The Generated Code for Database Tables

The Generated Code for Database Tables

1 INTRODUCTION

This topic will walk you through AspCoreGen 3.0 Razor's generated code.

1.1 READ THESE TUTORIALS IN ORDER

1. Database Settings Tab
2. Code Settings Tab
3. UI Settings Tab
4. App Settings Tab
5. Selected Tables Tab
6. Selected Views Tab
7. Generating Code

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

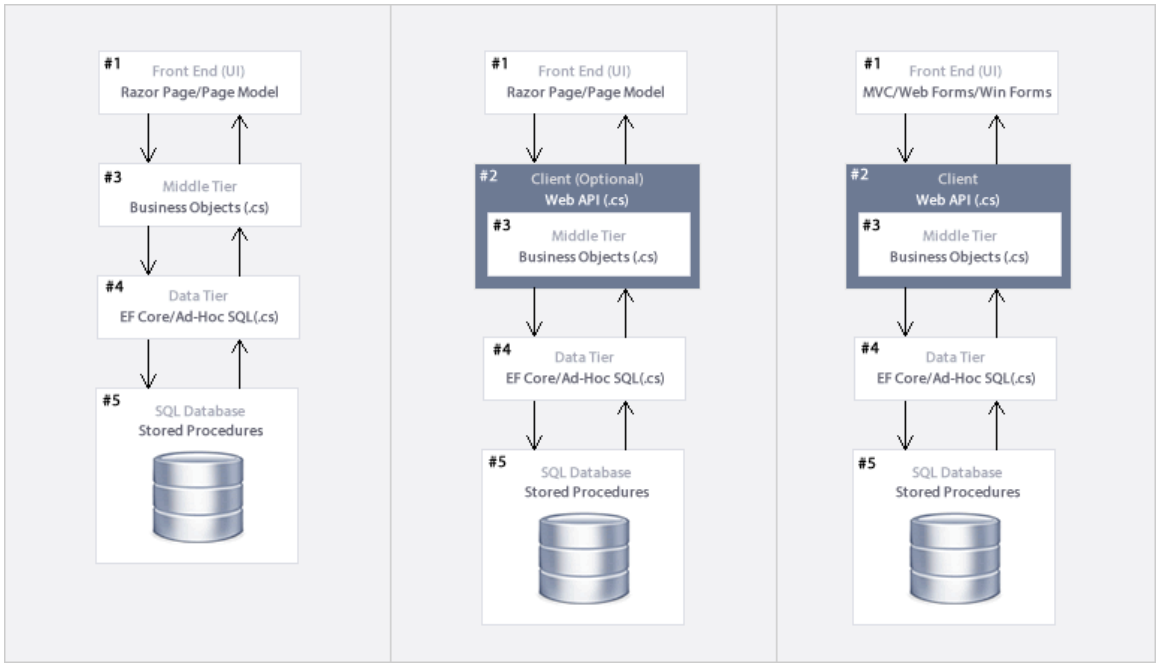
In the *App Settings Tutorial* there are 3 projects that can be generated in a solution:

- 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 Razor 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 are separated in different layers.



2.1 FRONT END

User Interface or Presentation Layer. Razor Pages and Razor Page Models, Partial Razor Pages and Partial Razor Page Models, JavaScript, CSS, JQuery, and more.

2.2 MIDDLE-TIER/ MIDDLE LAYER

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

2.3 DATA-TIER/ DATA LAYER

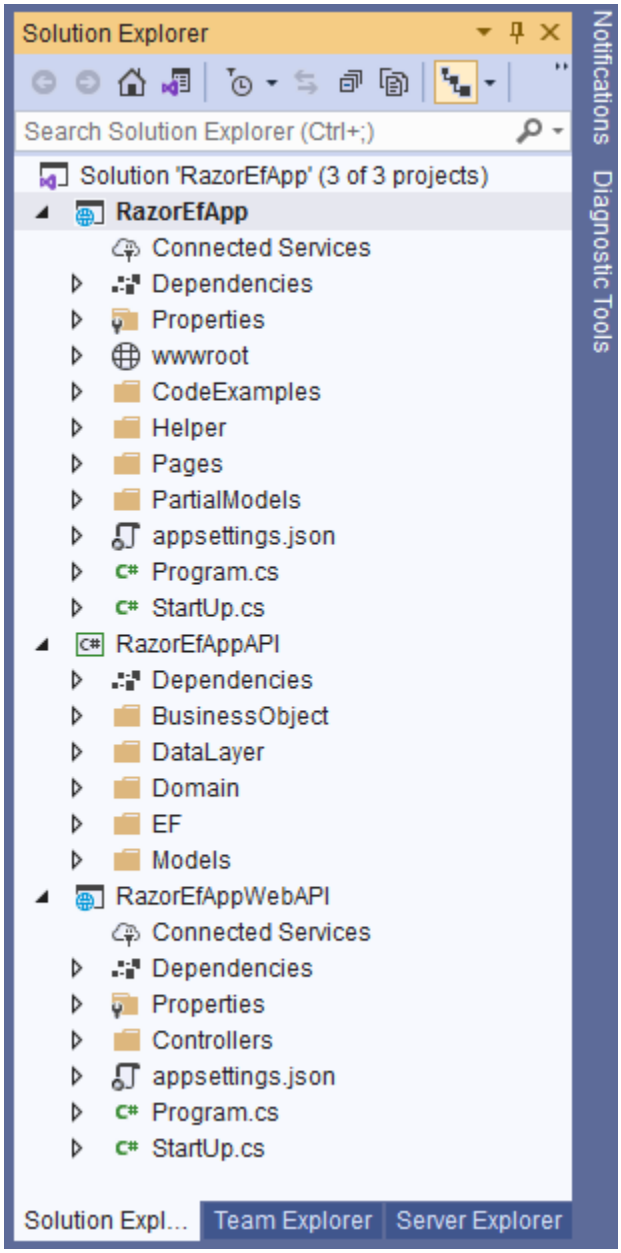
Class Files using Linq-to-Entities - Entity Framework Core or Ad-Hoc SQL.

2.4 SQL SCRIPTS

Stored Procedures.

3 GENERATED PROJECTS

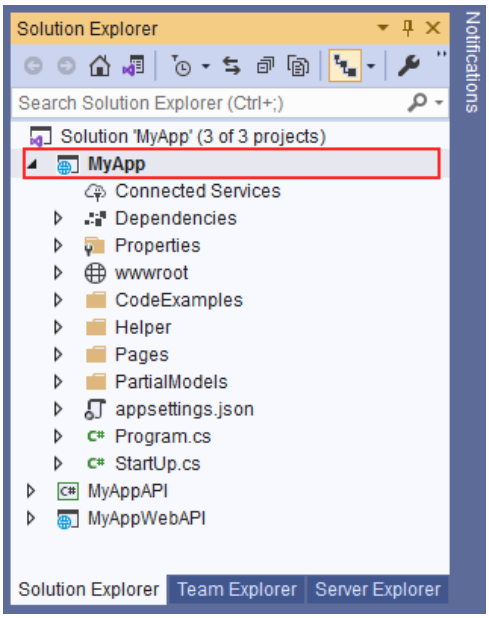
There are 3 projects that can be generated by AspCoreGen 3.0 Razor Professional Plus including the optional *Web API* project. In addition, if you chose *Stored Procedures* under the *Generated SQL Script* in the *Database Settings Tab*, these SQL scripts will be generated straight in your MS SQL Server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



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 Core Razor Page 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.

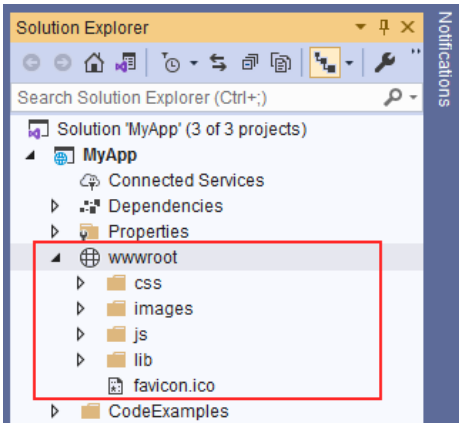


In this example, the *MyApp* project is the *Web Application Project* that was generated. Everything in this project is used to present users with an interface they can interact with, except the optional *CodeExamples* folder which contains *Class Files* for each of the database tables showing code examples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

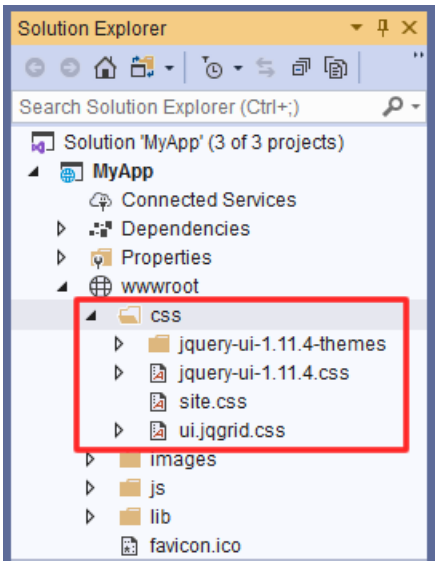
As shown in the *N-Tier Layering* above, the *Front End* (Razor Page) accesses the *Middle Tier* (class) to do any kind of operation. Alternatively, it can also access the *Web API* instead of the *Middle Tier* (class).

3.1.1 wwwroot

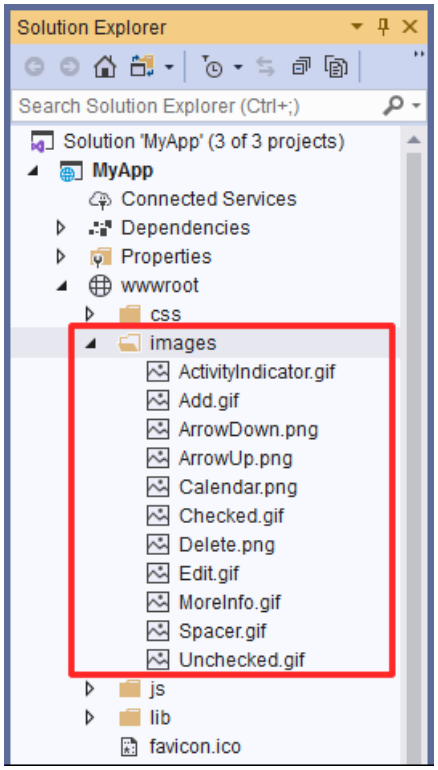
This folder is generally needed by ASP.NET Core Razor Page project as the *Web Root* of the project by default. You can place static files needed by the ASP.NET Core Razor Page project here. You can add folders and files and name them to whatever you like.



