

# Customization by Adding Your Own Code

## 1 CONTENTS

---

1	Introduction .....	2
1.1	Read these tutorials in order .....	2
2	Adding New Files .....	2
2.1	Where Can You Add Files? .....	2
2.2	What Files Can You Add? .....	3
2.3	Why Add New Files? .....	3
3	Adding Code to a Generated File .....	3
4	End-to-End Example on Adding Your Own File and Code .....	4
4.1	Generate Code Using AspCoreGen 6.0 MVC Professional Plus .....	4
4.2	The Tutorial .....	4

# Customization by Adding Your Own Code

## 1 INTRODUCTION

---

This topic will show you how to add your own code to the 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
8. The Generated Code for Database Tables/Views

Then follow these step-by-step instructions.

## 2 ADDING NEW FILES

---

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

### 2.1 WHERE CAN YOU ADD FILES?

You can add files to the following generated projects:

1. Web Application Project (Presentation Layer – UI)
2. Middle Layer Project (Class Library – Business Layer, Data Repository, Shared Libraries).
3. Web API Project (Optional – Web Services)

## 2.2 WHAT FILES CAN YOU ADD?

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

1. MVC View
2. Controller
3. Class Files
4. Images
5. CSS Files
6. JavaScript Files
7. And many, many more

## 2.3 WHY ADD NEW FILES?

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

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

## 3 ADDING CODE TO A GENERATED FILE

---

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

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

---

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

### 4.1 GENERATE CODE USING ASPCOREGEN 6.0 MVC PROFESSIONAL PLUS

You can generate your own Web Application using AspCoreGen 6.0 MVC Professional Plus and just follow along this tutorial. Make sure to:

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

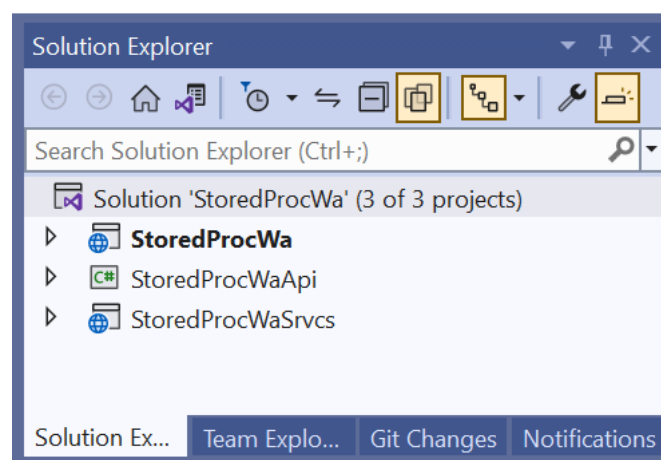
Or, you can download the sample *Generated Web Project Example* from our website:

<https://junnark.com/Product/AspCoreGen6MVC/GeneratedProjects>. Download #4, the *Stored Procedures Using Web API Sample Project*. Unzip the downloaded project and make sure to follow the instructions in the *Readme.txt* file.

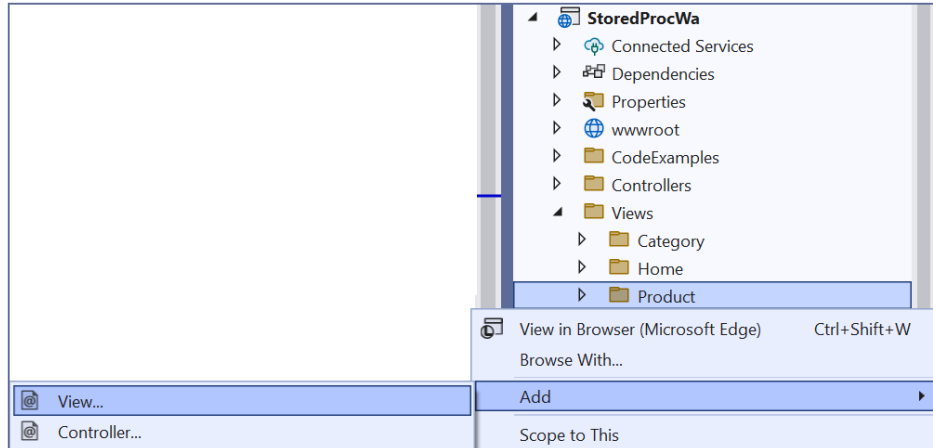
### 4.2 THE TUTORIAL

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

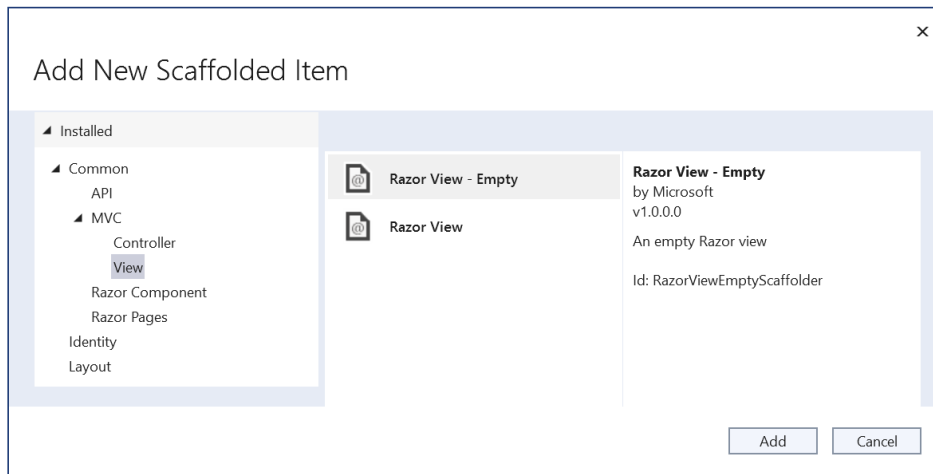
1. Open the *Generated Web Application (StoredProcWa.sln)* in *Visual Studio 2022*. This solution should have 3 projects: The *Web Application (StoredProcWa)*, the *Class Library (StoredProcWaApi)*, and the *Web API (StoredProcWaSrvcs)* projects.



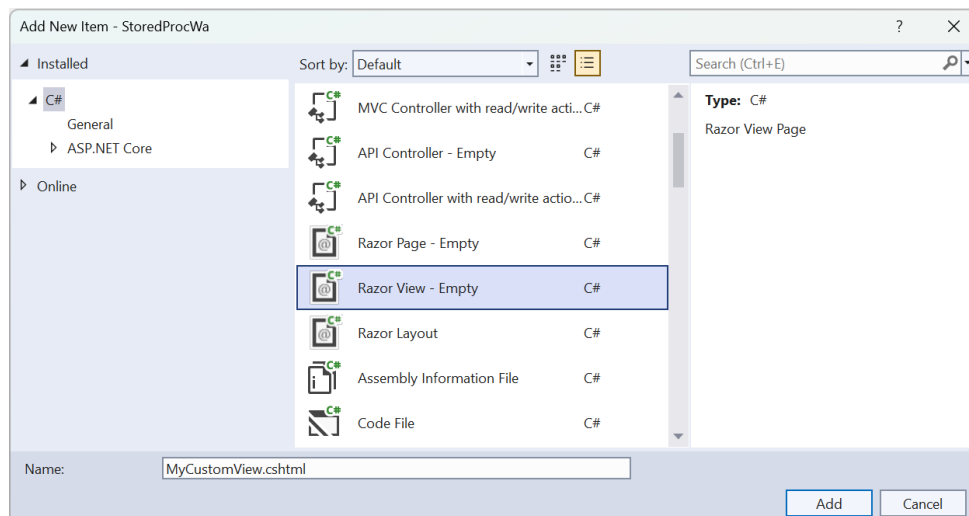
2. Add a new MVC View under the *Product* folder.



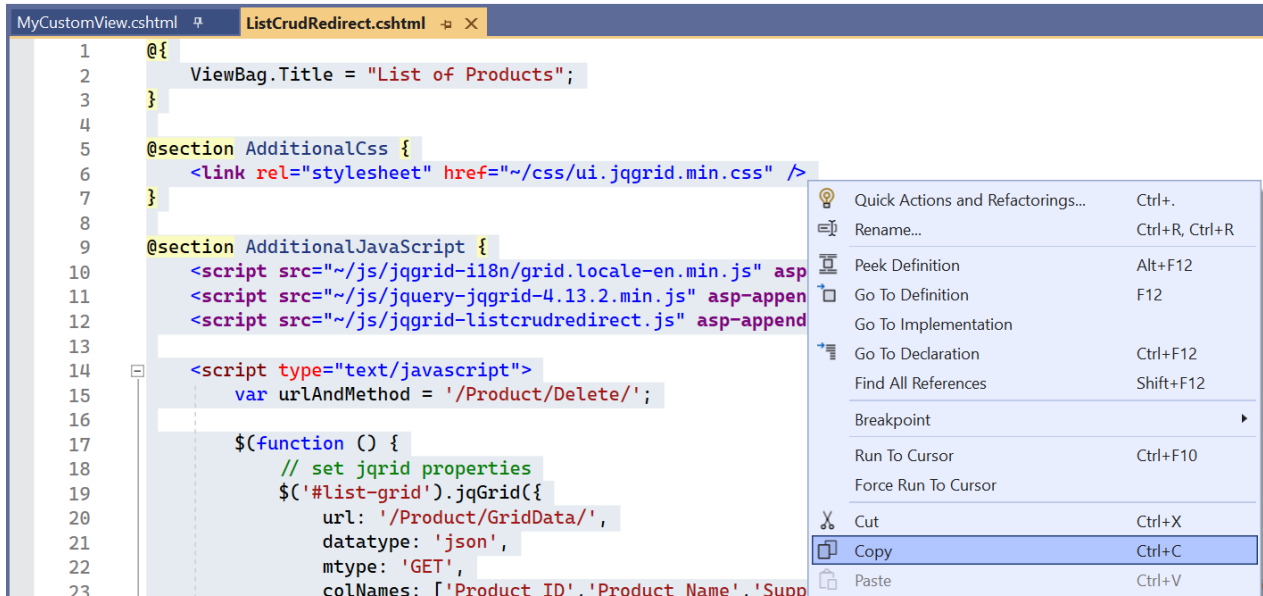
3. Choose *Razor View - Empty* and click the *Add* button.



