

# The Generated Code for Database Tables

1	Introduction.....	3
1.1	Read these tutorials in order .....	3
1.2	Generated Code for Database Tables .....	3
1.3	Generated Projects .....	3
2	N-Tier Layering.....	4
2.1	Front End .....	4
2.2	Middle-Tier/ Middle Layer .....	4
2.3	Data-Tier/ Data Layer.....	4
2.4	SQL Scripts .....	4
3	Generated Projects .....	5
3.1	Web Application Project .....	5
3.1.1	wwwroot.....	6
1.	css (folder): .....	7
2.	images (folder):.....	7
3.	js (folder):.....	8
4.	lib (folder): .....	8
5.	favicon.ico:.....	9
3.1.2	Controllers .....	9
3.1.2.1	The Controller - Used Like A Base Class.....	10
3.1.2.1.1	Code Separated By Regions .....	10
3.1.2.1.1.1	Actions Used by Their Respective Views .....	11
3.1.2.1.1.2	Public Methods .....	12
3.1.2.1.1.3	Private Methods .....	13
3.1.2.1.1.4	Methods that Return Data in JSON Format Use by the JQuery .....	13
3.1.2.2	The Controller - Empty .....	14
3.1.2.2.1.1	HomeController .....	14
3.1.3	Helper .....	15
1.	Functions.cs: .....	15
3.1.4	Views.....	16
3.1.4.1	Views Generated for Database Tables .....	17
3.1.4.2	Partial Views for Database Tables .....	17
3.1.4.3	Other Partial Views .....	18
3.1.4.3.1	_Layout.cshtml.....	19
3.1.4.3.2	_ValidationScriptPartial.cshtml.....	19

3.1.4.3.3	_ViewImports.cshtml .....	20
3.1.4.3.4	_ViewStart.cshtml .....	20
3.1.4.4	Index.cshtml View .....	21
3.1.5	appsettings.json .....	21
3.1.6	Program.cs .....	21
3.2	Middle Layer Project (Business Layer, Data Repository, Shared Libraries) .....	22
3.2.1	Business Layer (Middle Tier) .....	22
3.2.1.1	Partial Interface/Partial Class – Used Like A Base Interface/Class .....	23
3.2.1.2	Business Layer - Empty .....	24
3.2.2	Data Repository (Data Tier) .....	24
3.2.2.1	Partial Interface/Partial Class – Used Like A Base Interface/Class .....	25
3.2.2.2	Data Repository - Empty .....	26
3.2.3	Domain .....	26
3.2.3.1	CrudOperation.cs .....	27
3.2.3.2	FieldType.cs .....	27
3.2.4	Models .....	27
3.2.4.1	Partial Class – Used Like A Base Class .....	28
3.2.4.2	Models - Empty .....	29
3.2.5	View Models .....	30
3.2.5.1	Partial Class – Used Like A Base Class .....	30
3.2.5.2	View Model - Empty .....	31
3.2.5.2.1	ListSearch.cshtml .....	32
3.2.5.2.2	ListInline.cshtml .....	34
3.2.5.2.3	ListCrud.cshtml .....	35
3.2.5.2.4	ListByForeignKey.cshtml .....	36
3.2.5.3	Foreach View Models .....	37
3.2.5.4	Partial Class – Used Like A Base Class .....	37
3.2.5.5	Foreach View Model – Empty .....	38
3.3	Web API Project (Web Services) .....	40
3.3.1	LaunchSettings.json, appsettings.json, Program.cs .....	41
3.3.2	Controllers .....	41
3.3.2.1	The Controller - Used Like A Base Class .....	41
3.3.2.2	The Controller - Empty .....	42
3.3.2.3	Accessing Web API Controllers (Methods) .....	42
3.3.3	Swagger .....	44

# The Generated Code for Database Tables

## 1 INTRODUCTION

---

This topic will walk you through AspCoreGen 6.0 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

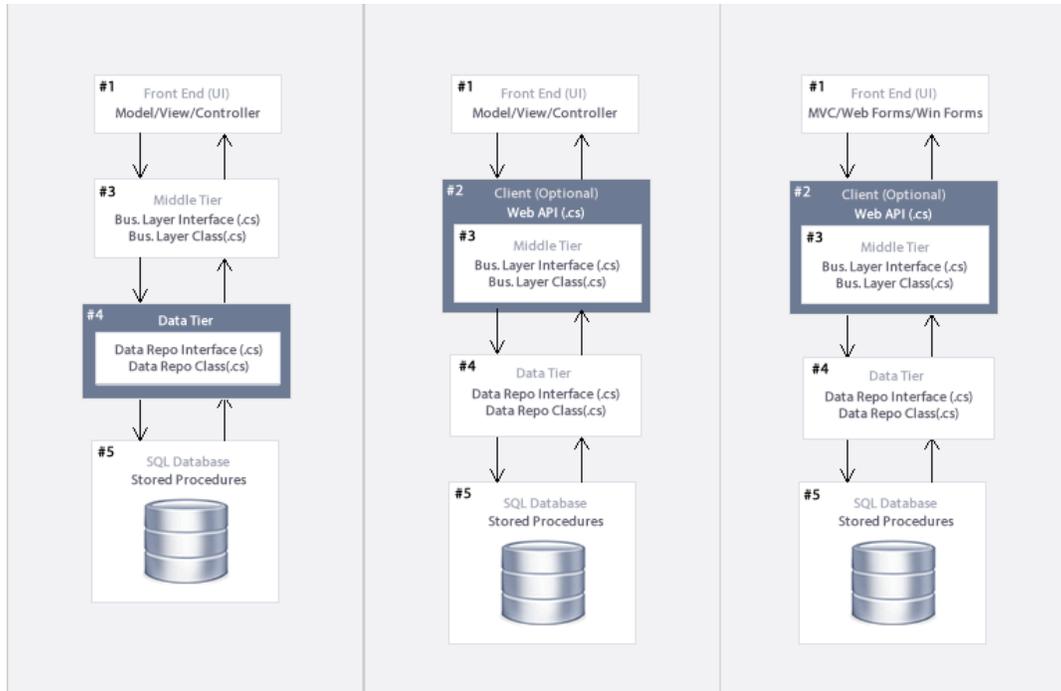
In the *App Settings Tutorial* there are 3 projects that can be generated in a solution:

- Web Application Project (User Interface)
- Middle Layer Project (Class Library Project – Business Layer, Data Repository, and Shared Libraries)
- Web API Project (Web Services - Optional)

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

## 2 N-TIER LAYERING

AspCoreGen 6.0 MVC generates code in an *n-tier* architecture. A presentation tier (the client), middle tier (business layer), data tier (data repository), and the database scripts such as stored procedures. Code are separated in different layers.



### 2.1 FRONT END

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

### 2.2 MIDDLE-TIER/ MIDDLE LAYER

1. Business Layer Interface and Class, Models, Views, View Models, etc. Or,
2. Web API (Optional). Optionally encapsulate calls to the Business Layer when generating Web API code.

### 2.3 DATA-TIER/ DATA LAYER

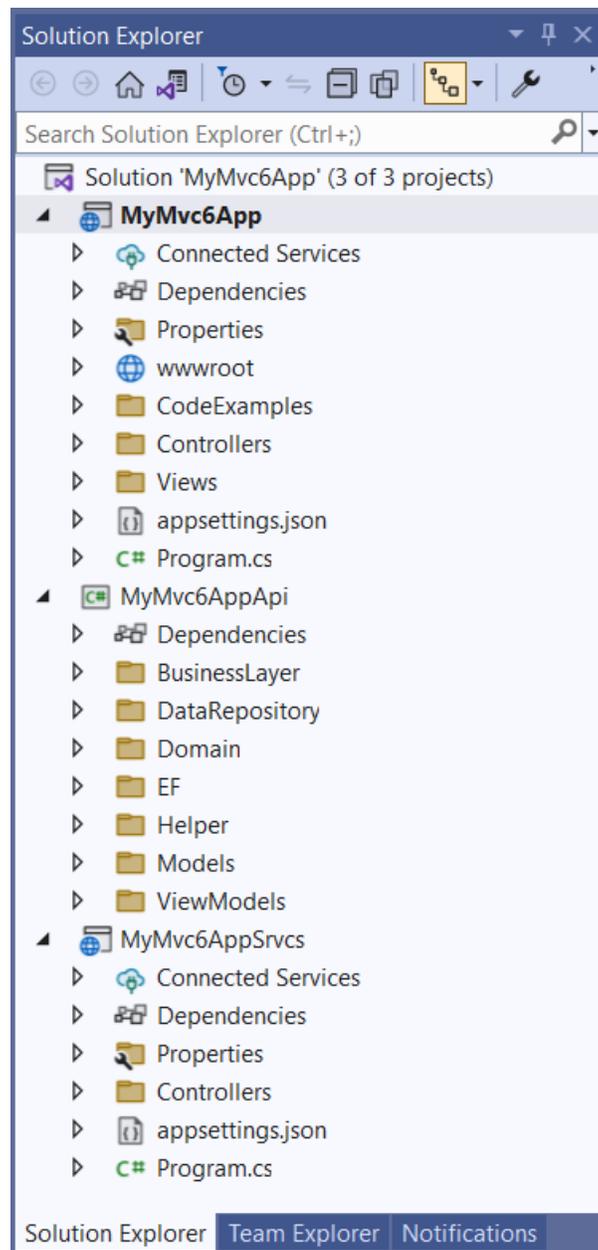
Data Repository Interface and Class, Class Files using Linq-to-Entities - Entity Framework Core or Ad-Hoc SQL.

### 2.4 SQL SCRIPTS

Stored Procedures.

## 3 GENERATED PROJECTS

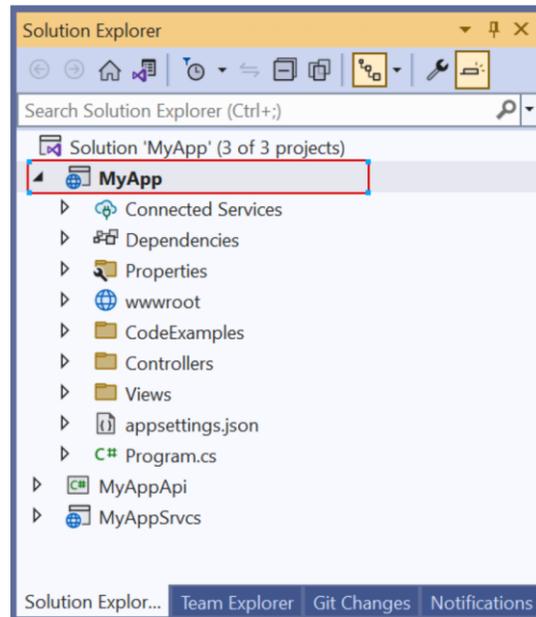
There are 3 projects that can be generated by AspCoreGen 6.0 MVC Professional Plus including the optional *Web API* project. When you chose *Stored Procedures* under the *Generated SQL Script* in the *Database Settings Tab*, these SQL scripts will be generated straight in your MS SQL Server Database's *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 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.

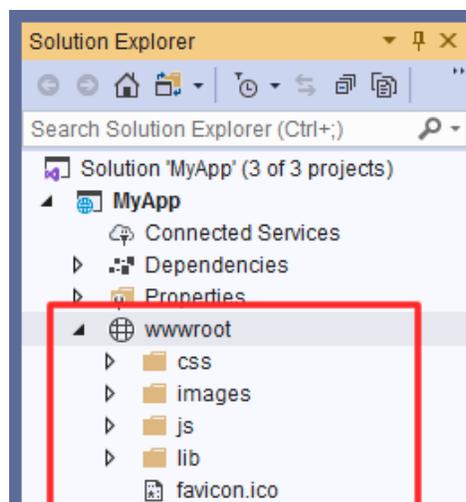


In this example, the *MyApp* project is the *Web Application Project* that was generated. Everything in this project are used to present users with an interface they can interact with, except the optional *CodeExamples* folder which contains *Class Files* for each of the database tables showing code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

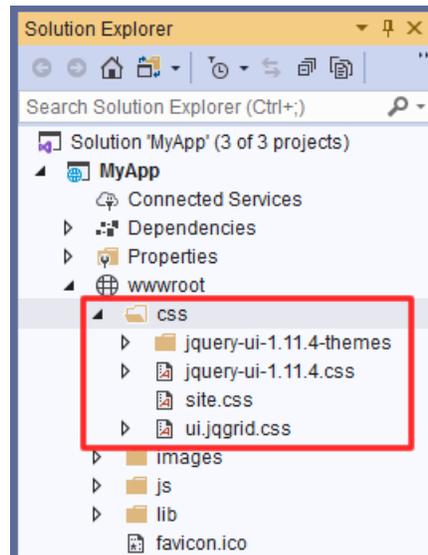
As shown in the *N-Tier Layering* above, the *Front End* (MVC view) accesses the *Middle Tier* (class) to do any kind of operation. Or it can also access the *Web API* instead of the *Middle Tier* (class).

### 3.1.1 wwwroot

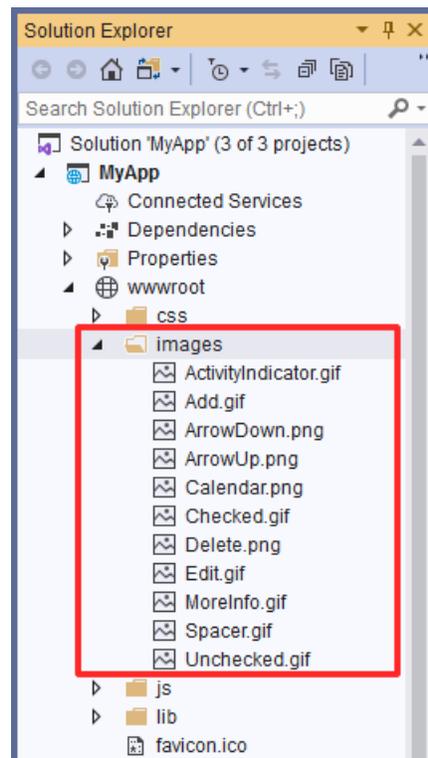
This folder is generally needed by ASP.NET Core MVC as the *Web Root* of the project by default. You can place static files needed by the ASP.NET Core MVC project here. You can add folders and files and name them to whatever you like.



