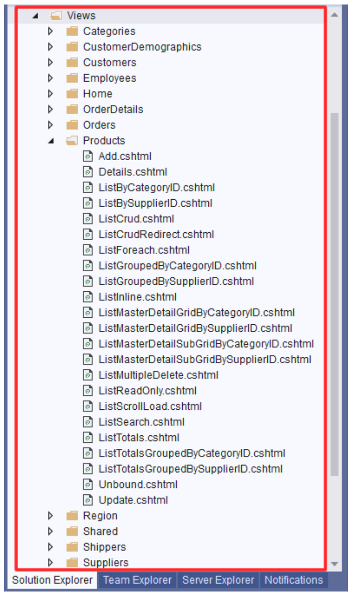
```
using ...

namespace MyApp
{
    /// <summary> Helper functions or methods used by the current application.
    99+ references
    public sealed class Functions
    {
        0 references
        private Functions()...

        /// <summary>
        /// Removes non-alphanumeric characters other than a Space and Dash.
        /// </summary>
        /// <param name="text">String text that needs to be filtered</param>
        /// <returns>Returns a String.  E.g. 12Abc#$ ef-g will return 12Abc ef-g</returns>
        0 references
        private static string RemoveSpecialChars(string text)...

        /// <summary>
        /// Returns values used in the SQL's Where statement when searching for a value.
        /// </summary>
        /// <param name="fieldName">Table's Column Name</param>
        /// <param name="data">Value being searched for or filtered</param>
        /// <param name="fieldType">String, Boolean, Numeric, Decimal</param>
        /// <returns>When searching for a String fieldType: E.g.  [MyColumnName] LIKE '%" + data + "%'"</returns>
        0 references
        public static string GetWhereValue(string fieldName, string data, FieldType fieldType)...

        /// <summary>
        /// Used in the ListForeach razor view.
        /// Gets the Start Page shown in the footer (pager) of the ListForeach grid along with page numbers.
        /// </summary>
        /// <param name="currentPage">Page number where the current grid is</param>
        /// <param name="numberOfPagesToShow">Number of pages to show in the pager between the First and Last links</param>
        /// <returns>Start Page in the pager</returns>
        11 references
        public static int GetPagerStartPage(int currentPage, int numberOfPagesToShow)...
```

3.1.4 Views

This folder is generally needed by ASP.NET Core MVC by default. It houses MVC *Views*. **You can add your own MVC *Views* here.** All the MVC *Views* generated by AspCoreGen 3.0 MVC will be overwritten the next time you generated code for the same project.

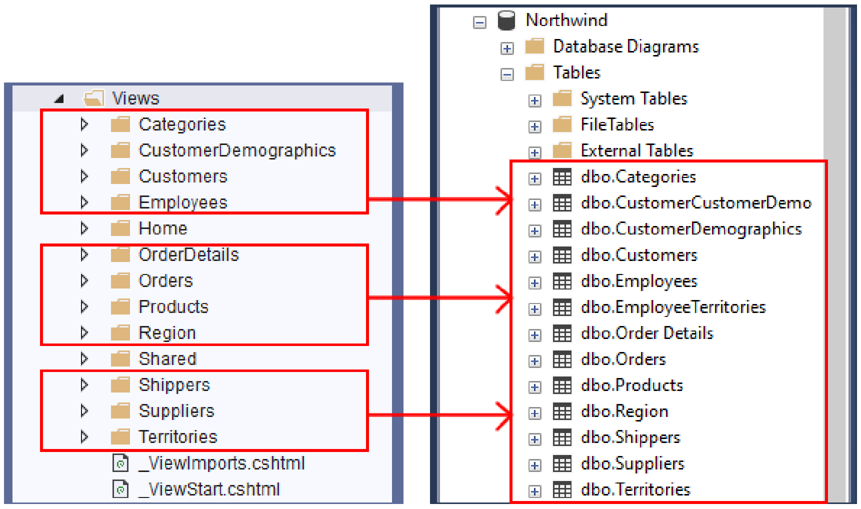
Note: Do not add any code in any of the generated MVC *Views*. Please see the *AppSettingsTab Tutorial*, page 5 (1.1.2 *Files That Will Always Be Overwritten*) for more information.

For more information on the different kinds of MVC *Views* generated by AspCoreGen 3.0 MVC, please see the *UISettingsTab Tutorial* on *Views to Generate*, starting in page 6.



3.1.4.1 *Views Generated for Database Tables*

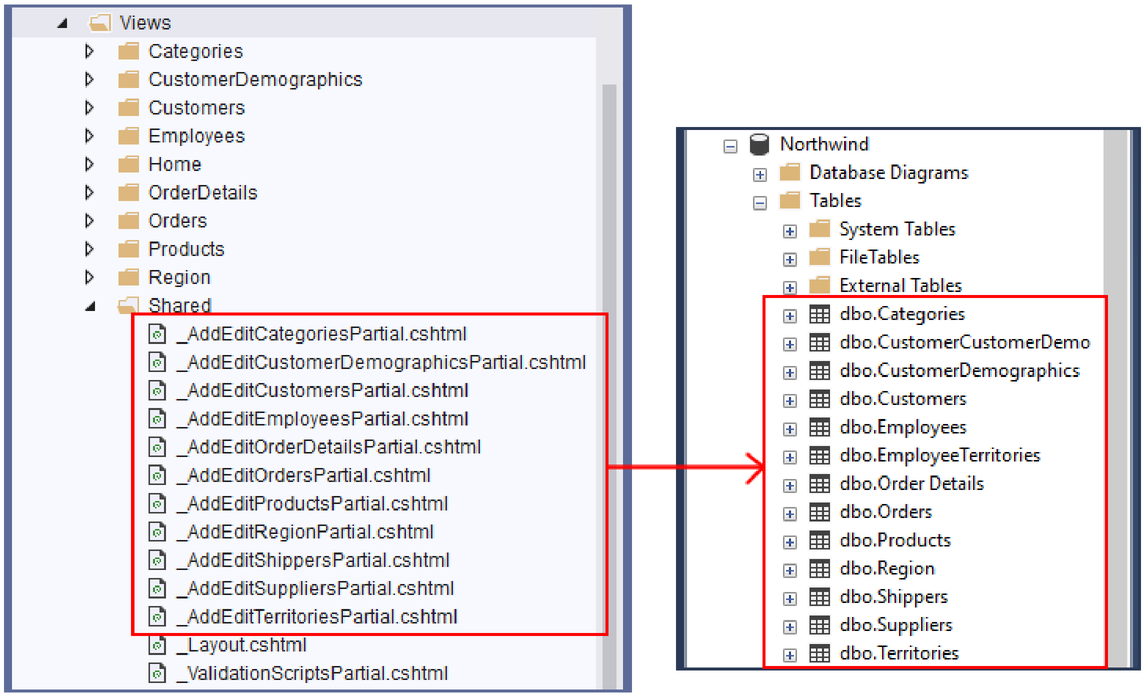
These MVC *Views* are generated based on the *Database Tables* you generated code for. Each *Folder* as shown below is directly related to a *Database Table*.



Views in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

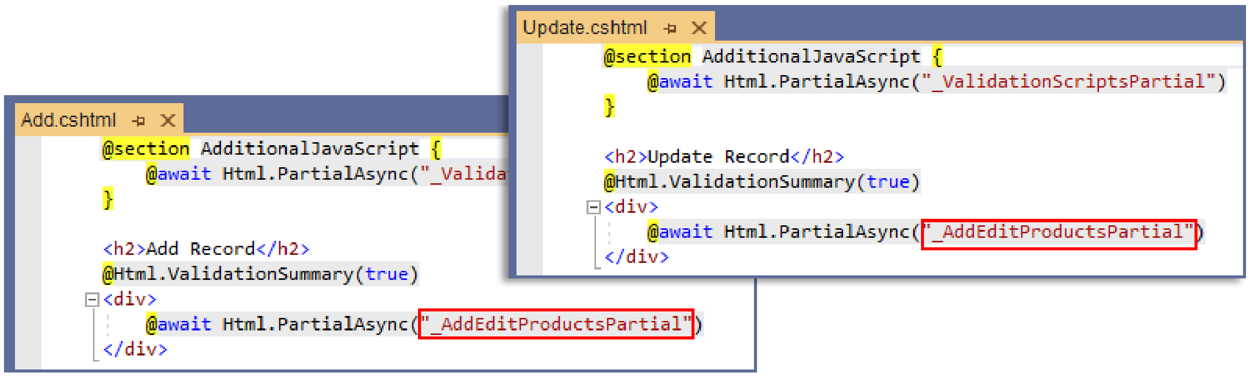
3.1.4.2 *Partial Views for Database Tables*

These *Partial Views* are generated based on the *Database Tables* you generated code for. Each *Partial View* is directly related to the respective *Database Table* as shown below and has a prefix *"_AddEdit". Partial Views* are located in the *Views/Shared Folder*. The ASP.NET Core MVC naming convention for *Partial Views* starts with an *Underscore* "_" prefix.


Views
Categories
CustomerDemographics
Customers
Employees
Home
OrderDetails
Orders
Products
Region
Shared
_AddEditCategoriesPartial.cshtml
_AddEditCustomerDemographicsPartial.cshtml
_AddEditCustomersPartial.cshtml
_AddEditEmployeesPartial.cshtml
_AddEditOrderDetailsPartial.cshtml
_AddEditOrdersPartial.cshtml
_AddEditProductsPartial.cshtml
_AddEditRegionPartial.cshtml
_AddEditShippersPartial.cshtml
_AddEditSuppliersPartial.cshtml
_AddEditTerritoriesPartial.cshtml
_Layout.cshtml
_ValidationScriptsPartial.cshtml
Northwind
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Categories
dbo.CustomerCustomerDemo
dbo.CustomerDemographics
dbo.Customers
dbo.Employees
dbo.EmployeeTerritories
dbo.Order Details
dbo.Orders
dbo.Products
dbo.Region
dbo.Shippers
dbo.Suppliers
dbo.Territories

Partial Views in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

Each *Partial View* is used by the *Add.cshtml* and *Update.cshtml* MVC *Views*.

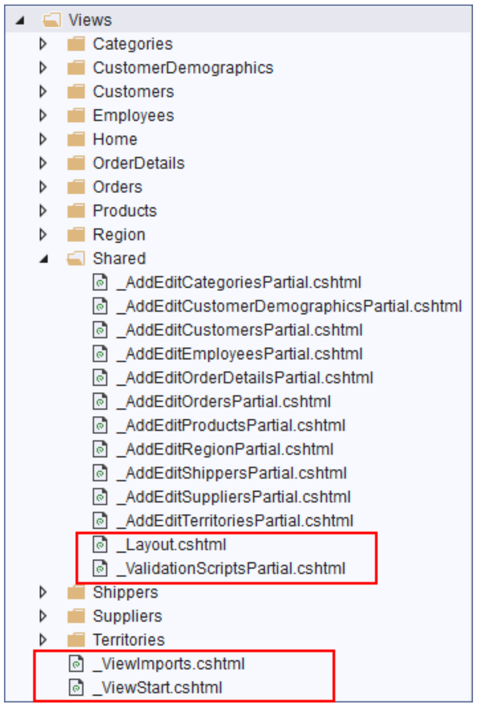


```
@section AdditionalJavaScript {
    @await Html.PartialAsync("_ValidationScriptsPartial")
}

<h2>Update Record</h2>
@Html.ValidationSummary(true)
<div>
    @await Html.PartialAsync("_AddEditProductsPartial")
</div>
```

```
@section AdditionalJavaScript {
    @await Html.PartialAsync("_Valida
}

<h2>Add Record</h2>
@Html.ValidationSummary(true)
<div>
    @await Html.PartialAsync("_AddEditProductsPartial")
</div>
```

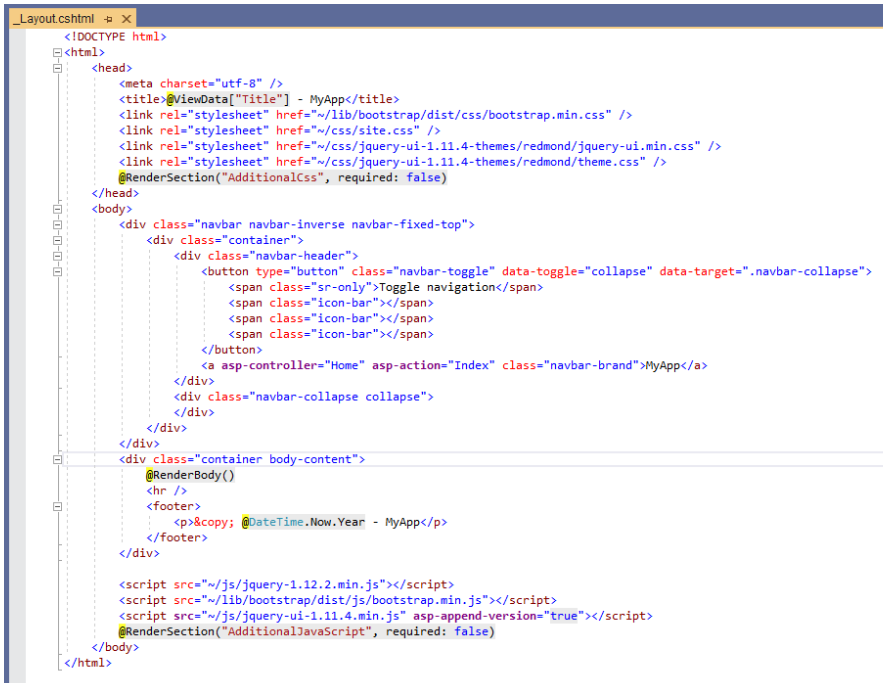
3.1.4.3 *Other Partial Views*

These are mainly ASP.NET Core MVC default *Partial Views*. The ASP.NET Core MVC naming convention for *Partial Views* starts with an *Underscore* "_" prefix.


Views
Categories
CustomerDemographics
Customers
Employees
Home
OrderDetails
Orders
Products
Region
Shared
_AddEditCategoriesPartial.cshtml
_AddEditCustomerDemographicsPartial.cshtml
_AddEditCustomersPartial.cshtml
_AddEditEmployeesPartial.cshtml
_AddEditOrderDetailsPartial.cshtml
_AddEditOrdersPartial.cshtml
_AddEditProductsPartial.cshtml
_AddEditRegionPartial.cshtml
_AddEditShippersPartial.cshtml
_AddEditSuppliersPartial.cshtml
_AddEditTerritoriesPartial.cshtml
_Layout.cshtml
_ValidationScriptsPartial.cshtml
Shippers
Suppliers
Territories
_ViewImports.cshtml
_ViewStart.cshtml

3.1.4.3.1 _Layout.cshtml

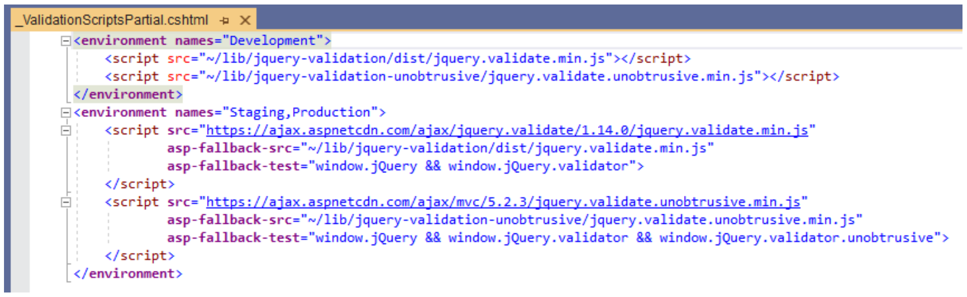
The *_Layout.cshtml* is a *Partial View* that is the default overall design or master page for all the MVC *Views* that incorporates it. MVC *Views* that incorporate the *_Layout.cshtml* starts it's code base where it shows the *@RenderBody()* code shown below. **You can change the overall design of all the generated MVC *Views* by changing all or a few code here.**



```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>@ViewData["Title"] - MyApp</title>
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
        <link rel="stylesheet" href="~/css/site.css" />
        <link rel="stylesheet" href="~/css/jquery-ui-1.11.4-themes/redmond/jquery-ui.min.css" />
        <link rel="stylesheet" href="~/css/jquery-ui-1.11.4-themes/redmond/theme.css" />
        @RenderSection("AdditionalCss", required: false)
    </head>
    <body>
        <div class="navbar navbar-inverse navbar-fixed-top">
            <div class="container">
                <div class="navbar-header">
                    <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                        <span class="sr-only">Toggle navigation</span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>
                    <a asp-controller="Home" asp-action="Index" class="navbar-brand">MyApp</a>
                </div>
                <div class="navbar-collapse collapse">
                </div>
            </div>
        </div>
        <div class="container body-content">
            @RenderBody()
            <hr />
            <footer>
                <p>&copy; @DateTime.Now.Year - MyApp</p>
            </footer>
        </div>

        <script src="~/js/jquery-1.12.2.min.js"></script>
        <script src="~/lib/bootstrap/dist/js/bootstrap.min.js"></script>
        <script src="~/js/jquery-ui-1.11.4.min.js" asp-append-version="true"></script>
        @RenderSection("AdditionalJavaScript", required: false)
    </body>
</html>
```

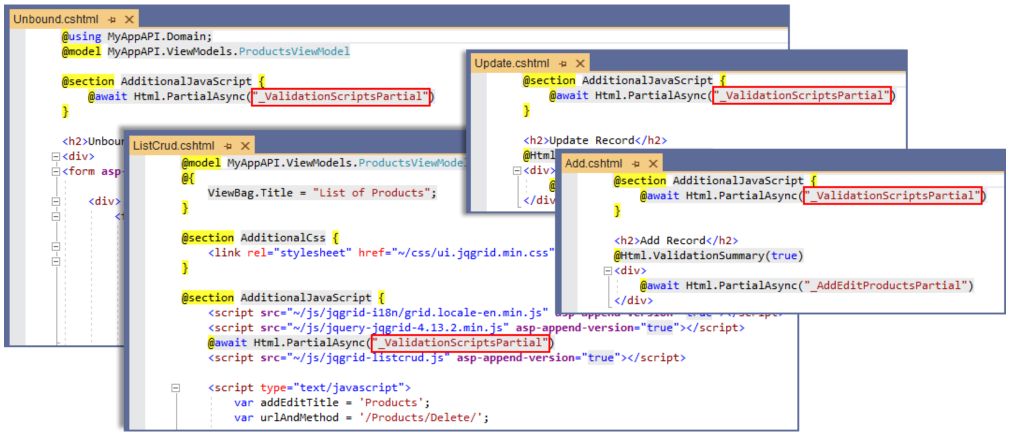
3.1.4.3.2 _ValidationScriptPartial.cshtml

The _*ValidationScriptPartial.cshtml* is a *Partial View* that references javascript (jQuery) libraries for use when validating controls for errors. **You can add your own code here**.