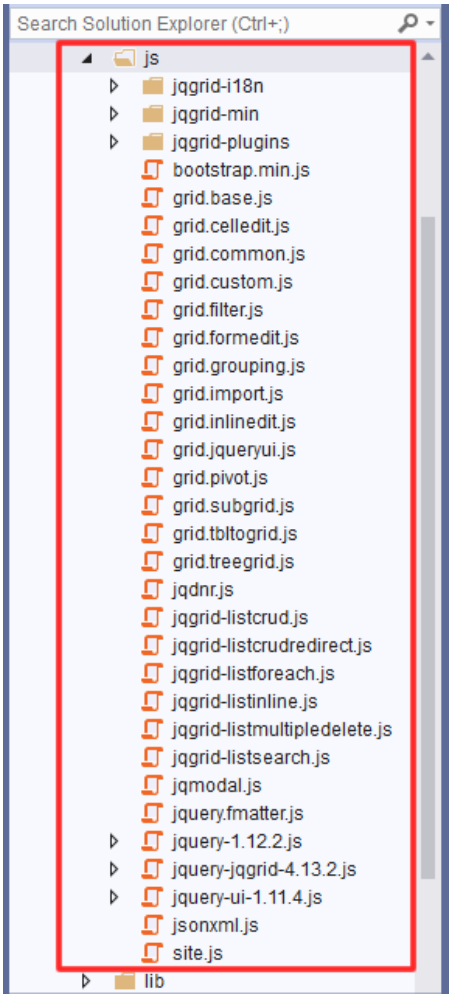
1. **css (folder):** Contains styles including 24 different JQuery-UI themes used by the project. You can add your own stylesheets here. You can also add and updates styles in the *site.css* stylesheet.



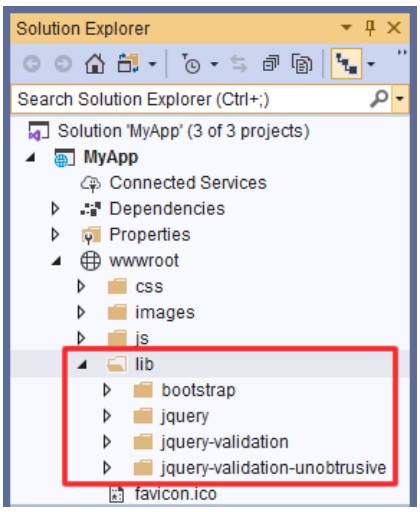
2. **images (folder):** Contains images used by the project. You can add your own images here.



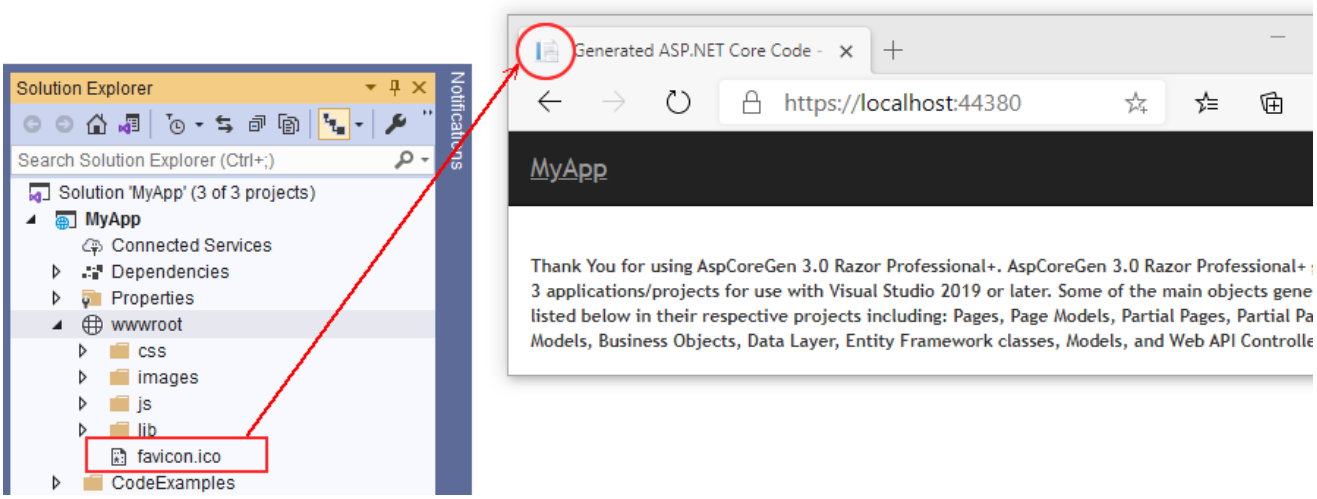
3. **js (folder):** Contains javascript files including JQGrid and JQuery plugins used by the project. You can add your own scripts here.



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



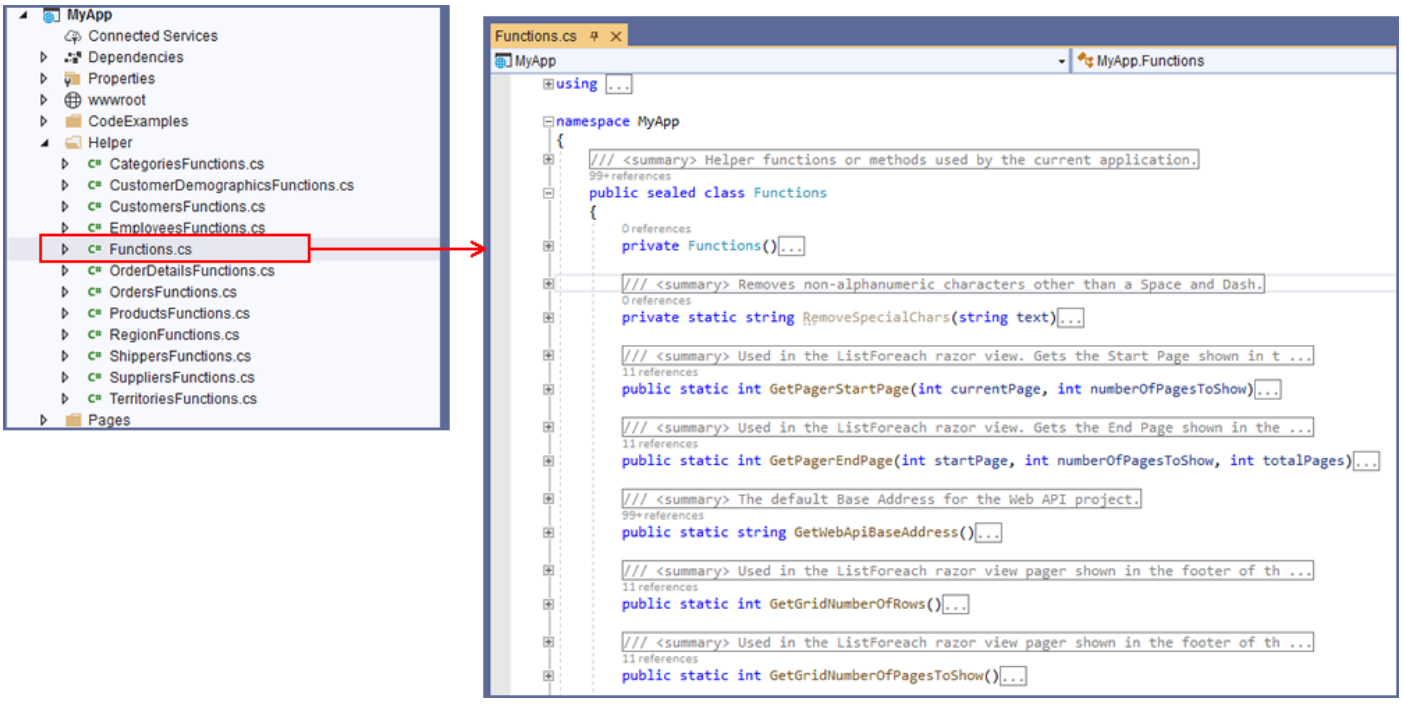
5. **favicon.ico:** An icon used by the browser as the default icon for your project. You can change this to your own icon (brand).