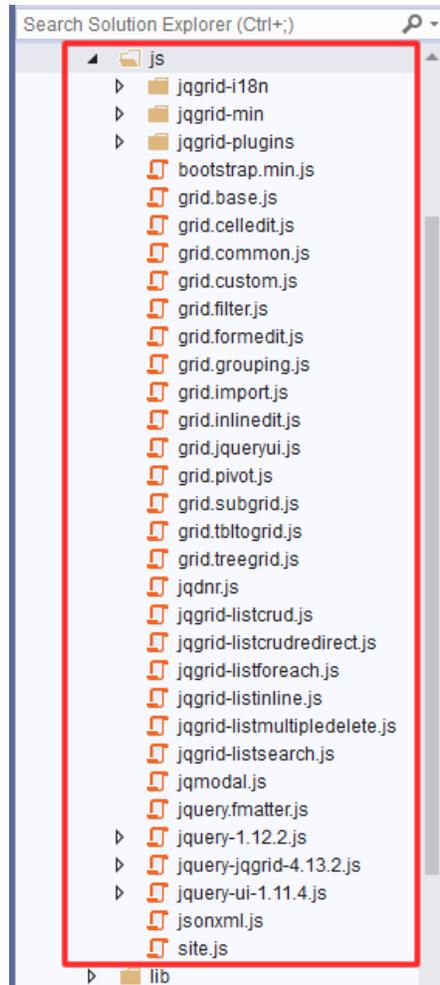
1. **css (folder):** Contains styles including 24 different JQuery-UI themes used by the project. You can add your own stylesheets here. You can also add and updates styles in the *site.css* stylesheet.



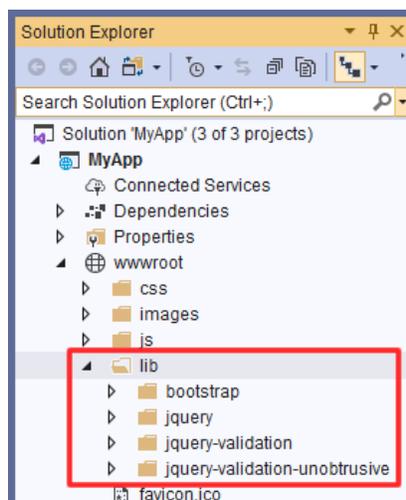
2. **images (folder):** Contains images used by the project. You can add your own images here.



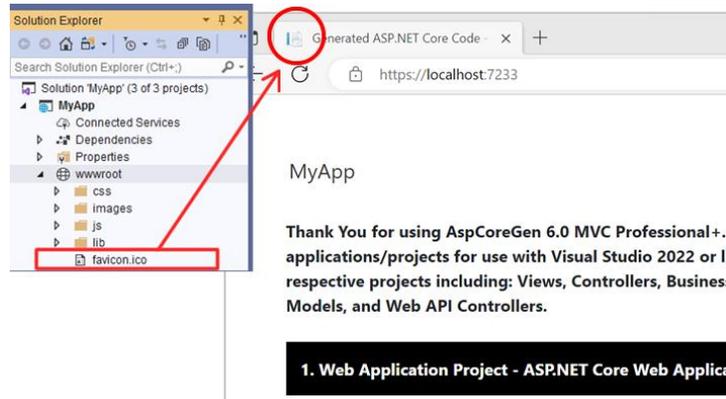
3. **js (folder):** Contains javascript files including JQGrid and JQuery plugins used by the project. You can add your own scripts here.



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 6.0 MVC to generate the code. You can add your own libraries here, however, **we recommend that you don't**.



- 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).

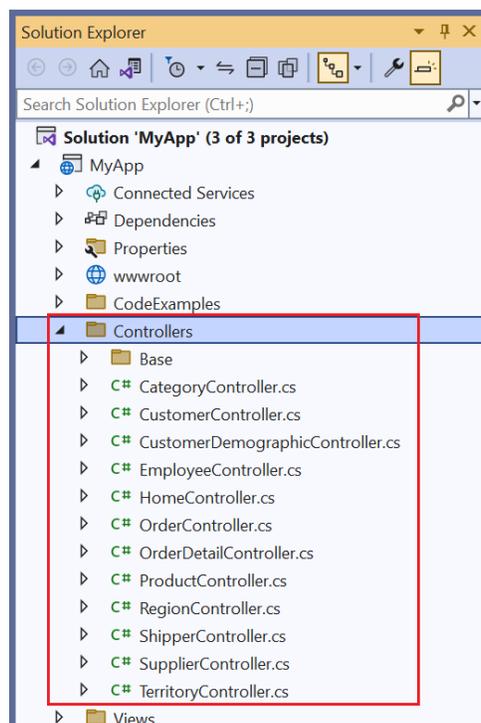


### 3.1.2 Controllers

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 6.0 MVC. There are **2 Controllers with the same name (partial class)** per database table, one directly under the Controllers folder, and the other under the Controllers/Base folder.

You can add your own *Methods* and or *Actions* in any existing *Controller* generated by AspCoreGen 6.0 MVC found directly beneath the Controller folder, these partial classes **will not be overwritten** when you regenerate 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 *Controller* generated in the Controller/Base folder.** Please see the *AppSettingsTab Tutorial*, page 4 (1.1.2 *Files That Will Always Be Overwritten*) for more information.

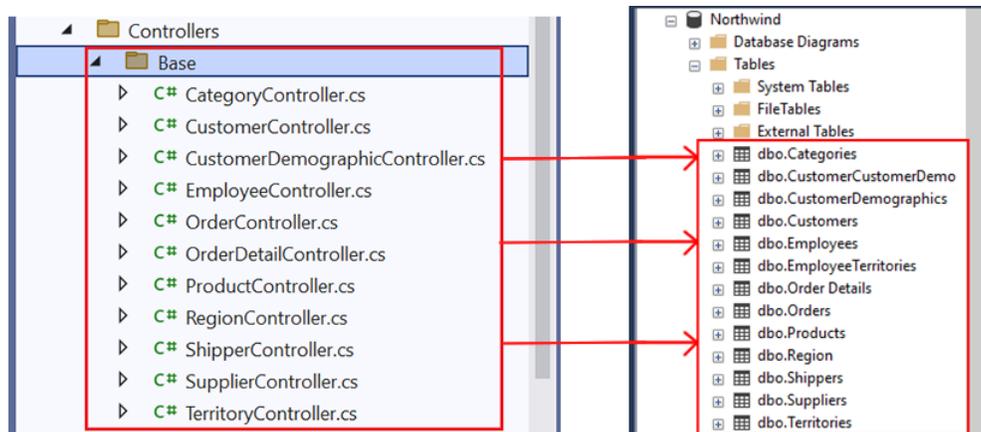


### 3.1.2.1 The Controller - Used Like A Base Class

**Note: Not a base class.** The code needed by the Controller are generated in these partial classes. These are the partial class files generated in the *Controllers\Base* folder. The naming convention used is: *TableNameController.cs*.

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

One *Partial Class* (in the *Controllers\Base* folder) is generated per *Database Table*. The example below shows that you generated code for *All Tables* for the *Northwind* database.



**Controllers (Partial Classes) 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.

```

ProductController.cs
MyApp
1 using Newtonsoft.Json;
2 using Microsoft.AspNetCore.Mvc;
3 using System;
4 using System.Linq;
5 using MyAppApi;
6 using MyAppApi.BusinessLayer;
7 using MyAppApi.Models;
8 using MyAppApi.ViewModels;
9 using MyAppApi.Domain;
10 using System.Net.Http;
11 using System.Collections.Generic;
12 using System.Threading.Tasks;
13
14 namespace MyApp.Controllers
15 {
16     /// <summary>
17     /// Works like the Base class for ProductController class.
18     /// ***** Do not make changes to this class *****
19     /// ** Put your additional code in the ProductController class under the Controller folder. **
20     /// *****
21     /// </summary>
22     public partial class ProductController : Controller
23     {
24         private Product _product;
25         private ProductViewModel _productViewModel;
26         private ProductForEachViewModel _productForEachViewModel;
27
28         // constructor
29         public ProductController(Product product, ProductViewModel productViewModel, ProductForEachViewModel productForEachViewModel)
30         {
31             _product = product;
32             _productViewModel = productViewModel;
33             _productForEachViewModel = productForEachViewModel;
34         }
35
36         actions used by their respective views
37
38         public methods
39
40         private methods
41
42         methods that return data in json format used by the jqgrid
43     }
44 }

```

### 3.1.2.1.1 Actions Used by Their Respective Views

These are the *Action* methods used by their respective MVC Views under the Views folder. The *ProductController* partial class shown below is located in the *Controller\Base* folder.

The screenshot displays the *ProductController.cs* code on the left and the *Solution Explorer* on the right. The code includes several action methods, each with a summary comment and a public method signature. The *Solution Explorer* shows the project structure, including the *Views* folder and its subfolders. Arrows indicate the mapping between the action methods and their respective view files:

- Index()* maps to *Add.cshtml*
- Add(string urlReferrer = null)* maps to *Details.cshtml*
- Add(ProductViewModel viewModel, string returnUrl)* maps to *ListByCategoryID.cshtml*
- Update(int id, string urlReferrer = null)* maps to *ListBySupplierID.cshtml*
- Update(int id, ProductViewModel viewModel, string returnUrl)* maps to *ListCrud.cshtml*
- Details(int id, string urlReferrer)* maps to *ListCrudRedirect.cshtml*
- ListCrudRedirect()* maps to *ListForEach.cshtml*
- ListReadOnly()* maps to *ListGroupedByCategoryID.cshtml*

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

For example, the **Add.cshtml** View under the **Product** folder will look for an **Add** Action method in the **ProductController**. In the same way, the **ListCrudRedirect.cshtml** View under the **Product** folder will look for a **ListCrudRedirect** Action method in the **ProductController**. 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 requires a **Get** and **Post** Action methods, while the **Details**, **ListCrudRedirect**, **ListReadOnly** MVC Views only require a **Get** Action method. Each ASP.NET Core MVC View require a **Get** Action method minimum by default.

**Note:** Since both the **ProductController** (in the **Controller\Base** folder) and the **ProductController** (directly under the **Controller** folder) are partial classes with the same name, the MVC View will look at both **ProductController(s)** to find its respective action. If you need to add new code (Actions, methods, etc) that is not generated by AspCoreGen 6.0 MVC, you can **add them in the ProductController directly under the Controller folder**.

### 3.1.2.1.2 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 **ProductController** (remember this inherits from the **ProductControllerBase**) with a **Public DeleteMultiple Method** as highlighted in the MVC View's code below: **'Product/DeleteMultiple'**. Code inside the Controller's **DeleteMultiple** method is executed and the control flow is returned back to the calling **ListMultipleDelete.cshtml** MVC View.

The image shows two code files side-by-side in Visual Studio. The left file is **ProductController.cs** and the right file is **ListMultipleDelete.cshtml**. A red arrow points from the JavaScript code in the right file to the **DeleteMultiple** method in the left file.

```

ProductController.cs
1 using ...
13
14 namespace MyApp.Controllers
15 {
16     /// <summary> Works like the Base class for ProductController class
17     2 references
18     public partial class ProductController : Controller
19     {
20         private Product _product;
21         private ProductViewModel _productViewModel;
22         private ProductForeachViewModel _productForeachViewModel;
23
24         // constructor
25         0 references
26         public ProductController(Product product, ProductViewModel prod
27
28         #region
29         #endregion
30
31         #region actions used by their respective views
32
33         #region public methods
34
35         /// <summary> POST: /Product/Delete/# Deletes a record based on
36         [HttpPost]
37         0 references
38         public async Task<IActionResult> Delete(int id) {...}
39
40         /// <summary> POST: /Product/DeleteMultiple/ids Deletes multiple
41         [HttpPost]
42         0 references
43         public async Task<IActionResult> DeleteMultiple(string ids) {...}
44
45         #endregion
46
47         #region private methods
48
49         #endregion
50
51         #region methods that return data in json format used by the jqgrid
52
53         #endregion
54     }
55 }
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
```

### 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
/// <references>
private async Task<ActionResult> GetAddViewModelAsync(string returnUrl = null) {...}

/// <summary> Gets the view model used by the Update razor view
/// <references>
private async Task<ActionResult> GetUpdateViewModelAsync(int id, string urlReferrer) {...}

/// <summary> Used when adding a new record or updating an existing record
/// <references>
private async Task<ActionResult> AddEditProductAsync(ProductViewModel viewModel, CrudOperation operation, bool isForListInline) {...}

/// <summary> Gets the view model based on the actionName
/// <references>
private async Task<ProductViewModel> GetViewModelAsync(string actionName,
string controllerName = "Product", string returnUrl = null, Product objProduct = null,
CrudOperation operation = CrudOperation.None, bool isFillSupplierDel = true, bool isFillCategoryDel = true) {...}

/// <summary> Validates the Add or Update operation and Saves the data when valid
/// <references>
private async Task<ActionResult> ValidateAndSave(CrudOperation operation, ProductViewModel productViewModel, string returnUrl, ...

/// <summary> Fills the Product model information by primary key
/// <references>
private async Task FillModelByPrimaryKeyAsync(int productId) {...}

/// <summary> Selects records as a collection (List) of MyAppApi.Models.Supplier ...
/// <references>
private async Task<List<MyAppApi.Models.Supplier>> GetSupplierDropDownListDataAsync() {...}

/// <summary> Selects records as a collection (List) of MyAppApi.Models.Category ...
/// <references>
private async Task<List<MyAppApi.Models.Category>> GetCategoryDropDownListDataAsync() {...}

/// <summary> Deserializes the modelString and then Adds a New Record or Updates ...
/// <references>
private async Task<ActionResult> ListInlineAddOrUpdate(string modelString, CrudOperation operation) {...}

/// <summary> Gets json-formatted data based on the List of Products for use by ...
/// <references>
private JsonResult GetJsonData(List<Product> objProductsList, int totalPages, int page, int totalRecords) {...}

/// <summary> Gets json-formatted data based on the List of Products for use by ...
/// <references>
private JsonResult GetJsonDataGroupedBySupplierID(List<Product> objProductsList, int totalPages, int page, int totalRecords) {...}

/// <summary> Gets json-formatted data based on the List of Products for use by ...
/// <references>
private JsonResult GetJsonDataGroupedByCategoryID(List<Product> objProductsList, int totalPages, int page, int totalRecords) {...}
#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 *ProductController*.

```
#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 ...
/// <references>
public async Task<ActionResult> GridData(string sidx, string sord, int page, int rows) {...}

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

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

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

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

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

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

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

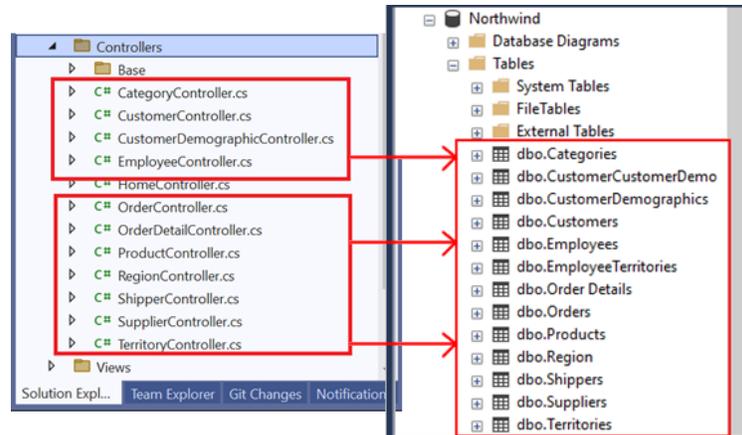
/// <summary> GET: /Products/GridDataByCategoryID Gets the json needed by the jq ...
/// <references>
public async Task<ActionResult> GridDataByCategoryID(string sidx, string sord, int page, int rows, int? categoryID) {...}
#endregion
```

```
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 jqgrid properties
    $( "#list-crud" ).jqgrid({
        url: "/Products/GridData/",
        datatype: "json",
        mtype: "GET",
        colNames: [ 'Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per

```

### 3.1.2.2 The Controller - Empty

These are the *Partial Classes* generated directly under the *Controllers* folder (not including everything inside the *Base* folder). The naming convention used is: **TableNameController.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 *Partial Class* files.