4. Name the new MVC View: *MyCustomView.cshtml* and then click the *Add* button.



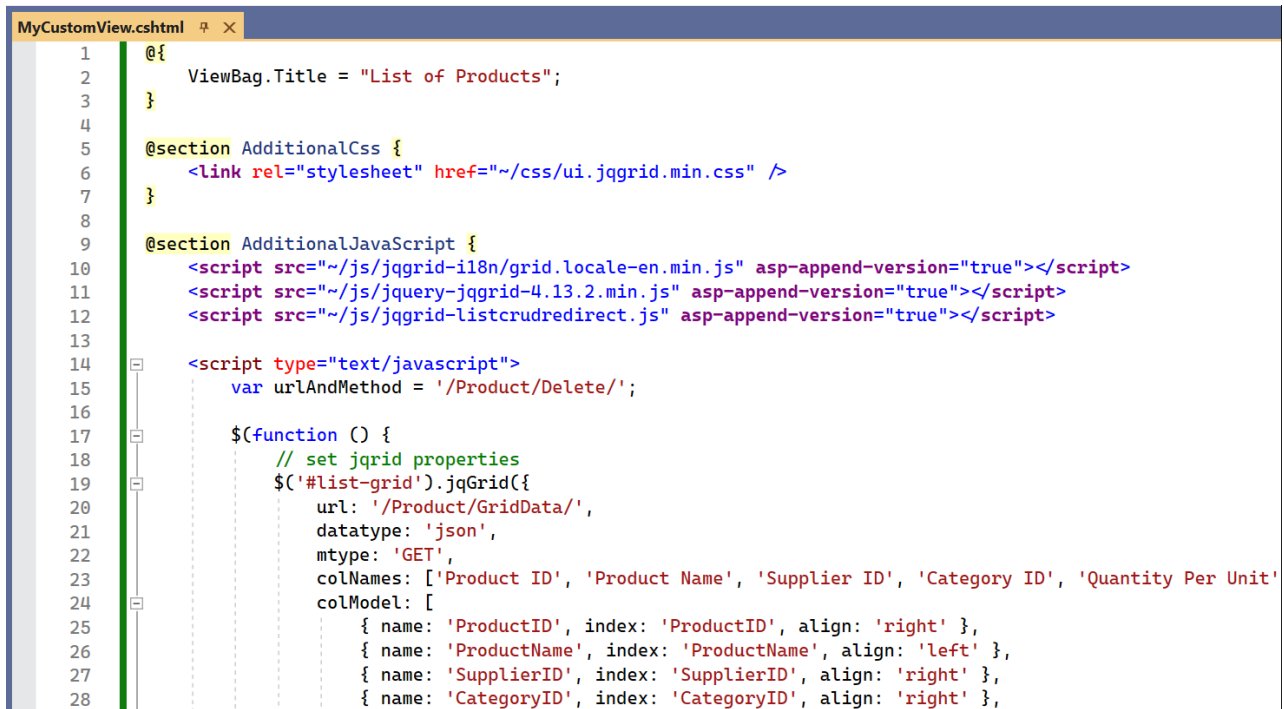
5. Delete all the commented code in the *MyCustomView.cshtml*. And then Open the *ListCrudRedirect.cshtml* under the *Product* folder and Copy all code to *MyCustomView.cshtml*.



```

1  @{}
2  ViewBag.Title = "List of Products";
3  }
4
5  @section AdditionalCss {
6  <link rel="stylesheet" href="~/css/ui.jqgrid.min.css" />
7  }
8
9  @section AdditionalJavaScript {
10 <script src="~/js/jqgrid-118n/grid.locale-en.min.js" asp-
11 <script src="~/js/jquery-jqgrid-4.13.2.min.js" asp-appen
12 <script src="~/js/jqgrid-listcrudredirect.js" asp-append
13
14 <script type="text/javascript">
15     var urlAndMethod = '/Product/Delete/';
16
17     $(function () {
18         // set jqrid properties
19         $('#list-grid').jqGrid({
20             url: '/Product/GridData/',
21             datatype: 'json',
22             mtype: 'GET',
23             colNames: ['Product ID', 'Product Name', 'Supp

```



```

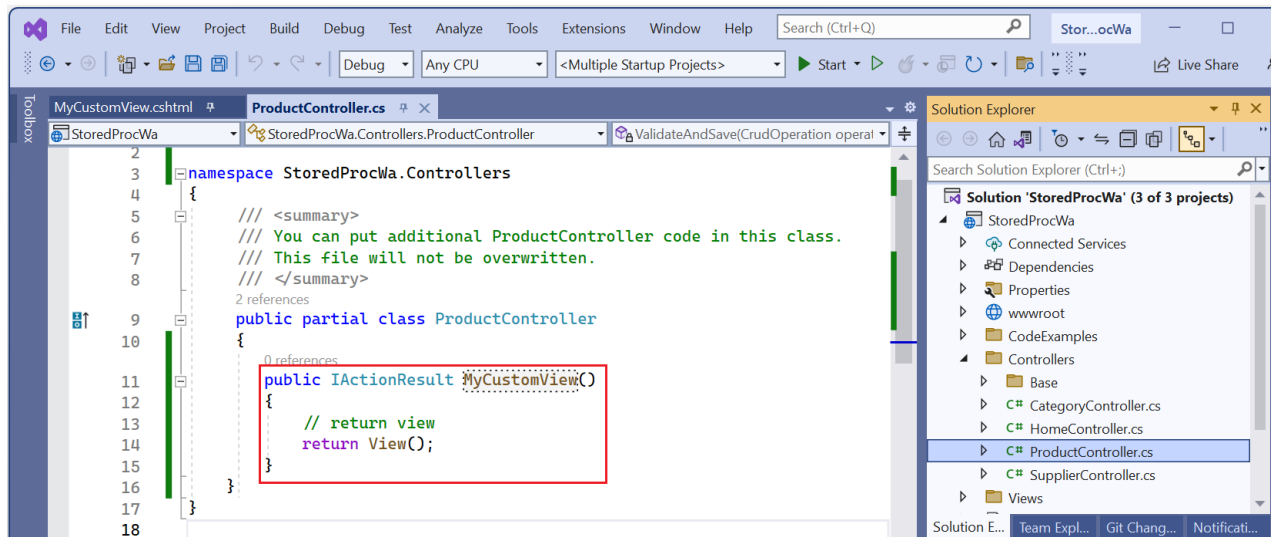
1  @{}
2  ViewBag.Title = "List of Products";
3  }
4
5  @section AdditionalCss {
6  <link rel="stylesheet" href="~/css/ui.jqgrid.min.css" />
7  }
8
9  @section AdditionalJavaScript {
10 <script src="~/js/jqgrid-118n/grid.locale-en.min.js" asp-append-version="true"></script>
11 <script src="~/js/jquery-jqgrid-4.13.2.min.js" asp-append-version="true"></script>
12 <script src="~/js/jqgrid-listcrudredirect.js" asp-append-version="true"></script>
13
14 <script type="text/javascript">
15     var urlAndMethod = '/Product/Delete/';
16
17     $(function () {
18         // set jqrid properties
19         $('#list-grid').jqGrid({
20             url: '/Product/GridData/',
21             datatype: 'json',
22             mtype: 'GET',
23             colNames: ['Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per Unit'],
24             colModel: [
25                 { name: 'ProductID', index: 'ProductID', align: 'right' },
26                 { name: 'ProductName', index: 'ProductName', align: 'left' },
27                 { name: 'SupplierID', index: 'SupplierID', align: 'right' },
28                 { name: 'CategoryID', index: 'CategoryID', align: 'right' },