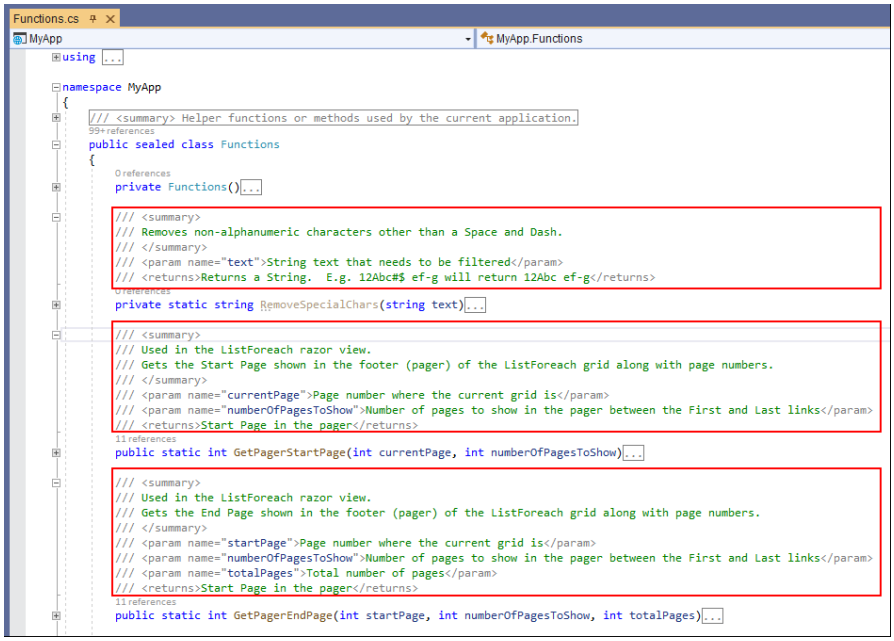
3.1.2 Helper

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.**



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

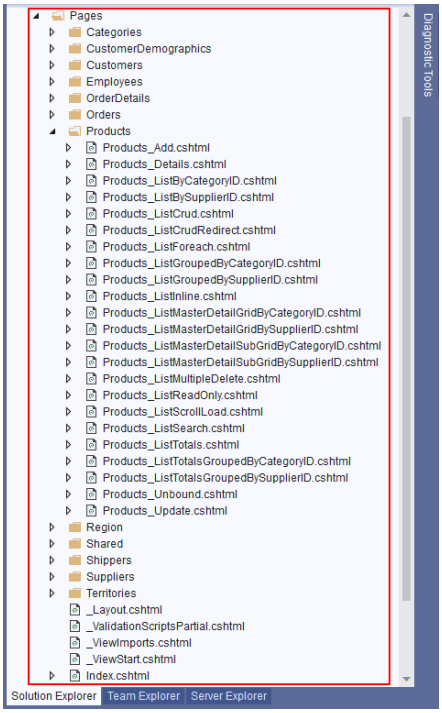


3.1.3 Razor Pages

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

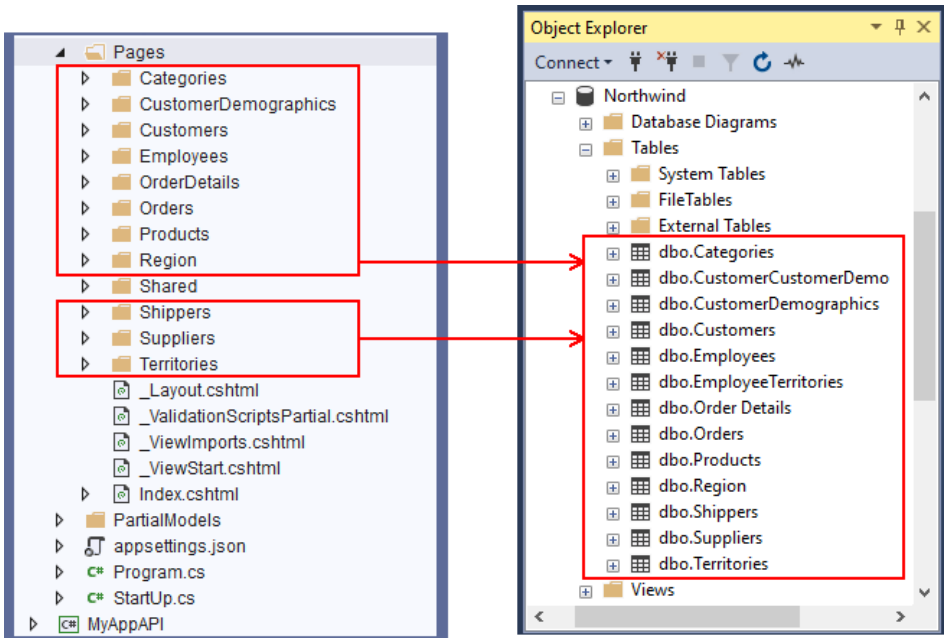
Note: Do not add any code in any of the generated Razor Pages. Please see the *AppSettingsTab Tutorial*, page 5 (1.1.2 *Files That Will Always Be Overwritten*) for more information.

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



3.1.3.1 Razor Pages Generated for Database Tables

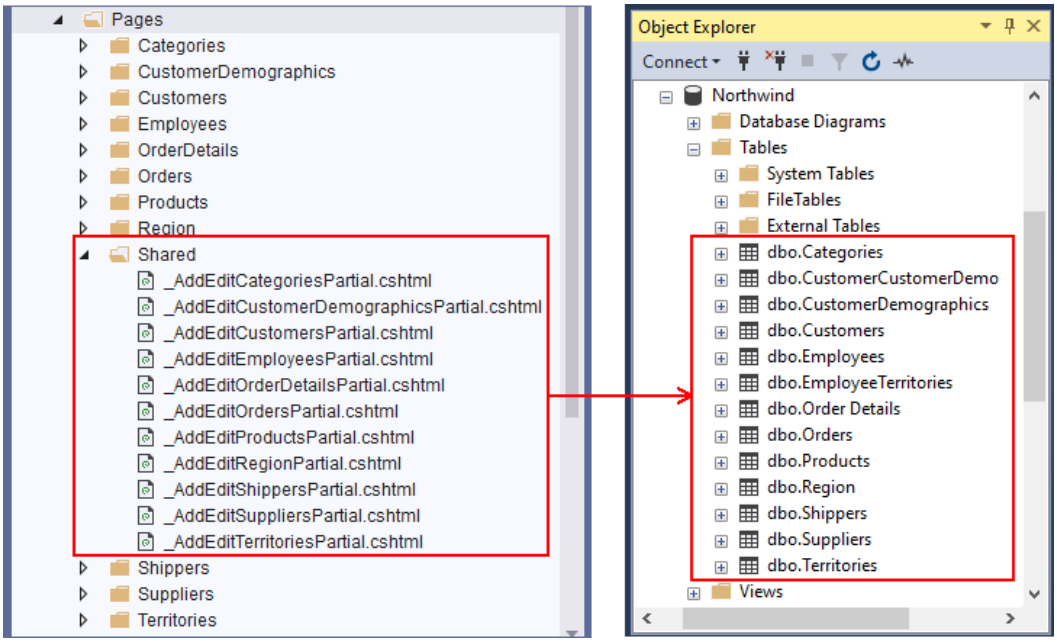
These Razor Pages are generated based on the *Database Tables* you generated code for. Each *Folder* as shown below is directly related to a *Database Table*.



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

3.1.3.2 Partial Razor Pages for Database Tables

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



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

Each *Partial Razor Page* is used by the ***Add.cshtml* and ***Update.cshtml* Razor Pages.

Products_Add.cshtml

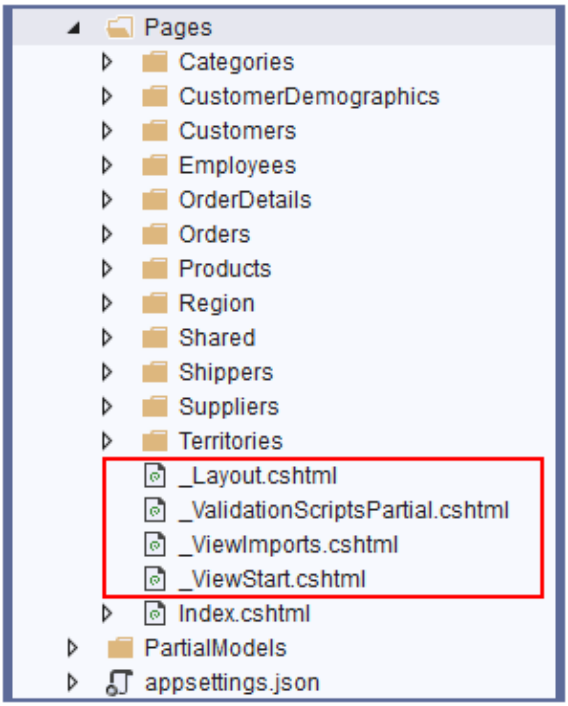
```
@page
@model MyApp.Pages.Products_AddModel
@section AdditionalJavaScript {
    @await Html.PartialAsync("_ValidationScriptsPartial")
}

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

Products_Update.cshtml

```
@page
@model MyApp.Pages.Products_UpdateModel
@section AdditionalJavaScript {
    @await Html.PartialAsync("_ValidationScriptsPartial")
}

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

3.1.3.3 Other Partial Razor Pages

These are mainly ASP.NET Core Razor Page project default *Partial Razor Pages*. The ASP.NET Core Razor Page project naming convention for *Partial Razor Pages* starts with an *Underscore* "_" prefix.



3.1.3.3.1 _Layout.cshtml

The *_Layout.cshtml* is a *Partial Razor Page* that is the default overall design or master page for all the Razor Pages that incorporates it. Razor Pages 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 Razor Pages by changing all or a few code here.**

_Layout.cshtml

```
<!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 href="/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>
```

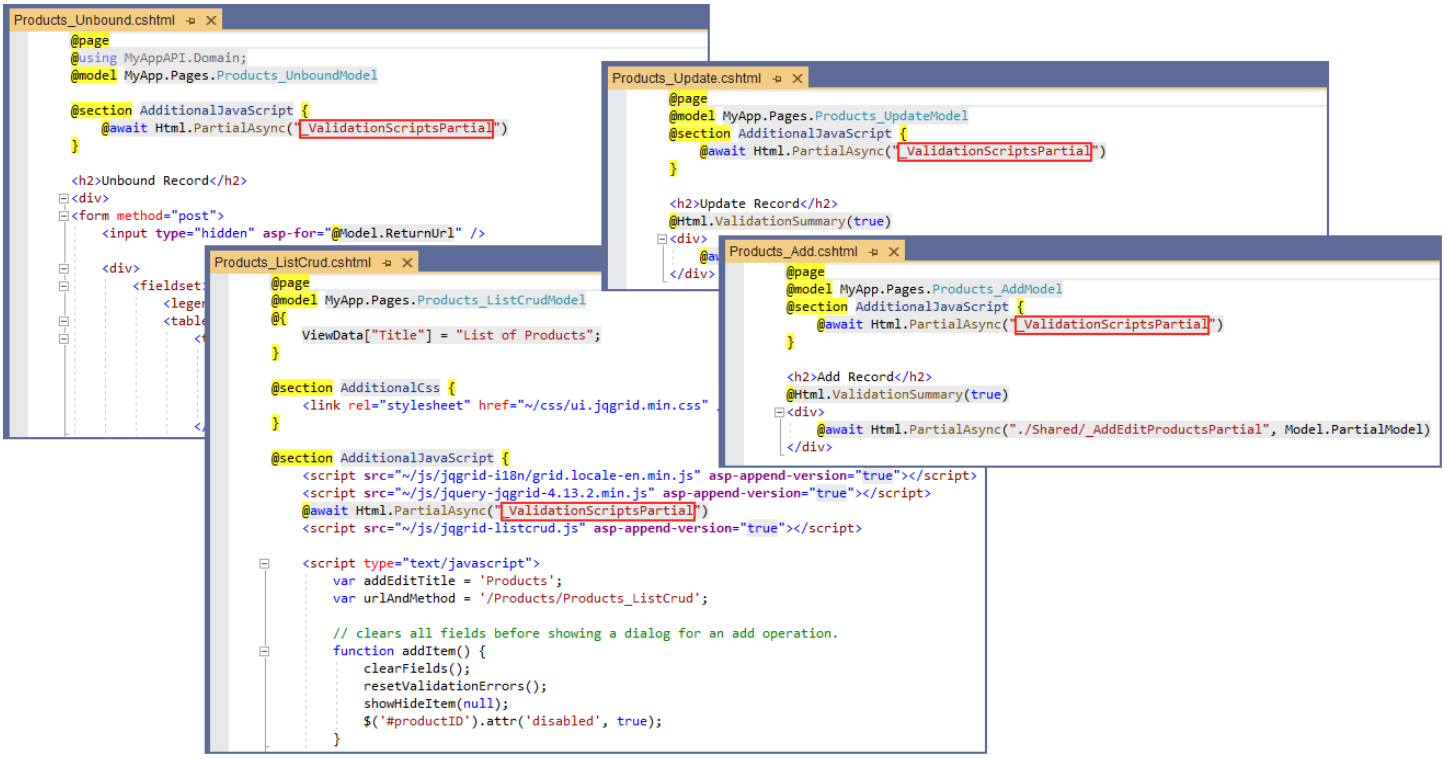
3.1.3.3.2 _ValidationScriptPartial.cshtml