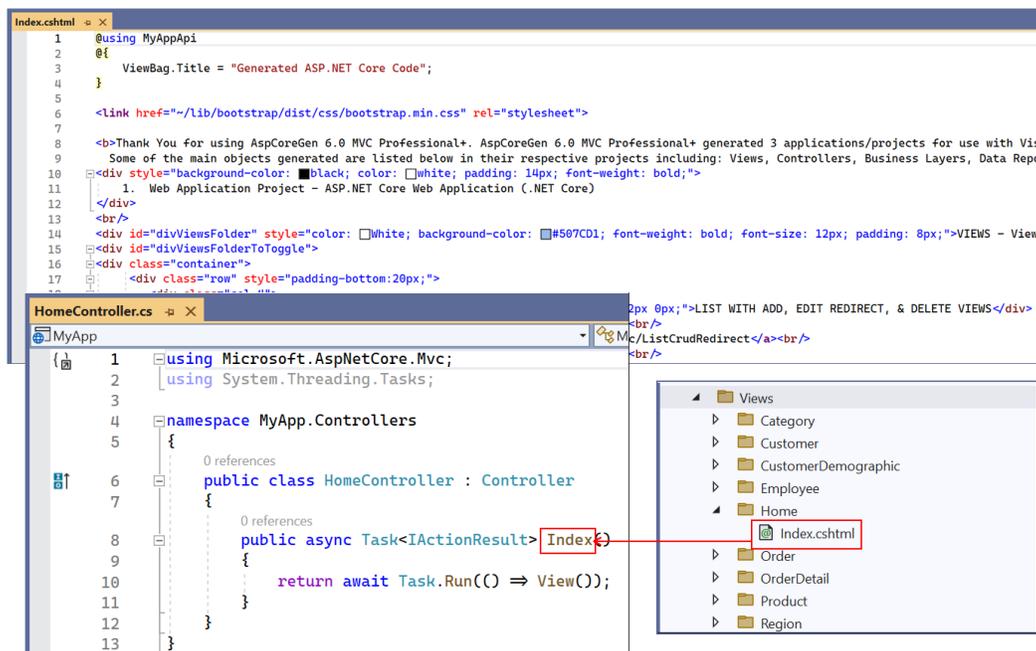


Controllers in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

#### 3.1.2.2.1.1 HomeController

The *HomeController.cs* is unlike the other *Controllers*. It 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*.



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 6.0 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 *Program* class as shown below.

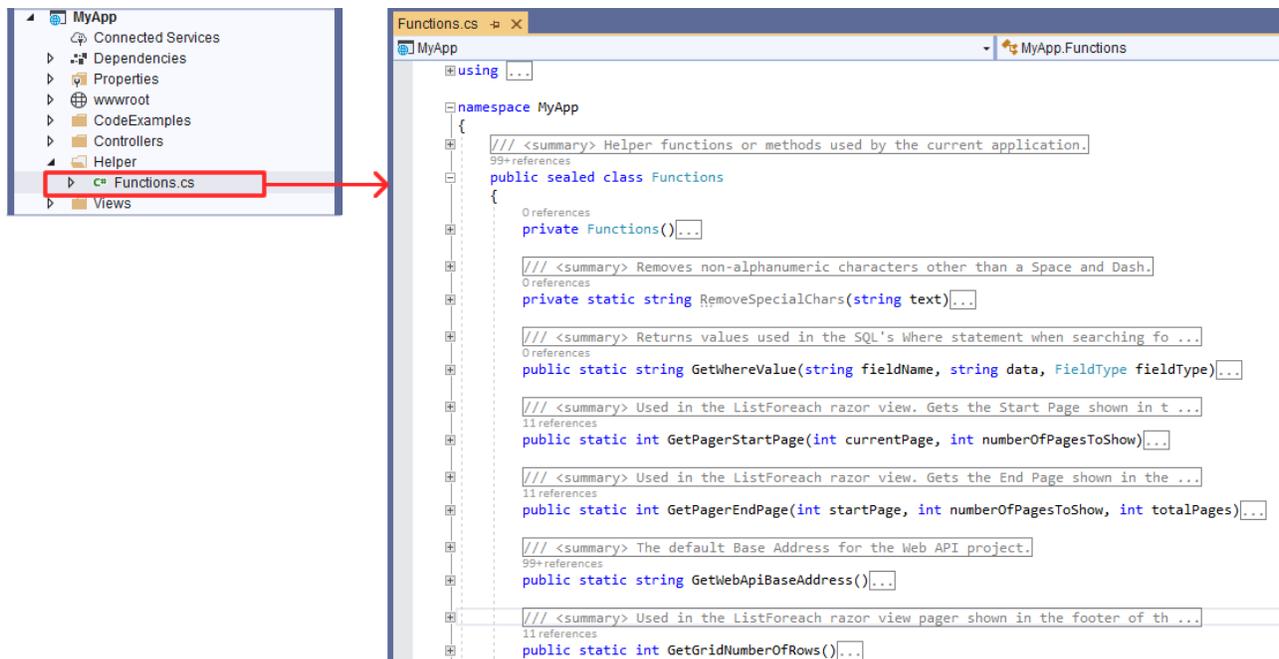
```

Program.cs
MyApp
1 using MyAppApi;
2
3 var builder = WebApplication.CreateBuilder(args);
4
5 // Add services to the container.
6 builder.Services.AddControllersWithViews();
7
8 // register services for dependency injection (di)
9 Functions.AddModelServices(builder.Services);
10 Functions.AddViewModelServices(builder.Services);
11
12 var app = builder.Build();
13
14 // Configure the HTTP request pipeline.
15 if (!app.Environment.IsDevelopment())
16 {
17     app.UseExceptionHandler("/Home/Error");
18     // The default HSTS value is 30 days. You may want to change this
19     app.UseHsts();
20 }
21
22 app.UseHttpsRedirection();
23 app.UseStaticFiles();
24
25 app.UseRouting();
26
27 app.UseAuthorization();
28
29 app.MapControllerRoute(
30     name: "default",
31     pattern: "{controller=Home}/{action=Index}/{id?}");
32
33 app.Run();
  
```

### 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.



```

MyApp
├── Connected Services
├── Dependencies
├── Properties
├── wwwroot
├── CodeExamples
├── Controllers
├── Helper
│   └── Functions.cs
└── Views
  
```

```

Functions.cs
MyApp
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.

```

Functions.cs -> x
MyApp
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) {...}
    }
}

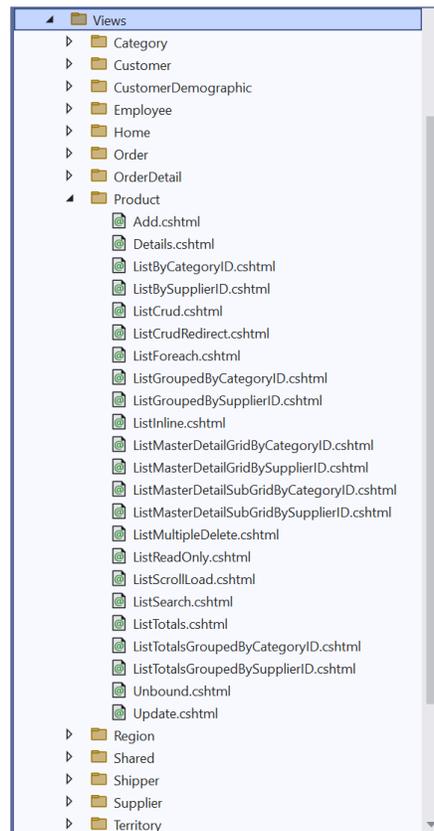
```

### 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 6.0 MVC will be overwritten the next time you generate code for the same project.

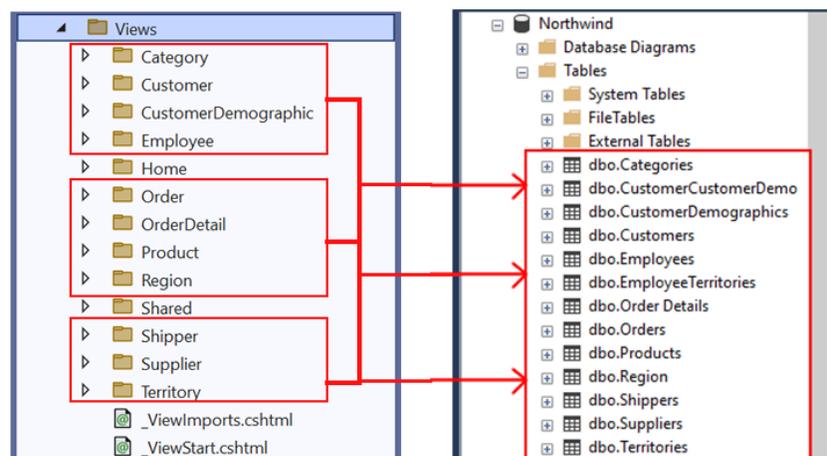
**Note: Do not add any code in any of the generated MVC Views.** Please see the *AppSettingsTab Tutorial*, page 5 (1.1.2 *Files That Will Always Be Overwritten*) for more information.

For more information on the different kinds of MVC Views generated by AspCoreGen 6.0 MVC, please see the *UISettingsTab Tutorial on Views to Generate*, starting in page 5.



### 3.1.4.1 Views Generated for Database Tables

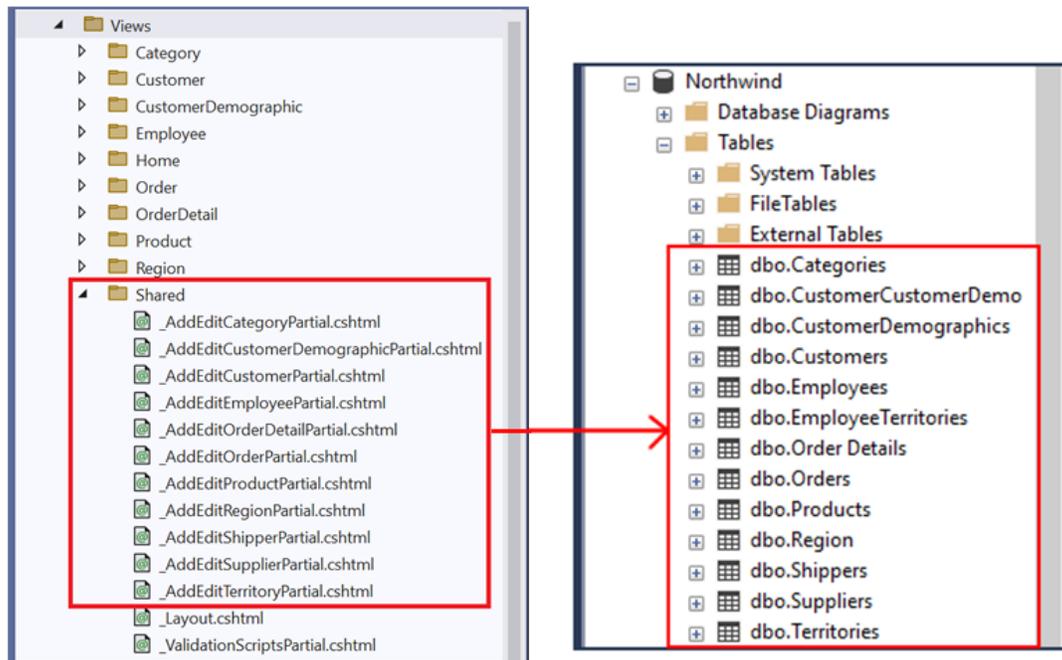
These MVC Views are generated based on the *Database Tables* you chose to generate code for. Each *Folder* as shown below is directly related to a *Database Table*.



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 chose to generate 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.



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.

```

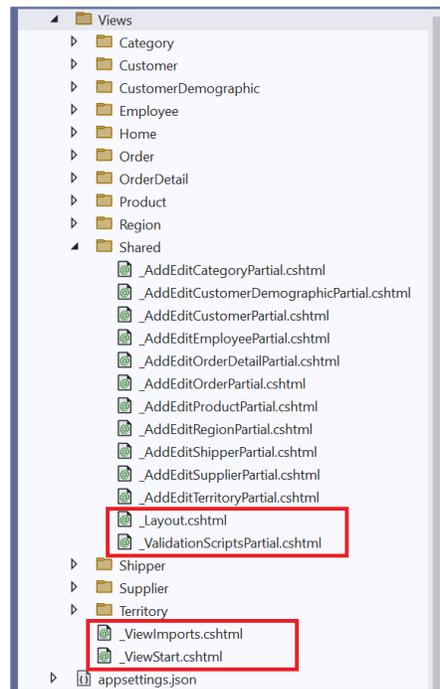
Add.cshtml
1  @section AdditionalJavaScript
2      @await Html.PartialAsync("_ValidationScriptsPartial")
3  }
4
5  <h2>Add Record</h2>
6  @Html.ValidationSummary
7  <div>
8      @await Html.PartialAsync("_AddEditProductPartial")
9  </div>
10

Update.cshtml
1  @section AdditionalJavaScript {
2      @await Html.PartialAsync("_ValidationScriptsPartial")
3  }
4
5  <h2>Update Record</h2>
6  @Html.ValidationSummary(true)
7  <div>
8      @await Html.PartialAsync("_AddEditProductPartial")
9  </div>
10

```

### 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.



### 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 incorporate 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.**

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8" />
5     <title>@ViewData["Title"] - MyApp</title>
6     <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.min.css" />
7     <link rel="stylesheet" href="/css/site.css" />
8     <link rel="stylesheet" href="/css/jquery-ui-1.11.4-themes/redmond/jquery-ui.min.css" />
9     <link rel="stylesheet" href="/css/jquery-ui-1.11.4-themes/redmond/theme.css" />
10    @RenderSection("AdditionalCss", required: false)
11  </head>
12  <body>
13    <div class="navbar navbar-inverse navbar-fixed-top">
14      <div class="container">
15        <div class="navbar-header">
16          <a asp-controller="Home" asp-action="Index" class="navbar-brand">MyApp</a>
17        </div>
18        <div class="navbar-collapse collapse">
19        </div>
20      </div>
21    </div>
22    <br />
23    <div class="container body-content">
24      @RenderBody()
25      <hr />
26      <footer>
27        <p>&copy; @DateTime.Now.Year - MyApp</p>
28      </footer>
29    </div>
30
31    <script src="/js/jquery-1.12.2.min.js"></script>
32    <script src="/lib/bootstrap/dist/js/bootstrap.min.js"></script>
33    <script src="/js/jquery-ui-1.11.4.min.js" asp-append-version="true"></script>
34    @RenderSection("AdditionalJavaScript", required: false)
35  </body>
36 </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.**

```

_VValidationScriptsPartial.cshtml - X
<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.