```

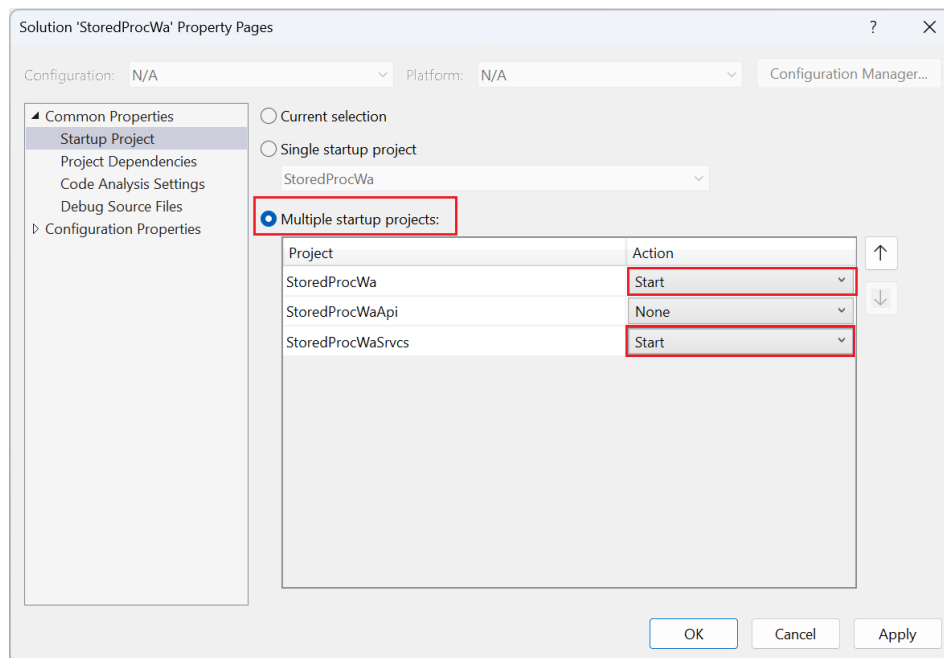
6. Now that we've added a new file (*MVC View*) to the generated *Web Application Project*, we will now add code to an existing generated file. We need to add an *Action Method* for the *MyCustomView.cshtml* in the respective *ProductController.cs*.

Again, please read the *App Settings Tab* document to see the list of generated files where you can add your own code to, **these files will not get overwritten even when you regenerate code for the same project.**

7. Open the *ProductController.cs* under the *Controllers* folder. Add an *Action Method* for the *MyCustomView.cshtml* in the respective *ProductController.cs* as shown in red below. Also add the using statements as shown below.



8. Right-click the Solution and click *Properties*. In the *Solution Property Pages* choose *Multiple startup projects*. Choose *Start* for both *StoredProcWa* (web application project) and the *StoredProcWaSrvcs* (web api project) and click *OK*.



9. Run the *Web Application* by pressing *F5* while in Visual Studio 2022. Two browsers will launch, one for the *Web Application* project and one for the *Web API* project. In the web application project's browser go to the *MyCustomView* MVC View. This page/view should look exactly like the *ListCrudRedirect.cshtml* MVC View.

StoredProcWa

List of Products

Add New Product

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 S	2	2	48 - 6 oz jars	\$22.00	53	0	0	<input type="checkbox"/>
5	Chef Anton's Gumbo	2	2	36 boxes	\$21.35	0	0	0	<input checked="" type="checkbox"/>
6	Grandma's Boysenbe	3	2	12 - 8 oz jars	\$25.00	120	0	25	<input type="checkbox"/>
7	Uncle Bob's Organic l	3	7	12 - 1 lb pkgs.	\$30.00	15	0	10	<input type="checkbox"/>
8	Northwoods Cranber	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"/>

Page 1 of 8 | View 1 - 10 of 77

10. Close the browser and go back to Visual Studio 2022.

11. Open the *ProductController.cs* under the *Controllers\Base* folder and then copy the *GridData* method to the *ProductController.cs* directly under the *Controllers* folder.

```

1  using ...
13
14 namespace StoredProcWa.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 productViewModel, ProductForeachViewModel productForeachViewModel)
27         {
28             // constructor
29             0 references
30             public ProductController(Product product, ProductViewModel productViewModel, ProductForeachViewModel productForeachViewModel)
31         }
32
33         actions used by their respective views
34
35         public methods
36
37         private methods
38
39         #region methods that return data in json format used by the jqgrid
40
41         /// <summary> GET: /Product/GridData Gets the json needed by the jqgrid for use ...
42         0 references
43         public async Task<IActionResult> GridData(string sidx, string sord, int page, int rows)
44         {
45             // get the total number of records
46             string responseBody1 = await Functions.HttpClientGetAsync("ProductApi/GetRecordCount/");
47             int totalRecords = JsonConvert.DeserializeObject<int>(responseBody1);
48
49             // get the index where to start retrieving records from and the total number of pages
50             (int startRowIndex, int totalPages) = Functions.GetStartRowIndexAndTotalPages(page, rows, totalRecords);
51
52             // get records
53             List<Product> objProductsList = null;
54             string responseBody2 = await Functions.HttpClientGetAsync("ProductApi/SelectSkipAndTake/?rows=" + rows);
55
56             // make sure responseBody2 is not empty before deserialization
57             if(!String.IsNullOrEmpty(responseBody2))
58                 objProductsList = JsonConvert.DeserializeObject<List<Product>>(responseBody2);
59
60             // return json data for use by the jqgrid
61             return this.JsonData(objProductsList, totalPages, page, totalRecords);
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

```



```

MyCustomView.cshtml  ProductController.cs  ProductController.cs
StoredProcWa  StoredProcWa.Controllers.ProductController  MyCustomView()

2 references
12 public partial class ProductController
13 {
0 references
14 public IActionResult MyCustomView()
15 {
16     // return view
17     return View();
18 }
19
0 references
20 public async Task<ActionResult> GridData(string sidx, string sord, int page, int rows)
21 {
22     // get the total number of records
23     string responseBody1 = await Functions.HttpClientGetAsync("ProductApi/GetRecordCount/");
24     int totalRecords = JsonConvert.DeserializeObject<int>(responseBody1);
25
26     // get the index where to start retrieving records from and the total number of pages
27     (int startRowIndex, int totalPages) = Functions.GetStartRowIndexAndTotalPages(page, rows, totalRecords);
28
29     // get records
30     List<Product> objProductsList = null;
31     string responseBody2 = await Functions.HttpClientGetAsync("ProductApi/SelectSkipAndTake/?rows=" + rows +
32
33     // make sure responseBody2 is not empty before deserialization
34     if (!String.IsNullOrEmpty(responseBody2))
35         objProductsList = JsonConvert.DeserializeObject<List<Product>>(responseBody2);
36
37     // return json data for use by the jqgrid
38     return this.GetJsonData(objProductsList, totalPages, page, totalRecords);
39 }
40 }
41

```

12. Change the name of the *GridData* method to *MyGridData*.

```

MyCustomView.cshtml  ProductController.cs  ProductController.cs
StoredProcWa  StoredProcWa.Controllers.ProductController  MyGridData(string sidx, string sord, int page, int rows)

2 references
12 public partial class ProductController
13 {
0 references
14 public IActionResult MyCustomView()
15 {
16     // return view
17     return View();
18 }
19
0 references
20 public async Task<ActionResult> MyGridData(string sidx, string sord, int page, int rows)
21 {
22     // get the total number of records
23     string responseBody1 = await Functions.HttpClientGetAsync("ProductApi/GetRecordCount/");
24     int totalRecords = JsonConvert.DeserializeObject<int>(responseBody1);
25
26     // get the index where to start retrieving records from and the total number of pages
27     (int startRowIndex, int totalPages) = Functions.GetStartRowIndexAndTotalPages(page, rows, totalRecords);

```

13. In the *MyCustomView.cshtml*, we are going to use the new *MyGridData* method that we added on the *ProductController.cs* as the source of the grid's data. To do this, simply change the *URL* property of *JQGrid* from *GridData* to *MyGridData* as shown below. Also change the title of the page.

```

MyCustomView.cshtml ProductController.cs
1  @{}
2  ViewBag.Title = "My Custom View";
3  }
4
5  @section AdditionalCss {
6  <link rel="stylesheet" href="~/css/ui.jqgrid.min.css" />
7  }
8
9  @section AdditionalJavaScript {
10 <script src="~/js/jqgrid-i18n/grid.locale-en.min.js" asp-append-version="true"></script>
11 <script src="~/js/jquery-jqgrid-4.13.2.min.js" asp-append-version="true"></script>
12 <script src="~/js/jqgrid-listcrudredirect.js" asp-append-version="true"></script>
13
14 <script type="text/javascript">
15     var urlAndMethod = '/Product/Delete/';
16
17     $(function () {
18         // set jqgrid properties
19         $('#list-grid').jqGrid({
20             url: '/Product/MyGridData/',
21             datatype: 'json',

```

14. Run the *Web Application* by pressing *F5* while in Visual Studio 2022. And then go to the *MyCustomView MVC View*. This page/view should look just like the *ListCrudRedirect.cshtml MVC View* with a new page title.