_ValidationScriptsPartial.cshtml

```
<environment names="Development">
    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
</environment>
<environment names="Staging,Production">
    <script src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.14.0/jquery.validate.min.js"
            asp-fallback-src="~/lib/jquery-validation/dist/jquery.validate.min.js"
            asp-fallback-test="window.jQuery && window.jQuery.validator">
    </script>
    <script src="https://ajax.aspnetcdn.com/ajax/mvc/5.2.3/jquery.validate.unobtrusive.min.js"
            asp-fallback-src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"
            asp-fallback-test="window.jQuery && window.jQuery.validator && window.jQuery.validator.unobtrusive">
    </script>
</environment>
```

It is used by MVC *Views*: *Add.cshtml, Update.cshtml, Unbound.cshtml,* and *ListCrud.cshtml* as shown below.


Unbound.cshtml

```
@using MyAppAPI.Domain;
@model MyAppAPI.ViewModels.ProductsViewModel

@section AdditionalJavaScript {
    @await Html.PartialAsync("_ValidationScriptsPartial")
}

<h2>Unbou
<div>
<form asp
    <div>
        <
```

Update.cshtml

```
@section AdditionalJavaScript {
    @await Html.PartialAsync("_ValidationScriptsPartial")
}

<h2>Update Record</h2>
@Html
<div>
    @
</div
```

ListCrud.cshtml

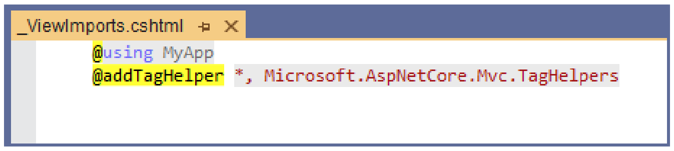
```
@model MyAppAPI.ViewModels.ProductsViewModel
@{
    ViewBag.Title = "List of Products";
}

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

@section AdditionalJavaScript {
    <script src="~/js/jqgrid-i18n/grid.locale-en.min.js"
    <script src="~/js/jquery-jqgrid-4.13.2.min.js" asp-append-version="true"></script>
    @await Html.PartialAsync("_ValidationScriptsPartial")
    <script src="~/js/jqgrid-listcrud.js" asp-append-version="true"></script>

    <script type="text/javascript">
        var addEditTitle = 'Products';
        var urlAndMethod = '/Products/Delete/';
```

Add.cshtml

```
@section AdditionalJavaScript {
    @await Html.PartialAsync("_ValidationScriptsPartial")
}

<h2>Add Record</h2>
@Html.ValidationSummary(true)
<div>
    @await Html.PartialAsync("_AddEditProductsPartial")
</div>
```

3.1.4.3.3 _ViewImports.cshtml

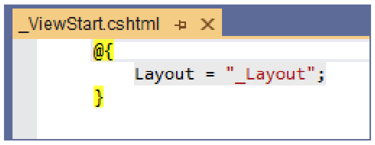
This *Partial View* imports directives that can be shared throughout all the generated MVC *Views*. **You can add your own code here**.


_ViewImports.cshtml

```
@using MyApp
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

3.1.4.3.4 _ViewStart.cshtml

By default, this *Partial View* is ran before any MVC *View*. **You can add your own code here**.


_ViewStart.cshtml

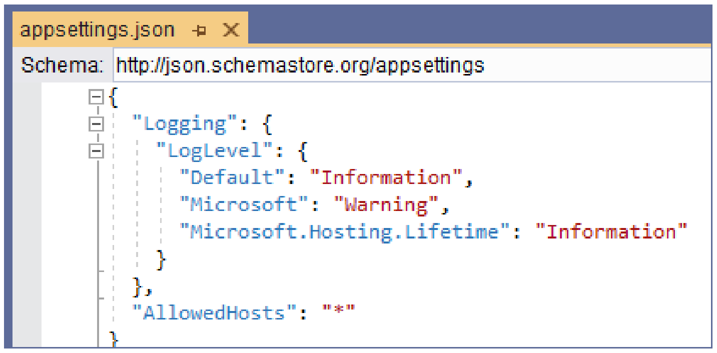
```
@{
    Layout = "_Layout";
}
```

3.1.4.4 *Index.cshtml View*

This MVC *View* is the default page of the *Web Application Project*. The *Index Action Method* can be found in the *HomeController*. Please read about the *HomeController* in page 14 for more information on the *Index View*.

3.1.5 appsettings.json

This is a settings json file used by the ASP.NET MVC Core *Web Application Project* by default. **You can add your own code here**.

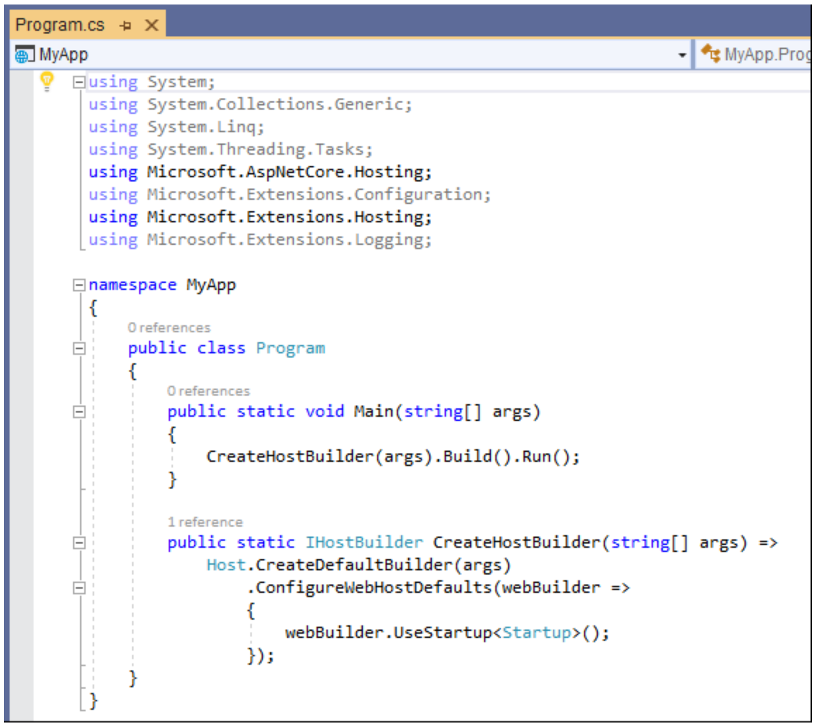


Schema: http://json.schemastore.org/appsettings

```json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

3.1.6 Program.cs

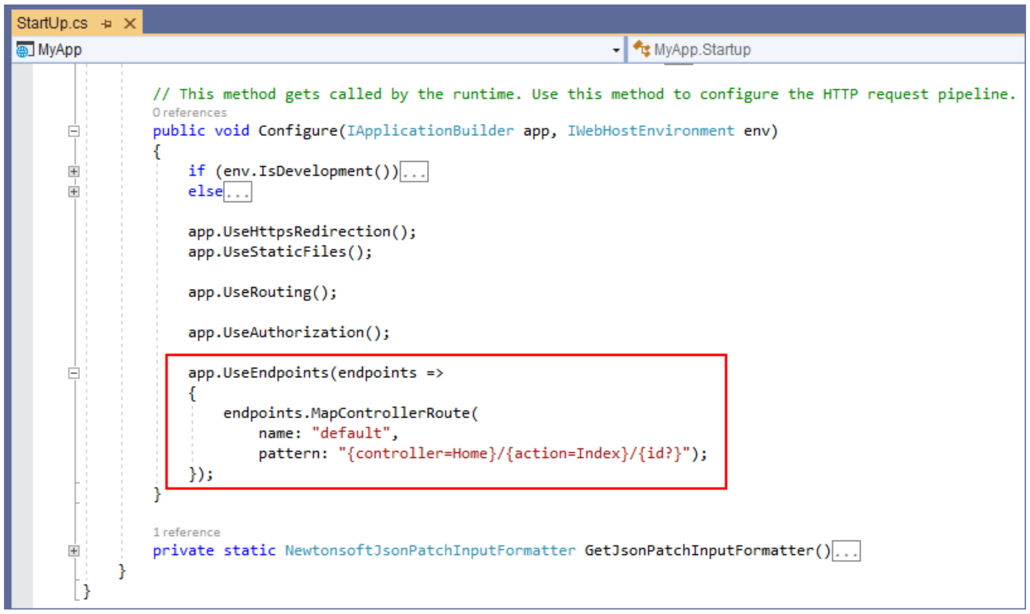
The *Program.cs Class* is the entry point to the ASP.NET MVC Core *Web Application Project* by default. An ASP.NET MVC Core web application project is technically a *Console* app. Just like any *Console* app, execution of the app starts at the *Program Class's **Main()** Method.* **You can add your own code here**.



```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace MyApp
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

3.1.7 StartUp.cs

The *StartUp Class* also starts execution before any ASP.NET MVC Core *View* runs and is called by the *Program Class* as shown above, *"webBuilder.UserStartup<**Startup**>()".* Here, you can set or configure services that can be used globally in the *Web Application Project* by default. **You can add your own code here**.

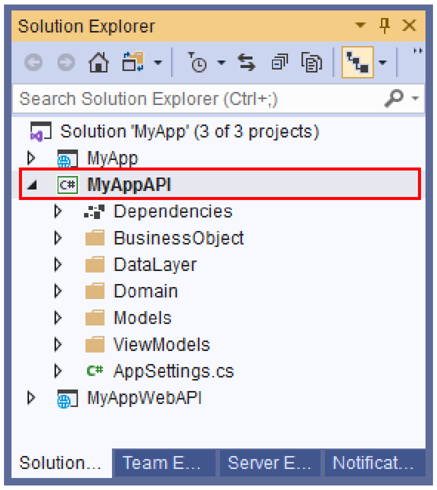


```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())...
    else...