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

_ValidationScriptsPartial.cshtml

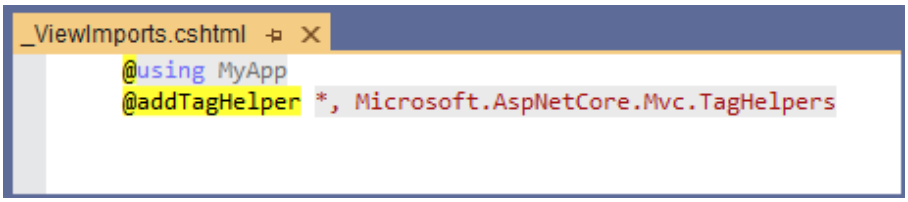
```
<environment names="Development">
    <script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
    <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>
</environment>
<environment names="Staging,Production">
    <script src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.14.0/jquery.validate.min.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 Razor Pages: *Add.cshtml, Update.cshtml, Unbound.cshtml,* and *ListCrud.cshtml* as shown below.



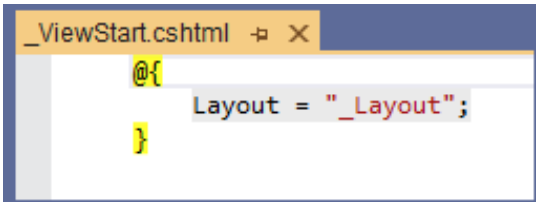
3.1.3.3.3 _ViewImports.cshtml

This *Partial Razor Page* imports directives that can be shared throughout all the generated Razor Pages. **You can add your own code here**.



3.1.3.3.4 _ViewStart.cshtml

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

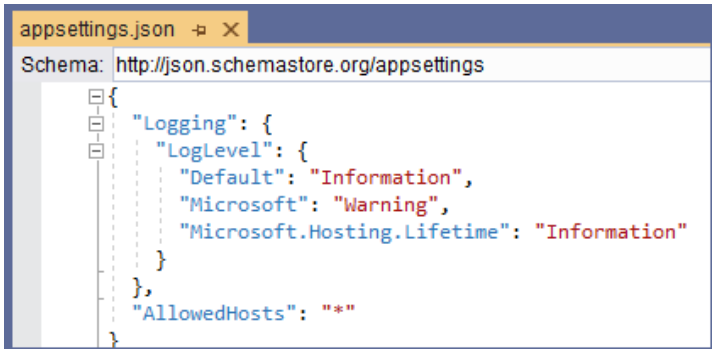


3.1.3.4 Index.cshtml Razor Page

This Razor Page is the default page of the *Web Application Project.*

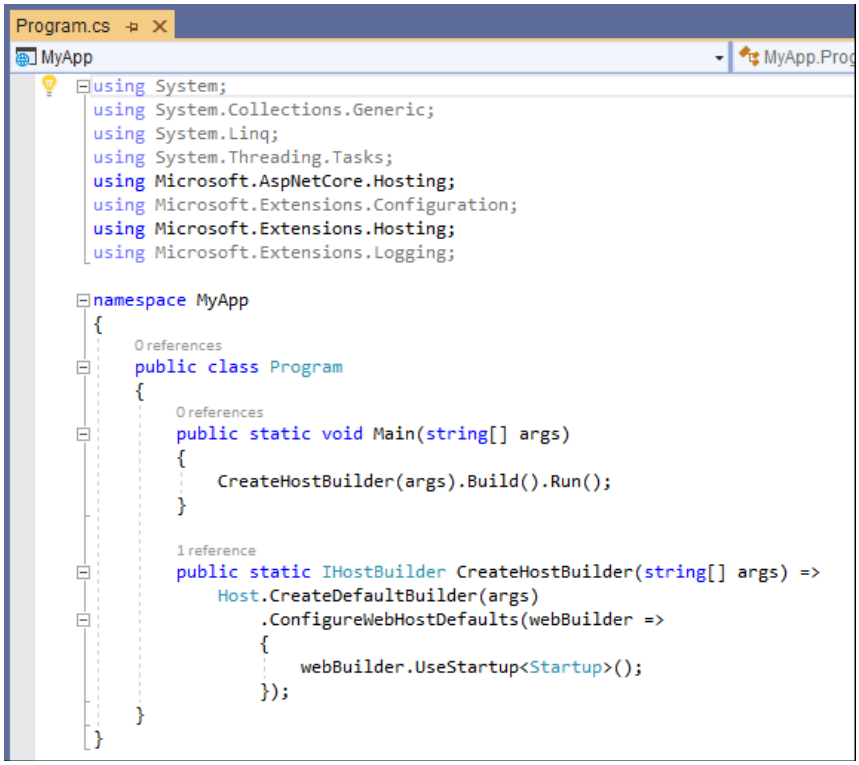
3.1.4 appsettings.json

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



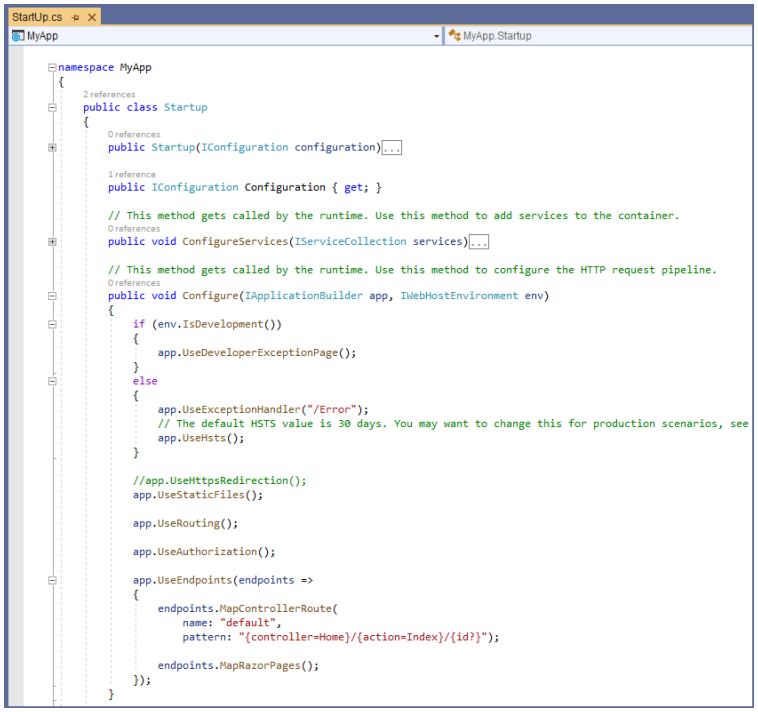
3.1.5 Program.cs

The *Program.cs Class* is the entry point to the ASP.NET Core Razor Page *Web Application Project* by default. An ASP.NET Core Razor Page 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**.



3.1.6 StartUp.cs

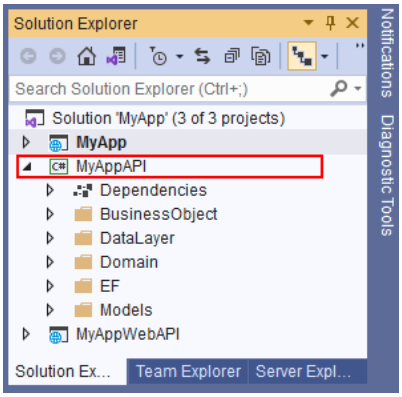
The *StartUp Class* also starts execution before any ASP.NET *Razor Page* runs and is called by the *Program Class* as shown above, *"webBuilder.UserStartup<**Startup**>()".* Here, you can set or configure services that can be used globally in the *Web Application Project* by default. **You can add your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



3.2 BUSINESS OBJECT AND DATA LAYER API PROJECT

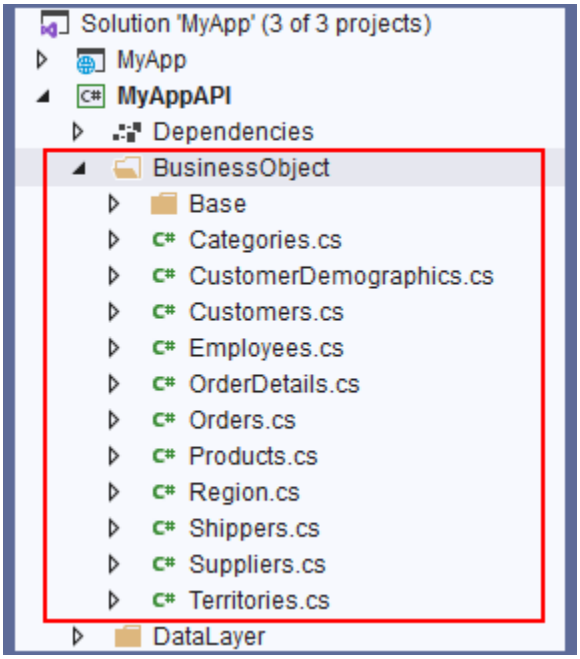
The generated *Business Object and Data Layer API Project* contains the *Middle-Tier* and *Data-Tier* part of the N-tier layer generated code (and other classes as well). This is a *Class Library* core project. ***Classes/Objects* here can be reused by other projects/clients except the *Data Layer Classes*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



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 Razor 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 of 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 Razor), let us say somewhere in the *Razor Page Model* 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.