My Custom View - StoredProcWa x +

https://localhost:7233/Product/MyCustomView

StoredProcWa

My Custom View

Add New Product

Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit	Unit Price	Units In Stock	Units On Order
1	Chai	1	1	10 boxes x 20 bags	\$18.00	39	0
2	Chang	1	1	24 - 12 oz bottles	\$19.00	17	40
3	Aniseed Syrup	1	2	12 - 550 ml bottles	\$10.00	13	70
4	Chef Anton's Cajun S	2	2	48 - 6 oz jars	\$22.00	53	0
5	Chef Anton's Gumbo	2	2	36 boxes	\$21.35	0	0
6	Grandma's Boysenbe	3	2	12 - 8 oz jars	\$25.00	120	0
7	Uncle Bob's Organic	3	7	12 - 1 lb pkgs.	\$30.00	15	0
8	Northwoods Cranber	3	2	12 - 12 oz jars	\$40.00	6	0
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	\$97.00	29	0
10	Ikura	4	8	12 - 200 ml jars	\$31.00	31	0

Page 1 of 8 10

15. Close the browser and go back to Visual Studio 2022.

16. Again, open the *ProductController.cs* under the *Controllers\Base* folder and then copy the *GetJsonData* method to the *ProductController.cs* directly under the *Controllers* folder.

```

710 // <summary> Gets json-formatted data based on the List of Products for use by ...
711 6 references
712 private JsonResult GetJsonData(List<Product> objProductsList, int totalPages, int page, int totalRecords)
713 {
714     // return a null in json for use by the jqgrid
715     if (objProductsList is null)
716         return Json("{ total = 0, page = 0, records = 0, rows = null }");
717
718     // create a serialized json object for use by the jqgrid
719     var jsonData = new
720     {
721         total = totalPages,
722         page,
723         records = totalRecords,
724         rows = (
725             from objProduct in objProductsList
726             select new
727             {
728                 id = objProduct.ProductID,
729                 cell = new string[] {
730                     objProduct.ProductID.ToString(),
731                     objProduct.ProductName,
732                     objProduct.SupplierID.HasValue ? objProduct.SupplierID.Value.ToString() : "",
733                     objProduct.CategoryID.HasValue ? objProduct.CategoryID.Value.ToString() : "",
734                     objProduct.QuantityPerUnit,
735                     objProduct.UnitPrice.HasValue ? objProduct.UnitPrice.Value.ToString() : "",
736                     objProduct.UnitsInStock.HasValue ? objProduct.UnitsInStock.Value.ToString() : "",
737                     objProduct.UnitsOnOrder.HasValue ? objProduct.UnitsOnOrder.Value.ToString() : "",
738                     objProduct.ReorderLevel.HasValue ? objProduct.ReorderLevel.Value.ToString() : "",
739                     objProduct.Discontinued.ToString()
740                 }
741             }
742             ).ToArray()
743     };
744
745     return Json(jsonData);
746 }
747
748
749
750
751

```

```

12 public partial class ProductController
13 {
14     0 references
15     public IActionResult MyCustomView(...)
16
17     0 references
18     public async Task<IActionResult> MyGridData(string sidx, string sord, int page, int rows) ...
19
20     6 references
21     private JsonResult GetJsonData(List<Product> objProductsList, int totalPages, int page, int totalRecords)
22     {
23         // return a null in json for use by the jqgrid
24         if (objProductsList is null)
25             return Json("{ total = 0, page = 0, records = 0, rows = null }");
26
27         // create a serialized json object for use by the jqgrid
28         var jsonData = new
29         {
30             total = totalPages,
31             page,
32             records = totalRecords,
33             rows = (
34                 from objProduct in objProductsList
35                 select new
36                 {
37                     id = objProduct.ProductID,
38                     cell = new string[] {
39                         objProduct.ProductID.ToString(),
40                         objProduct.ProductName,
41                         objProduct.SupplierID.HasValue ? objProduct.SupplierID.Value.ToString() : "",
42                         objProduct.CategoryID.HasValue ? objProduct.CategoryID.Value.ToString() : "",
43                         objProduct.QuantityPerUnit,
44                         objProduct.UnitPrice.HasValue ? objProduct.UnitPrice.Value.ToString() : "",
45                         objProduct.UnitsInStock.HasValue ? objProduct.UnitsInStock.Value.ToString() : "",
46                         objProduct.UnitsOnOrder.HasValue ? objProduct.UnitsOnOrder.Value.ToString() : "",
47                         objProduct.ReorderLevel.HasValue ? objProduct.ReorderLevel.Value.ToString() : "",
48                         objProduct.Discontinued.ToString()
49                     }
50                 }
51                 ).ToArray()
52         };
53
54         return Json(jsonData);
55     }
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76

```

17. Change the name of the *GetJsonData* method to *GetJsonData4MyCustomGrid*.

```

12 public partial class ProductController
13 {
14     0 references
15     public IActionResult MyCustomView(...)
16
17     0 references
18     public async Task<IActionResult> MyGridData(string sidx, string sord, int page, int rows)
19     {
20         // get the total number of records
21         string responseBody1 = await Functions.HttpClientGetAsync("ProductApi/GetRecordCount/");
22         int totalRecords = JsonConvert.DeserializeObject<int>(responseBody1);
23
24         // get the index where to start retrieving records from and the total number of pages
25         (int startRowIndex, int totalPages) = Functions.GetStartRowIndexAndTotalPages(page, rows, totalR
26
27         // get records
28         List<Product> objProductsList = null;
29         string responseBody2 = await Functions.HttpClientGetAsync("ProductApi/SelectSkipAndTake/?rows="
30
31         // make sure responseBody2 is not empty before deserialization
32         if (!string.IsNullOrEmpty(responseBody2))
33             objProductsList = JsonConvert.DeserializeObject<List<Product>>(responseBody2);
34
35         // return json data for use by the jqgrid
36         return this.GetJsonData4MyCustomGrid(objProductsList, totalPages, page, totalRecords);
37     }
38
39     1 reference
40     private JsonResult GetJsonData4MyCustomGrid(List<Product> objProductsList, int totalPages, int page,
41     {
42         // return a null in json for use by the jqgrid
43         if (objProductsList is null)
44             return Json("{ total = 0, page = 0, records = 0, rows = null }");
45     }

```

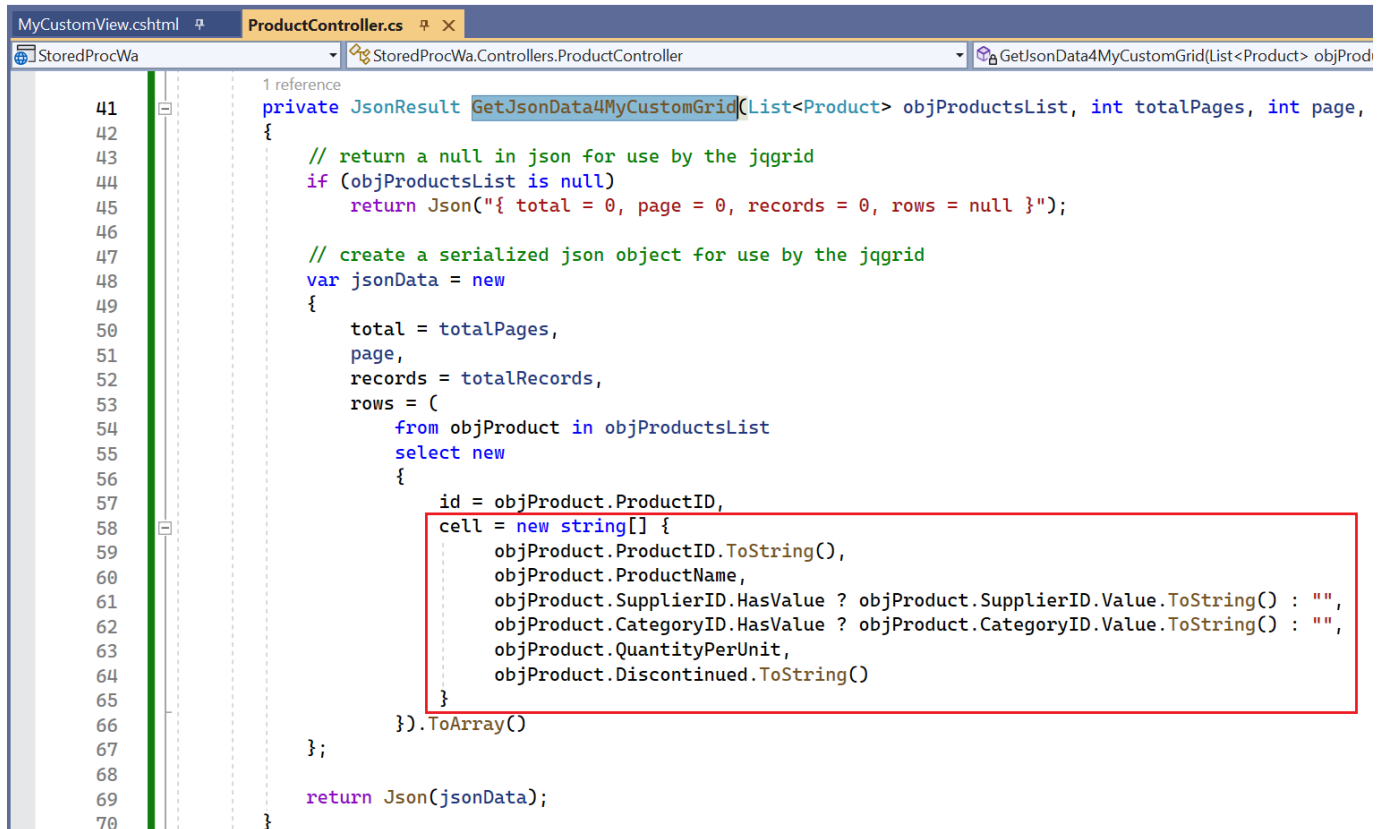
18. Let's remove the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* from the grid. In the *MyCustomView*, delete the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* in the *colNames* and *colModel* properties of the *JQGrid*. The code should look like the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

```

14 <script type="text/javascript">
15     var urlAndMethod = '/Product/Delete/';
16
17     $(function () {
18         // set jqgrid properties
19         $('#list-grid').jqGrid({
20             url: '/Product/MyGridData/',
21             datatype: 'json',
22             mtype: 'GET',
23             colNames: ['Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per Unit', 'Discontinued', '', ''],
24             colModel: [
25                 { name: 'ProductID', index: 'ProductID', align: 'right' },
26                 { name: 'ProductName', index: 'ProductName', align: 'left' },
27                 { name: 'SupplierID', index: 'SupplierID', align: 'right' },
28                 { name: 'CategoryID', index: 'CategoryID', align: 'right' },
29                 { name: 'QuantityPerUnit', index: 'QuantityPerUnit', align: 'left' },
30                 { name: 'Discontinued', index: 'Discontinued', align: 'center', formatter: 'checkbox' },
31                 { name: 'editoperation', index: 'editoperation', align: 'center', width: 40, sortable: false, title: false },
32                 { name: 'deleteoperation', index: 'deleteoperation', align: 'center', width: 40, sortable: false, title: false }
33             ],
34             pager: $('#list-pager'),

```

19. In the *ProductController* under the *GetJsonData4MyCustomGrid* method, delete the lines of code that pertains to the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level*. The code should look like the one shown below after deletion.