The image shows four overlapping code snippets from different MVC views, each demonstrating the use of the `_ValidationScriptsPartial` partial view. The snippets are:

- Unbound.cshtml**: Shows `@await Html.PartialAsync("_ValidationScriptsPartial");` within an `@section AdditionalJavaScript` block.
- Update.cshtml**: Shows `@await Html.PartialAsync("_ValidationScriptsPartial");` within an `@section AdditionalJavaScript` block.
- ListCrud.cshtml**: Shows `@await Html.PartialAsync("_ValidationScriptsPartial");` within an `@section AdditionalJavaScript` block, along with other scripts and a `ViewBag` assignment.
- Add.cshtml**: Shows `@await Html.PartialAsync("_ValidationScriptsPartial");` within an `@section AdditionalJavaScript` block, along with a `ValidationSummary` and another partial view.

### 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 - X
@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 - X
@{
  Layout = "_Layout";
}

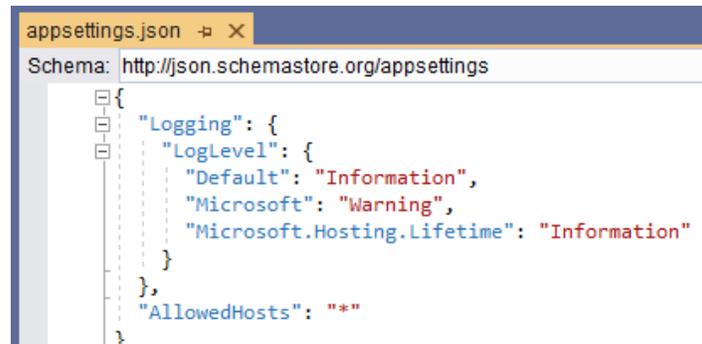
```

### 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*.

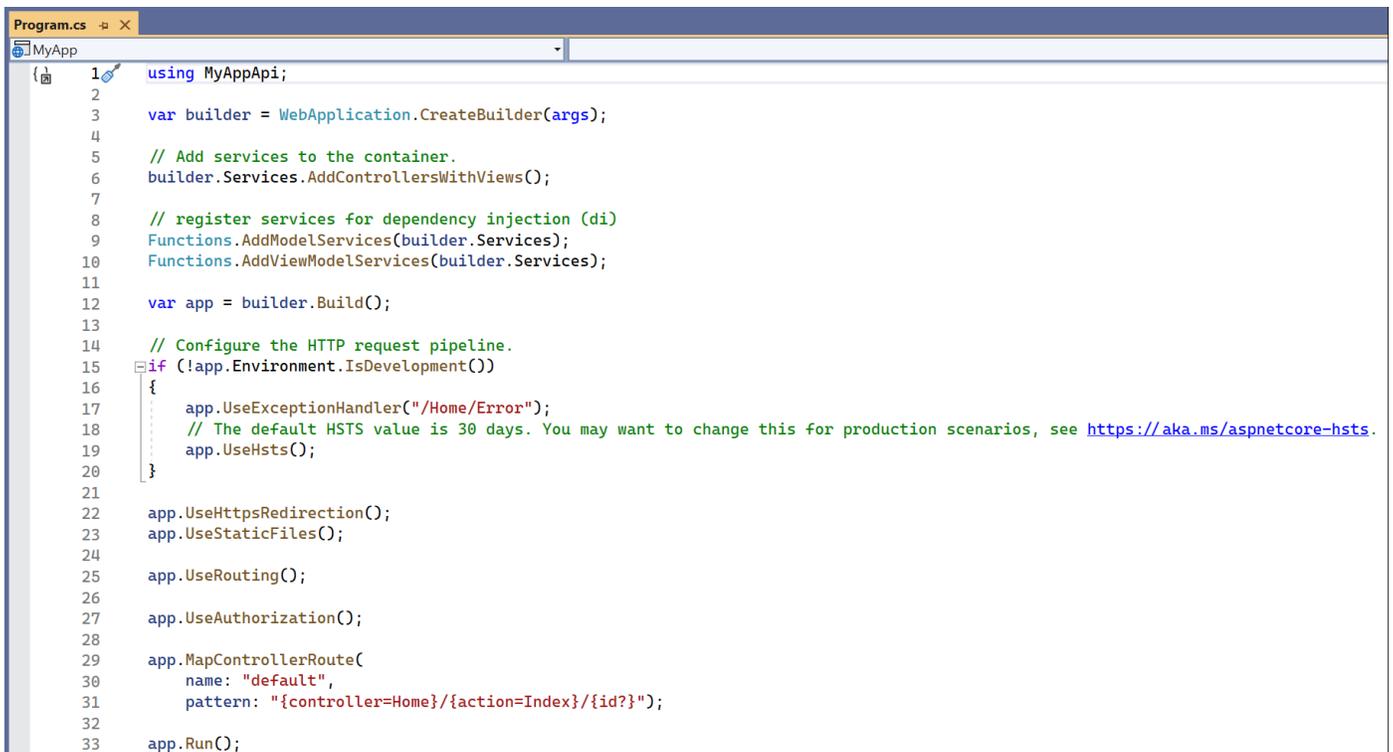
### 3.1.5 appsettings.json

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.**



### 3.1.6 Program.cs

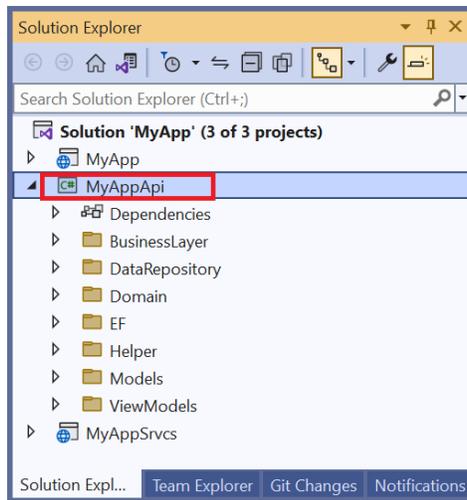
The *Program.cs Class* is the entry point to the ASP.NET MVC Core *Web Application Project* by default. An ASP.NET MVC Core web application project is technically a *Console app*. Just like any *Console app*, execution of the app starts at the *Program Class's Main() Method*. **You can add your own code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



## 3.2 MIDDLE LAYER PROJECT (BUSINESS LAYER, DATA REPOSITORY, SHARED LIBRARIES)

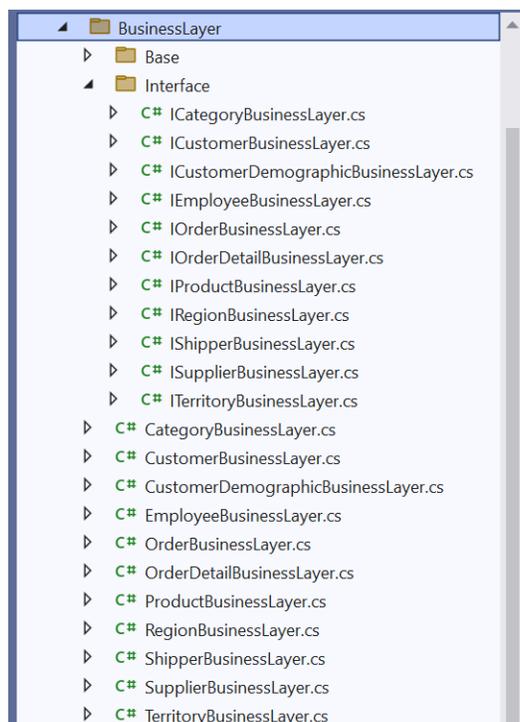
The generated *Middle Layer Project* contains the *Middle-Tier* and *Data-Tier* part of the N-tier layer generated code (and shared libraries as well). This is a *Class Library* project. **Classes/Interfaces here can be reused by other projects/clients.**

The *Middle Layer 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 Business Layer (Middle Tier)

The *Business Layer (Middle Tier) Interface and Class Files* are located in the *BusinessLayer Folder*.



The *Business Layer's* (or *Middle Tier*) main purpose is to serve as a client's (a program's) only access to the Business Layer. The *Business Layer's* purpose is to calculate things. For the purposes of AspCoreGen 6.0 MVC code generation, in most parts, there really is nothing to calculate, instead, the *Business Layer* classes simply returns data handed to it by the *Data Repository* (Data Tier), or carries and passes the CRUD\* operations that the *Data Repository* 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 *Business Layer Class's Method* should readily return that logic.

For example (just an example and not generated by AspCoreGen 6.0 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 Layer (Middle-Tier Class)*, "GetFullName()" is a *Public Method* in the "User" *Business Layer Class*. Somewhere in the "GetFullName()" *Method*, it's calculating the first name and last name of the user, return a full name, e.g.

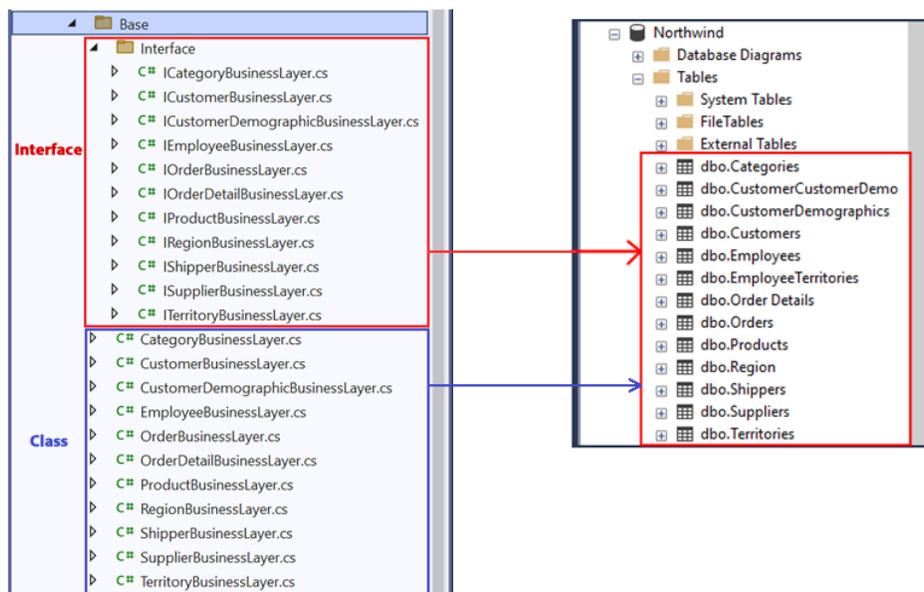
```
return User.FirstName + " " + User.LastName;
```

### 3.2.1.1 Partial Interface/Partial Class – Used Like A Base Interface/Class

These are the interface and class files generated in the *BusinessLayer\Base* folder.

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

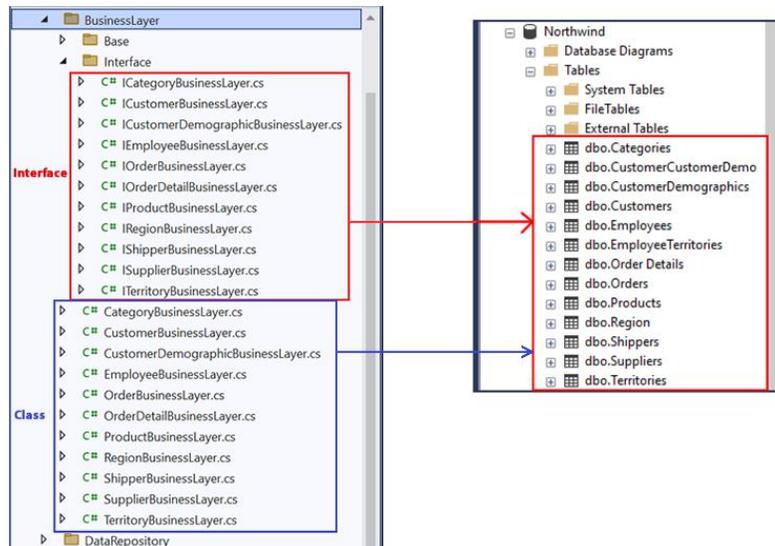
One *Partial Interface and Class* (in the Base folder) is generated per *Database Table*. The example below shows that you generated code for *All Tables* for the *Northwind* database.



Interfaces/Classes in Visual Studio under Base Folder (Left) – Database Tables in MS SQL Server (Right)

### 3.2.1.2 Business Layer - Empty

These are the interface and class files generated directly under the *BusinessLayer* folder (not including everything inside the *Base* folder). The naming convention used is: **TableNameBusinessLayer.cs**. One *Business Layer interface* and *class* is generated per *Database Table*.



Interfaces/Classes in Visual Studio directly under BusinessLayer Folder (Left) – Database Tables in MS SQL Server (Right)

You can add code in these *Interface* and *Class* files. You access all the *Business Layer* methods and properties using these *interfaces* and *classes*.

These are the *Interfaces/Classes* that **any client** should access. You can also access the *Web API Project's* public methods when you generate the optional *Web API* project.

**Note 1:** When you generate the optional *Web API Project*, AspCoreGen 6.0 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 Layer 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 6.0 MVC, you can also access the *Business Layer Classes* directly if you want to. Again, please refer to *Note 1* above.

### 3.2.2 Data Repository (Data Tier)

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

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

**Note 2:** Most of the time, a ***Data Repository Class*** should only be called by their respective ***Business Layer Class***. *Data Repository Interfaces/Classes* have an "internal" access modifier to prevent clients outside of the *Middle Layer Project* access.

**Note 3:** Since each *Data Repository Interface/Class* have an “internal” access modifier, any (*Interface/Class, Method*) code you create in the *Business Layer and Data Repository API Project* will be able to access these objects. Again, no *Interface/Class* should access a *Data Repository Interface/Class* other than a *Business Layer Interface/Class*, please see *Note 1*.

These *Data Repository (Data Tier) Interface/Class Files* are located in the *DataRepository Folder*.

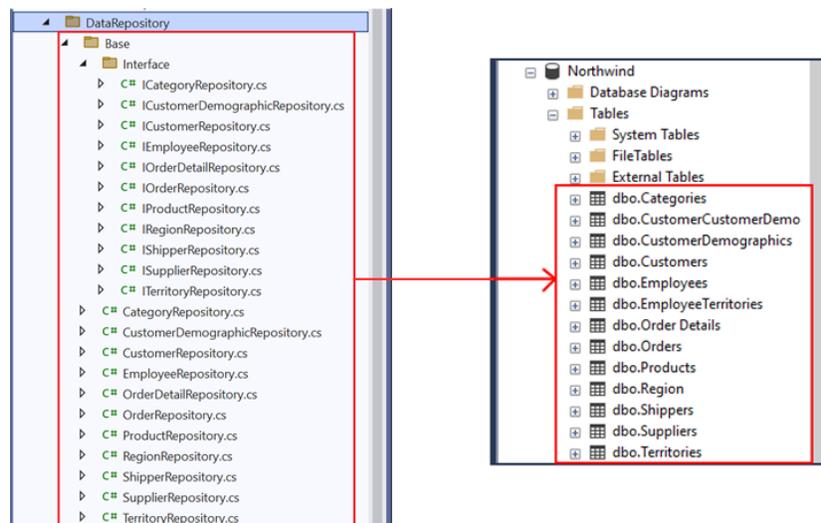


### 3.2.2.1 Partial Interface/Partial Class – Used Like A Base Interface/Class

These are the interface and class files generated in the *DataRepository\Base* folder.

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

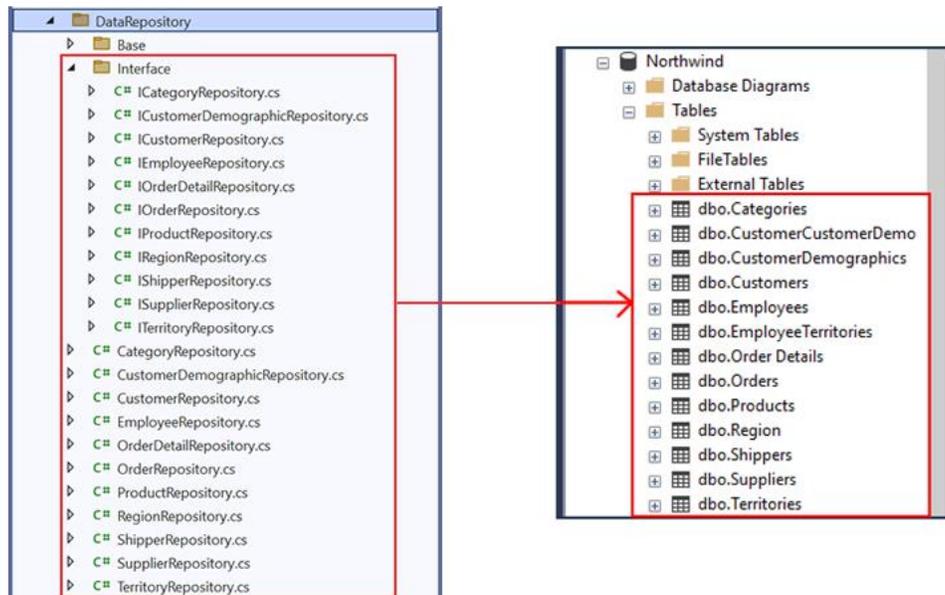
One *Partial Interface and Class* (in the *Base* folder) is generated per *Database Table*. The example below shows that you generated code for *All Tables* for the *Northwind* database.



**Interfaces/Classes in Visual Studio under Base Folder (Left) – Database Tables in MS SQL Server (Right)**

### 3.2.2.2 Data Repository - Empty

These are the interface and class files generated directly under the *DataRepository* folder (not including everything inside the *Base* folder). The naming convention used is: **TableNameDataRepository.cs**. One *Data Repository* interface and class is generated per *Database Table*.



Interfaces/Classes in Visual Studio directly under DataRepository Folder (Left) – Database Tables in MS SQL Server (Right)

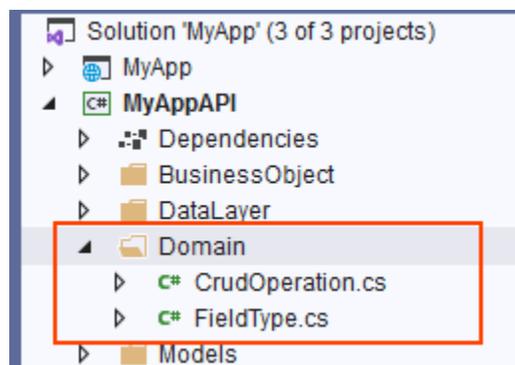
You can add code in these *Interface/Class* files. You access all the *Data Repository* methods and properties using this *Interface/Class*.

These are the *Interfaces/Classes* that *Business Layer Interfaces/Classes* should access.

**Note 1:** Only a *Business Layer Interface/Class* should access their respective *Data Repository 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. When not doing an *Add* or *Update* operation, use *None*.