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}

1 reference
private static NewtonsoftJsonPatchInputFormatter GetJsonPatchInputFormatter()...
    }
}
```

3.2 Business Object and Data Layer API Project

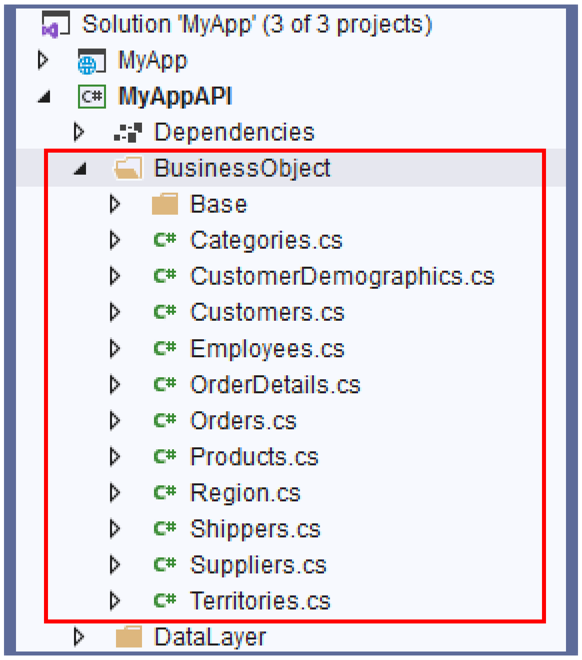
The generated *Business Object and Data Layer API Project* contains the *Middle-Tier* and *Data-Tier* part of the N-tier layer generated code (and other classes as well). This is a *Class Library* core project. ***Classes/Objects* here can be reused by other projects/clients except the *Data Layer Classes* and the *AppSettings Class*.**

The *Business Object and Data Layer API Project* is referenced by the *Web Application Project* and *Web API Project* for use. You can also reference this project from other projects that you add to the generated *Solution* or altogether copy the whole project to your own custom projects, and many more possibilities for reuse.



3.2.1 Middle Tier (Business Object)

The *Business Object (Middler Tier) Class Files* are located in the *BusinessObject Folder*.


Solution 'MyApp' (3 of 3 projects)
MyApp
MyAppAPI
Dependencies
BusinessObject
Base
Categories.cs
CustomerDemographics.cs
Customers.cs
Employees.cs
OrderDetails.cs
Orders.cs
Products.cs
Region.cs
Shippers.cs
Suppliers.cs
Territories.cs
DataLayer

The *Middle-Tier*'s (or *Middle Layer*) main purpose is to serve as a client's (a program's) only access to the *Business Objects*. A *Business Object*'s purpose is to calculate things. For the purposes of AspCoreGen 3.0 MVC code generation, in most parts, there really is nothing to calculate, instead, the *Business Object* classes just returns data handed to it by the Data Layer classes, or carries and passes the CRUD* operations that the *Data Layer* need to handle.

The *Calculations* we are talking about here are not just math problems, instead, these are logic that the *Client* (controller, asp.net web form, web api, wcf program etc.) needs. For most parts, any *Client* should not be doing any kind of *Calculation*, instead, a line code referencing a *Middle-Tier Class's Method* should readily return that logic.

For example (this is just an example and not generated by AspCoreGen 3.0 MVC), let's say somewhere in the *Controller* it needs the full name of a person.

var fullName = User.GetFullName();

In this example, *"User"* is the *Business Object (Middle-Tier Class)*, "*GetFullName()*" is a *Public Method* in the *"User" Business Object Class.* Somewhere in the "*GetFullName()*" *Method,* it's calculating the first name and last name of the user, return a full name, e.g.

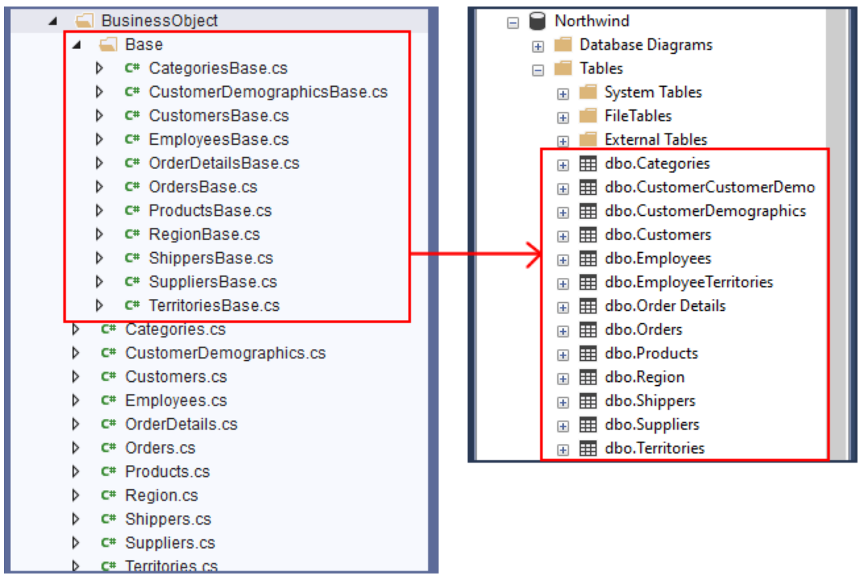
return User.FirstName + " " + User.LastName;

3.2.1.1 *Parent (Base) Class*

These are the class files generated in the *Base* folder. The naming convention used is: *TableName**Base**.cs*.

Do not add any code in these *Class* files.

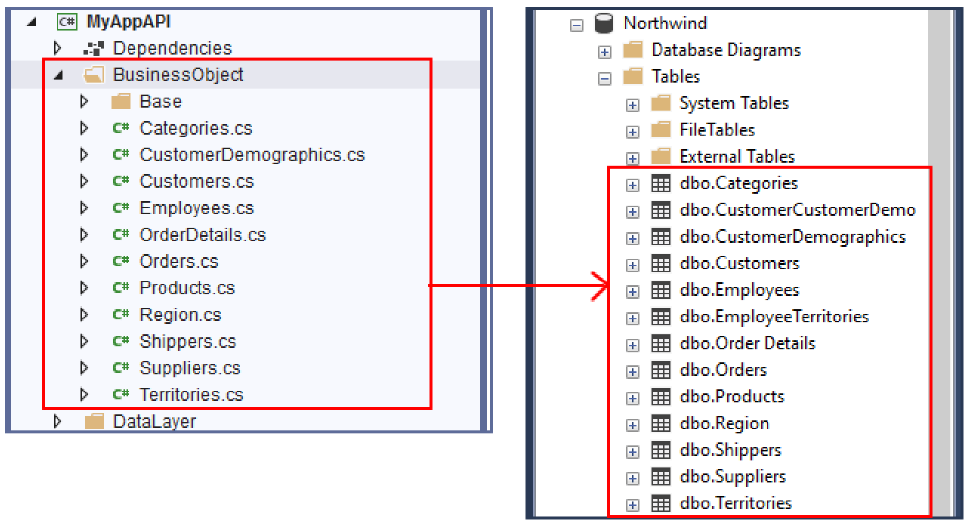
One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.



Base (Parent) Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

3.2.1.2 *Child Class – The Business Object*

These are the *Class* files generated directly under the *BusinessObject* folder (not including everything inside the *Base* folder). The naming convention used is: ***TableName**.cs*. One *Business Object Class* is generated per *Database Table*.



Child Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

You can add code in these *Class* files. You access all the *Business Object* methods and properties using this Class.

These are the *Classes* that **any client** should access, the *Middle Tier Classes* unless you also generated the optional *Web API Project*. Otherwise, you can also access the *Web API Project's* public methods.

Note 1: When you generate the optional *Web API Project,* AspCoreGen 3.0 MVC's generated code will always access *Web API Methods* from clients like the *Controller* class. These *Web API Methods* encapsulates calls to the related/respective *Business Object Methods* as shown in the *N-Tier Layering* in page 4.

Note 2: You don't always have to access the *Web API Methods* (from any client) generated by AspCoreGen 3.0 MVC, you can also access the *Business Object Classes* directly if you want to. Again, please refer to *Note 1* above.

3.2.2 Data Layer

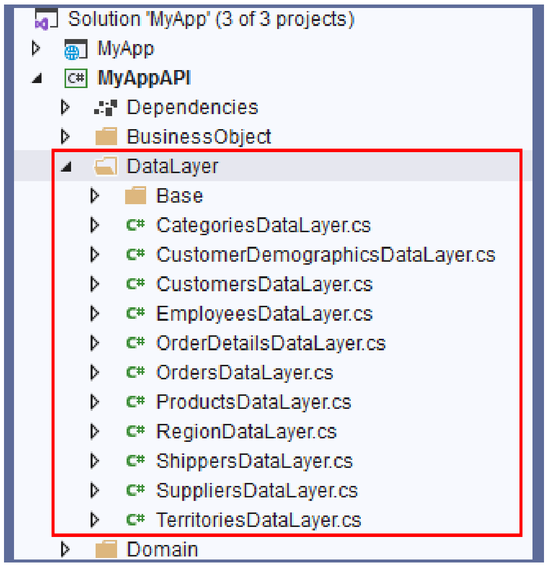
The *Data Tier*'s (or *Data Layer*) main purpose is to interact with the database. It does all the CRUD* operations.

Note 1: The *Data Layer* is called by the *Middle Tier*, and once the CRUD operation is done it returns the control back to the *Middle Tier*.

Note 2: A *Data Layer Class* should only be called by their respective *Middle Layer Class*. *Data Layer Classes* have an *"internal"* access modifier to prevent clients outside of the *Business Object and Data Layer API Project* access.

Note 3: Since *Data Layer Classes* have an *"internal"* access modifier, any (*Class, Method*) code you create in the *Business Object and Data Layer API Project* will be able to access these *Classes*. Again, no Class should access a *Data Tier Class* other than a *Middle Tier Class*, please see *Note 1*.

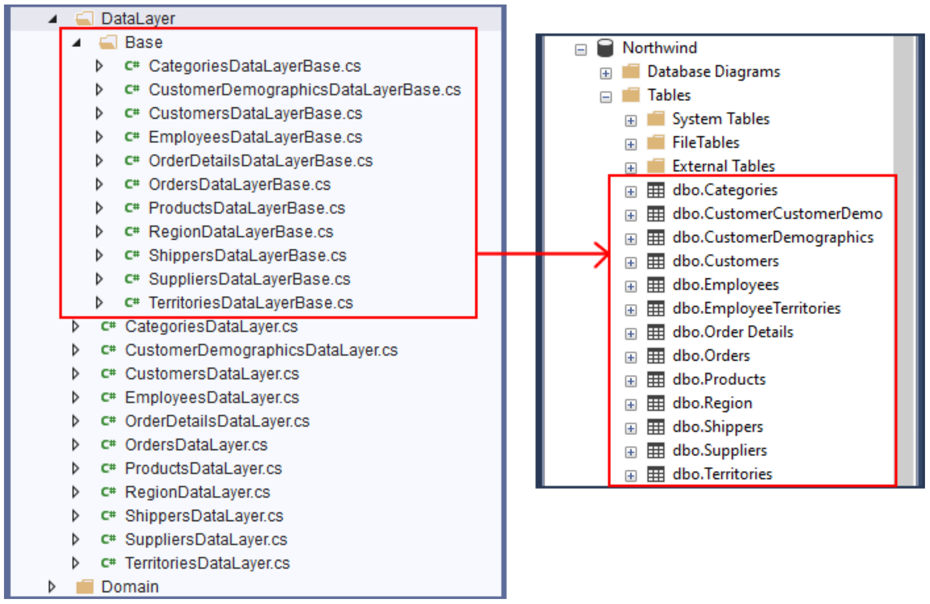
The *Data Tier* (*Data Layer*) *Class Files* are located in the *DataLayer Folder*.



3.2.2.1 *Parent (Base) Class*

These are the class files generated in the *Base* folder. The naming convention used is: *TableName**DataLayerBase**.cs*. **Do not add any code in these *Class* files.**

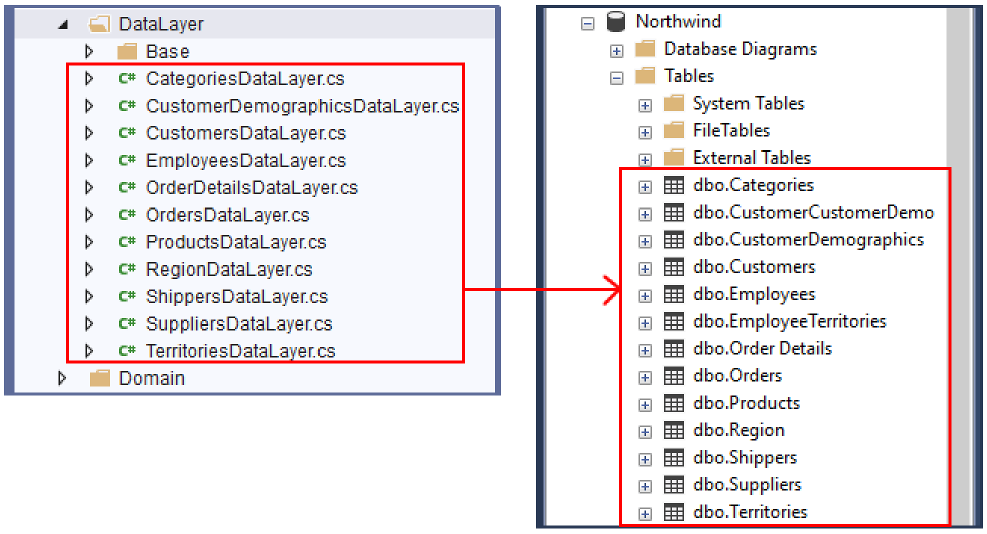
One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.



Base (Parent) Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

3.2.2.2 *Child Class – The Data Layer*

These are the *Class* files generated directly under the *DataLayer* folder (not including everything inside the *Base* folder). The naming convention used is: *TableName**DataLayer**.cs*. One *Data Layer Class* is generated per *Database Table*.



Child Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

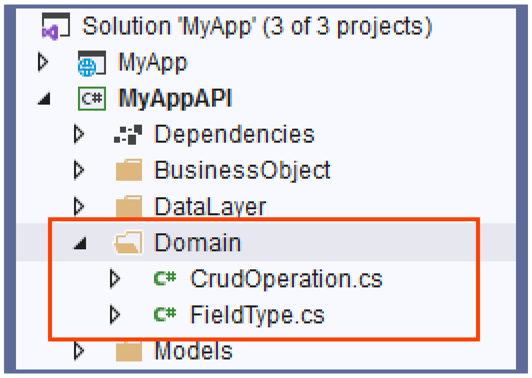
You can add code in these *Class* files. You access all the *Data Layer* methods and properties using this Class.

These are the *Classes* that *Middle Tier Classes* should access.

Note 1: Only a *Middle Tier Class* should access their respective *Data Layer Class*.

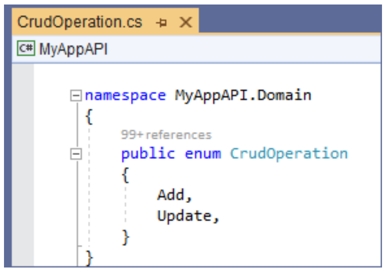
3.2.3 Domain

The *Domain Folder* contains 2 reusable *enum* type objects; the *CrudOperation.cs* and *FieldType.cs*.



3.2.3.1 CrudOperation.cs

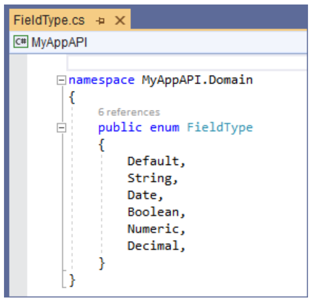
The *CrudOperation enum* is used to determine whether an *Add* or *Update* operation needs to be handled.



```
namespace MyAppAPI.Domain
{
    99+ references
    public enum CrudOperation
    {
        Add,
        Update,
    }
}
```

3.2.3.2 FieldType.cs

The *FieldType enum* is used to determine a field's type before executing an operation.



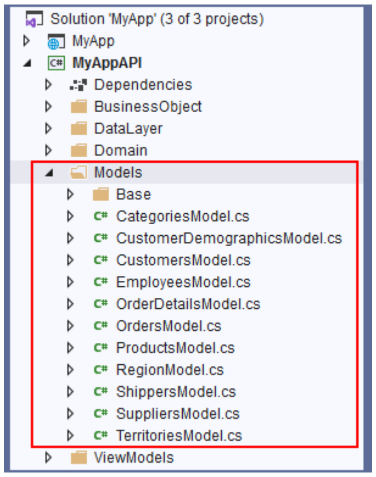
```
namespace MyAppAPI.Domain
{
    6 references
    public enum FieldType
    {
        Default,
        String,
        Date,
        Boolean,
        Numeric,
        Decimal,
    }
}
```

3.2.4 Models

These are *Classes* that contains *Properties* for each of the *Database Table* you generated code for. A *Property* is equivalent to a *Field* or *Column* in the respective *Database Table.* *Models* is the *"M"* in MVC. Sometimes *Models* are misinterpreted as the *Data Layer* part of MVC, in this case, it is not.

So why are *Models* generated in the *Business Object and Data Layer API Project* instead of the *Web Application Project* where the MVC *Views* and *Controllers* are generated in (after all it's called Models, Views, Controllers, hence MVC)? Simple, ***Models* are reusable**.

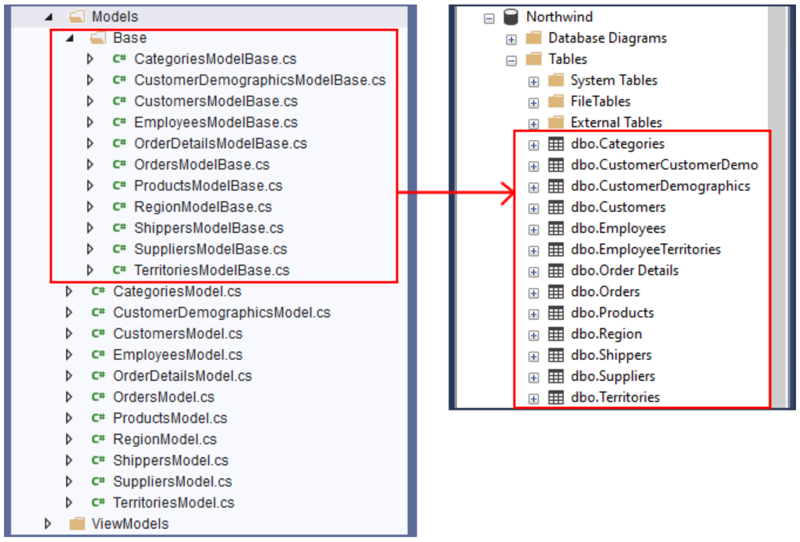
The *Models* are located in the *Models Folder*.



3.2.4.1 Parent (Base) Class

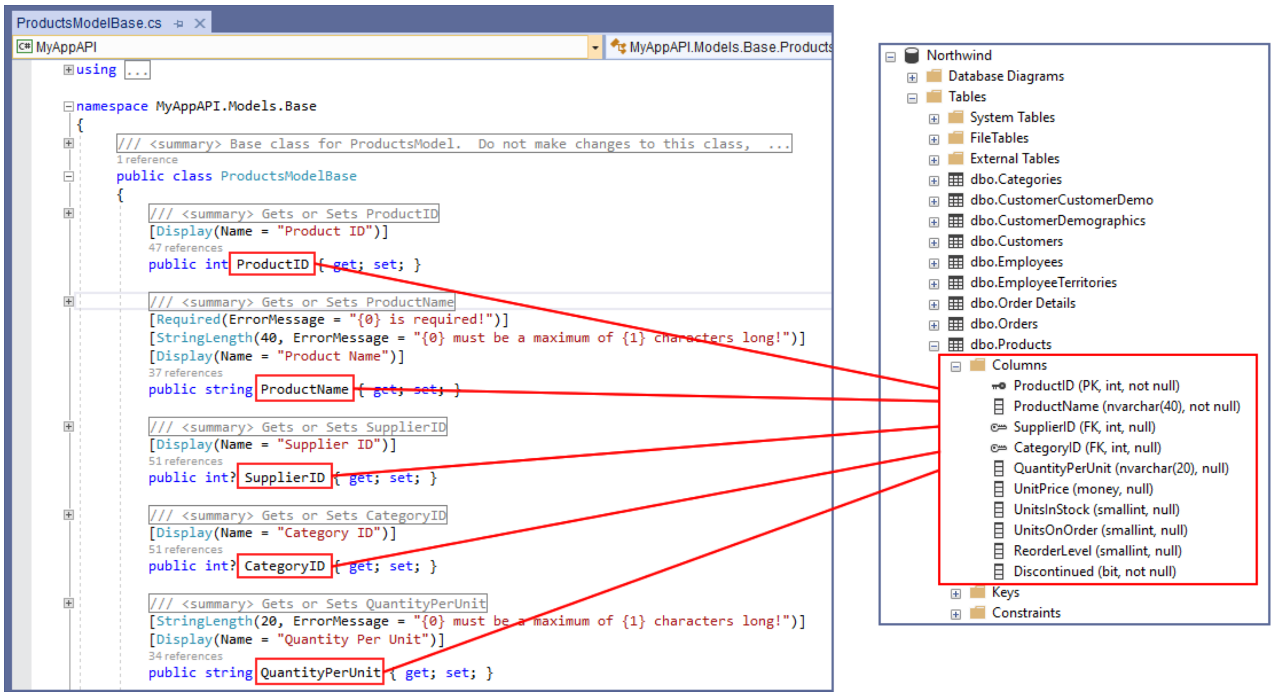
These are the class files generated in the *Base* folder. The naming convention used is: *TableName**ModelBase**.cs*. **Do not add any code in these *Class* files.**

One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.



Base (Parent) Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

Here's an example of the *Products Table Columns* (*Fields*) in the *Northwind* database in relation to the generated *Model*.