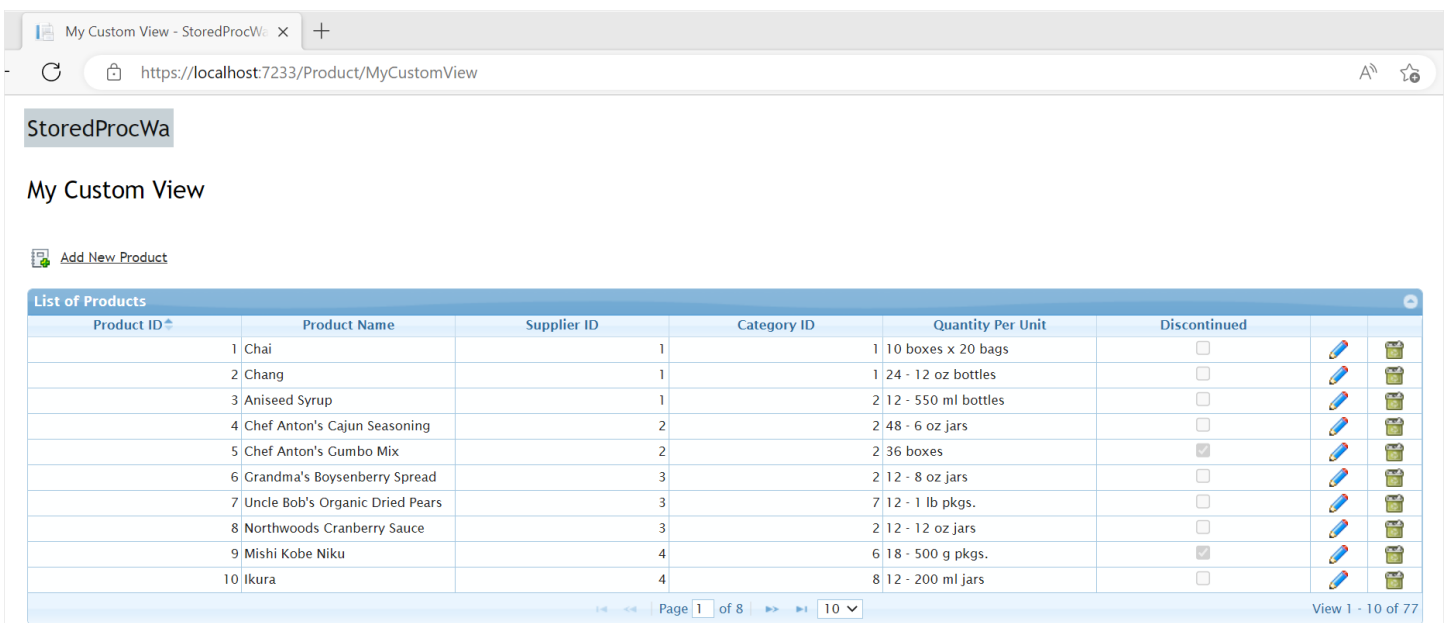


```

41 1 reference
42 private JsonResult GetJsonData4MyCustomGrid(List<Product> objProductsList, int totalPages, int page,
43 {
44     // return a null in json for use by the jqgrid
45     if (objProductsList is null)
46         return Json("{ total = 0, page = 0, records = 0, rows = null }");
47
48     // create a serialized json object for use by the jqgrid
49     var jsonData = new
50     {
51         total = totalPages,
52         page,
53         records = totalRecords,
54         rows = (
55             from objProduct in objProductsList
56             select new
57             {
58                 id = objProduct.ProductID,
59                 cell = new string[] {
60                     objProduct.ProductID.ToString(),
61                     objProduct.ProductName,
62                     objProduct.SupplierID.HasValue ? objProduct.SupplierID.Value.ToString() : "",
63                     objProduct.CategoryID.HasValue ? objProduct.CategoryID.Value.ToString() : "",
64                     objProduct.QuantityPerUnit,
65                     objProduct.Discontinued.ToString()
66                 }
67             }
68         ).ToArray()
69     };
70     return Json(jsonData);

```

20. Run the *Web Application* by pressing *F5* while in Visual Studio 2022. And then go to the *MyCustomView MVC View*. The *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* should no longer be displayed on the grid.



My Custom View - StoredProcWa

My Custom View

Add New Product

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

Page 1 of 8 View 1 - 10 of 77

21. Close the browser and go back to Visual Studio 2022.
22. Go back to the *ProductController* in #19 and update the *SupplierID* and *CategoryID* to show the *CompanyName* and *CategoryName* respectively.

```

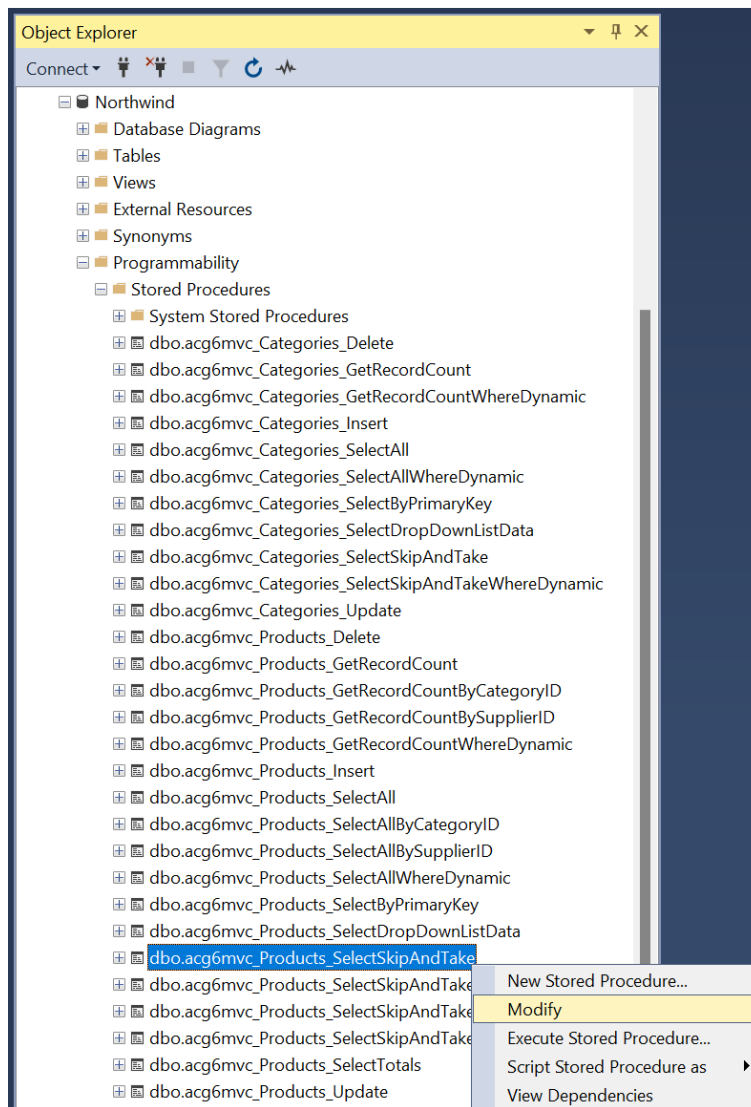
41 private JsonResult GetJsonData4MyCustomGrid(List<Product> objProductsList, int totalPages, int page, int totalRecords)
42 {
43     // return a null in json for use by the jqgrid
44     if (objProductsList is null)
45         return Json("{ total = 0, page = 0, records = 0, rows = null }");
46
47     // create a serialized json object for use by the jqgrid
48     var jsonData = new
49     {
50         total = totalPages,
51         page,
52         records = totalRecords,
53         rows = (
54             from objProduct in objProductsList
55             select new
56             {
57                 id = objProduct.ProductID,
58                 cell = new string[] {
59                     objProduct.ProductID.ToString(),
60                     objProduct.ProductName,
61                     objProduct.SupplierID.HasValue ? objProduct.CompanyName + " (" + objProduct.SupplierID.Value.ToString() + ")" : "",
62                     objProduct.CategoryID.HasValue ? objProduct.CategoryName + " (" + objProduct.CategoryID.Value.ToString() + ")" : "",
63                     objProduct.QuantityPerUnit,
64                     objProduct.Discontinued.ToString()
65                 }
66             }).ToArray()
67     };

```

23. Now we will change the display on the *Supplier ID* and *Category ID*. Instead of showing just the IDs for these foreign keys, we will show the *Company Name (Supplier)* and *Category Name (Category)* respectively. To do this, we need to:
  - a. Create a new *Stored Procedure*.
  - b. Create 2 new *Properties as Models* for *Company Name* and *Category Name*.
  - c. Create a new *Data Repository* method.
  - d. Create a new *Business Layer* method.
  - e. Create a new *Web API* method.