```

1 namespace MyAppApi.Domain
2 {
3     /// <summary>
4     /// Enum for Add or Update (CRUD) operation.
5     /// ***** Do not make changes to this enum *****
6     /// </summary>
7     /// </summary>
8     public enum CrudOperation
9     {
10        /// <summary>
11        /// Add, insert, or create a new record
12        /// </summary>
13        Add,
14
15        /// <summary>
16        /// Update an existing record
17        /// </summary>
18        Update,
19
20        /// <summary>
21        /// Not an Add or Update operation
22        /// </summary>
23        None,
24    }
25 }

```

### 3.2.3.2 FieldType.cs

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

```

1 namespace MyAppAPI.Domain
2 {
3     6 references
4     public enum FieldType
5     {
6         Default,
7         String,
8         Date,
9         Boolean,
10        Numeric,
11        Decimal,
12    }
13 }

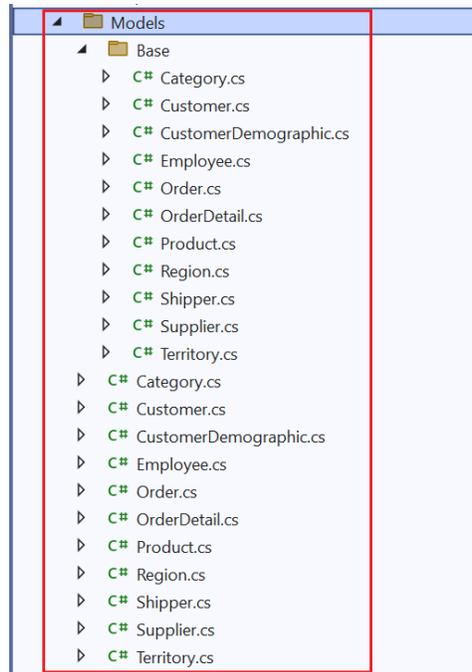
```

### 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 their respective *Database Table*. *Models* is the "M" in MVC. Sometimes *Models* are misinterpreted as the *Data Tier* part of MVC, in this case, it is not.

So why are *Models* generated in the *Middle Layer 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**.

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

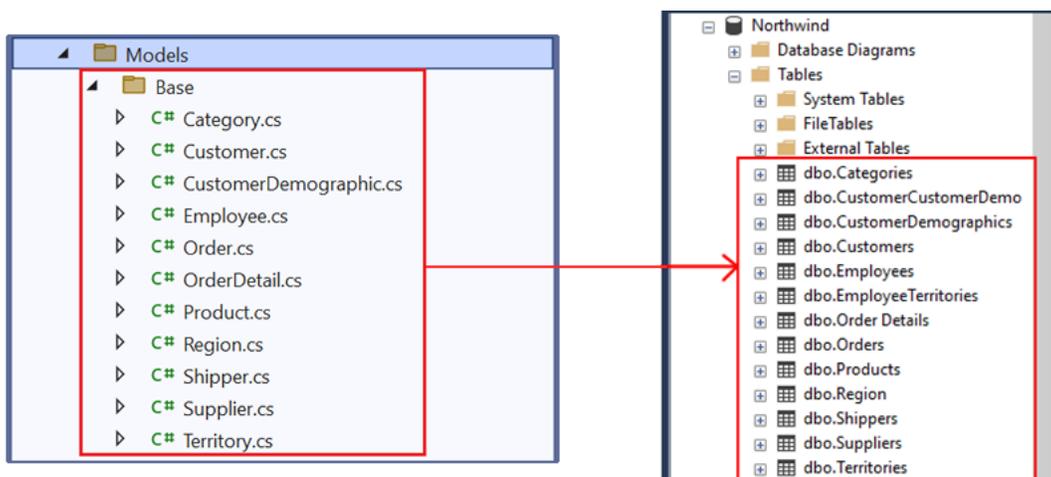


### 3.2.4.1 Partial Class – Used Like A Base Class

These are the class files generated in the *Models\Base* folder.

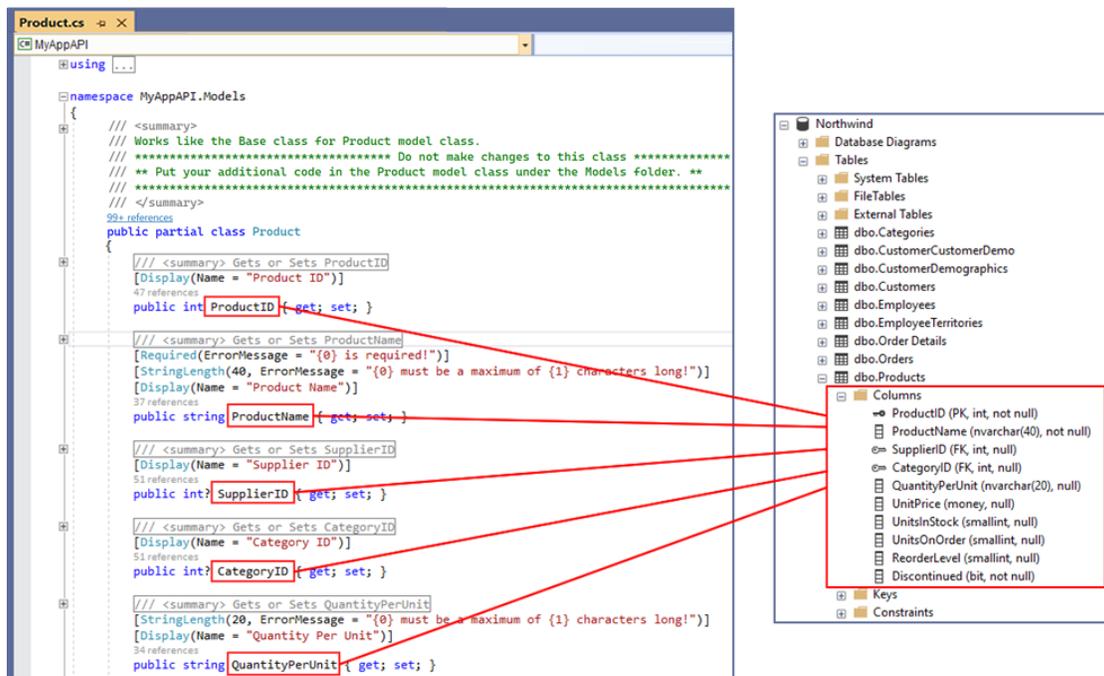
**Do not add any code in these *Class* files.**

One *Partial Class* (in the Base folder) is generated per *Database Table*. The example below shows that you generated code for *All Tables* for the *Northwind* database.



**Classes in Visual Studio under Base Folder (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*.

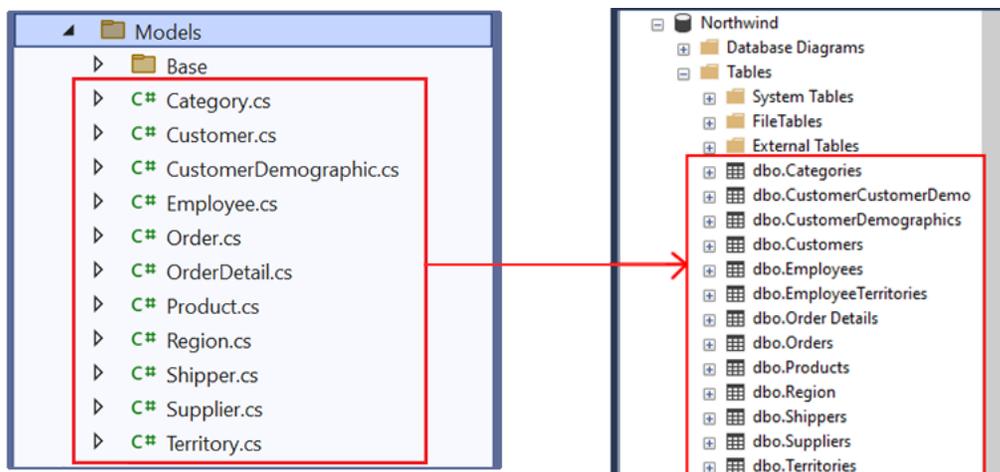


Partial Model Class in Visual Studio under Base Folder (Left) – Product Database Table Columns in MS SQL Server (Right)

### 3.2.4.2 Models - Empty

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

You can add code in these *Class* files.



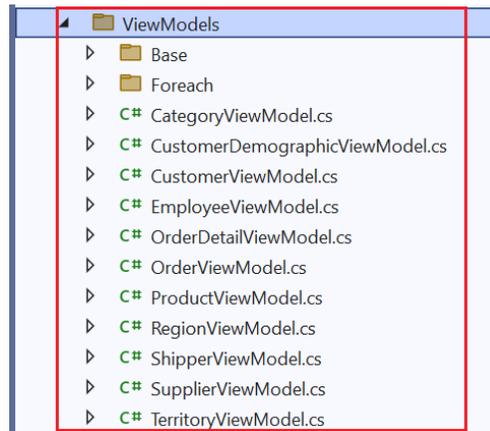
Classes in Visual Studio directly under Models Folder (Left) – Database Tables in MS SQL Server (Right)

### 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 *Middle Layer Project* instead of the *Web Application Project* where the MVC *Views* and *Controllers* are generated in? Simple, ***ViewModels* are reusable.**

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

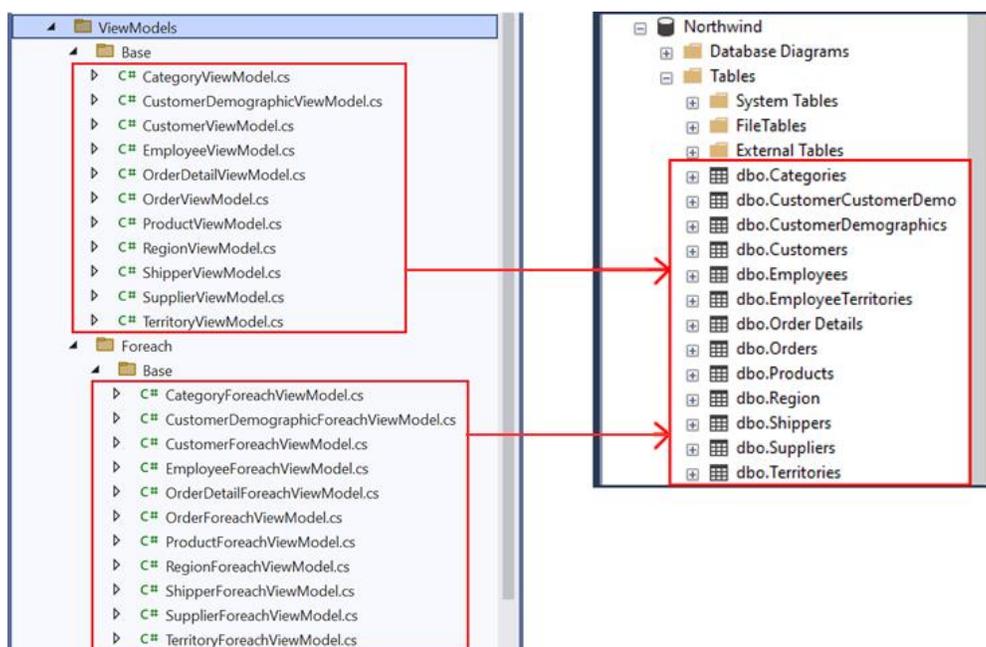


#### 3.2.5.1 Partial Class – Used Like A Base Class

These are the class files generated in the *ViewModels\Base* folder.

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

One *Partial Class* (in the *Base* folder) is generated per *Database Table*. The example below shows that you generated code for *All Tables* for the *Northwind* database.



**Partial View Model Class in Visual Studio under Base Folder (Left) – Product Database Table Columns in MS SQL Server (Right)**

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