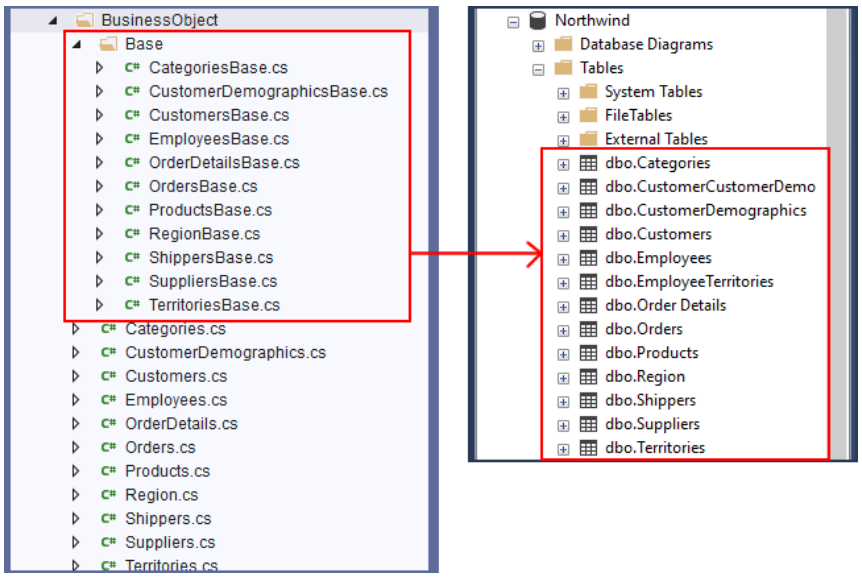
return User.FirstName + " " + User.LastName;

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.

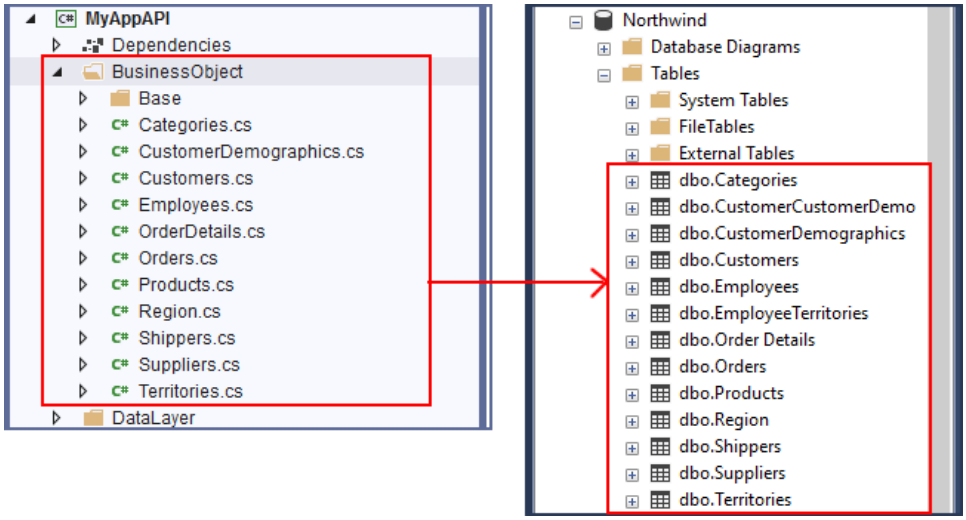
One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.



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 Razor's generated code will always access *Web API Methods* from clients like the *Razor Page Model* 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 do not always have to access the *Web API Methods* (from any client) generated by AspCoreGen 3.0 Razor, 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

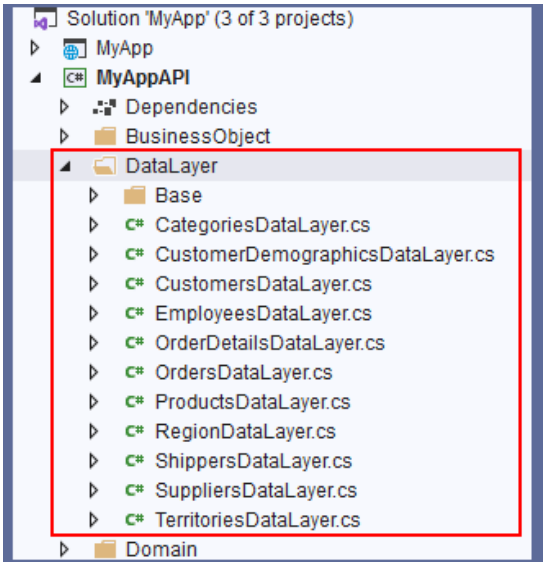
The *Data Tier*'s (or *Data Layer*) main purpose is to interact with the database. It does all the CRUD* operations.

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*.

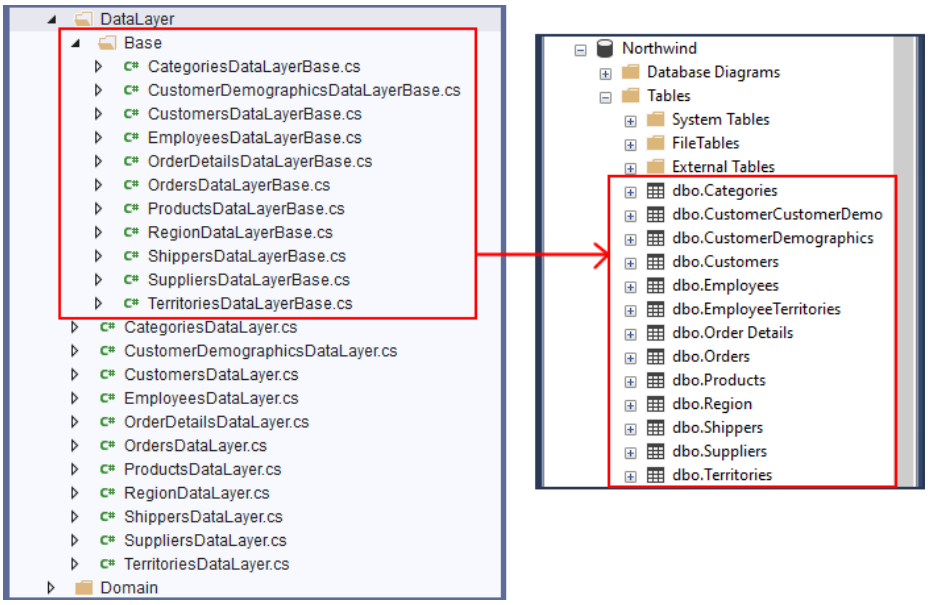
The *Data Tier* (*Data Layer*) *Class Files* are located in the *DataLayer Folder*.



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.**

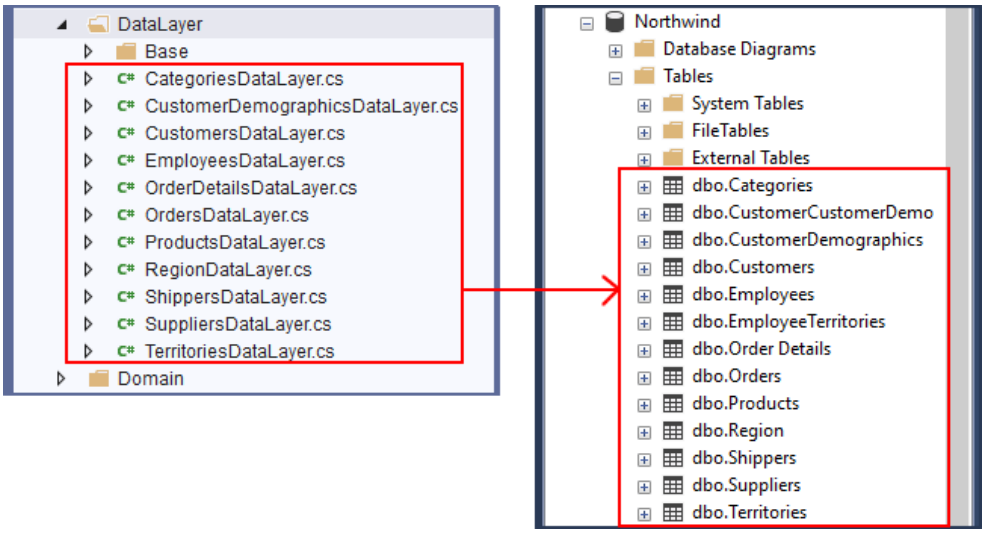
One *Class* is generated per *Database Table* you generated code for. The example below shows that you generated code for *All Tables* for the *Northwind* database.



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)

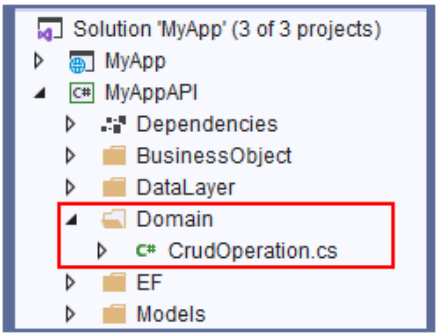
You can add code in these *Class* files. You access all the *Data Layer* methods and properties using this Class.

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

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

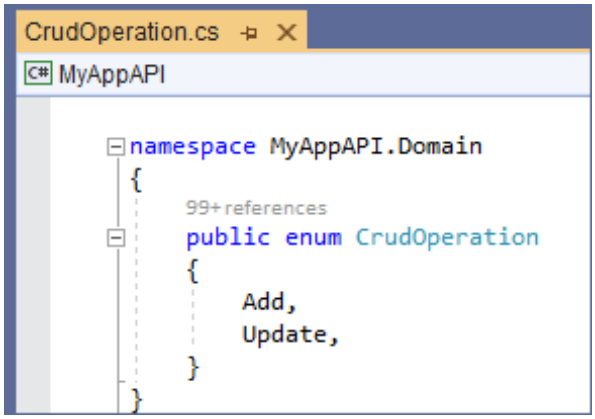
3.2.3 Domain

The *Domain Folder* contains 1 reusable *enum* type object; the *CrudOperation.cs*.



3.2.3.1 CrudOperation.cs

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

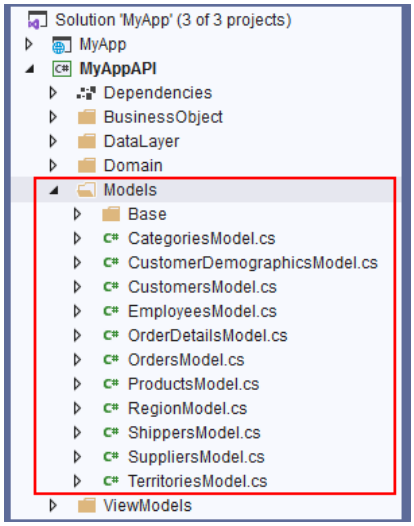


3.2.4 Models

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*.

So why are *Models* generated in the *Business Object and Data Layer API Project* instead of the *Web Application Project* where the Razor Page*s* and the respective Razor Page Models are? Simple, ***Models* are reusable**.

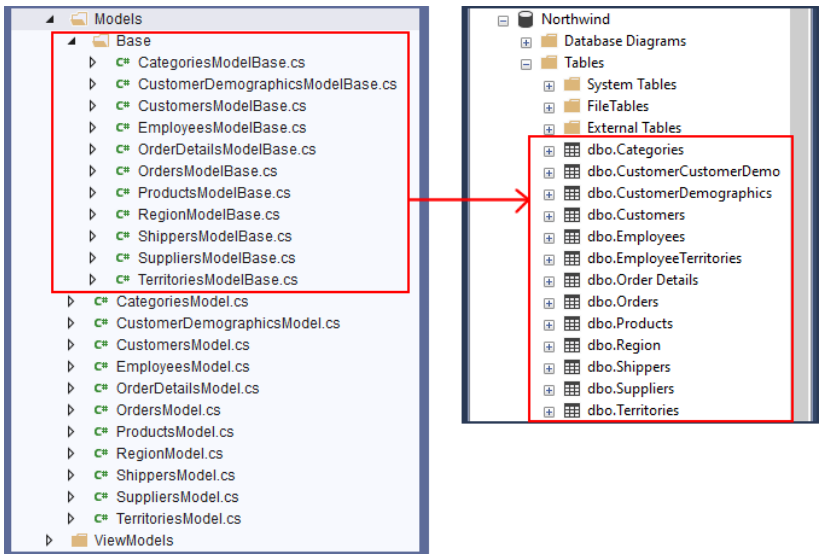
The *Models* are located in the *Models Folder*.



3.2.4.1 Parent (Base) Class

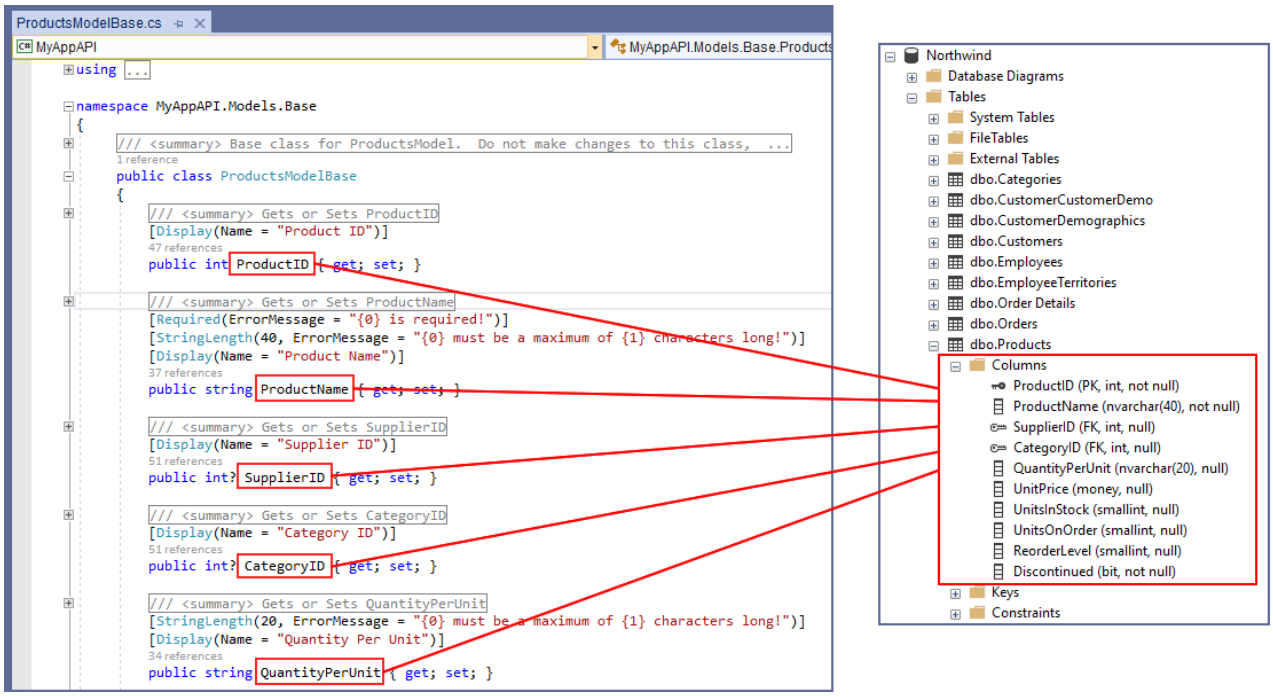