```
ProductsModelBase.cs
MyAppAPI                                   MyAppAPI.Models.Base.Products
using ...

namespace MyAppAPI.Models.Base
{
    /// <summary> Base class for ProductsModel.  Do not make changes to this class,  ...
    1 reference
    public class ProductsModelBase
    {
        /// <summary> Gets or Sets ProductID
        [Display(Name = "Product ID")]
        47 references
        public int ProductID { get; set; }

        /// <summary> Gets or Sets ProductName
        [Required(ErrorMessage = "{0} is required!")]
        [StringLength(40, ErrorMessage = "{0} must be a maximum of {1} characters long!")]
        [Display(Name = "Product Name")]
        37 references
        public string ProductName { get; set; }

        /// <summary> Gets or Sets SupplierID
        [Display(Name = "Supplier ID")]
        51 references
        public int? SupplierID { get; set; }

        /// <summary> Gets or Sets CategoryID
        [Display(Name = "Category ID")]
        51 references
        public int? CategoryID { get; set; }

        /// <summary> Gets or Sets QuantityPerUnit
        [StringLength(20, ErrorMessage = "{0} must be a maximum of {1} characters long!")]
        [Display(Name = "Quantity Per Unit")]
        34 references
        public string QuantityPerUnit { get; set; }
```

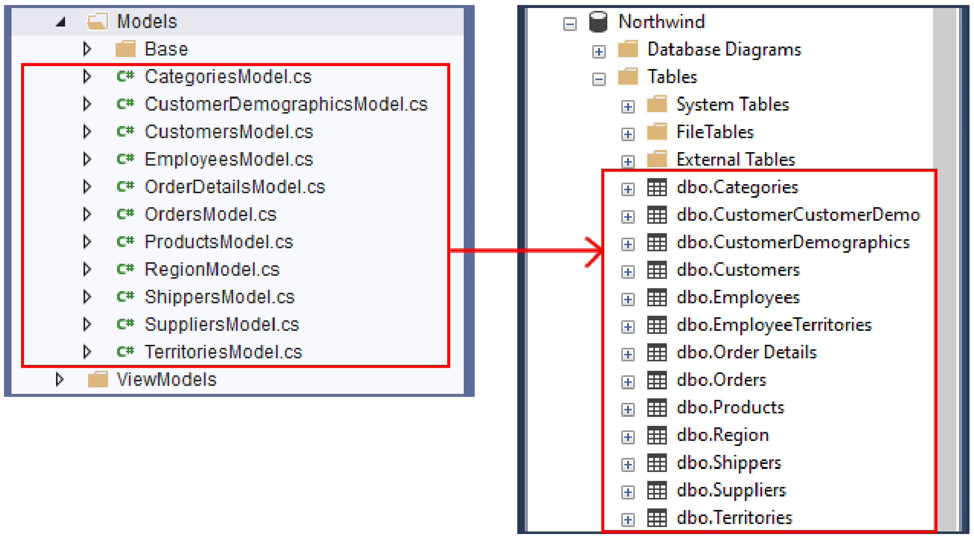
```
Northwind
  Database Diagrams
  Tables
    System Tables
    FileTables
    External Tables
    dbo.Categories
    dbo.CustomerCustomerDemo
    dbo.CustomerDemographics
    dbo.Customers
    dbo.Employees
    dbo.EmployeeTerritories
    dbo.Order Details
    dbo.Orders
    dbo.Products
      Columns
        ProductID (PK, int, not null)
        ProductName (nvarchar(40), not null)
        SupplierID (FK, int, null)
        CategoryID (FK, int, null)
        QuantityPerUnit (nvarchar(20), null)
        UnitPrice (money, null)
        UnitsInStock (smallint, null)
        UnitsOnOrder (smallint, null)
        ReorderLevel (smallint, null)
        Discontinued (bit, not null)
      Keys
      Constraints
```

Model (Base) Class in Visual Studio (Left) – Products Database Table Columns in MS SQL Server (Right)

3.2.4.2 *Child Class – The Model*

These are the *Class* files generated directly under the *Models* folder (not including everything inside the *Base* folder). The naming convention used is: *TableName**Model**.cs*. One *Model Class* is generated per *Database Table*.

You can add code in these *Class* files.



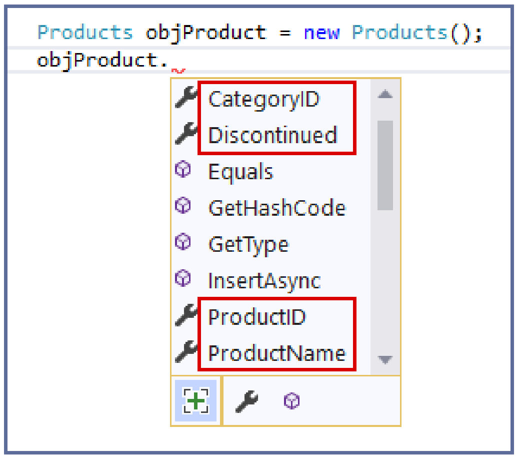
```
Models
  Base
  CategoriesModel.cs
  CustomerDemographicsModel.cs
  CustomersModel.cs
  EmployeesModel.cs
  OrderDetailsModel.cs
  OrdersModel.cs
  ProductsModel.cs
  RegionModel.cs
  ShippersModel.cs
  SuppliersModel.cs
  TerritoriesModel.cs
ViewModels
```

```
Northwind
  Database Diagrams
  Tables
    System Tables
    FileTables
    External Tables
    dbo.Categories
    dbo.CustomerCustomerDemo
    dbo.CustomerDemographics
    dbo.Customers
    dbo.Employees
    dbo.EmployeeTerritories
    dbo.Order Details
    dbo.Orders
    dbo.Products
    dbo.Region
    dbo.Shippers
    dbo.Suppliers
    dbo.Territories
```

Child Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

Models are inherited by the respective *Business Object Class*. So when you instantiate a *Business Object*, you will have access to the *Models* (*Properties*) for that *Business Object*.

For example, when you instantiate a *Products Business Object*, you will also be able to access the inherited *Models* and of course all the other objects in the *Products Business Object*. See below.



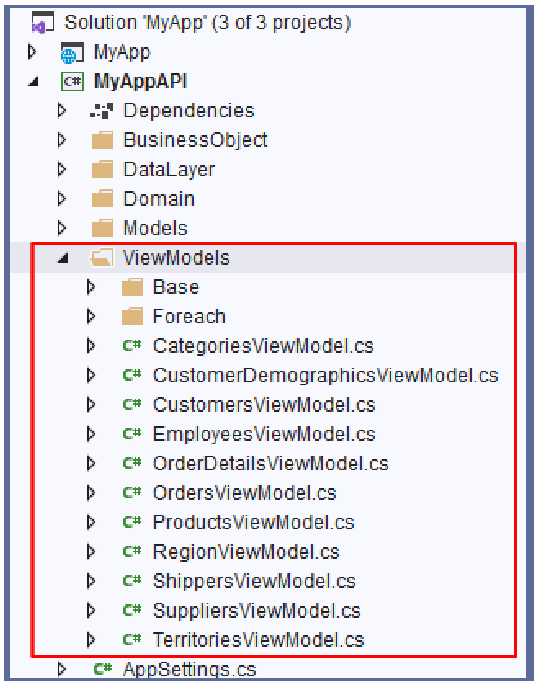
```
Products objProduct = new Products();
objProduct.
```

3.2.5 View Models

These are *Classes* that contains models (properties) used by MVC *Views*, hence the name *ViewModels*.

So why are *ViewModels* generated in the *Business Object and Data Layer API Project* instead of the *Web Application Project* where the MVC *Views* and *Controllers* are generated in? Simple, ***ViewModels* are reusable**.

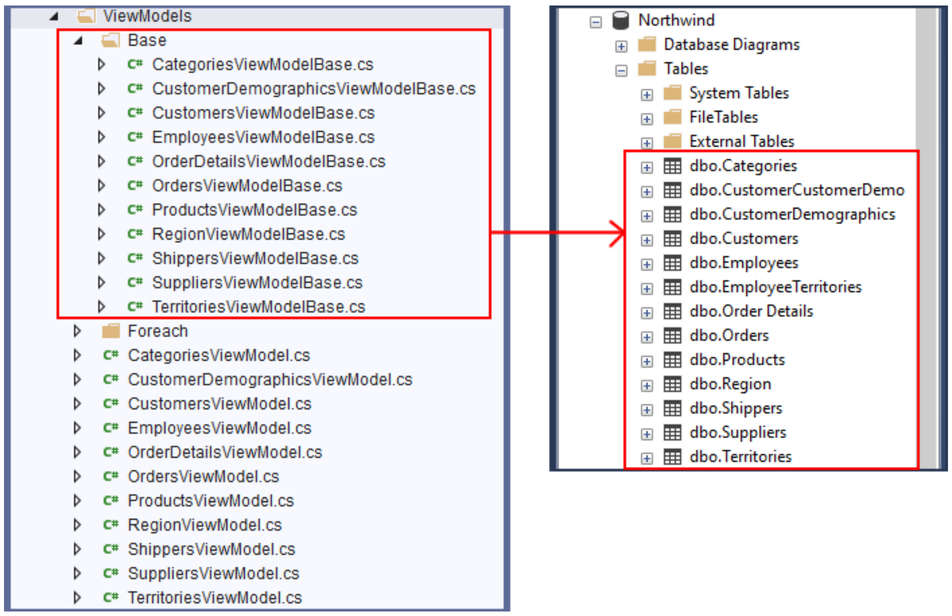
The *ViewModels* are located in the *ViewModels Folder*.



3.2.5.1 Parent (Base) Class

These are the class files generated in the *Base* folder. The naming convention used is: *TableName****ViewModelBase****.cs*. **Do not add any code in these *Class* files.**

One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.

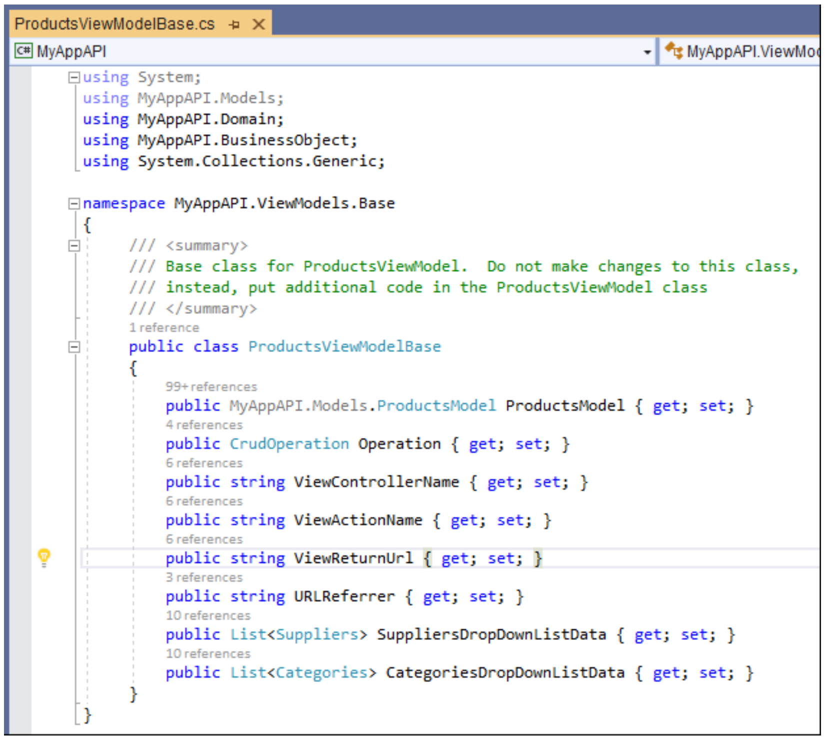


ViewModels
- Base
  - CategoriesViewModelBase.cs
  - CustomerDemographicsViewModelBase.cs
  - CustomersViewModelBase.cs
  - EmployeesViewModelBase.cs
  - OrderDetailsViewModelBase.cs
  - OrdersViewModelBase.cs
  - ProductsViewModelBase.cs
  - RegionViewModelBase.cs
  - ShippersViewModelBase.cs
  - SuppliersViewModelBase.cs
  - TerritoriesViewModelBase.cs
- Foreach
- CategoriesViewModel.cs
- CustomerDemographicsViewModel.cs
- CustomersViewModel.cs
- EmployeesViewModel.cs
- OrderDetailsViewModel.cs
- OrdersViewModel.cs
- ProductsViewModel.cs
- RegionViewModel.cs
- ShippersViewModel.cs
- SuppliersViewModel.cs
- TerritoriesViewModel.cs

Northwind
- Database Diagrams
- Tables
  - System Tables
  - FileTables
  - External Tables
  - dbo.Categories
  - dbo.CustomerCustomerDemo
  - dbo.CustomerDemographics
  - dbo.Customers
  - dbo.Employees
  - dbo.EmployeeTerritories
  - dbo.Order Details
  - dbo.Orders
  - dbo.Products
  - dbo.Region
  - dbo.Shippers
  - dbo.Suppliers
  - dbo.Territories

Base (Parent) Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

Here's an example of the *ProductsViewModelBase* code.

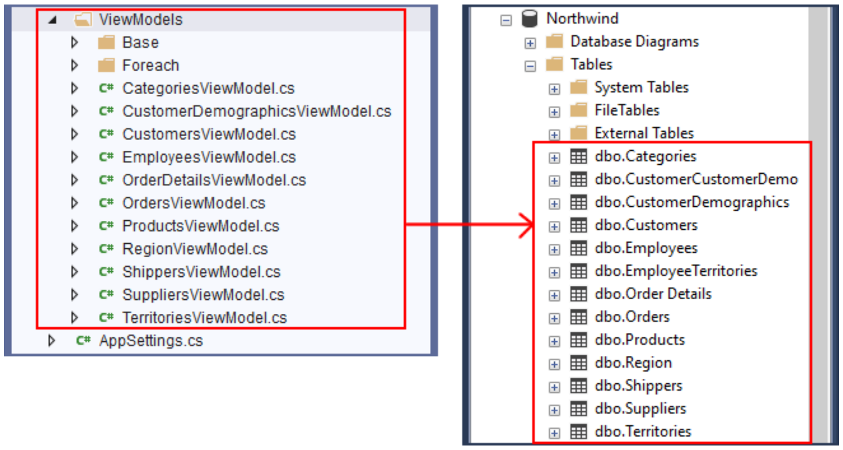


```
using System;
using MyAppAPI.Models;
using MyAppAPI.Domain;
using MyAppAPI.BusinessObject;
using System.Collections.Generic;

namespace MyAppAPI.ViewModels.Base
{
    /// <summary>
    /// Base class for ProductsViewModel.  Do not make changes to this class,
    /// instead, put additional code in the ProductsViewModel class
    /// </summary>
    1 reference
    public class ProductsViewModelBase
    {
        99+ references
        public MyAppAPI.Models.ProductsModel ProductsModel { get; set; }
        4 references
        public CrudOperation Operation { get; set; }
        6 references
        public string ViewControllerName { get; set; }
        6 references
        public string ViewActionName { get; set; }
        6 references
        public string ViewReturnUrl { get; set; }
        3 references
        public string URLReferrer { get; set; }
        10 references
        public List<Suppliers> SuppliersDropDownListData { get; set; }
        10 references
        public List<Categories> CategoriesDropDownListData { get; set; }
    }
}
```

3.2.5.2 *Child Class – The Model*

These are the *Class* files generated directly under the *ViewModels* folder (not including everything inside the *Base* and *Foreach* folders). The naming convention used is: *TableName**ViewModel**.cs*. One *ViewModel Class* is generated per *Database Table*.

You can add code in these *Class* files.



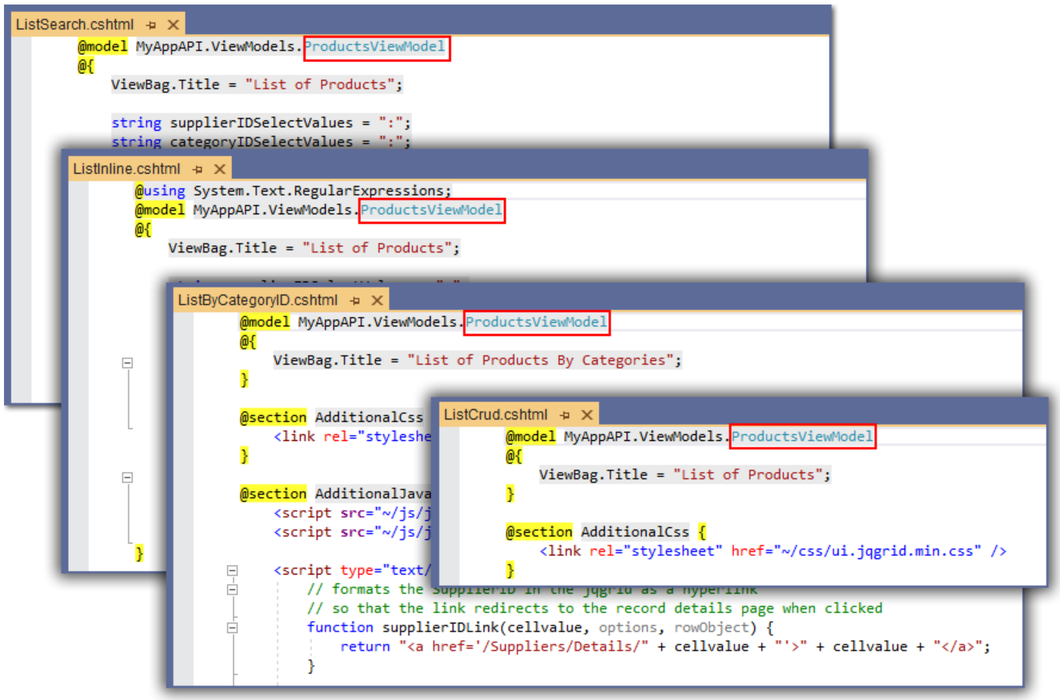
```
◢ ViewModels
    ▷ Base
    ▷ Foreach
    ▷ C# CategoriesViewModel.cs
    ▷ C# CustomerDemographicsViewModel.cs
    ▷ C# CustomersViewModel.cs
    ▷ C# EmployeesViewModel.cs
    ▷ C# OrderDetailsViewModel.cs
    ▷ C# OrdersViewModel.cs
    ▷ C# ProductsViewModel.cs
    ▷ C# RegionViewModel.cs
    ▷ C# ShippersViewModel.cs
    ▷ C# SuppliersViewModel.cs
    ▷ C# TerritoriesViewModel.cs
▷ C# AppSettings.cs
```

```
⊟ Northwind
    ⊞ Database Diagrams
    ⊟ Tables
        ⊞ System Tables
        ⊞ FileTables
        ⊞ External Tables
        ⊞ dbo.Categories
        ⊞ dbo.CustomerCustomerDemo
        ⊞ dbo.CustomerDemographics
        ⊞ dbo.Customers
        ⊞ dbo.Employees
        ⊞ dbo.EmployeeTerritories
        ⊞ dbo.Order Details
        ⊞ dbo.Orders
        ⊞ dbo.Products
        ⊞ dbo.Region
        ⊞ dbo.Shippers
        ⊞ dbo.Suppliers
        ⊞ dbo.Territories
```

Child Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

These *ViewModels (ProductsViewModel* in the example*)* are referenced and used by the following MVC *Views*:

1. *ListSearch.cshtml*
2. *ListInline.cshtml*
3. *ListCrud.cshtml*
4. *ListBy**ForeignKey**.cshtml*



ListSearch.cshtml