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

24. Create a new **Stored Procedure** named `acg6mvc_Product_MyCustomSelectSkipAndTake` in the *Northwind Database* using *Microsoft SQL Server Management Studio*. Go to the *Stored Procedures* folder under *Programmability* and *Modify* the `acg6mvc_Product_SelectSkipAndTake` Stored Procedure.



25. This will open up the `acg6mvc_Product_SelectSkipAndTake` Stored Procedure on a window.

```

SQLQuery1.sql - 19_G-320junker (54)*
USE [Northwind]
GO
/***** Object: StoredProcedure [dbo].[acg6mvc_Products_SelectSkipAndTake]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[acg6mvc_Products_SelectSkipAndTake]
(
    @start int,
    @numberOfRows int,
    @sortByExpression varchar(200)
)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @numberOfRowsToSkip int = @start;

    SELECT
        [ProductID],
        [ProductName],
        [SupplierID],
        [CategoryID],
        [QuantityPerUnit],
        [UnitPrice],
        [UnitsInStock],
        [UnitsOnOrder],
        [ReorderLevel],
        [Discontinued]
    FROM [dbo].[Products]
    ORDER BY
        CASE WHEN @sortByExpression = 'ProductID' THEN [ProductID] END,
        CASE WHEN @sortByExpression = 'ProductID desc' THEN [ProductID] END DESC,

```

26. Modify the *Stored Procedure*. Change the *ALTER* keyword to *CREATE*. Change the *Stored Procedure* name to *acg6mvc\_Product\_MyCustomSelectSkipAndTake*. Add *INNER JOINS* to the *Suppliers* and *Categories* tables. Remove references to the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* columns.

```

SQLQuery1.sql - 19...G-320junnark (54)* X
USE [Northwind]
GO
/***** Object: StoredProcedure [dbo].[acg6mvc_Products_SelectSkipAndTake]    Script Date: 2
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE PROCEDURE [dbo].[acg6mvc_Products_MyCustomSelectSkipAndTake]
(
    @start int,
    @numberOfRows int,
    @sortByExpression varchar(200)
)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @numberOfRowsToSkip int = @start;

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

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

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

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

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

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

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

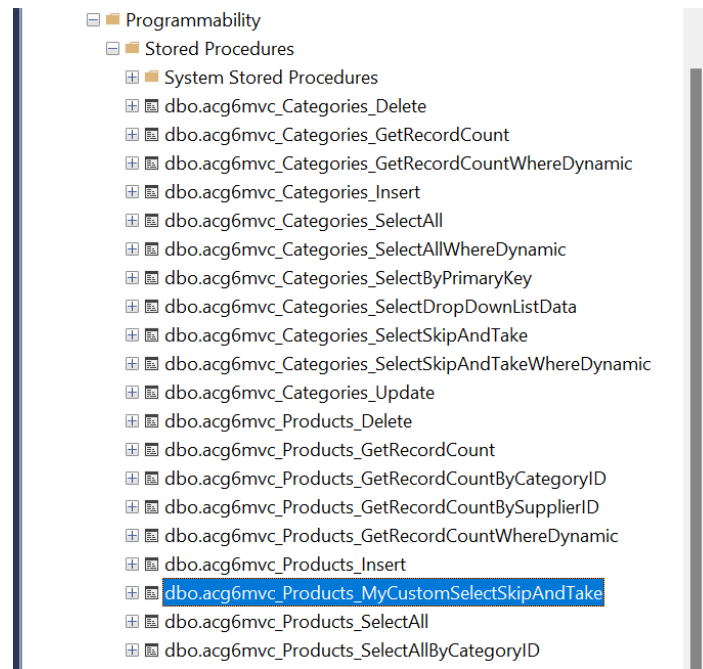
        CASE WHEN @sortByExpression = 'Discontinued' THEN prod.[Discontinued] END,
        CASE WHEN @sortByExpression = 'Discontinued desc' THEN prod.[Discontinued] END DESC

    OFFSET @numberOfRowsToSkip ROWS
    FETCH NEXT @numberOfRows ROWS ONLY
END

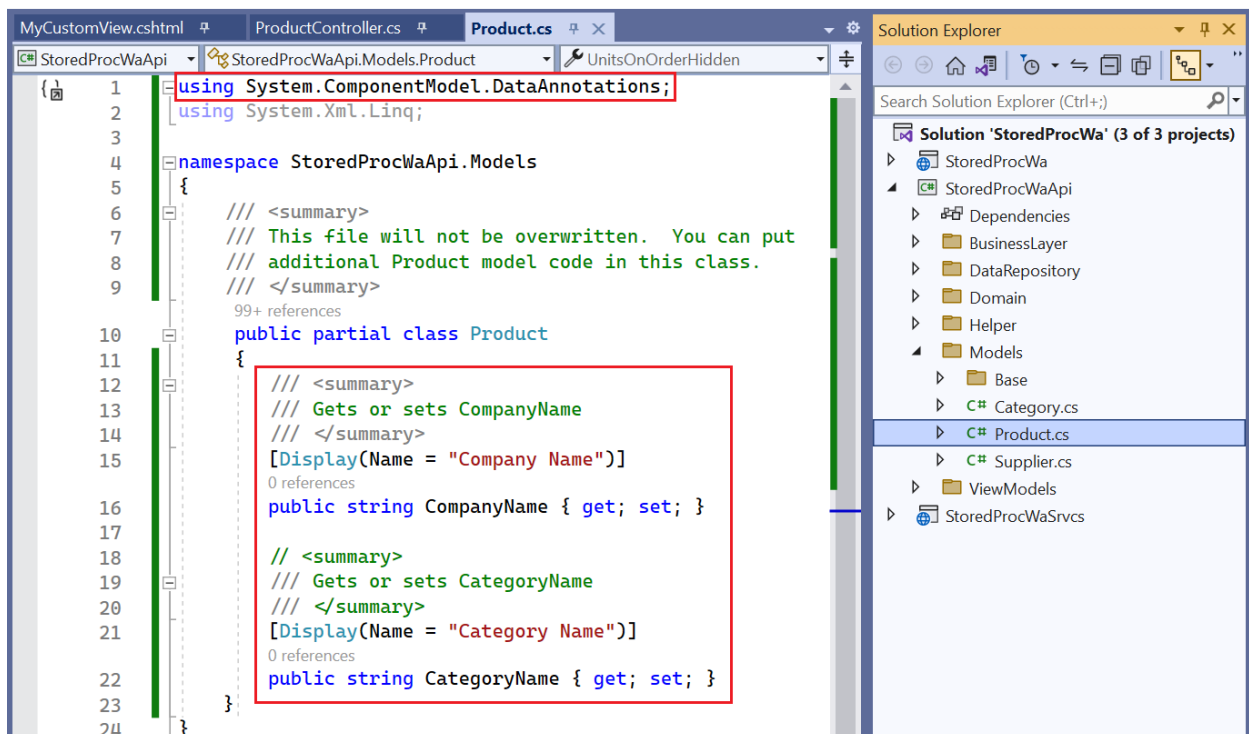
```



27. Make sure to click *Execute* in the *Microsoft SQL Server Management Studio's* menu to create the *acg6mvc\_Product\_MyCustomSelectSkipAndTake* Stored Procedure. When you refresh the Stored Procedures, the *acg6mvc\_Product\_MyCustomSelectSkipAndTake* should now be displayed.



28. Create 2 new *Properties as Models* for *CompanyName* and *CategoryName*. Open the *ProductModel.cs* located in the *StoredProcWaApi (Middle Layer Project)* under the *Models* folder. Add the *CompanyName (Suppliers Database Table)* and *CategoryName (Categories Database Table)* properties. Also add the using statement as shown below.