These are the class files generated in the *Base* folder. The naming convention used is: *TableName**ModelBase**.cs*. **Do not add any code in these *Class* files.**

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

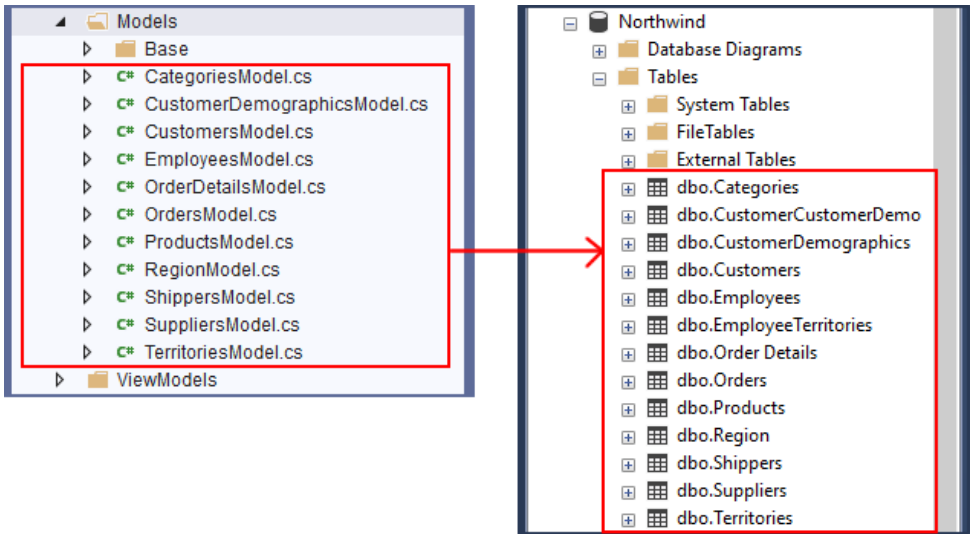


Model (Base) Class in Visual Studio (Left) – Products Database Table Columns in MS SQL Server (Right)

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*.

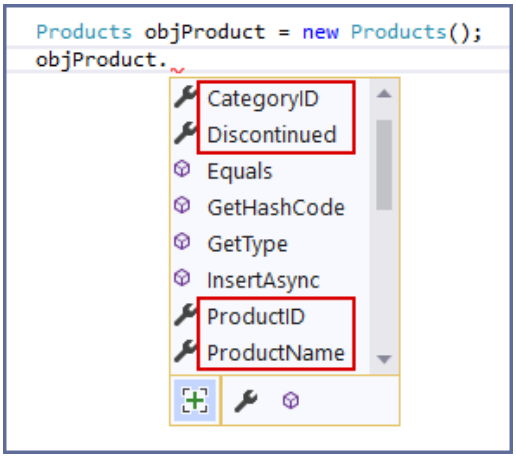
You can add code in these *Class* files.



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

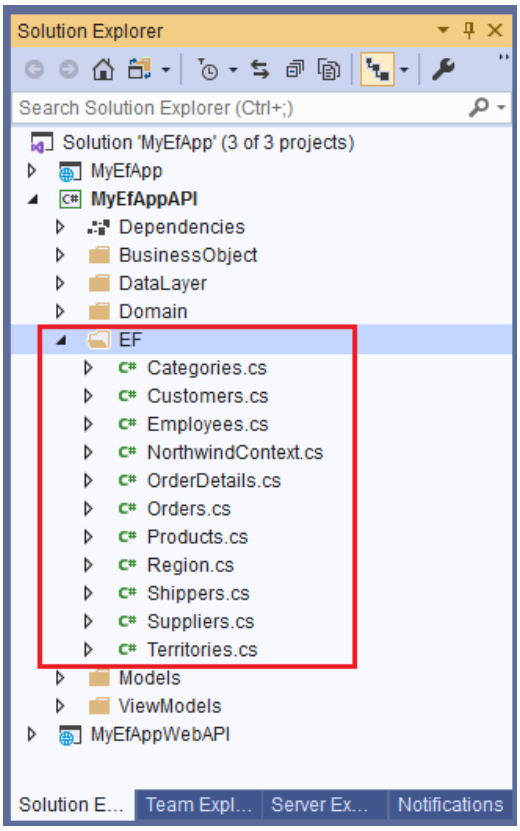
Models are inherited by the respective *Business Object Class*. So when you instantiate a *Business Object*, you will have access to the *Models* (*Properties*) for that *Business Object*.

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.



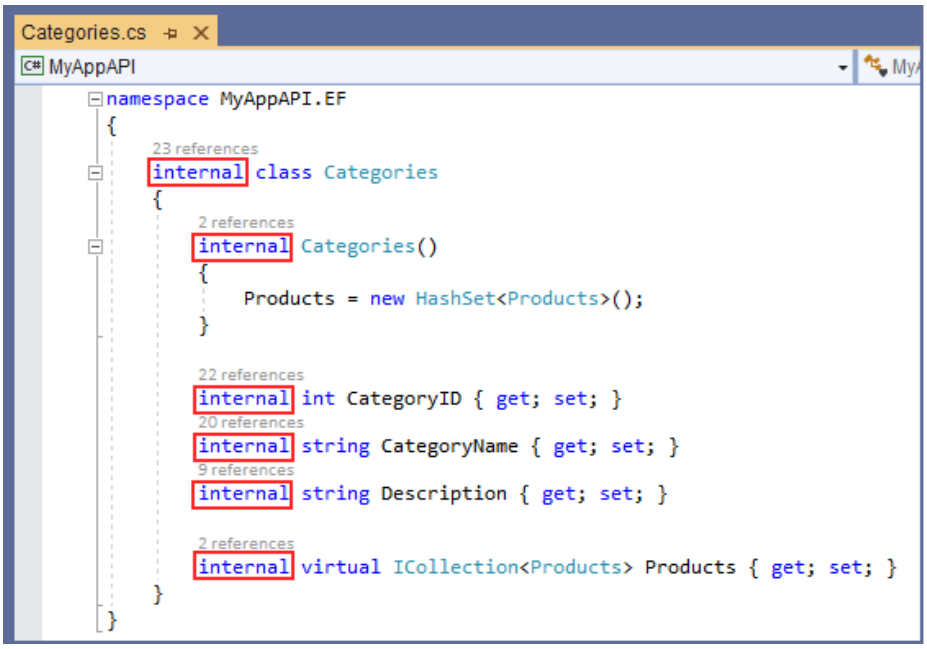
3.2.5 EF (Entity Framework)

These classes are generated in the *EF* folder when you choose *Use Linq-to-Entities (Entity Framework Core)* in the *Database Settings* tab under the *Generated SQL* group as discussed in page 6 of the *DatabaseSettingsTab* tutorial document.



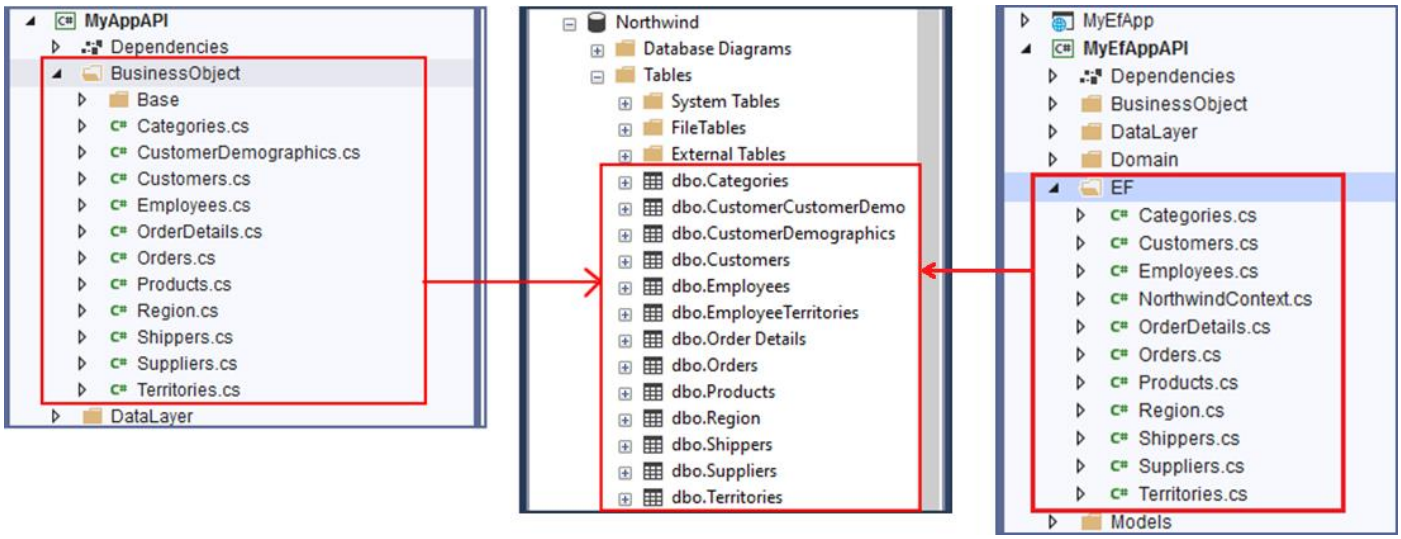
3.2.5.1 Generated Entity Model

The generated *Entity Models/Classes* are very similar to the ones that can be generated by Visual Studio when you issue a *Scaffold-DbContext* command (we will not discuss this command here because it is outside the scope of this tutorial). One of the main difference is that each *Entity Model/Class* and all of the respective properties are *internal* while the ones generated by Visual Studio are public.



3.2.5.2 Why internal?

The generated *BusinessObject* (public) classes have the same name as the generated *Entity Models* (internal). Clients that calls the *Middle Tier/BusinessObject* classes expects a *BusinessObject* class as a return and NOT an *Entity Model.*



Business Object Classes (Left), MS SQL Database (Middle), Entity Models/Classes (Right)

The example below shows that the call to a *Middle Tier BusinessObject* expects a *List of Categories* (*BusinessObject*).

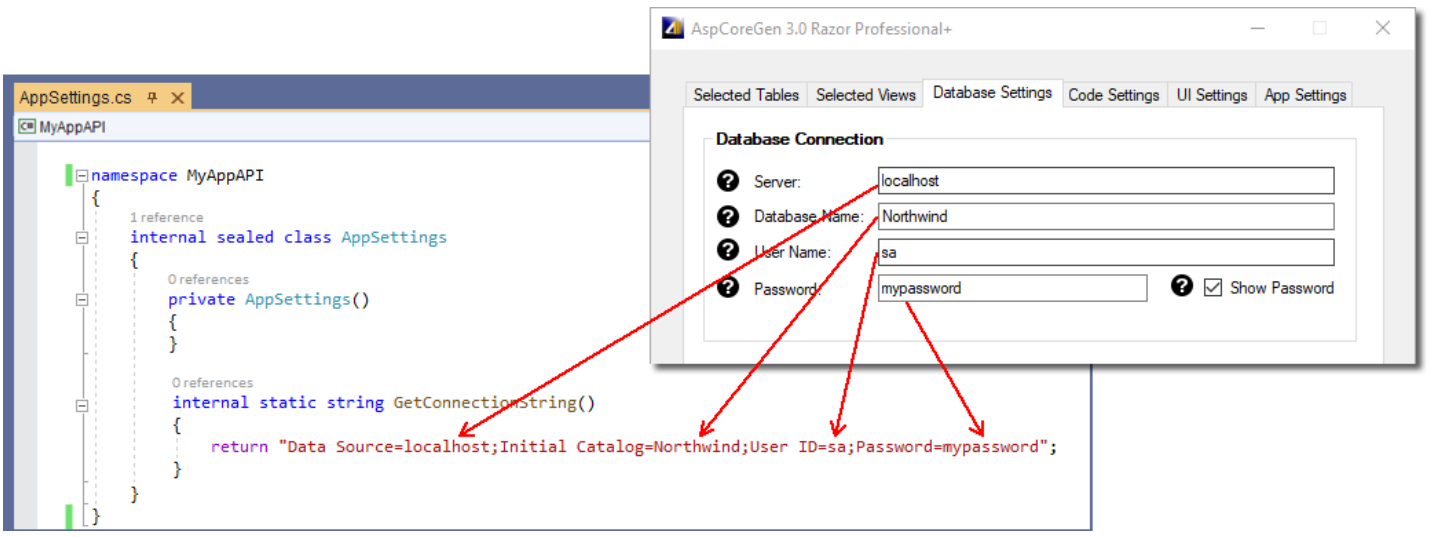
```
public async Task<IActionResult> OnGetGridDataAsync(string sidx, string sord, int _page, int rows)
{
    // get the index where to start retrieving records from
    // 0 = starts from the beggining, 10 means skip the first 10 records and start from record 11
    int startRowIndex = ((_page * rows) - rows);