```
@model MyAppAPI.ViewModels.ProductsViewModel
@{
    ViewBag.Title = "List of Products";

    string supplierIDSelectValues = ":";
    string categoryIDSelectValues = ":";
```

ListInline.cshtml

```
@using System.Text.RegularExpressions;
@model MyAppAPI.ViewModels.ProductsViewModel
@{
    ViewBag.Title = "List of Products";
```

ListByCategoryID.cshtml

```
@model MyAppAPI.ViewModels.ProductsViewModel
@{
    ViewBag.Title = "List of Products By Categories";
}

@section AdditionalCss
    <link rel="styleshe
}

@section AdditionalJava
    <script src="~/js/j
    <script src="~/js/j

    <script type="text/
        // formats the
        // so that the link redirects to the record details page when clicked
        function supplierIDLink(cellvalue, options, rowObject) {
            return "<a href='/Suppliers/Details/" + cellvalue + "'>" + cellvalue + "</a>";
        }
```

ListCrud.cshtml

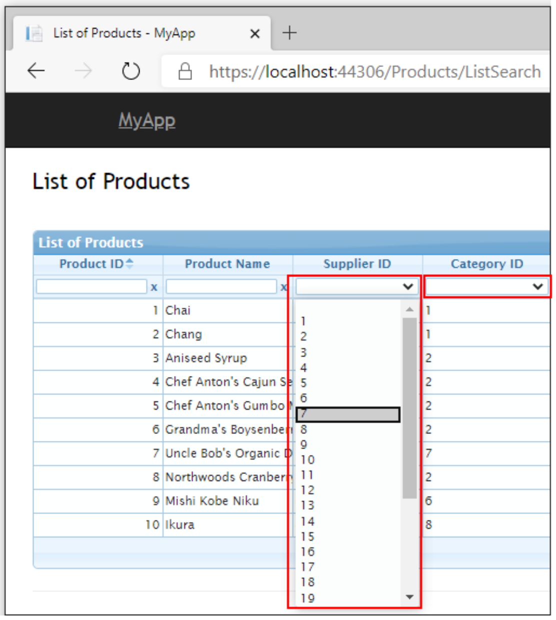
```
@model MyAppAPI.ViewModels.ProductsViewModel
@{
    ViewBag.Title = "List of Products";
}

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

Note: By default ASP.NET MVC *Views* use "***Model***" as a *Keyword*. Also by default, you can set the MVC *View*'s *Model* following the *@model* directive. Here's an example on how to set an MVC *View*'s *Model:*

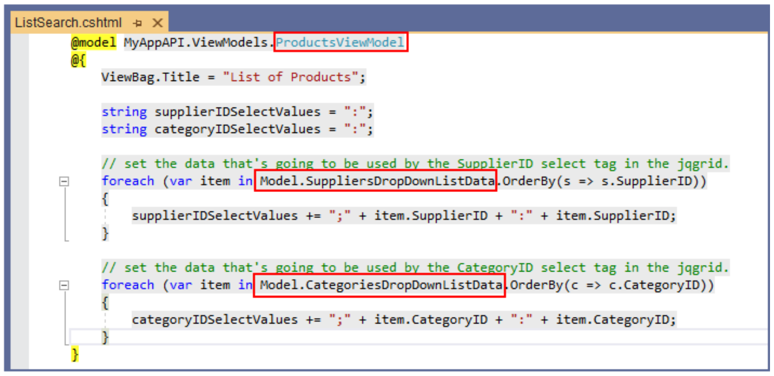
```
@model MyAppAPI.ViewModels.ProductsViewModel
```

3.2.5.2.1 ListSearch.cshtml

This MVC *View* uses the *ProductsViewModel* as its *Model*. It uses the MVC *View*'s *Model* (*ProductsViewModel*) to fill the *Select* tags for the *SupplierID* and *CategoryID* using the MVC *View's Model*, the *SuppliersDropDownListData* and *CategoriesDropDownListData* respectively, these are *Properties* of the *ProductsViewModel* as shown in the *ProductsViewModelBase* code example in page 30.

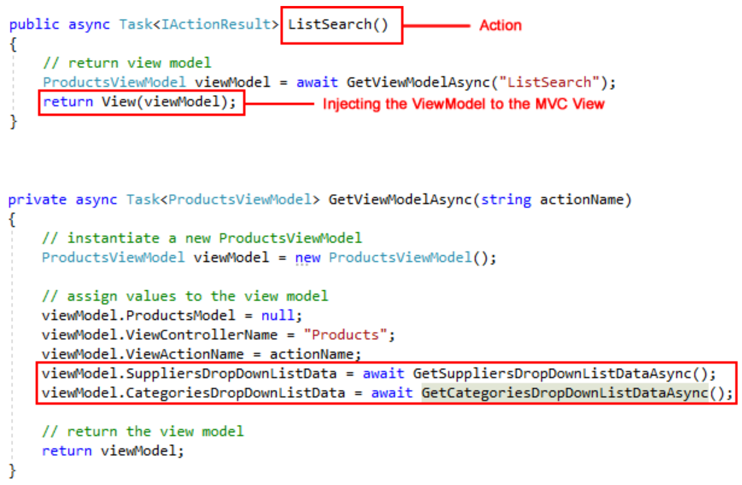


Product ID	Product Name	Supplier ID	Category ID
1	Chai		1
2	Chang		1
3	Aniseed Syrup		2
4	Chef Anton's Cajun Se		2
5	Chef Anton's Gumbo		2
6	Grandma's Boysenber		2
7	Uncle Bob's Organic D		7
8	Northwoods Cranberr		2
9	Mishi Kobe Niku		6
10	Ikura		8



```
@model MyAppAPI.ViewModels.ProductsViewModel
@{
    ViewBag.Title = "List of Products";

    string supplierIDSelectValues = ":";
    string categoryIDSelectValues = ":";

    // set the data that's going to be used by the SupplierID select tag in the jqgrid.
    foreach (var item in Model.SuppliersDropDownListData.OrderBy(s => s.SupplierID))
    {
        supplierIDSelectValues += ";" + item.SupplierID + ":" + item.SupplierID;
    }

    // set the data that's going to be used by the CategoryID select tag in the jqgrid.
    foreach (var item in Model.CategoriesDropDownListData.OrderBy(c => c.CategoryID))
    {
        categoryIDSelectValues += ";" + item.CategoryID + ":" + item.CategoryID;
    }
}
```

The *ViewModel* used by the *ListSearch.cshtml View* is assigned from the respective *Get Action* method found in the *Controller (Base)*, it is then injected to the *ListSearch.cshtml View.* See code example below.

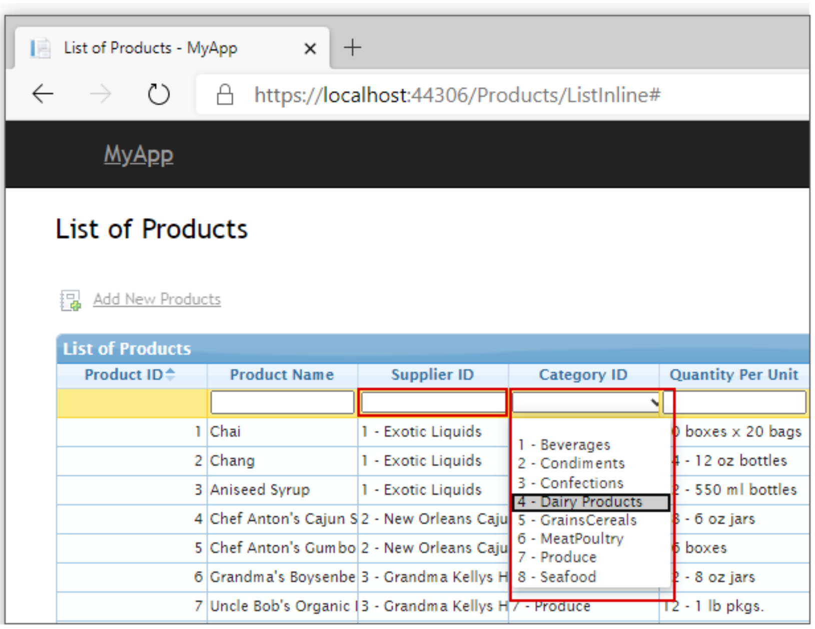


```
public async Task<IActionResult> ListSearch()
{
    // return view model
    ProductsViewModel viewModel = await GetViewModelAsync("ListSearch");
    return View(viewModel);
}


private async Task<ProductsViewModel> GetViewModelAsync(string actionName)
{
    // instantiate a new ProductsViewModel
    ProductsViewModel viewModel = new ProductsViewModel();

    // assign values to the view model
    viewModel.ProductsModel = null;
    viewModel.ViewControllerName = "Products";
    viewModel.ViewActionName = actionName;
    viewModel.SuppliersDropDownListData = await GetSuppliersDropDownListDataAsync();
    viewModel.CategoriesDropDownListData = await GetCategoriesDropDownListDataAsync();

    // return the view model
    return viewModel;
}
```

3.2.5.2.2 ListInline.cshtml

This MVC *View* uses the *ProductsViewModel* as its *Model*.

The *ListInline.cshtml* uses the MVC *View's Model* (*ProductsViewModel*) to fill the *Select* tags for the *SupplierID* and *CategoryID* in the dialog shown below using the MVC *View's Model*, the *SuppliersDropDownListData* and *CategoriesDropDownListData* respectively, these are *Properties* of the *ProductsViewModel* as shown in the *ProductsViewModelBase* code example in page 30.

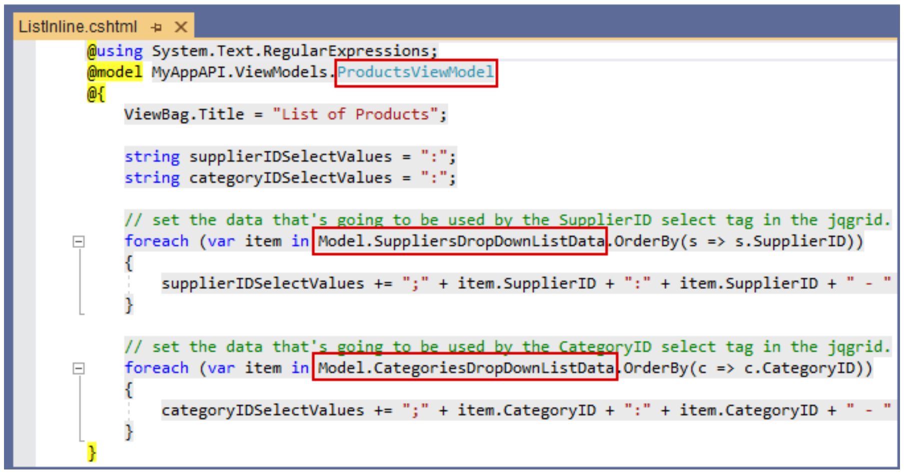


List of Products

List of Products

Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit
1	Chai	1 - Exotic Liquids		0 boxes x 20 bags
2	Chang	1 - Exotic Liquids		4 - 12 oz bottles
3	Aniseed Syrup	1 - Exotic Liquids		2 - 550 ml bottles
4	Chef Anton's Cajun S	2 - New Orleans Caju		8 - 6 oz jars
5	Chef Anton's Gumbo	2 - New Orleans Caju		6 boxes
6	Grandma's Boysenbe	3 - Grandma Kellys H		2 - 8 oz jars
7	Uncle Bob's Organic	3 - Grandma Kellys H	7 - Produce	12 - 1 lb pkgs.

```
1 - Beverages
2 - Condiments
3 - Confections
4 - Dairy Products
5 - GrainsCereals
6 - MeatPoultry
7 - Produce
8 - Seafood
```



```
@using System.Text.RegularExpressions;
@model MyAppAPI.ViewModels.ProductsViewModel
@{
    ViewBag.Title = "List of Products";

    string supplierIDSelectValues = ":";
    string categoryIDSelectValues = ":";

    // set the data that's going to be used by the SupplierID select tag in the jqgrid.
    foreach (var item in Model.SuppliersDropDownListData.OrderBy(s => s.SupplierID))
    {
        supplierIDSelectValues += ";" + item.SupplierID + ":" + item.SupplierID + " - "
    }

    // set the data that's going to be used by the CategoryID select tag in the jqgrid.
    foreach (var item in Model.CategoriesDropDownListData.OrderBy(c => c.CategoryID))
    {
        categoryIDSelectValues += ";" + item.CategoryID + ":" + item.CategoryID + " - "
    }
}
```

The *ViewModel* used by the *ListInline.cshtml View* is assigned from the respective *Get Action* method found in the *Controller (Base)*, it is then injected to the *ListInline.cshtml View.* See code example below.

```
public async Task<IActionResult> ListInline()
{
    // return view model
    ProductsViewModel viewModel = await GetViewModelAsync("ListInline");
    return View(viewModel);
}
```

Action

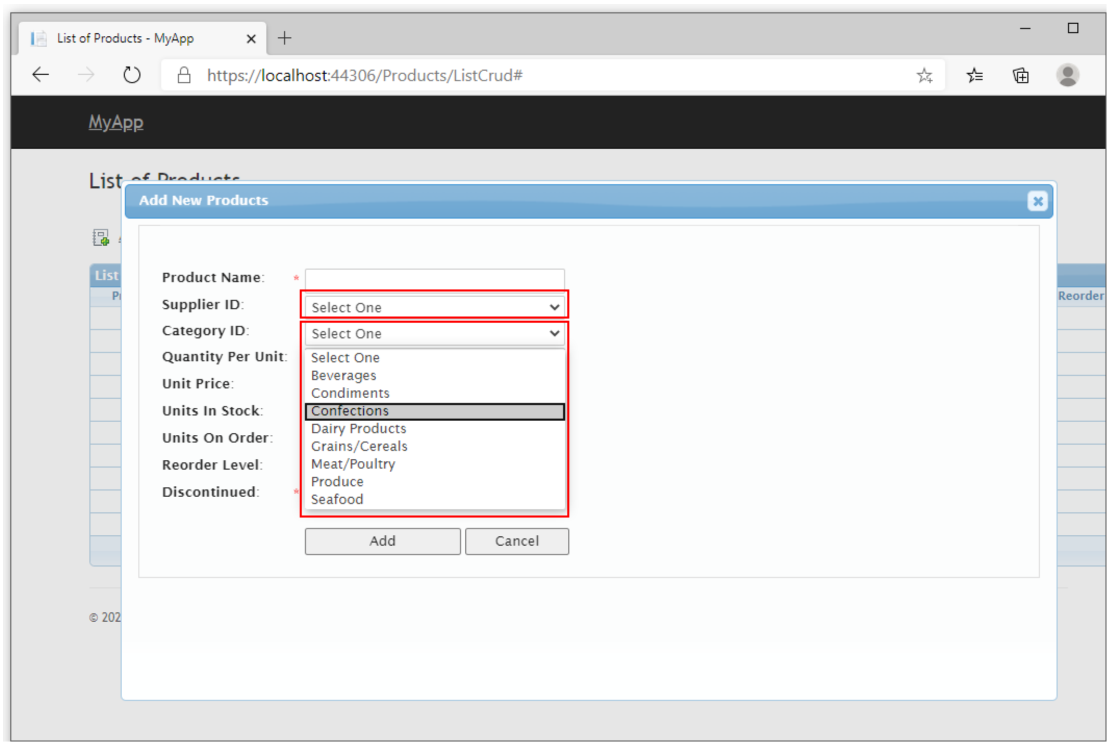
Injecting the ViewModel to the MVC View

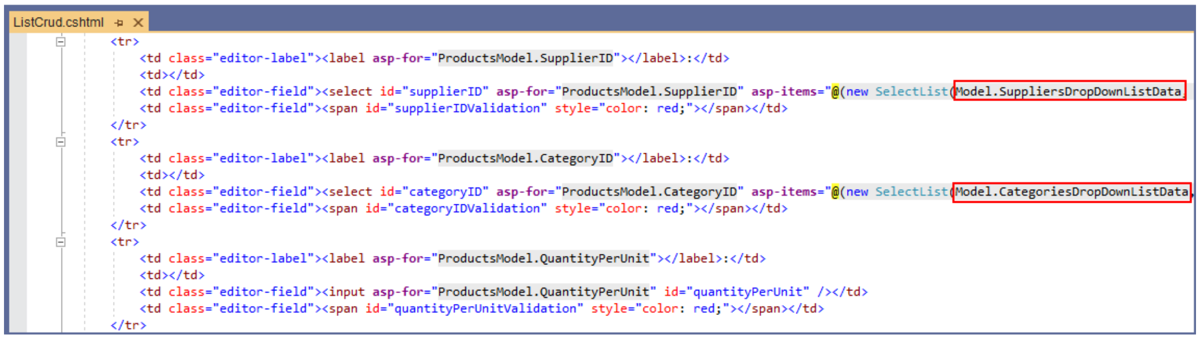
3.2.5.2.3 ListCrud.cshtml

This MVC *View* uses the *ProductsViewModel* as its *Model*.

In this MVC *View*, when you *Add a New Record* or *Update an Existing Record*, a dialog pops up.