29. Create a new *Data Repository* method. In *Visual Studio*, open the following:

- IProductRepository* – An interface, found under the *DataRepository\Base\Interface* folder. **Used like a base interface.**
- IProductRepository* – An interface, found under the *DataRepository\Interface* folder. **We can add or update code here.**
- ProductRepository* – A class, found under the *DataRepository\Base* folder. **Used like a base class.**
- ProductRepository* – A class, found under the *DataRepository* folder. **We can add or update code here.**

30. Copy the *SelectSkipAndTakeAsync* method from the *IProductRepository* (used like a base **interface**) to the *IProductRepository* under the *DataRepository\Interface* folder. And then rename the method to *MyCustomSelectSkipAndTakeAsync*. Also add the *using System Data* reference.

```

40
41     /// <summary>
42     /// Selects Products table records sorted by the sortByExpression and returns records from the startRowIndex
43     /// </summary>
44     internal Task<DataTable> SelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows);
45
  
```

```

1     using System.Data;
2
3     namespace StoredProcWApi.DataRepository
4     {
5         /// <summary>
6         /// This file will not be overwritten.
7         /// You can put additional IProductRepository members in this interface.
8         /// </summary>
9         public partial interface IProductRepository
10        {
11            /// <summary>
12            /// This is just an example on how to add your own method. You can delete this.
13            /// Implement this in the ProductRepository class under the DataRepository folder.
14            /// </summary>
15            internal string ExampleRepositoryMember();
16
17            internal Task<DataTable> MyCustomSelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows);
18        }
19    }
  
```

31. Copy the *SelectSkipAndTakeAsync* method from the *ProductRepository* (used like a base **class**) to the *ProductRepository* directly under the *DataRepository* folder. And then rename the method to

*MyCustomSelectSkipAndTakeAsync*. Also add the *using System Data* reference. Also add “*MyCustom*” to the stored procedure name: *acg6mvc\_Products\_MyCustomSelectSkipAndTake*.

```

91
92     /// <summary>
93     /// Selects Products table records sorted by the sortByExpression and returns records from the startRowIndex with rows (
94     /// </summary>
95     2 references
96     async Task<DataTable> IProductRepository.SelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows)
97     {
98         string storedProcName = "[dbo].[acg6mvc_Products_SelectSkipAndTake]";
99         return await this.SelectSharedAsync(storedProcName, null, null, sortByExpression, startRowIndex, rows);
100     }

```

```

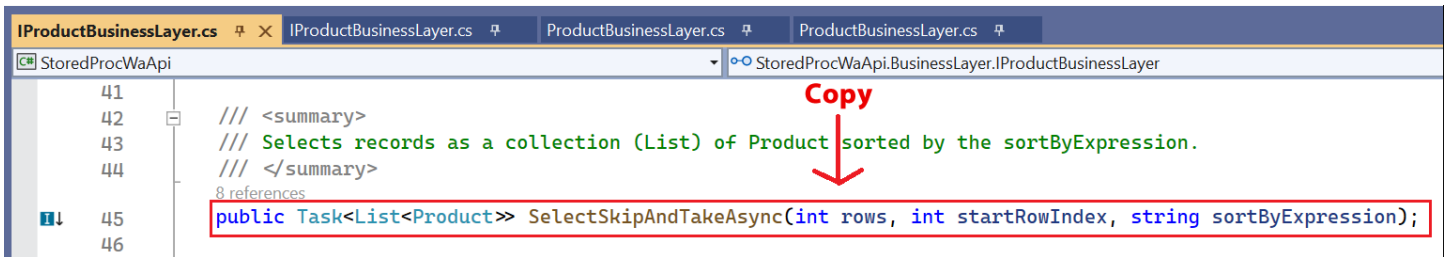
1  using StoredProcWaApi.DataRepository.Helper;
2  using System.Data.SqlClient;
3  using System.Data;
4
5  namespace StoredProcWaApi.DataRepository
6  {
7      /// <summary>
8      /// This file will not be overwritten.
9      /// You can put additional ProductRepository code in this class.
10     /// Here, you can implement additional code you placed in the IProductRepository interface found directly under the DataRepository
11     /// </summary>
12     3 references
13     public partial class ProductRepository
14     {
15         /// <summary>
16         /// This is just an example custom member. You can delete this.
17         /// </summary>
18         1 reference
19         string IProductRepository.ExampleRepositoryMember()...
20
21
22         2 references
23         async Task<DataTable> IProductRepository.MyCustomSelectSkipAndTakeAsync(string sortByExpression, int startRowIndex, int rows)
24         {
25             string storedProcName = "[dbo].[acg6mvc_Products_MyCustomSelectSkipAndTake]";
26             return await this.SelectSharedAsync(storedProcName, null, null, sortByExpression, startRowIndex, rows);
27         }
28     }

```

32. Create new *Business Layer* method(s). In *Visual Studio*, open the following:

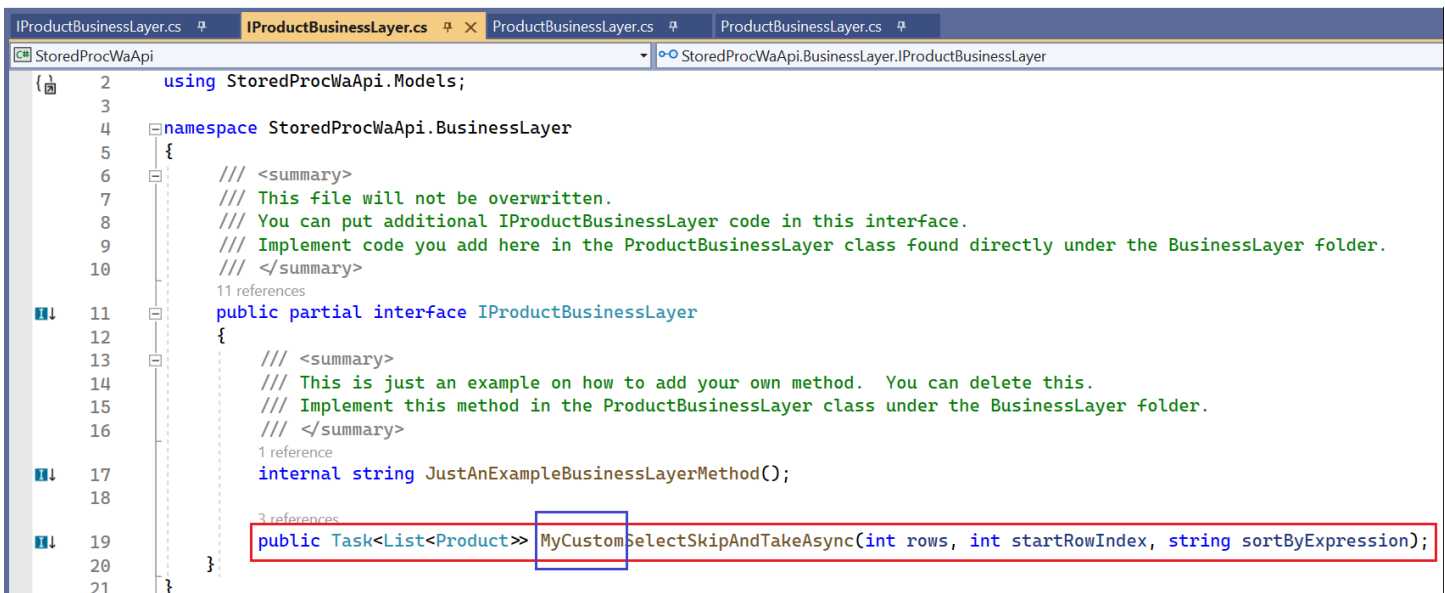
- IProductBusinessLayer* – An interface, found under the *BusinessLayer\Base\Interface* folder. **Used like a base interface.**
- IProductBusinessLayer* – An interface, found under the *BusinessLayer\Interface* folder. **We can add or update code here.**
- ProductBusinessLayer* – A class, found under the *BusinessLayer\Base* folder. **Used like a base class.**
- ProductBusinessLayer* – A class, found under the *BusinessLayer* folder. **We can add or update code here.**

33. Copy the *SelectSkipAndTakeAsync* method from the *IProductBusinessLayer* (used like a base **interface**) to the *IProductBusinessLayer* under the *BusinessLayer\Interface* folder. And then rename the method to ***MyCustomSelectSkipAndTakeAsync***.



```

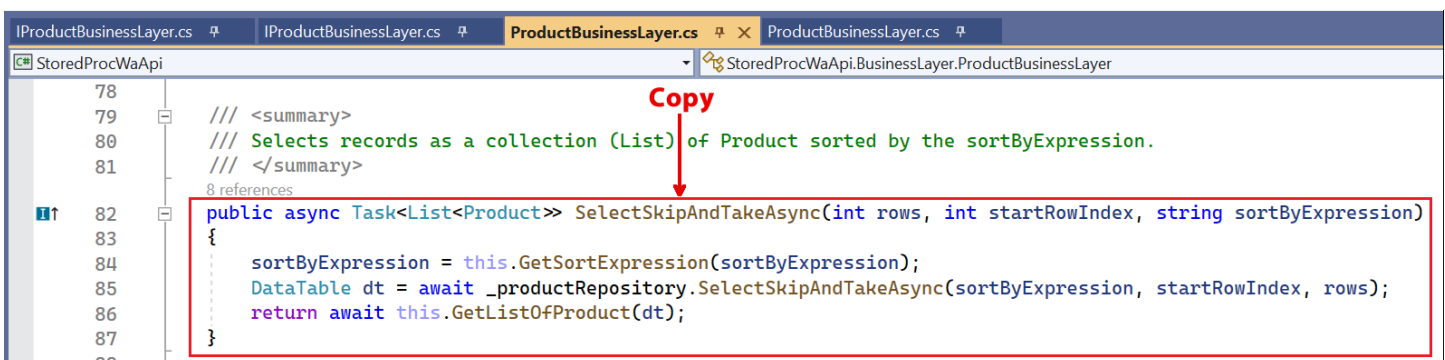
41
42     /// <summary>
43     /// Selects records as a collection (List) of Product sorted by the sortByExpression.
44     /// </summary>
45     public Task<List<Product>> SelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression);
46
  