    // get the total number of records
    int totalRecords = await Categories.GetRecordCountAsync();

    // get records
    List<Categories> objCategoriesCol = await Categories.SelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);

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

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

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* are saved 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*.

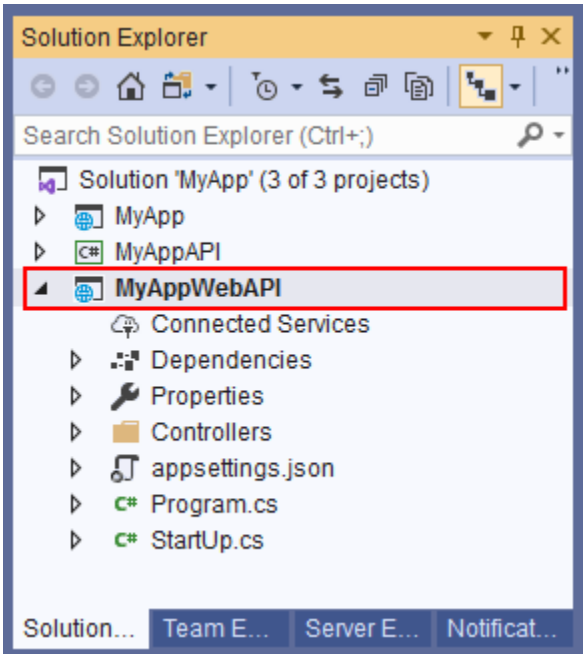


3.3 WEB API PROJECT

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 4, the *Web Application Project (ASP.NET Core Razor Pages)* and other clients are seen accessing the *Web APIs* instead of 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 *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*.