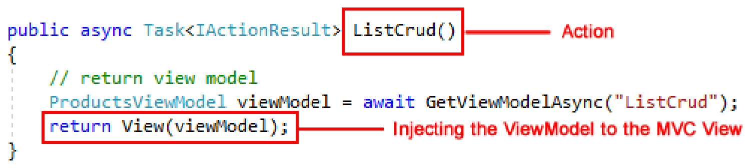
The *ListCrud.cshtml* uses the MVC *View*'s *Model* (*ProductsViewModel*) to fill the *Select* tags for the *SupplierID* and *CategoryID* in the dialog shown below using the MVC *View's Model*, the *SuppliersDropDownListData* and *CategoriesDropDownListData* respectively, these are *Properties* of the *ProductsViewModel* as shown in the *ProductsViewModelBase* code example in page 30.





```
<tr>
    <td class="editor-label"><label asp-for="ProductsModel.SupplierID"></label>:</td>
    <td></td>
    <td class="editor-field"><select id="supplierID" asp-for="ProductsModel.SupplierID" asp-items="@(new SelectList(Model.SuppliersDropDownListData,
    <td class="editor-field"><span id="supplierIDValidation" style="color: red;"></span></td>
</tr>
<tr>
    <td class="editor-label"><label asp-for="ProductsModel.CategoryID"></label>:</td>
    <td></td>
    <td class="editor-field"><select id="categoryID" asp-for="ProductsModel.CategoryID" asp-items="@(new SelectList(Model.CategoriesDropDownListData,
    <td class="editor-field"><span id="categoryIDValidation" style="color: red;"></span></td>
</tr>
<tr>
    <td class="editor-label"><label asp-for="ProductsModel.QuantityPerUnit"></label>:</td>
    <td></td>
    <td class="editor-field"><input asp-for="ProductsModel.QuantityPerUnit" id="quantityPerUnit" /></td>
    <td class="editor-field"><span id="quantityPerUnitValidation" style="color: red;"></span></td>
</tr>
```

The *ViewModel* used by the *ListInline.cshtml View* is assigned from the respective *Get Action* method found in the *Controller (Base)*, it is then injected to the *ListInline.cshtml View*. See code example below.

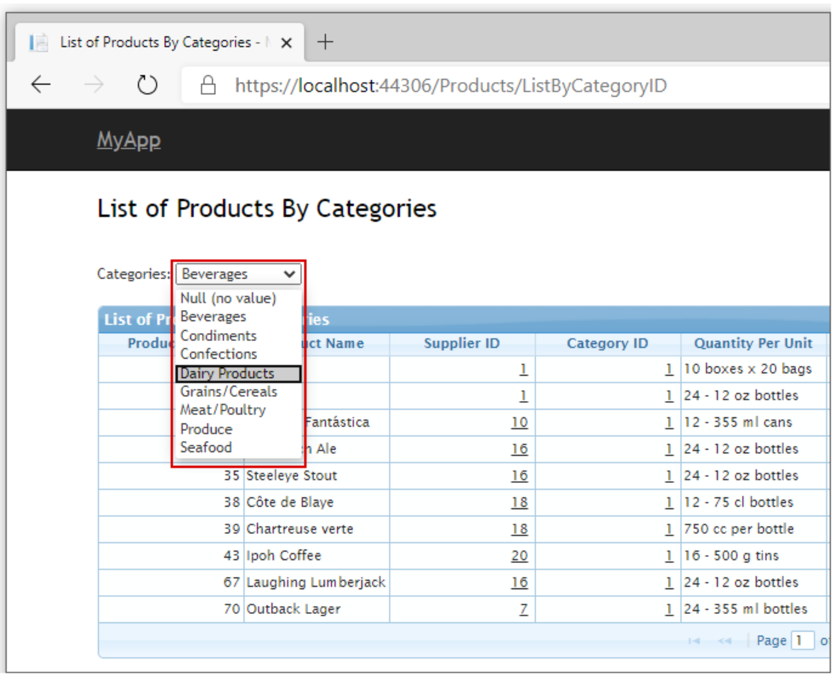


```
public async Task<IActionResult> ListCrud()
{
    // return view model
    ProductsViewModel viewModel = await GetViewModelAsync("ListCrud");
    return View(viewModel);
}
```

3.2.5.2.4 ListBy*ForeignKey*.cshtml

This MVC *View* uses the *ProductsViewModel* as its *Model*.

The *ListBy**ForeignKey**.cshtml* uses the MVC *View*'s *Model* (*ProductsViewModel*) to fill the *Select* tag for the *ForeignKey* (*CategoryID)* in the dialog shown below using the MVC *View's Model*, the *CategoriesDropDownListData*, this is one of the *Properties* of the *ProductsViewModel* as shown in the *ProductsViewModelBase* code example in page 30.



List of Products By Categories -
https://localhost:44306/Products/ListByCategoryID
MyApp

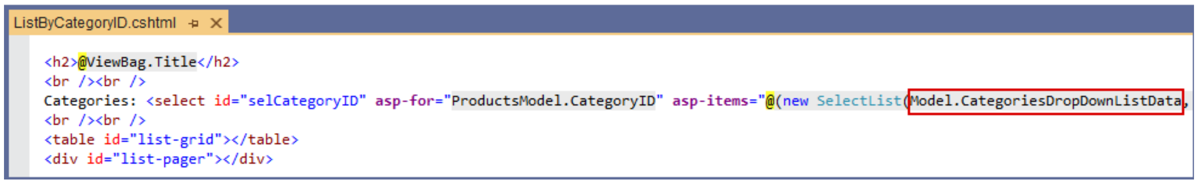
List of Products By Categories

Categories: Beverages

- Null (no value)
- Beverages
- Condiments
- Confections
- Dairy Products
- Grains/Cereals
- Meat/Poultry
- Produce
- Seafood

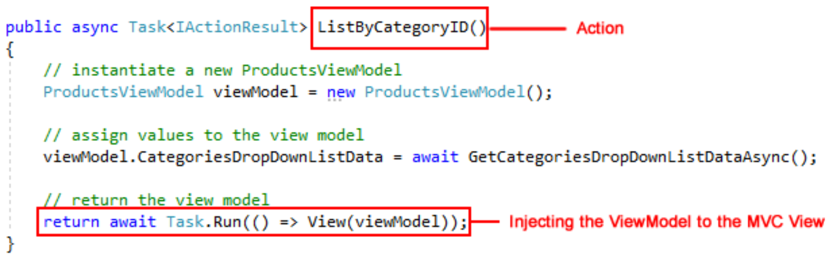
List of Pr ies

Produc	uct Name	Supplier ID	Category ID	Quantity Per Unit
		1	1	10 boxes x 20 bags
		1	1	24 - 12 oz bottles
	Fantástica	10	1	12 - 355 ml cans
	n Ale	16	1	24 - 12 oz bottles
35	Steeleye Stout	16	1	24 - 12 oz bottles
38	Côte de Blaye	18	1	12 - 75 cl bottles
39	Chartreuse verte	18	1	750 cc per bottle
43	Ipoh Coffee	20	1	16 - 500 g tins
67	Laughing Lumberjack	16	1	24 - 12 oz bottles
70	Outback Lager	7	1	24 - 355 ml bottles



```
<h2>@ViewBag.Title</h2>
<br /><br />
Categories: <select id="selCategoryID" asp-for="ProductsModel.CategoryID" asp-items="@(new SelectList(Model.CategoriesDropDownListData,
<br /><br />
<table id="list-grid"></table>
<div id="list-pager"></div>
```

The *ViewModel* used by the *ListBy**ForeignKey**.cshtml View* is assigned from the respective *Get Action* method found in the *Controller (Base)*, it is then injected to the *ListBy**ForeignKey**.cshtml View.* You will notice that code in the *Controller*'s *Action Method* only assigns one *ViewModel Property* compared to the *ListSearch.cshtml*, *ListInline.cshtml*, and *ListCrud.cshtml*, the *CategoriesDropDownListData*. Because it only needs data for one *Select Tag (Categories)* as seen in the image above.

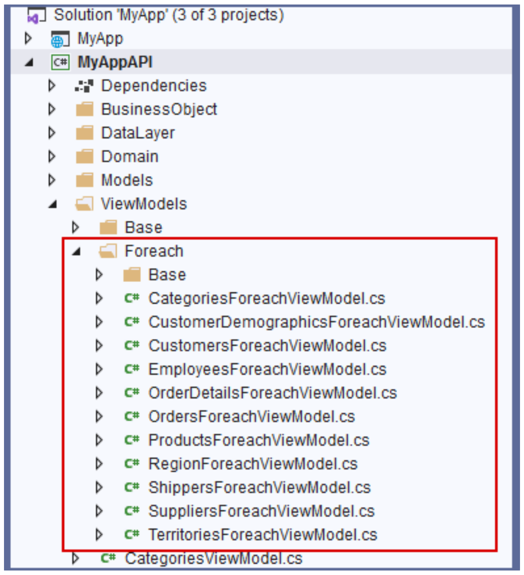


```
public async Task<IActionResult> ListByCategoryID()
{
    // instantiate a new ProductsViewModel
    ProductsViewModel viewModel = new ProductsViewModel();

    // assign values to the view model
    viewModel.CategoriesDropDownListData = await GetCategoriesDropDownListDataAsync();

    // return the view model
    return await Task.Run(() => View(viewModel));
}
```

3.2.5.3 *Foreach View Models*

These are *Classes* that contains models (properties) used by the *ListForeach.cshtml* MVC *Views*.

The *ForeachViewModels* are located in the *ViewModels/Foreach Folder*.

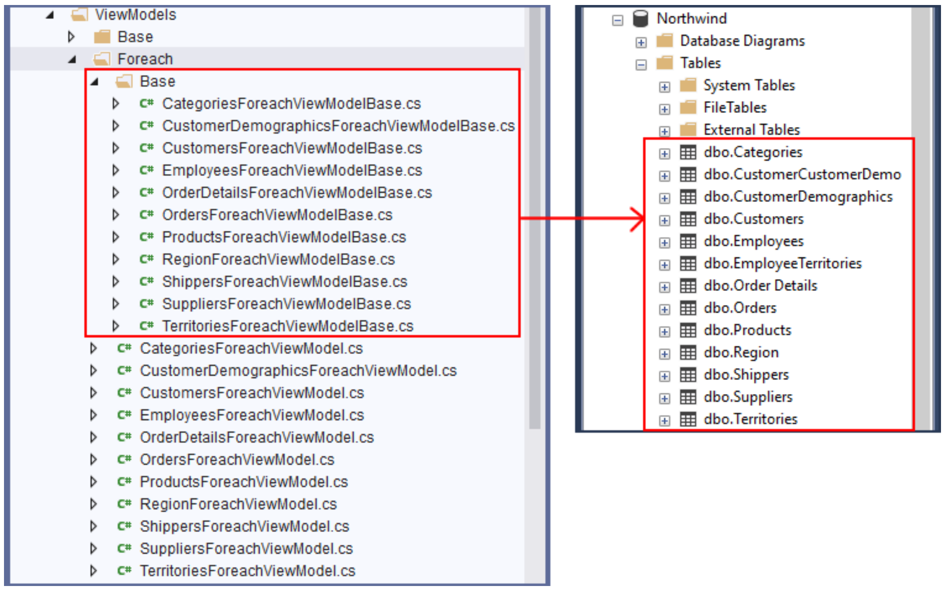


Solution 'MyApp' (3 of 3 projects)
MyApp
MyAppAPI
Dependencies
BusinessObject
DataLayer
Domain
Models
ViewModels
Base
Foreach
Base
CategoriesForeachViewModel.cs
CustomerDemographicsForeachViewModel.cs
CustomersForeachViewModel.cs
EmployeesForeachViewModel.cs
OrderDetailsForeachViewModel.cs
OrdersForeachViewModel.cs
ProductsForeachViewModel.cs
RegionForeachViewModel.cs
ShippersForeachViewModel.cs
SuppliersForeachViewModel.cs
TerritoriesForeachViewModel.cs
CategoriesViewModel.cs

3.2.5.4 *Parent (Base) Class*

These are the class files generated in the *Foreach/Base* folder. The naming convention used is: *TableName**ForeachViewModelBase**.cs*. **Do not add any code in these *Class* files.**

One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.

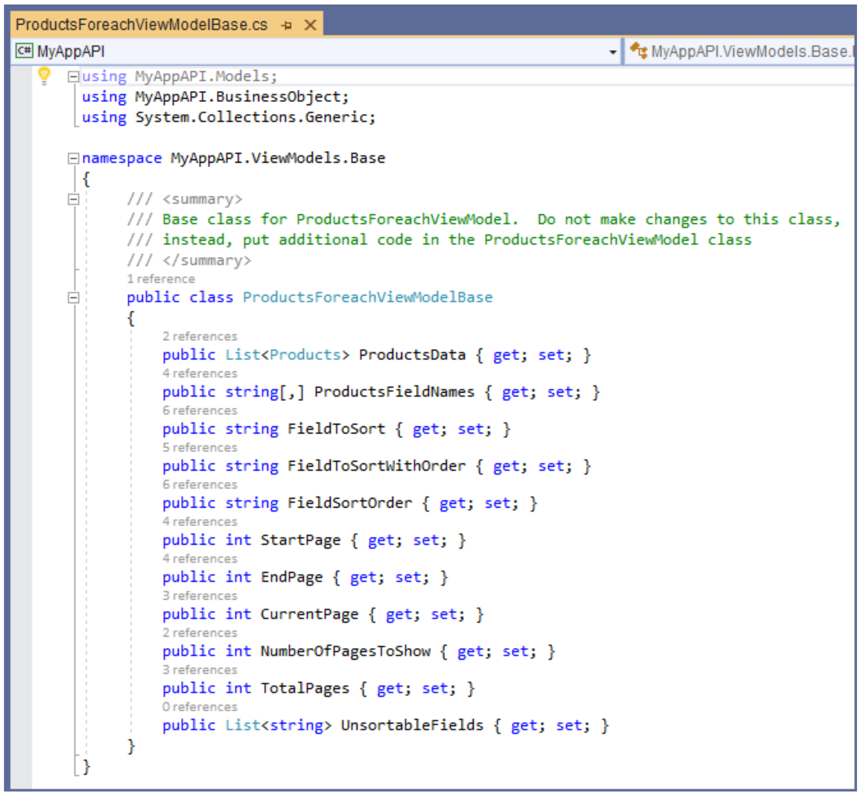


ViewModels
Base
Foreach
Base
CategoriesForeachViewModelBase.cs
CustomerDemographicsForeachViewModelBase.cs
CustomersForeachViewModelBase.cs
EmployeesForeachViewModelBase.cs
OrderDetailsForeachViewModelBase.cs
OrdersForeachViewModelBase.cs
ProductsForeachViewModelBase.cs
RegionForeachViewModelBase.cs
ShippersForeachViewModelBase.cs
SuppliersForeachViewModelBase.cs
TerritoriesForeachViewModelBase.cs
CategoriesForeachViewModel.cs
CustomerDemographicsForeachViewModel.cs
CustomersForeachViewModel.cs
EmployeesForeachViewModel.cs
OrderDetailsForeachViewModel.cs
OrdersForeachViewModel.cs
ProductsForeachViewModel.cs
RegionForeachViewModel.cs
ShippersForeachViewModel.cs
SuppliersForeachViewModel.cs
TerritoriesForeachViewModel.cs

Northwind
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Categories
dbo.CustomerCustomerDemo
dbo.CustomerDemographics
dbo.Customers
dbo.Employees
dbo.EmployeeTerritories
dbo.Order Details
dbo.Orders
dbo.Products
dbo.Region
dbo.Shippers
dbo.Suppliers
dbo.Territories

Base (Parent) Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

Here's an example of the *ProductsForeachViewModelBase* code.