```

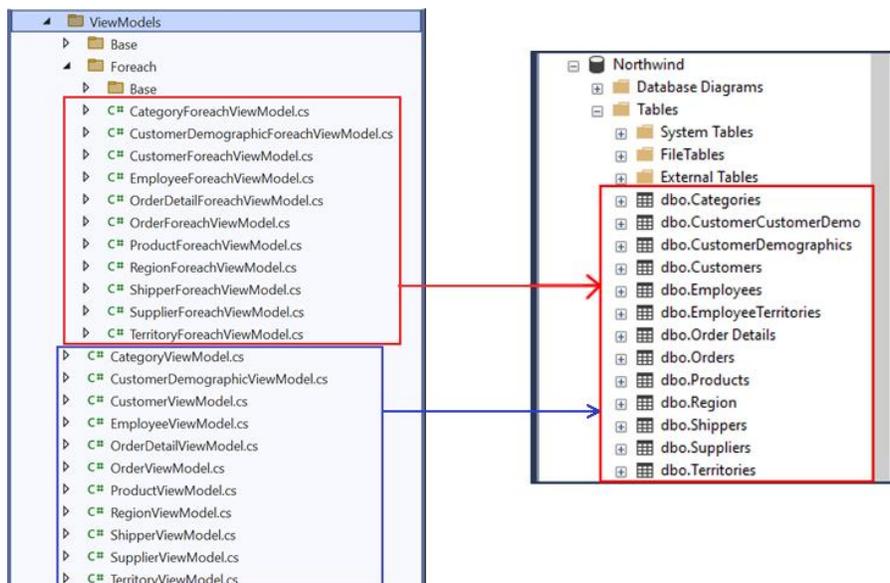
ProductViewModel.cs
MyAppApi
using ...
namespace MyAppApi.ViewModels
{
    /// <summary>
    /// Works like the Base class for ProductViewModel.
    /// ***** Do not make changes to this class *****
    /// ** Put your additional code in the ProductViewModel class under the ViewModel folder. **
    /// *****
    /// </summary>
    34 references
    public partial class ProductViewModel
    {
        15 /// <summary> The model used by the MVC view
        99+ references
        18 public MyAppApi.Models.Product Product { get; set; }
        20 /// <summary> Add new record or Update existing record
        3 references
        23 public CrudOperation Operation { get; set; }
        24
        25 /// <summary> Controller Name used by the MVC view
        3 references
        28 public string ViewControllerName { get; set; }
        29
        30 /// <summary> Action Name used by the MVC view
        3 references
        33 public string ViewActionName { get; set; }
        34
        35 /// <summary> URL where the current MVC view redirects to after the operation.
        8 references
        38 public string ViewReturnUrl { get; set; }
        39
        40 /// <summary> Data used by the Supplier drop down list control
        13 references
        43 public List<Models.Supplier> SupplierDropDownListData { get; set; }
        44
        45 /// <summary> Data used by the Category drop down list control
        13 references
        48 public List<Models.Category> CategoryDropDownListData { get; set; }
        49
    }
    50

```

### 3.2.5.2 View Model - Empty

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

You can add code in these *Class* files.



Classes in Visual Studio directly under ViewModels Folder (Left) – Database Tables in MS SQL Server (Right)

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

1. *ListSearch.cshtml*
2. *ListInline.cshtml*
3. *ListCrud.cshtml*
4. *ListByForeignKey.cshtml*

```

ListSearch.cshtml
@model MyAppAPI.ViewModels.ProductViewModel
@{
    ViewBag.Title = "List of Products";

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

ListInline.cshtml
@using System.Text.RegularExpressions;
@model MyAppAPI.ViewModels.ProductViewModel
@{
    ViewBag.Title = "List of Products";
}

ListByCategoryID.cshtml
@model MyAppAPI.ViewModels.ProductViewModel
@{
    ViewBag.Title = "List of Products By Categories";
}
@section AdditionalCss
{
    <link rel="stylesheet" href="~/css/ui.jqgrid.min.css" />
}
@section AdditionalJava
{
    <script src="~/js/jqgrid.min.js" />
    <script src="~/js/jqgrid.locale.js" />
    <script type="text/javascript">
        // formats the SupplierID in the jqgrid as a hyperlink
        // 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>";
        }
    </script>
}

ListCrud.cshtml
@model MyAppAPI.ViewModels.ProductViewModel
@{
    ViewBag.Title = "List of Products";
}
@section AdditionalCss {
    <link rel="stylesheet" href="~/css/ui.jqgrid.min.css" />
}

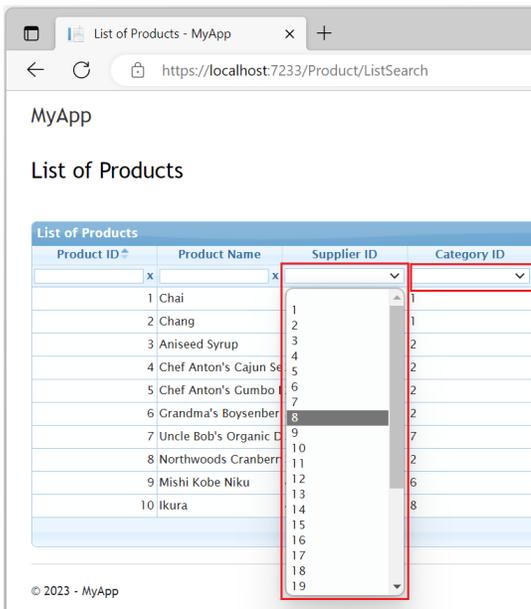
```

**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.ProductForeachViewModel
```

### 3.2.5.2.1 ListSearch.cshtml

This MVC View uses the *ProductViewModel* as its Model. It uses the MVC View’s Model (*ProductViewModel*) to fill the *Select* tags for the *SupplierID* and *CategoryID* using the MVC View’s Model, the *SupplierDropDownListData* and *CategoryDropDownListData* respectively, these are *Properties* of the *ProductViewModel* as shown in the code example in page 32.



```

1 @model MyAppApi.ViewModels.ProductViewModel
2 @{}
3 ViewBag.Title = "List of Products";
4
5 string supplierIDSelectValues = ":";
6 string categoryIDSelectValues = ":";
7
8 // set the data that's going to be used by the SupplierID select tag in the jqgrid.
9 if (Model.SupplierDropDownListData != null)
10 {
11     foreach (var item in Model.SupplierDropDownListData, OrderBy(s => s.SupplierID))
12     {
13         supplierIDSelectValues += ";" + item.SupplierID + ":" + item.SupplierID;
14     }
15 }
16
17 // set the data that's going to be used by the CategoryID select tag in the jqgrid.
18 if (Model.CategoryDropDownListData != null)
19 {
20     foreach (var item in Model.CategoryDropDownListData, OrderBy(c => c.CategoryID))
21     {
22         categoryIDSelectValues += ";" + item.CategoryID + ":" + item.CategoryID;
23     }
24 }
25

```

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.

```

/// <summary>
/// GET: /Product/ListSearch/
/// Gets the view model used by the ListSearch razor view
/// </summary>
0 references
public async Task<IActionResult> ListSearch() ——— Action
{
    // return view model
    _productViewModel = await this.GetViewModelAsync("ListSearch");
    return View(_productViewModel); ——— Injecting the ViewModel to the MVC View
}

/// <summary>
/// Gets the view model based on the actionName
/// </summary>
8 references
private async Task<ProductViewModel> GetViewModelAsync(string actionName,
string controllerName = "Product", string returnUrl = null, Product objProduct = null,
CrudOperation operation = CrudOperation.None, bool isFillSupplierDdl = true, bool isFillCategoryDdl = true)
{
    // assign values to the view model
    _productViewModel.Product = objProduct;
    _productViewModel.Operation = operation;
    _productViewModel.ViewControllerName = controllerName;
    _productViewModel.ViewActionName = actionName;
    _productViewModel.ViewReturnUrl = returnUrl;

    if (isFillSupplierDdl)
        _productViewModel.SupplierDropDownListData = await this.GetSupplierDropDownListDataAsync();
    else
        _productViewModel.SupplierDropDownListData = null;

    if (isFillCategoryDdl)
        _productViewModel.CategoryDropDownListData = await this.GetCategoryDropDownListDataAsync();
    else
        _productViewModel.CategoryDropDownListData = null;

    // return the view model
    return _productViewModel;
}

```

### 3.2.5.2.2 ListInline.cshtml

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

The *ListInline.cshtml* uses the MVC View's *Model* (*ProductViewModel*) to fill the *Select* tags for the *SupplierID* and *CategoryID* in the dialog shown below using the MVC View's *Model*, the *SupplierDropDownListData* and *CategoryDropDownListData* respectively, these are *Properties* of the *ProductViewModel* as shown in 32.

Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit	Unit Price
1	Chai	1 - Exotic Liquids		0 boxes x 20 bags	\$18.00
2	Chang	1 - Exotic Liquids	1 - Beverages	4 - 12 oz bottles	\$19.00
3	Aniseed Syrup	1 - Exotic Liquids	2 - Condiments	2 - 550 ml bottles	\$10.00
4	Chef Anton's Cajun S	2 - New Orleans Cajun	3 - Confections	8 - 6 oz jars	\$22.00
5	Chef Anton's Gumbo	2 - New Orleans Cajun	4 - Dairy Products	6 boxes	\$21.35
6	Grandma's Boysenbe	3 - Grandma Kellys H	5 - GrainsCereals	2 - 8 oz jars	\$25.00
7	Uncle Bob's Organic	3 - Grandma Kellys H	6 - MeatPoultry	12 - 1 lb pkgs.	\$30.00
8	Northwoods Cranber	3 - Grandma Kellys H	7 - Produce	12 - 12 oz jars	\$40.00

```

ListInline.cshtml
1  @using System.Text.RegularExpressions;
2  @model MyAppApi.ViewModels.ProductViewModel
3  @{
4      ViewBag.Title = "List of Products";
5
6      string supplierIDSelectValues = ":";
7      string categoryIDSelectValues = ":";
8
9      // set the data that's going to be used by the SupplierID select tag in the jqgrid.
10     if (Model.SupplierDropDownListData != null)
11     {
12         foreach (var item in Model.SupplierDropDownListData.OrderBy(s => s.SupplierID))
13         {
14             supplierIDSelectValues += ";" + item.SupplierID + ":" + item.SupplierID + " - " + Regex.Replace
15         }
16     }
17
18     // set the data that's going to be used by the CategoryID select tag in the jqgrid.
19     if (Model.CategoryDropDownListData != null)
20     {
21         foreach (var item in Model.CategoryDropDownListData.OrderBy(c => c.CategoryID))
22         {
23             categoryIDSelectValues += ";" + item.CategoryID + ":" + item.CategoryID + " - " + Regex.Replace
24         }
25     }
26 }

```

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

```

public async Task<IActionResult> ListInline() ——— Action
{
    // return view model
    _productViewModel = await this.GetViewModelAsync("ListInline");
    return View(_productViewModel); ——— Injecting the ViewModel to the MVC View
}

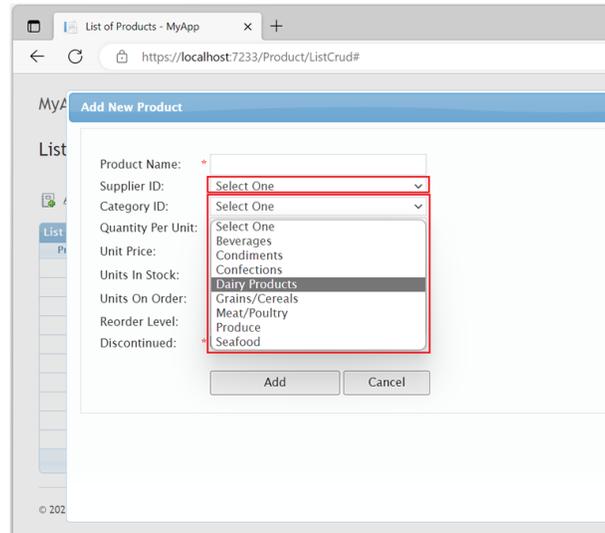
```

### 3.2.5.2.3 ListCrud.cshtml

This MVC *View* uses the *ProductViewModel* 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 (ProductViewModel)* 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 *ProductViewModel* as shown in the *ProductViewModelBase* code example in page 30.



```

225 <tr>
226 <td class="editor-label"><label asp-for="Product.SupplierID"></label></td>
227 <td></td>
228 <td class="editor-field">
229     @if (Model.SupplierDropDownListData != null)
230     {
231         <select id="supplierID" asp-for="Product.SupplierID" asp-items="@((new SelectList(Model.SupplierDropDownListData, "SupplierID", "CompanyName")))><option value="">Select One</option></select>
232     }
233     else
234     {
235         <select id="supplierID"><option value="">Select One</option></select>
236     }
237 </td>
238 <td class="editor-field"><span id="supplierIDValidation" style="color: red;"></span></td>
239 </tr>
240 <tr>
241 <td class="editor-label"><label asp-for="Product.CategoryID"></label></td>
242 <td></td>
243 <td class="editor-field">
244     @if (Model.CategoryDropDownListData != null)
245     {
246         <select id="categoryID" asp-for="Product.CategoryID" asp-items="@((new SelectList(Model.CategoryDropDownListData, "CategoryID", "CategoryName")))><option value="">Select One</option></select>
247     }
248     else
249     {
250         <select id="categoryID"><option value="">Select One</option></select>
251     }
252 </td>
253 <td class="editor-field"><span id="categoryIDValidation" style="color: red;"></span></td>
254 </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.