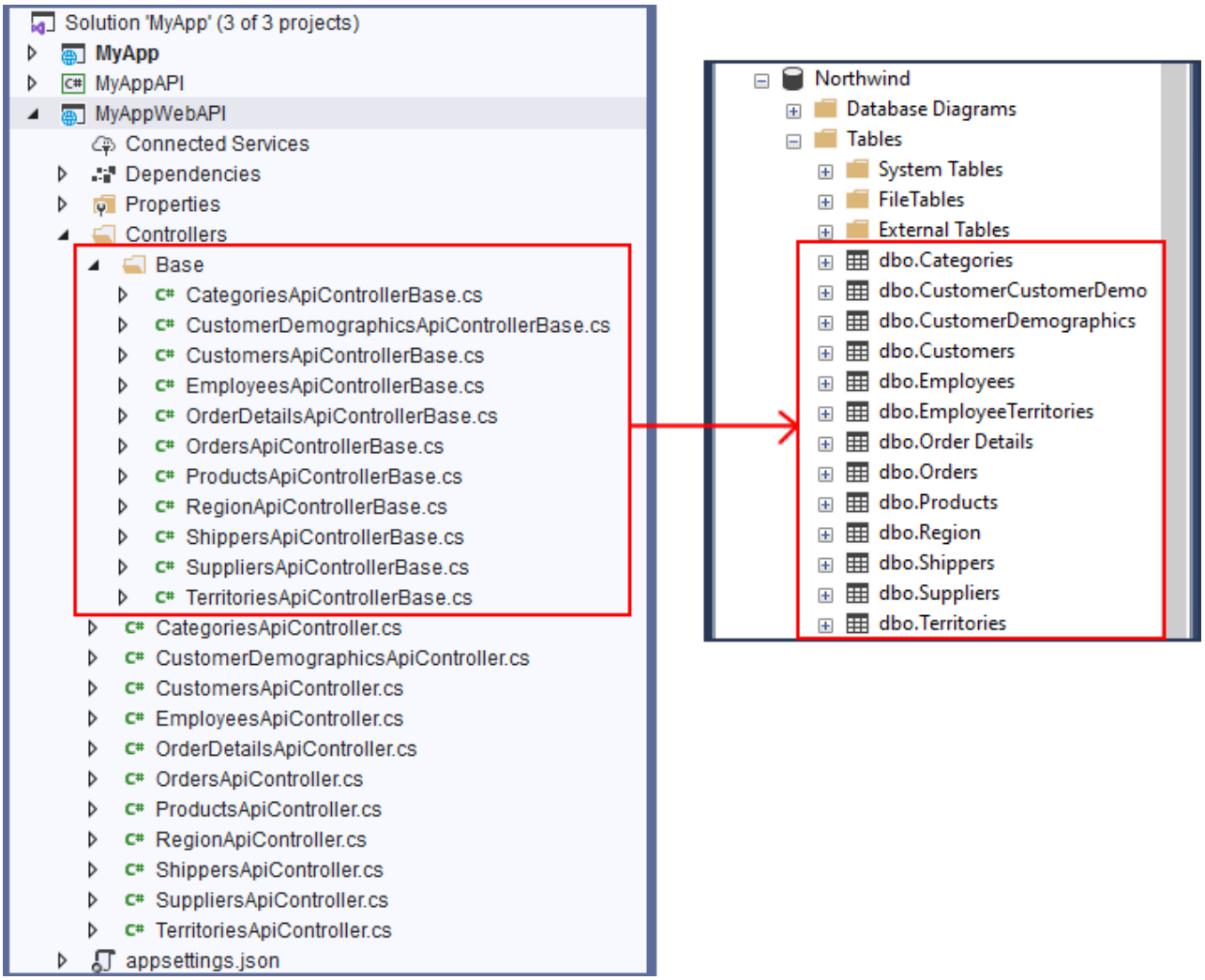
This folder is generally needed by ASP.NET Core MVC by default. It houses *Controllers* that can be used as *Web APIs*.

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.

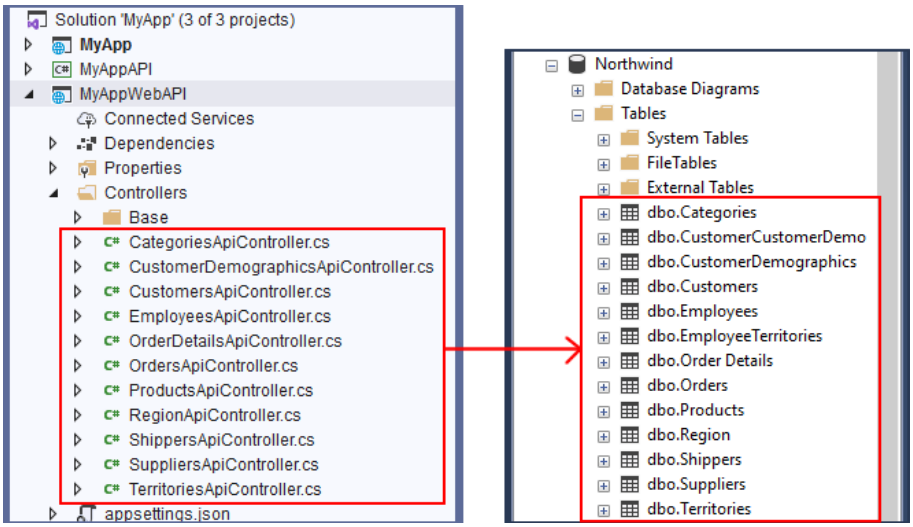


Base Controllers in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

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)

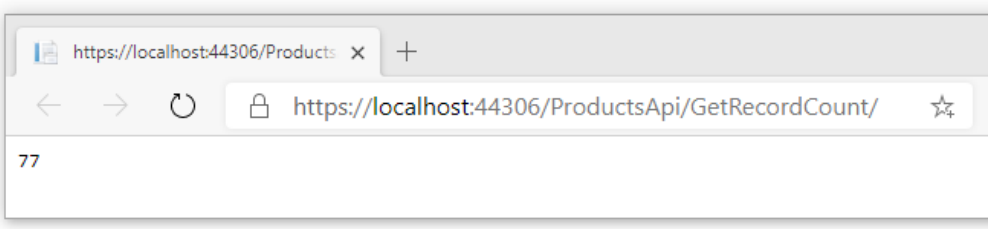
3.3.2.3 Accessing Web API Controllers

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

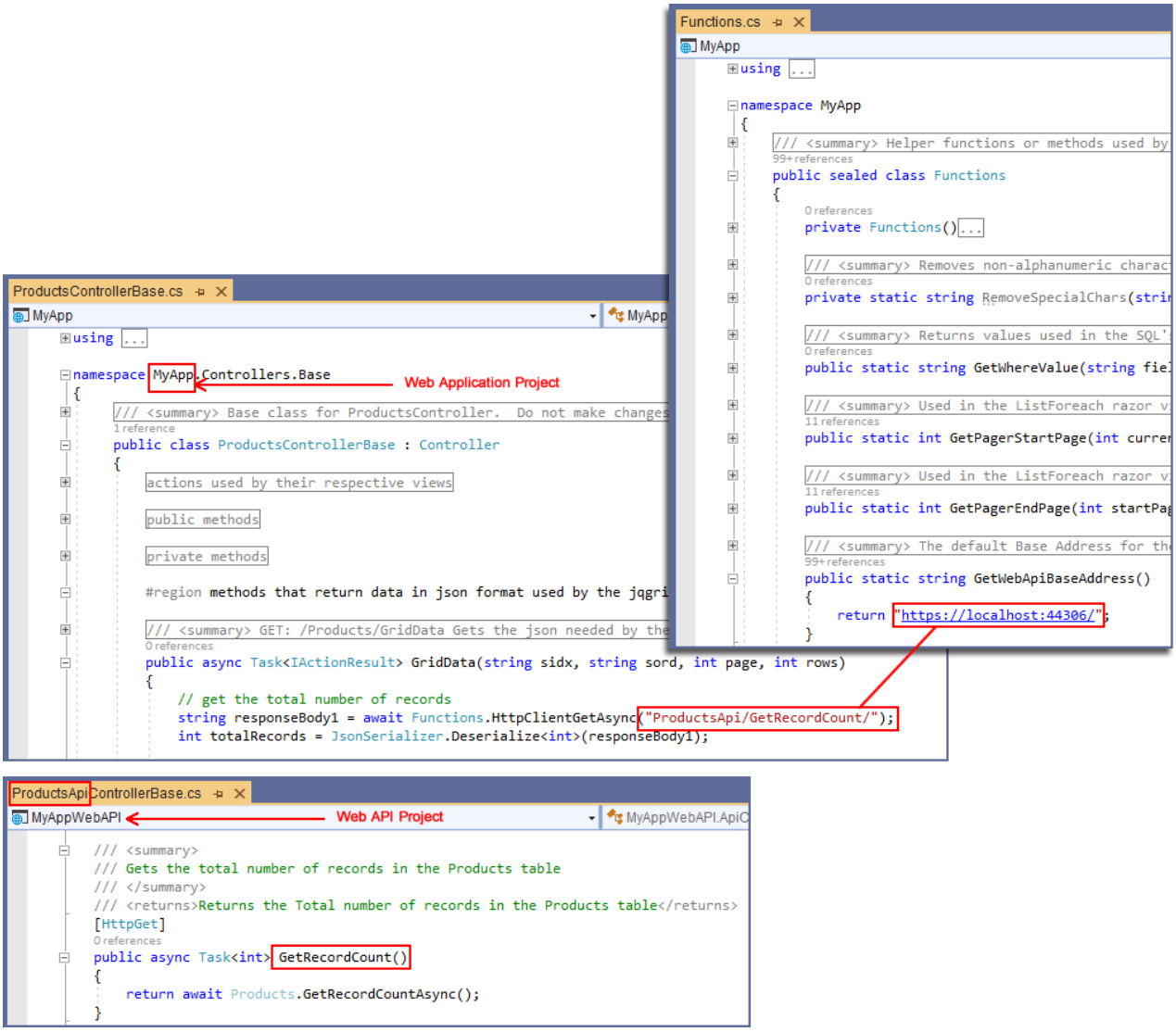
For example, we need to make an *HttpClient Get Request* call from the *Web Application Project's Razor Page Model (e.g. Products_ListForeach.cshtml.cs)* to access the *GetRecordCount() Method* in the *Web API Controller (Products****API****ControllerBase)*.

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()***)



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"*.



* 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 Razor Professional Plus that came with your purchase. These tutorials are available to customers and are included in a link on your invoice when you purchase AspCoreGen 3.0 Razor Professional.

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

End of tutorial.