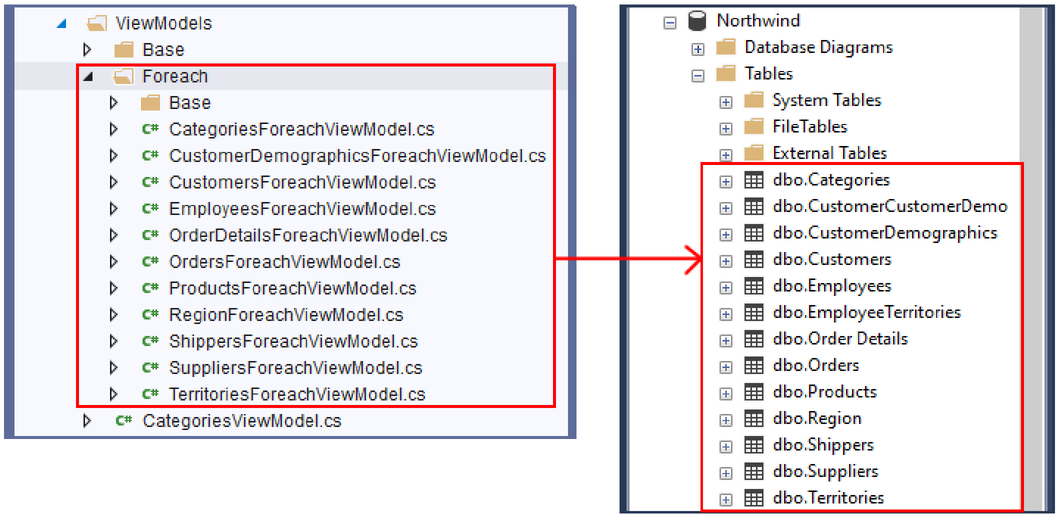


```
using MyAppAPI.Models;
using MyAppAPI.BusinessObject;
using System.Collections.Generic;

namespace MyAppAPI.ViewModels.Base
{
    /// <summary>
    /// Base class for ProductsForeachViewModel.  Do not make changes to this class,
    /// instead, put additional code in the ProductsForeachViewModel class
    /// </summary>
    1 reference
    public class ProductsForeachViewModelBase
    {
        2 references
        public List<Products> ProductsData { get; set; }
        4 references
        public string[,] ProductsFieldNames { get; set; }
        6 references
        public string FieldToSort { get; set; }
        5 references
        public string FieldToSortWithOrder { get; set; }
        6 references
        public string FieldSortOrder { get; set; }
        4 references
        public int StartPage { get; set; }
        4 references
        public int EndPage { get; set; }
        3 references
        public int CurrentPage { get; set; }
        2 references
        public int NumberOfPagesToShow { get; set; }
        3 references
        public int TotalPages { get; set; }
        0 references
        public List<string> UnsortableFields { get; set; }
    }
}
```

3.2.5.5 *Child Class – The Model*

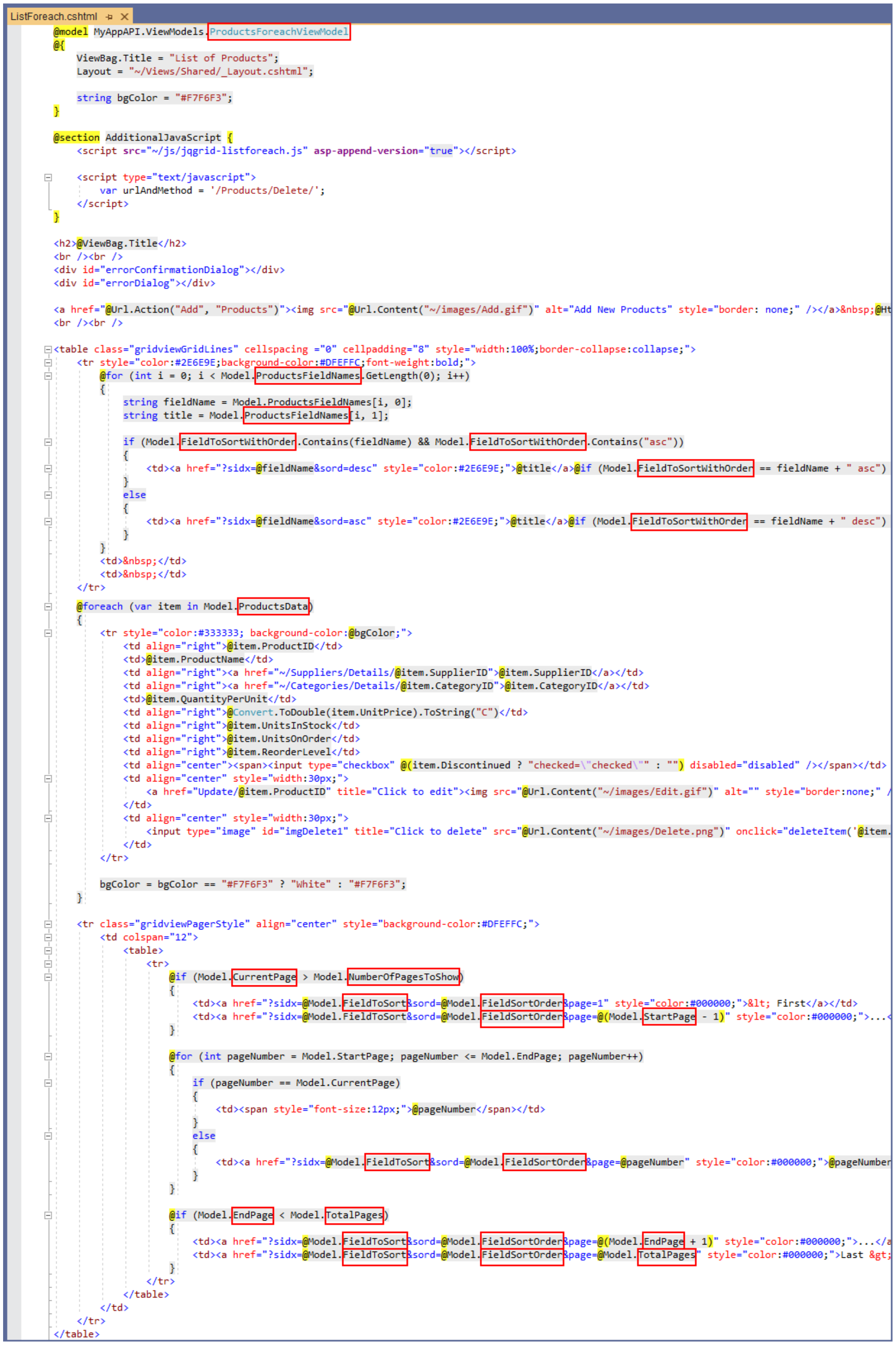
These are the *Class* files generated directly under the *ViewModels/Foreach* folder (not including everything inside the *Foreach/Base* folder). The naming convention used is: *TableName**ForeachViewModel**.cs*. One *ForeachViewModel Class* is generated per *Database Table*.

You can add code in these *Class* files.


ViewModels
Base
Foreach
Base
CategoriesForeachViewModel.cs
CustomerDemographicsForeachViewModel.cs
CustomersForeachViewModel.cs
EmployeesForeachViewModel.cs
OrderDetailsForeachViewModel.cs
OrdersForeachViewModel.cs
ProductsForeachViewModel.cs
RegionForeachViewModel.cs
ShippersForeachViewModel.cs
SuppliersForeachViewModel.cs
TerritoriesForeachViewModel.cs
CategoriesViewModel.cs
Northwind
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Categories
dbo.CustomerCustomerDemo
dbo.CustomerDemographics
dbo.Customers
dbo.Employees
dbo.EmployeeTerritories
dbo.Order Details
dbo.Orders
dbo.Products
dbo.Region
dbo.Shippers
dbo.Suppliers
dbo.Territories

Child Classes in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

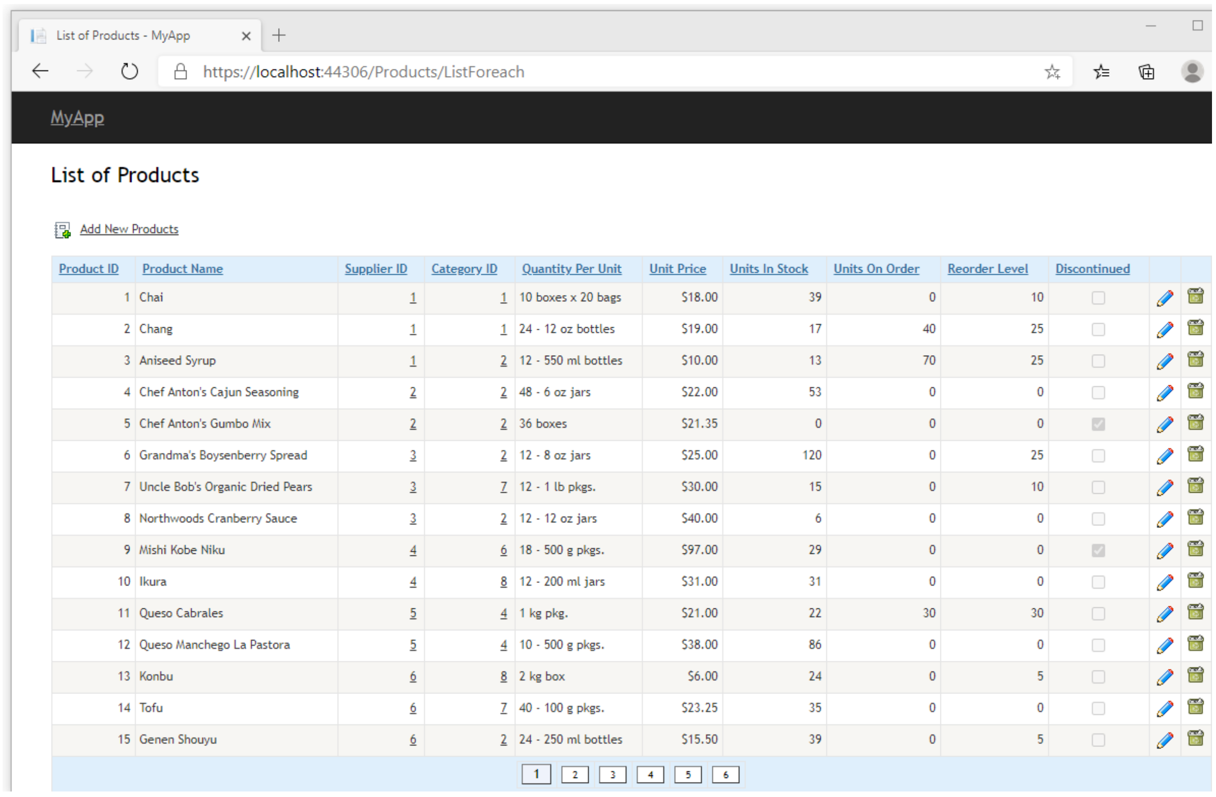
Here's an example on how the *ListForeach.cshtml* MVC *View* uses the MVC *View*'s *Model* (*ProductsForeachViewModel*) to manually build the data grid. The snapshot below shows the *ProductsForeachViewModel's Properties* referenced in the *ListForeach.cshtml* MVC *View.*



ListForeach.cshtml

```
@model MyAppAPI.ViewModels.ProductsForeachViewModel
@{
    ViewBag.Title = "List of Products";
    Layout = "~/Views/Shared/_Layout.cshtml";

    string bgColor = "#F7F6F3";
}

@section AdditionalJavaScript {
    <script src="~/js/jqgrid-listforeach.js" asp-append-version="true"></script>

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

<h2>@ViewBag.Title</h2>
<br /><br />
<div id="errorConfirmationDialog"></div>
<div id="errorDialog"></div>

<a href="@Url.Action("Add", "Products")"><img src="@Url.Content("~/images/Add.gif")" alt="Add New Products" style="border: none;" /></a>&nbsp;@Ht
<br /><br />

<table class="gridviewGridLines" cellspacing ="0" cellpadding="8" style="width:100%;border-collapse:collapse;">
    <tr style="color:#2E6E9E;background-color:#DFEFFC;font-weight:bold;">
        @for (int i = 0; i < Model.ProductsFieldNames.GetLength(0); i++)
        {
            string fieldName = Model.ProductsFieldNames[i, 0];
            string title = Model.ProductsFieldNames[i, 1];

            if (Model.FieldToSortWithOrder.Contains(fieldName) && Model.FieldToSortWithOrder.Contains("asc"))
            {
                <td><a href="?sidx=@fieldName&sord=desc" style="color:#2E6E9E;">@title</a>@if (Model.FieldToSortWithOrder == fieldName + " asc")
            }
            else
            {
                <td><a href="?sidx=@fieldName&sord=asc" style="color:#2E6E9E;">@title</a>@if (Model.FieldToSortWithOrder == fieldName + " desc")
            }
        }
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>

    @foreach (var item in Model.ProductsData)
    {
        <tr style="color:#333333; background-color:@bgColor;">
            <td align="right">@item.ProductID</td>
            <td>@item.ProductName</td>
            <td align="right"><a href="~/Suppliers/Details/@item.SupplierID">@item.SupplierID</a></td>
            <td align="right"><a href="~/Categories/Details/@item.CategoryID">@item.CategoryID</a></td>
            <td>@item.QuantityPerUnit</td>
            <td align="right">@Convert.ToDouble(item.UnitPrice).ToString("C")</td>
            <td align="right">@item.UnitsInStock</td>
            <td align="right">@item.UnitsOnOrder</td>
            <td align="right">@item.ReorderLevel</td>
            <td align="center"><span><input type="checkbox" @(item.Discontinued ? "checked=\"checked\"" : "") disabled="disabled" /></span></td>
            <td align="center" style="width:30px;">
                <a href="Update/@item.ProductID" title="Click to edit"><img src="@Url.Content("~/images/Edit.gif")" alt="" style="border:none;" /
            </td>
            <td align="center" style="width:30px;">
                <input type="image" id="imgDelete1" title="Click to delete" src="@Url.Content("~/images/Delete.png")" onclick="deleteItem('@item.
            </td>
        </tr>

        bgColor = bgColor == "#F7F6F3" ? "White" : "#F7F6F3";
    }

    <tr class="gridviewPagerStyle" align="center" style="background-color:#DFEFFC;">
        <td colspan="12">
            <table>
                <tr>
                    @if (Model.CurrentPage > Model.NumberOfPagesToShow)
                    {
                        <td><a href="?sidx=@Model.FieldToSort&sord=@Model.FieldSortOrder&page=1" style="color:#000000;">&lt; First</a></td>
                        <td><a href="?sidx=@Model.FieldToSort&sord=@Model.FieldSortOrder&page=@(Model.StartPage - 1)" style="color:#000000;">...<
                    }

                    @for (int pageNumber = Model.StartPage; pageNumber <= Model.EndPage; pageNumber++)
                    {
                        if (pageNumber == Model.CurrentPage)
                        {
                            <td><span style="font-size:12px;">@pageNumber</span></td>
                        }
                        else
                        {
                            <td><a href="?sidx=@Model.FieldToSort&sord=@Model.FieldSortOrder&page=@pageNumber" style="color:#000000;">@pageNumber
                        }
                    }

                    @if (Model.EndPage < Model.TotalPages)
                    {
                        <td><a href="?sidx=@Model.FieldToSort&sord=@Model.FieldSortOrder&page=@(Model.EndPage + 1)" style="color:#000000;">...</a
                        <td><a href="?sidx=@Model.FieldToSort&sord=@Model.FieldSortOrder&page=@Model.TotalPages" style="color:#000000;">Last &gt;
                    }
                </tr>
            </table>
        </td>
    </tr>
</table>
```

Here's the *ListForeach.cshtml* MVC *View* in action.



List of Products

Add New Products

Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit	Unit Price	Units In Stock	Units On Order	Reorder Level	Discontinued		
1	Chai	1	1	10 boxes x 20 bags	$18.00	39	0	10	☐	✏	🗑
2	Chang	1	1	24 - 12 oz bottles	$19.00	17	40	25	☐	✏	🗑
3	Aniseed Syrup	1	2	12 - 550 ml bottles	$10.00	13	70	25	☐	✏	🗑
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	$22.00	53	0	0	☐	✏	🗑
5	Chef Anton's Gumbo Mix	2	2	36 boxes	$21.35	0	0	0	☑	✏	🗑
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	$25.00	120	0	25	☐	✏	🗑
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	$30.00	15	0	10	☐	✏	🗑
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	$40.00	6	0	0	☐	✏	🗑
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	$97.00	29	0	0	☑	✏	🗑
10	Ikura	4	8	12 - 200 ml jars	$31.00	31	0	0	☐	✏	🗑
11	Queso Cabrales	5	4	1 kg pkg.	$21.00	22	30	30	☐	✏	🗑
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	$38.00	86	0	0	☐	✏	🗑
13	Konbu	6	8	2 kg box	$6.00	24	0	5	☐	✏	🗑
14	Tofu	6	7	40 - 100 g pkgs.	$23.25	35	0	0	☐	✏	🗑
15	Genen Shouyu	6	2	24 - 250 ml bottles	$15.50	39	0	5	☐	✏	🗑

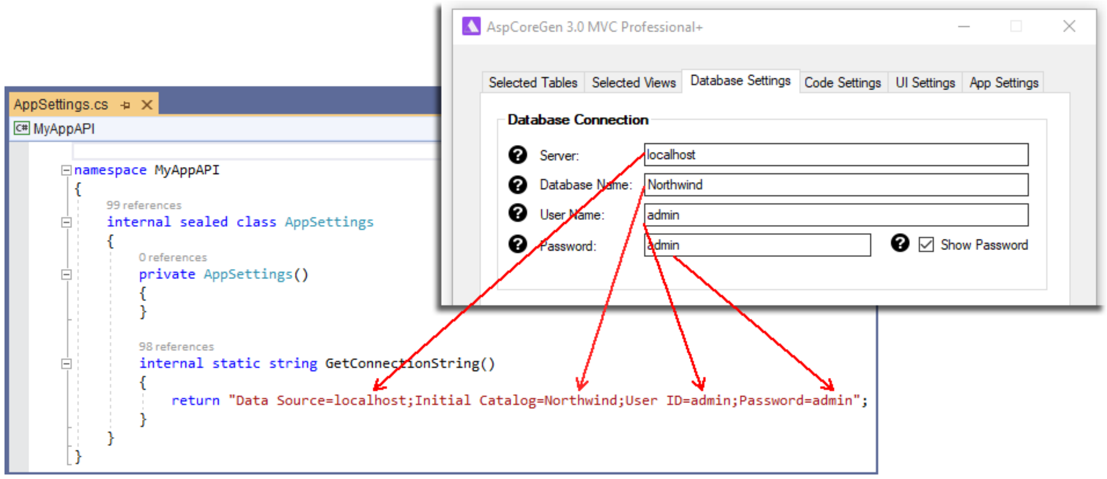
1 2 3 4 5 6

3.2.6 AppSettings.cs

The *AppSettings.cs* is only generated when you choose *Use Stored Procedures* or *Use Ad Hoc/Dynamic SQL* under the *Generated SQL* group in the *Database Settings Tab*. It has one method: *GetConnectionString()*. The *Database Connection* fields you entered under the *Database Settings Tab* ares aved here in a *Database Connection String* format.

The goal of the *GetConnectionString()* method is to simply return the *Database Connection String*.