```

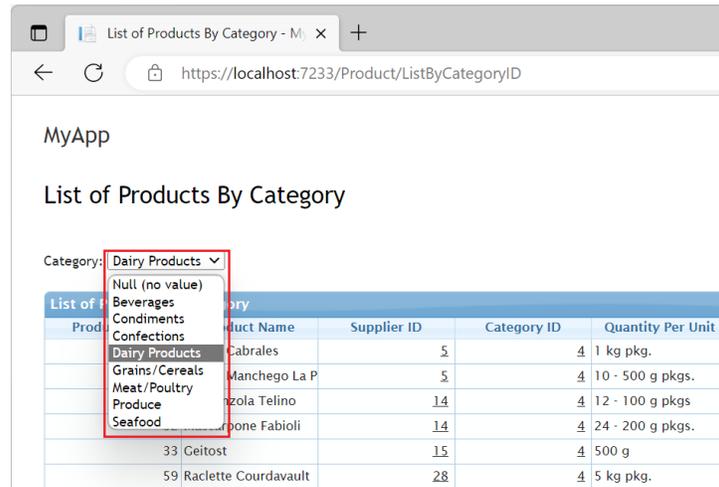
public async Task<IActionResult> ListCrud() ——— Action
{
    // return view model
    _productViewModel = await this.GetViewModelAsync("ListCrud");
    return View(_productViewModel); ——— Injecting the ViewModel to the MVC View
}

```

### 3.2.5.2.4 ListByForeignKey.cshtml

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

The *ListByForeignKey.cshtml* uses the MVC View's *Model (ProductViewModel)* 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 *ProductViewModel* as shown in the example code in page 32.



```

ListByCategoryId.cshtml
61 |         width: '1200'
62 |     });
63 |     });
64 | </script>
65 | }
66 |
67 | <h2>@ViewBag.Title</h2>
68 | <br /><br />
69 |
70 | @if (Model.CategoryDropDownListData != null)
71 | {
72 |     <text>Category: </text> <select id="selCategoryId" asp-for="Product.CategoryID" asp-items="@((new SelectList(Model.CategoryDropDownListData)
73 | }
74 | else
75 | {
76 |     <text>Category: </text> <select id="selCategoryId"><option value="">Category table data is required</option></select>
77 | }
78 | <br /><br />
79 | <table id="list-grid"></table>
80 | <div id="list-pager"></div>

```

The *ViewModel* used by the *ListByForeignKey.cshtml* View is assigned from the respective *Get Action* method found in the *Controller (Base)*, it is then injected to the *ListByForeignKey.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() ——— Action
{
    // get records
    List<Category> objCategoriesList = null;
    string responseBody = await Functions.HttpClientGetAsync("CategoryApi/SelectCategoryDropDownListData/");

    // make sure responseBody is not empty before deserialization
    if(!String.IsNullOrEmpty(responseBody))
        objCategoriesList = JsonConvert.DeserializeObject<List<Category>>(responseBody);

    // assign values to the view model
    _productViewModel.CategoryDropDownListData = objCategoriesList;

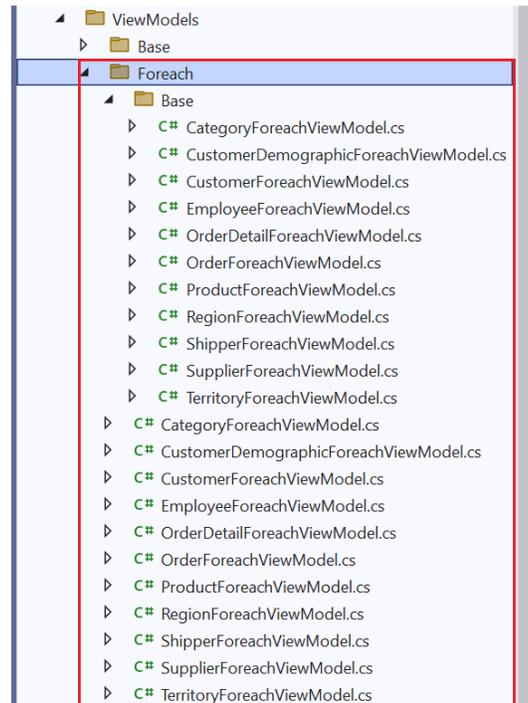
    // return the view model
    return View(_productViewModel); ——— Injecting the ViewModel to the MVC View
}

```

### 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.

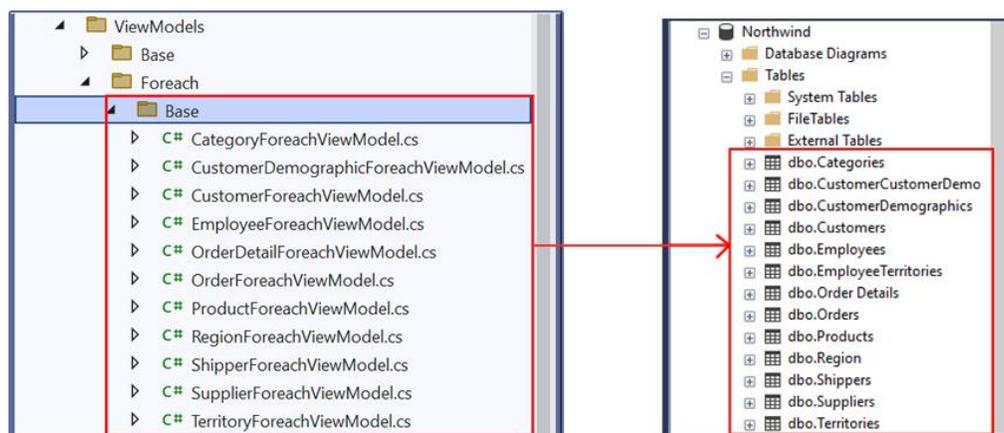


### 3.2.5.4 Partial Class – Used Like A Base Class

These are the class files generated in the *ViewModels\Foreach\Base* folder.

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

One *Partial Class* (in the Base folder) is generated per *Database Table*. The example below shows that you generated code for *All Tables* for the *Northwind* database.



Partial ForeacjViewModel in Visual Studio under Base Folder (Left) – Product Database Table Columns in MS SQL Server (Right)

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

```

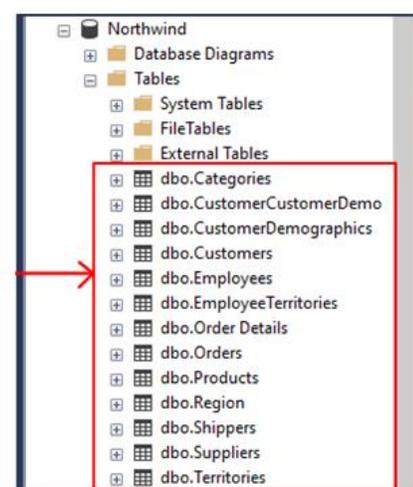
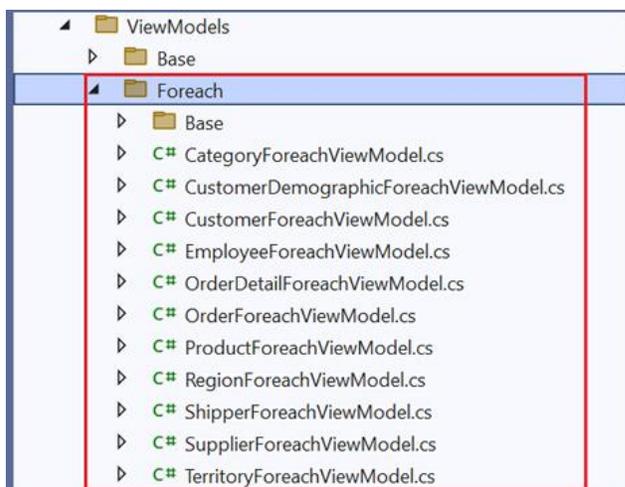
1  using MyAppApi.Models;
2  using MyAppApi.BusinessLayer;
3  using System.Collections.Generic;
4
5  namespace MyAppApi.ViewModels
6  {
7      /// <summary>
8      /// Works like the Base class for ProductForeachViewModel
9      /// ***** Do not make changes to this class *****
10     /// ** Put your additional code in the ProductForeachViewModel class under the ViewModels/Foreach folder. **
11     /// *****
12     /// </summary>
13     public partial class ProductForeachViewModel
14     {
15         3 references
16         public List<Product> ProductData { get; set; }
17         4 references
18         public string[,] ProductFieldNames { get; set; }
19         6 references
20         public string FieldToSort { get; set; }
21         5 references
22         public string FieldToSortWithOrder { get; set; }
23         6 references
24         public string FieldSortOrder { get; set; }
25         4 references
26         public int StartPage { get; set; }
27         4 references
28         public int EndPage { get; set; }
29         3 references
30         public int CurrentPage { get; set; }
31         2 references
32         public int NumberOfPagesToShow { get; set; }
33         3 references
34         public int TotalPages { get; set; }
35         0 references
36         public List<string> UnsortableFields { get; set; }
37     }

```

### 3.2.5.5 Foreach View Model – Empty

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

You can add code in these *Class* files.



Classes in Visual Studio directly under ViewModels\Foreach Folder (Left) – Database Tables in MS SQL Server (Right)



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

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	<input type="checkbox"/>
2	Chang	1	1	24 - 12 oz bottles	\$19.00	17	40	25	<input type="checkbox"/>
3	Aniseed Syrup	1	2	12 - 550 ml bottles	\$10.00	13	70	25	<input type="checkbox"/>
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	\$22.00	53	0	0	<input type="checkbox"/>
5	Chef Anton's Gumbo Mix	2	2	36 boxes	\$21.35	0	0	0	<input checked="" type="checkbox"/>
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	\$25.00	120	0	25	<input type="checkbox"/>
7	Uncle Bob's Organic Dried Pears	2	7	12 - 1 lb pkgs.	\$30.00	15	0	10	<input type="checkbox"/>
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	\$40.00	6	0	0	<input type="checkbox"/>
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	\$97.00	29	0	0	<input checked="" type="checkbox"/>
10	Ikura	4	8	12 - 200 ml jars	\$31.00	31	0	0	<input type="checkbox"/>
11	Queso Cabrales	5	4	1 kg pkg.	\$21.00	22	30	30	<input type="checkbox"/>
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs.	\$38.00	86	0	0	<input type="checkbox"/>
13	Konbu	6	8	2 kg box	\$6.00	24	0	5	<input type="checkbox"/>
14	Tofu	6	7	40 - 100 g pkgs.	\$23.25	35	0	0	<input type="checkbox"/>
15	Genen Shouyu	6	2	24 - 250 ml bottles	\$15.50	39	0	5	<input type="checkbox"/>

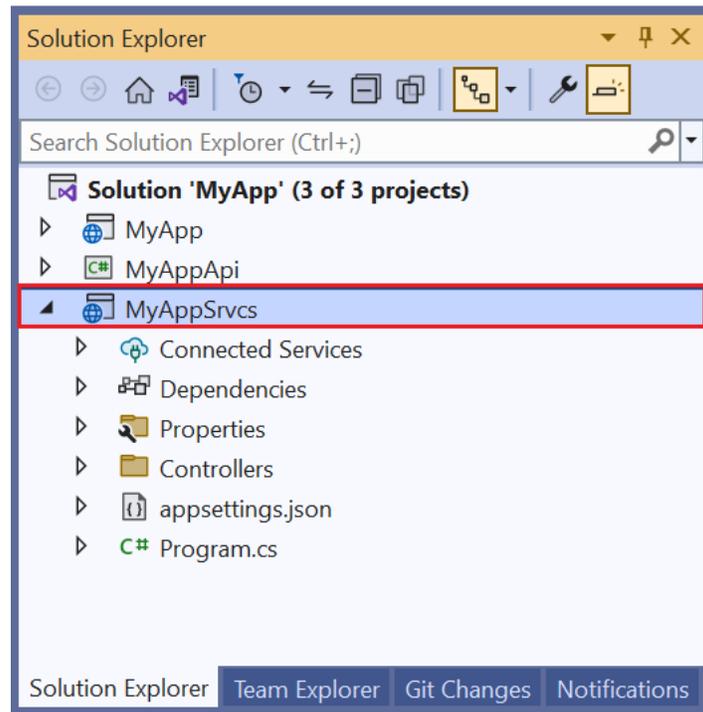
### 3.3 WEB API PROJECT (WEB SERVICES)

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 5, the *Web Application Project (ASP.NET MVC Core)* and other clients are seen accessing the *Web APIs* instead of directly accessing the *Business Layer (Middle Tier Objects)*.

These *Web APIs* encapsulates the *Middle Layer (Business Layer)*. As mentioned in this document, clients can either access the generated *Web APIs* or the *Middle Layer (Business Layers)* 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 Layers)*.

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\* (see note).

**Note:** Although the *Web API Project* does not have a user interface, the generated *Web API methods* can be tested in the **Swagger Index page**. The Swagger Index page can be used to test the generated *Web API methods*, you may also supply it to (software) clients so they can have an idea on how your *Web API methods* work (can be accessed). See page 44.



### 3.3.1 LaunchSettings.json, appsettings.json, Program.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 *Controller's Methods* that can be used as *Web APIs*.

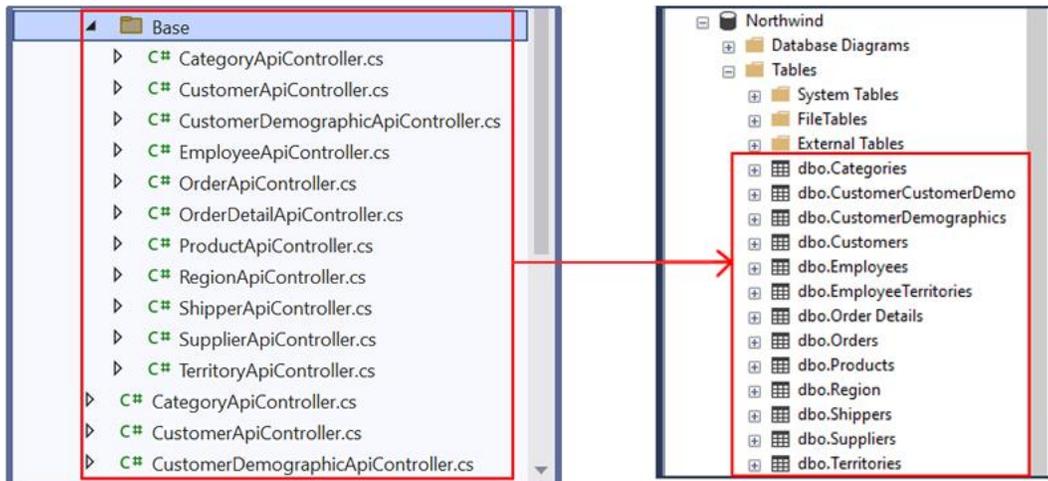
**Note:** The *Controllers* in the *Web API Project* and the *Web Application Project* are similar in nature. Please read about the *Controllers* under the *Web Application Project* in page 10 for more information.

#### 3.3.2.1 The Controller - Used Like A Base Class

**Note: Not a base class.** The code needed by the Controller are generated in these partial classes. These are the partial class files generated in the *Controller\Base* folder. The naming convention used is: ***TableNameApiController.cs***.

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

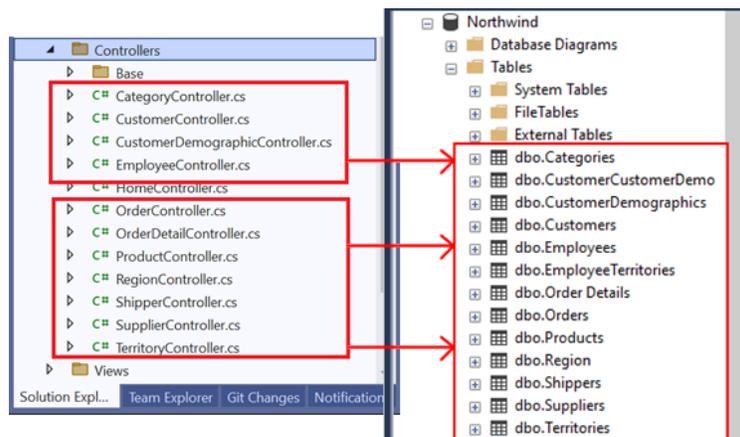
One *Partial Class* (in the *Controllers\Base* folder) is generated per *Database Table*. The example below shows that you generated code for *All Tables* for the *Northwind* database.



Web API Controllers (Partial Classes) in Visual Studio (Left) – Database Tables in MS SQL Server (Right)

### 3.3.2.2 The Controller - Empty

These are the *Partial Classes* generated directly under the *Controllers* folder (not including everything inside the *Base* folder). The naming convention used is: **TableNameApiController.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 *Partial Class* files.**



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

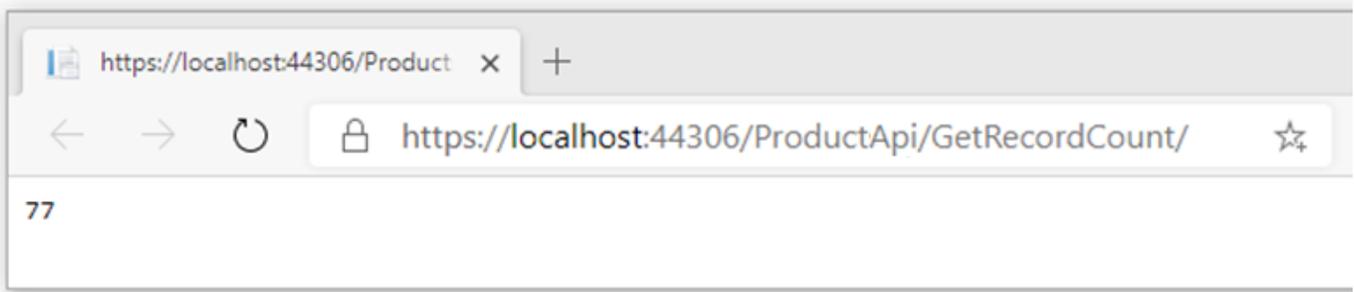
### 3.3.2.3 Accessing Web API Controllers (Methods)

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* (*ProductControllerBase*) to access the *GetRecordCount()* Method in the *Web API Controller* (*ProductApiControllerBase*).

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 (**ProductAPI**, 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 Middle Layer Project) plus the "*ProductAPI/GetRecordCount*".

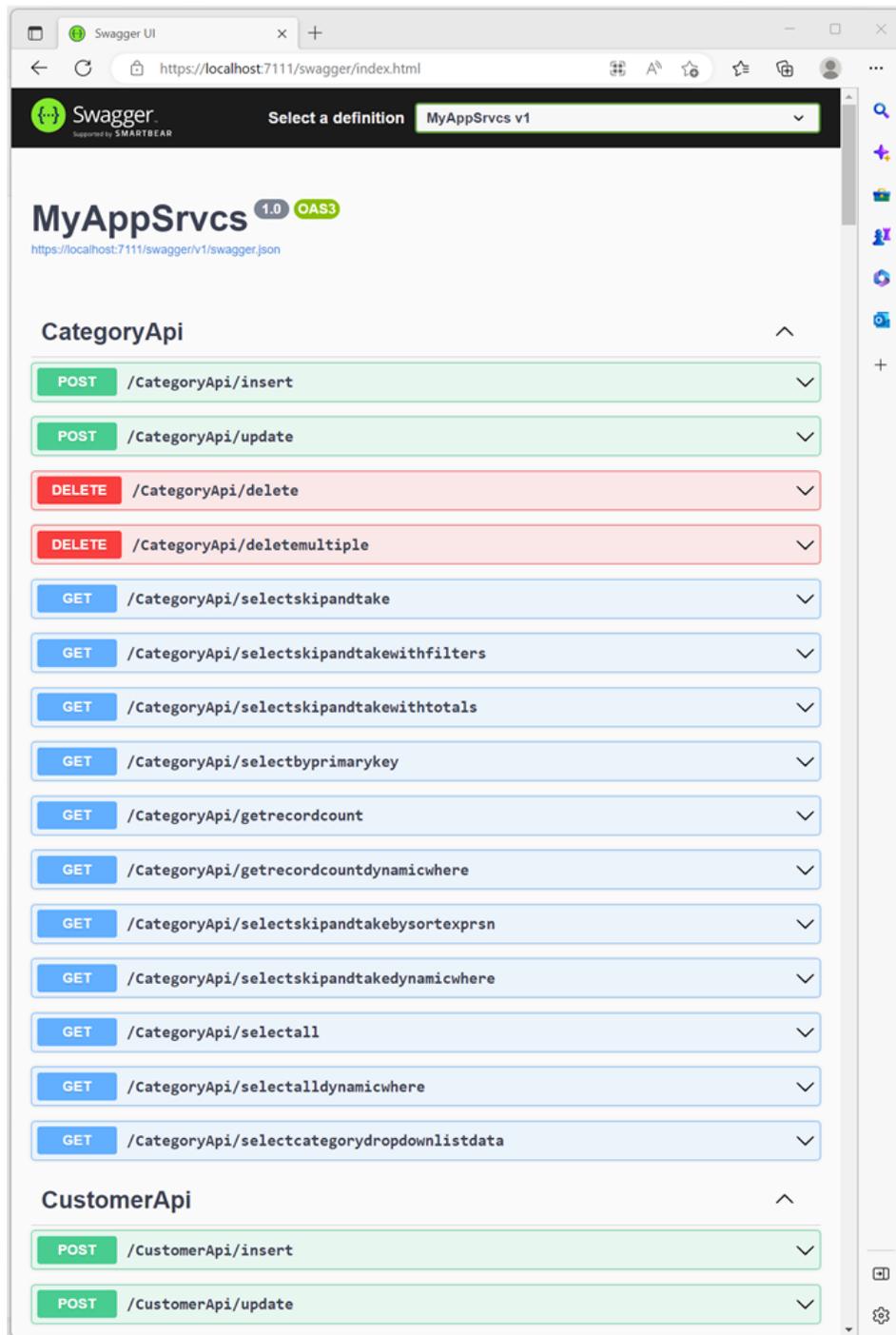
```

ProductController.cs
1  using ...
13
14 namespace MyApp.Controllers — Web Application
15 {
16     <summary> Works like the Base class for ProductController class. ***** ...
17     </summary>
18     public partial class ProductController : Controller
19     {
20         private Product _product;
21         private ProductViewModel _productViewModel;
22         private ProductForeachViewModel _productForeachViewModel;
23
24         // constructor
25         public ProductController(Product product, ProductViewModel productViewModel, ProductForeachVi
26
27         actions used by their respective views
28
29         public methods
30
31         private methods
32
33         #region methods that return data in json format used by the jqgrid
34
35         <summary> GET: /Product/GridData Gets the json needed by the jqgrid for use ...
36         </summary>
37         public async Task<IActionResult> GridData(string sidx, string sord, int page, int rows)
38         {
39             // get the total number of records
40             string responseBody1 = await Functions.HttpClientGetAsync("ProductApi/GetRecordCount/");
41             int totalRecords = JsonConvert.DeserializeObject<int>(responseBody1);
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567

```

### 3.3.3 Swagger

When you run both the *Web Application Project* and the *Web API Project* at the same time in Visual Studio, 2 browser instances launches on the screen, one for each project. The Web App's home page shows the list of objects that was generated while the Web API's home page launches a *Swagger User Interface* showing a list of web API endpoints.

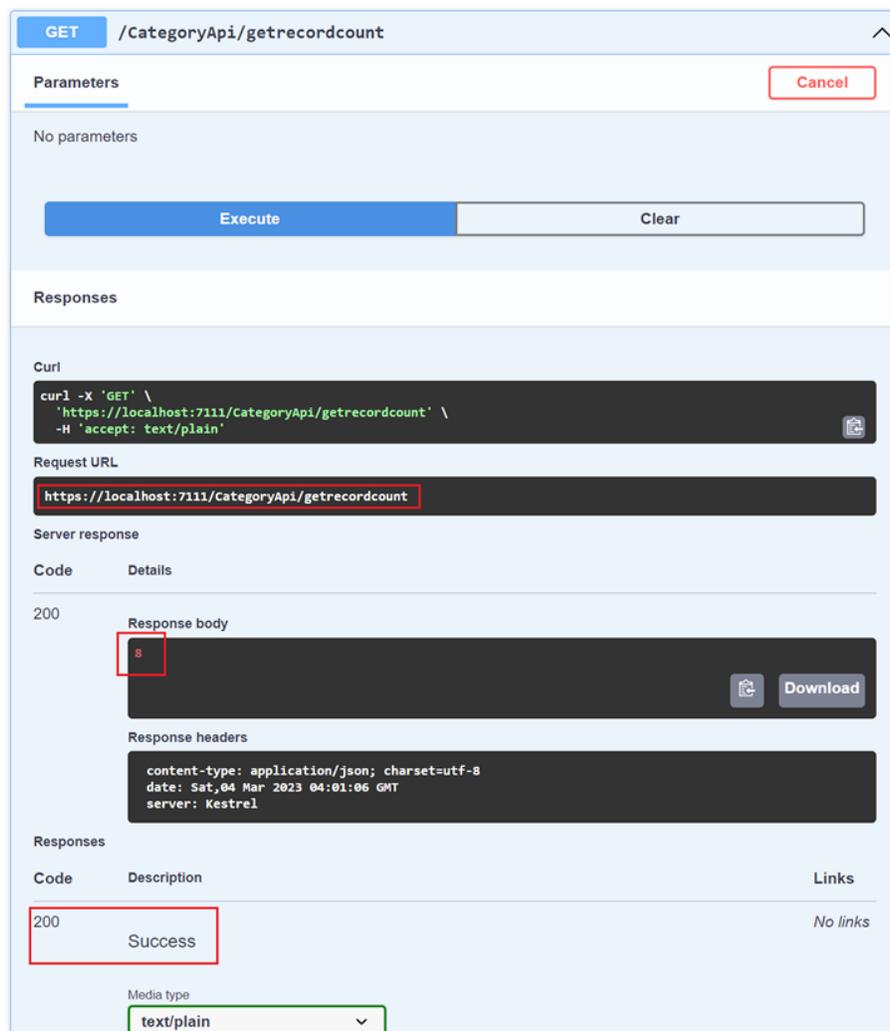


#### 3.3.3.1 Testing An Endpoint (Web API Method)

Click an Endpoint in the list, for example the `/CategoryApi/getrecordcount`. And then click the *Try it out* button. And then click the *Execute* button.



The result will show the *Request URL*, you can use this in your code to call this endpoint. Also shows the *Response Body* **8** (*getrecordcount* endpoint simply returns a number), which is the *total number of records* in the *Categories* database table. And the *Response Code*, *200* means *Success* (the web API call was successful).



Now try the `/CategoryApi/selectskipandtake` endpoint. Click the *Try it out* button. This endpoint requires 4 parameters:

1. **sidx** – Field to sort.
2. **sord** – Sort order. *asc* or *blank* (nothing) for ascending order, and *desc* for descending order.
3. **rows** – Number of rows you want returned.
4. **page** – based on the number of rows you're requesting, there may be several pages to return, this is the *page number* you want returned. For example, if there are 37 records in the *Categories* database table, and the *rows* you requested is 5, then there will be 7 pages, you can request any number from 1 to 7.

Name	Description
sidx string (query)	CategoryName
sord string (query)	sord
page integer(\$int32) (query)	1
ROWS integer(\$int32) (query)	5

The *Response* shows a *Code 200* (success), *5 records* returned (in descending order by *CategoryName*) in *json format*.

Request URL

```
https://localhost:7111/CategoryApi/selectskipandtake?sidx=CategoryName&page=1&rows=5
```

Server response

Code	Details
200	Response body

```
[
  {
    "categoryID": 1,
    "categoryName": "Beverages",
    "description": "Soft drinks, coffees, teas, beers, and ales"
  },
  {
    "categoryID": 2,
    "categoryName": "Condiments",
    "description": "Sweet and savory sauces, relishes, spreads, and seasonings"
  },
  {
    "categoryID": 3,
    "categoryName": "Confections",
    "description": "Desserts, candies, and sweet breads"
  },
  {
    "categoryID": 4,
    "categoryName": "Dairy Products",
    "description": "Cheeses"
  },
  {
    "categoryID": 5,
    "categoryName": "Grains/Cereals",
    "description": "Breads, crackers, pasta, and cereal"
  }
]
```

Response headers

```
content-type: application/json; charset=utf-8
date: Sat, 04 Mar 2023 04:27:38 GMT
server: Kestrel
```

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

You can read end-to-end tutorials on more subjects on using AspCoreGen 6.0 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 6.0 MVC Professional.

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

End of tutorial.