Do not add code to this *Class*.



```
namespace MyAppAPI
{
    99 references
    internal sealed class AppSettings
    {
        0 references
        private AppSettings()
        {
        }

        98 references
        internal static string GetConnectionString()
        {
            return "Data Source=localhost;Initial Catalog=Northwind;User ID=admin;Password=admin";
        }
    }
}
```

Database Connection

Server:	localhost
Database Name:	Northwind
User Name:	admin
Password:	admin

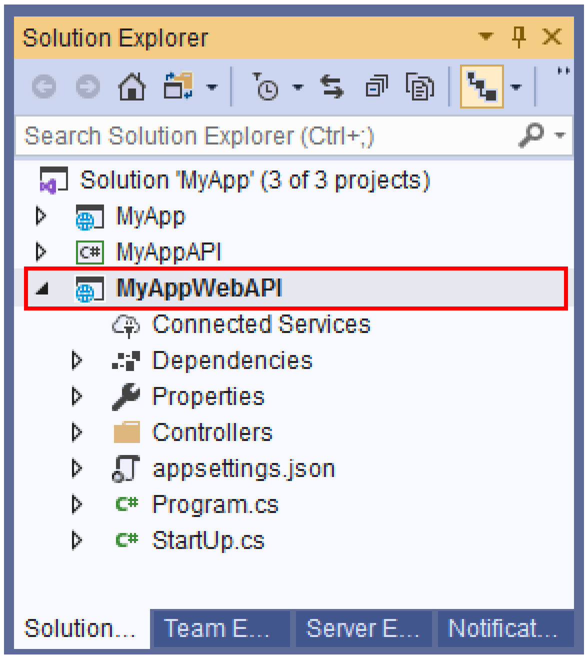
[✓] Show Password

3.3 Web API Project

The generated *Web API Project* is an optional project. This is an ASP.NET MVC API core project. The application's main purpose is to serve as *Web APIs* to clients such as the *Web Application Project*. In the *Ntier-Layering* illustrations #2 and #3 in page 3, the *Web Application Project (ASP.NET MVC Core)* and other clients are seen accessing the *Web APIs* instead or directly accessing the *Middle Tier Objects*.

These *Web APIs* encapsulates the *Middle Layer (Business Objects).* As mentioned in this document, clients can either access the generated *Web APIs* or the *Middle Layer (Business Objects)* directly. But, when you generate the optional *Web API Project*, the generated code will directly reference the *Web APIs* instead of the the *Middle Layer (Business Objects).*

The main difference between the generated *Web Application Project* and the *Web API Project* is that the *Web API Project* only contains *Controllers (Web APIs)* as the main objects of the project, and it does not have a user interface.



3.3.1 LaunchSettings.json, appsettings.json, Program.cs, and StartUp.cs

These are similar objects as the ones seen in the *Web Application Project*. Please see the *Web Application Project* for more information about these objects.

3.3.2 Controllers

The *Controllers* are the *Web APIs*.

This folder is generally needed by ASP.NET Core MVC by default. It houses *Controllers* that can be used as *Web APIs*.

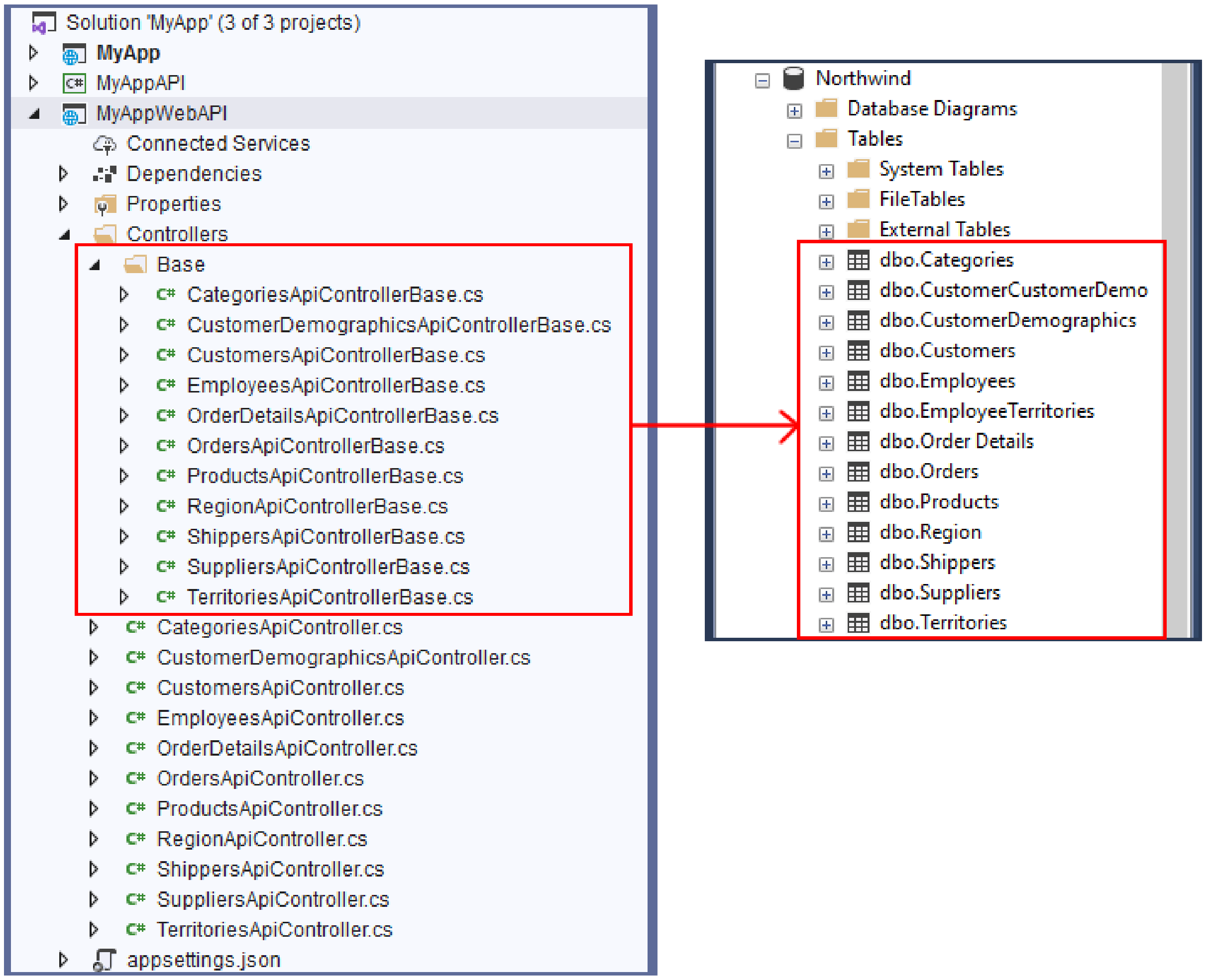
Note: The *Controllers* in the *Web API Project* and the *Web Application Project* are similar in nature. Please read about the *Controllers* under the *Web Application Project* in page 8 for more information.

3.3.2.1 Parent (Base) Class

These are the class files generated in the *Base* folder. The naming convention used is: ***TableName**APIController**Base**.cs*. Because it's just a regular *Class* file, ASP.NET Core MVC does not really recognize it other than it being a *Class* file.

Do not add any code in these *Class* files.

One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.



```
Solution 'MyApp' (3 of 3 projects)
  MyApp
  MyAppAPI
  MyAppWebAPI
    Connected Services
    Dependencies
    Properties
    Controllers
      Base
        CategoriesApiControllerBase.cs
        CustomerDemographicsApiControllerBase.cs
        CustomersApiControllerBase.cs
        EmployeesApiControllerBase.cs
        OrderDetailsApiControllerBase.cs
        OrdersApiControllerBase.cs
        ProductsApiControllerBase.cs
        RegionApiControllerBase.cs
        ShippersApiControllerBase.cs
        SuppliersApiControllerBase.cs
        TerritoriesApiControllerBase.cs
      CategoriesApiController.cs
      CustomerDemographicsApiController.cs
      CustomersApiController.cs
      EmployeesApiController.cs
      OrderDetailsApiController.cs
      OrdersApiController.cs
      ProductsApiController.cs
      RegionApiController.cs
      ShippersApiController.cs
      SuppliersApiController.cs
      TerritoriesApiController.cs
    appsettings.json
```

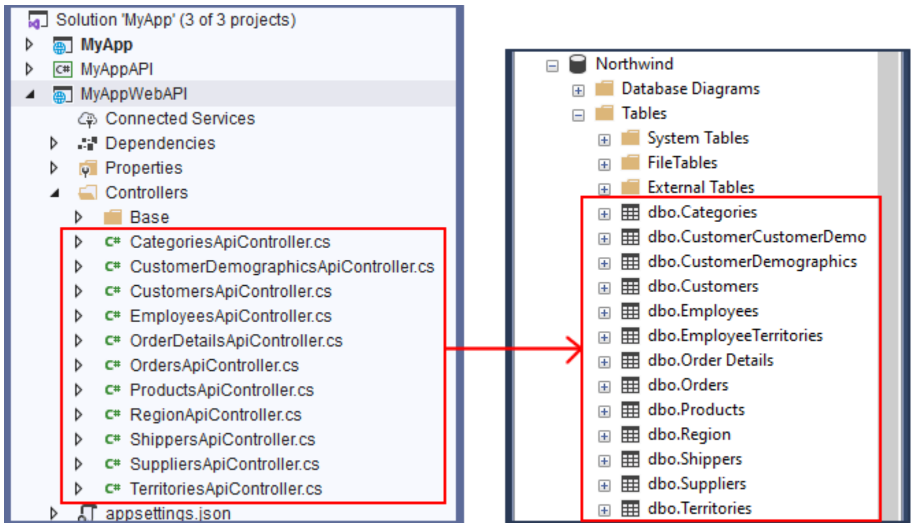
```
Northwind
  Database Diagrams
  Tables
    System Tables
    FileTables
    External Tables
    dbo.Categories
    dbo.CustomerCustomerDemo
    dbo.CustomerDemographics
    dbo.Customers
    dbo.Employees
    dbo.EmployeeTerritories
    dbo.Order Details
    dbo.Orders
    dbo.Products
    dbo.Region
    dbo.Shippers
    dbo.Suppliers
    dbo.Territories
```

Base Controllers in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

3.3.2.2 Child Class – The Controller (Web API)

These are the *Class* files generated directly under the *Controllers* folder (not including everything inside the *Base* folder). The naming convention used is: ***TableName****APIController.cs*. ASP.NET Core MVC recognizes this as a *Controller* by default because of the suffix "*Controller*" in the name. One *Controller* is generated per *Database Table*.

You can add code in these *Class* files.



Child Controllers in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

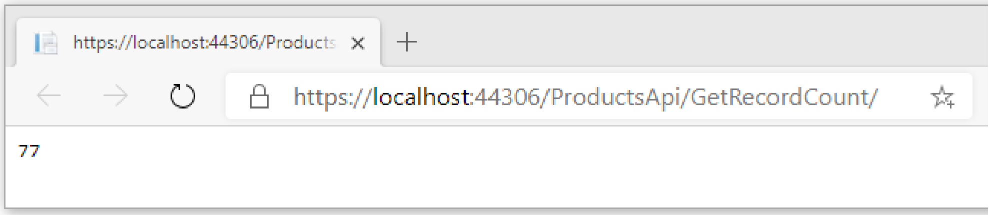
3.3.2.3 Accessing Web API Controllers

Just like mentioned above, the *Web API Project* does not have a user interface just like the *Web Application Project's* MVC *Views*. We need to access *Web API Controllers* via code using *HttpClient* calls.

For example, we need to make an *HttpClient Get Request* call from the *Web Application Project's Controller (ProductsControllerBase)* to access the *GetRecordCount() Method* in the *Web API Controller (Products**API**ControllerBase)*.

To make an *HttpClient Get Request* call, we use the:

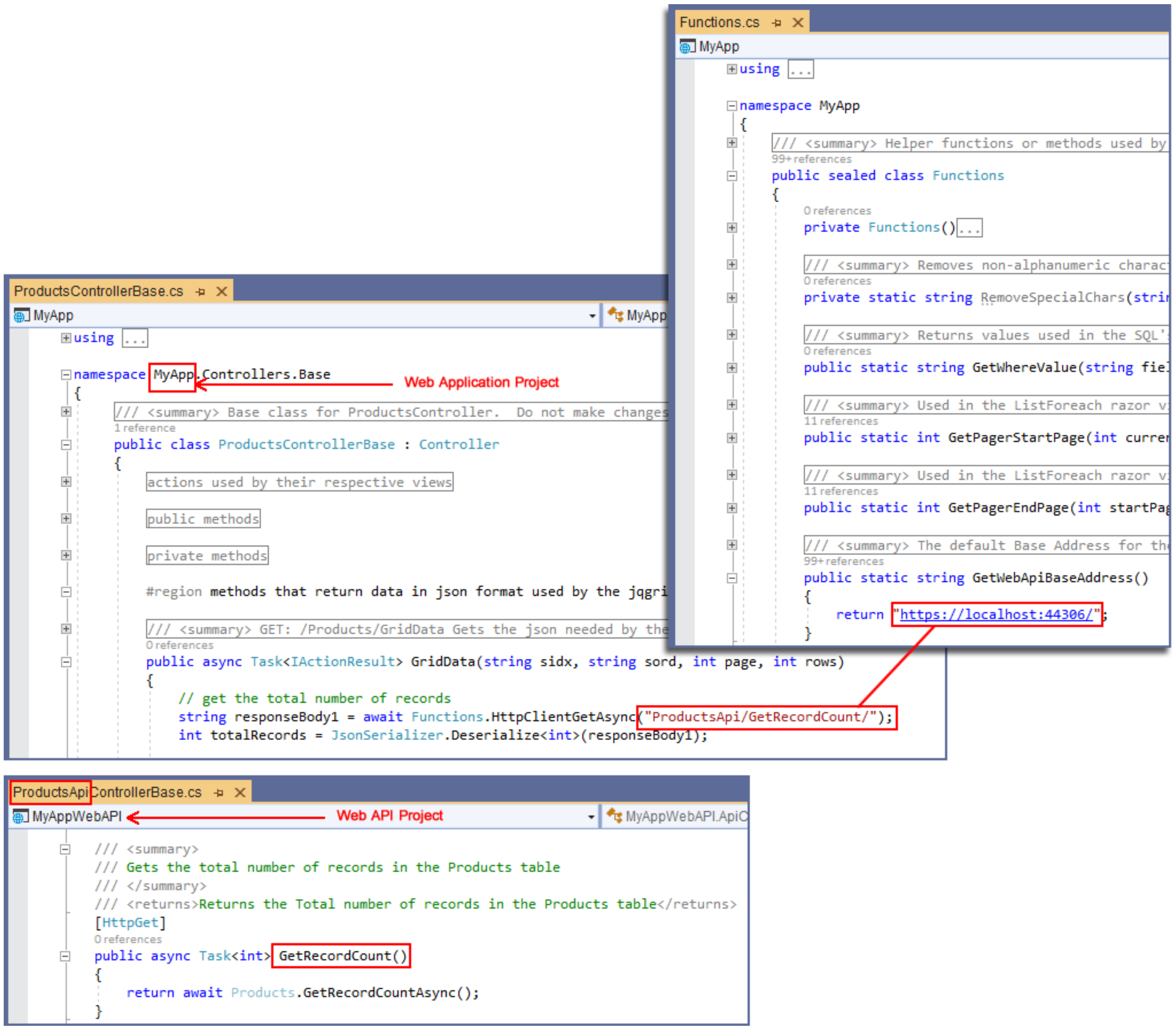
1. *Web API Project's Web Address (URL,* ***https://localhost:44306/****)*
2. *Controller's* name (***ProductsAPI***, minus the word *"Controller"*),
3. And the *Method* name (***GetRecordCount()***)



https://localhost:44306/Products
https://localhost:44306/ProductsApi/GetRecordCount/

```
77
```

The example below shows that *GetRecordCount() (Web API Project)* was called from the *GridData Method (Web Application Project)* using the *Web API*'s base URL *"https://localhost:44306/" (Functions Class in the Web Application Project)* plus the *"ProductsAPI/GetRecordCount".*



```
using ...

namespace MyApp
{
    /// <summary> Helper functions or methods used by
    99+ references
    public sealed class Functions
    {
        0 references
        private Functions()...

        /// <summary> Removes non-alphanumeric charac
        0 references
        private static string RemoveSpecialChars(stri

        /// <summary> Returns values used in the SQL'
        0 references
        public static string GetWhereValue(string fie

        /// <summary> Used in the ListForeach razor v
        11 references
        public static int GetPagerStartPage(int curre

        /// <summary> Used in the ListForeach razor v
        11 references
        public static int GetPagerEndPage(int startPa

        /// <summary> The default Base Address for th
        99+ references
        public static string GetWebApiBaseAddress()
        {
            return "https://localhost:44306/";
        }
```

```
using ...

namespace MyApp.Controllers.Base
{
    /// <summary> Base class for ProductsController.  Do not make changes
    1 reference
    public class ProductsControllerBase : Controller
    {
        actions used by their respective views

        public methods

        private methods

        #region methods that return data in json format used by the jqgri

        /// <summary> GET: /Products/GridData Gets the json needed by the
        0 references
        public async Task<IActionResult> GridData(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);
```

```
        /// <summary>
        /// Gets the total number of records in the Products table
        /// </summary>
        /// <returns>Returns the Total number of records in the Products table</returns>
        [HttpGet]
        0 references
        public async Task<int> GetRecordCount()
        {
            return await Products.GetRecordCountAsync();
        }
```

*   CRUD means Create, Retrieve, Update, and Delete. These are database operations.

You can read end-to-end tutorials on more subjects on using AspCoreGen 3.0 MVC 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 MVC Professional.

Note: Some features shown here are not available in the Express Edition.

End of tutorial.