```



```

2     using StoredProcWaApi.Models;
3
4     namespace StoredProcWaApi.BusinessLayer
5     {
6         /// <summary>
7         /// This file will not be overwritten.
8         /// You can put additional IProductBusinessLayer code in this interface.
9         /// Implement code you add here in the ProductBusinessLayer class found directly under the BusinessLayer folder.
10        /// </summary>
11        public partial interface IProductBusinessLayer
12        {
13            /// <summary>
14            /// This is just an example on how to add your own method. You can delete this.
15            /// Implement this method in the ProductBusinessLayer class under the BusinessLayer folder.
16            /// </summary>
17            internal string JustAnExampleBusinessLayerMethod();
18
19            public Task<List<Product>> MyCustomSelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression);
20        }
21    }
  
```

34. Copy the *SelectSkipAndTakeAsync*, *GetListOfProduct*, and *CreateProductFromDataRowAsync* methods from the *ProductBusinessLayer* (used like a base **class**) to the *ProductBusinessLayer* directly under the *BusinessLayer* folder. And then rename the methods to ***MyCustomSelectSkipAndTakeAsync***, ***MyCustomGetListOfProduct***, and ***MyCustomCreateProductFromDataRowAsync*** respectively. Also add the *using System Data* reference.



```

78
79     /// <summary>
80     /// Selects records as a collection (List) of Product sorted by the sortByExpression.
81     /// </summary>
82     public async Task<List<Product>> SelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
83     {
84         sortByExpression = this.GetSortExpression(sortByExpression);
85         DataTable dt = await _productRepository.SelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
86         return await this.GetListOfProduct(dt);
87     }
88
  
```

IPProductBusinessLayer.cs IPProductBusinessLayer.cs ProductBusinessLayer.cs ProductBusinessLayer.cs

StoredProcWaApi StoredProcWaApi.BusinessLayer.ProductBusinessLayer

```

292
293 /// <summary> Gets a List of Products based on the sql script.
8 references
299 private async Task<List<Product>> GetListOfProduct (DataTable dt)
300 {
301     List<Product> objProductsList = null;
302
303     // build the list of Products
304     if (dt != null && dt.Rows.Count > 0)
305     {
306         objProductsList = new List<Product>();
307
308         foreach (DataRow dr in dt.Rows)
309         {
310             Product objProduct = await this.CreateProductFromDataRowAsync(dr);
311             objProductsList.Add(objProduct);
312         }
313     }
314
315     return objProductsList;
316 }

```

Copy

IPProductBusinessLayer.cs IPProductBusinessLayer.cs ProductBusinessLayer.cs ProductBusinessLayer.cs

StoredProcWaApi StoredProcWaApi.BusinessLayer.ProductBusinessLayer

```

? references
321 private async Task<Product> CreateProductFromDataRowAsync(DataRow dr)
322 {
323     // instantiate the Product model
324     Product objProduct = new();
325
326     // assign values to the model
327     objProduct.ProductID = (int)dr["ProductID"];
328     objProduct.ProductName = dr["ProductName"].ToString();
329
330     if (dr["SupplierID"] != System.DBNull.Value)
331     {
332         int supplierID = (int)dr["SupplierID"];
333         objProduct.SupplierID = supplierID;
334         objProduct.Supplier = await _supplierBusinessLayer.SelectByPrimaryKeyAsync(supplierID);
335     }
336     else
337     {
338         objProduct.SupplierID = null;
339         objProduct.Supplier = null;
340     }
341
342     if (dr["CategoryID"] != System.DBNull.Value)
343     {
344         int categoryID = (int)dr["CategoryID"];
345         objProduct.CategoryID = categoryID;
346         objProduct.Category = await _categoryBusinessLayer.SelectByPrimaryKeyAsync(categoryID);
347     }
348     else
349     {
350         objProduct.CategoryID = null;
351         objProduct.Category = null;
352     }
353
354     if (dr["QuantityPerUnit"] != System.DBNull.Value)
355         objProduct.QuantityPerUnit = dr["QuantityPerUnit"].ToString();
356     else
357         objProduct.QuantityPerUnit = null;
358
359     if (dr["UnitPrice"] != System.DBNull.Value)
360         objProduct.UnitPrice = (decimal)dr["UnitPrice"];
361     else
362         objProduct.UnitPrice = null;
363
364     if (dr["UnitsInStock"] != System.DBNull.Value)
365         objProduct.UnitsInStock = (int)dr["UnitsInStock"];
366     else
367         objProduct.UnitsInStock = null;
368
369     if (dr["UnitsOnOrder"] != System.DBNull.Value)
370         objProduct.UnitsOnOrder = (int)dr["UnitsOnOrder"];
371     else
372         objProduct.UnitsOnOrder = null;
373
374     if (dr["ReorderLevel"] != System.DBNull.Value)
375         objProduct.ReorderLevel = (int)dr["ReorderLevel"];
376     else
377         objProduct.ReorderLevel = null;
378     objProduct.Discontinued = (bool)dr["Discontinued"];
379
380     return objProduct;
381 }

```

Copy

```

IProductBusinessLayer.cs  IProductBusinessLayer.cs  ProductBusinessLayer.cs  ProductBusinessLayer.cs
StoredProcWaApi  StoredProcWaApi.BusinessLayer.ProductBusinessLayer

1  using StoredProcWaApi.Models;
2  using System.Data;
3
4  namespace StoredProcWaApi.BusinessLayer
5  {
6      /// <summary>
7      /// Here, you can implement additional code you placed in the IProductBusinessLayer interface found directly under the Busin
8      /// This file will not be overwritten.
9      /// You can put additional ProductBusinessLayer code in this class.
10     /// </summary>
11     public partial class ProductBusinessLayer
12     {
13         /// <summary>
14         /// This is just an example on how to add your own method. You can delete this.
15         /// </summary>
16         string IProductBusinessLayer.JustAnExampleBusinessLayerMethod()
17         {
18         }
19     }
20
21     public async Task<List<Product>> MyCustomSelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
22     {
23         sortByExpression = this.GetSortExpression(sortByExpression);
24         DataTable dt = await _productRepository.MyCustomSelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
25         return await this.MyCustomGetListOfProduct(dt);
26     }
27
28     private async Task<List<Product>> MyCustomGetListOfProduct(DataTable dt)
29     {
30         List<Product> objProductsList = null;
31
32         // build the list of Products
33         if (dt != null && dt.Rows.Count > 0)
34         {
35             objProductsList = new List<Product>();
36
37             foreach (DataRow dr in dt.Rows)
38             {
39                 Product objProduct = await this.MyCustomCreateProductFromDataRowAsync(dr);
40                 objProductsList.Add(objProduct);
41             }
42         }
43
44         return objProductsList;
45     }
46
47     private async Task<Product> MyCustomCreateProductFromDataRowAsync(DataRow dr)
48     {
49         // instantiate the Product model
50         Product objProduct = new();
51
52         // assign values to the model
53         objProduct.ProductID = (int)dr["ProductID"];

```

35. Remove references for *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* in the **MyCustomCreateProductFromDataRowAsync** method, and then add references to *CompanyName* and *CategoryName*.

36.

```

47 private async Task<Product> MyCustomCreateProductFromDataRowAsync(DataRow dr)
48 {
49     // instantiate the Product model
50     Product objProduct = new();
51
52     // assign values to the model
53     objProduct.ProductID = (int)dr["ProductID"];
54     objProduct.ProductName = dr["ProductName"].ToString();
55
56     if (dr["SupplierID"] != System.DBNull.Value)
57     {
58         int supplierID = (int)dr["SupplierID"];
59         objProduct.SupplierID = supplierID;
60         objProduct.Supplier = await _supplierBusinessLayer.SelectByPrimaryKeyAsync(suppl
61     }
62     else
63     {
64         objProduct.SupplierID = null;
65         objProduct.Supplier = null;
66     }
67
68     if (dr["CategoryID"] != System.DBNull.Value)
69     {
70         int categoryID = (int)dr["CategoryID"];
71         objProduct.CategoryID = categoryID;
72         objProduct.Category = await _categoryBusinessLayer.SelectByPrimaryKeyAsync(categ
73     }
74     else
75     {
76         objProduct.CategoryID = null;
77         objProduct.Category = null;
78     }
79
80     if (dr["QuantityPerUnit"] != System.DBNull.Value)
81         objProduct.QuantityPerUnit = dr["QuantityPerUnit"].ToString();
82     else
83         objProduct.QuantityPerUnit = null;
84
85     if (dr["CompanyName"] != System.DBNull.Value)
86         objProduct.CompanyName = dr["CompanyName"].ToString();
87     else
88         objProduct.CompanyName = null;
89
90     if (dr["CategoryName"] != System.DBNull.Value)
91         objProduct.CategoryName = dr["CategoryName"].ToString();
92     else
93         objProduct.CategoryName = null;
94
95     objProduct.Discontinued = (bool)dr["Discontinued"];
96
97     return objProduct;
98 }

```

37. Go back to the *ProductController* under the *Controllers* folder. Rename the Web API call by prefixing it with **MyCustom** (**MyCustomSelectSkipAndTake**).

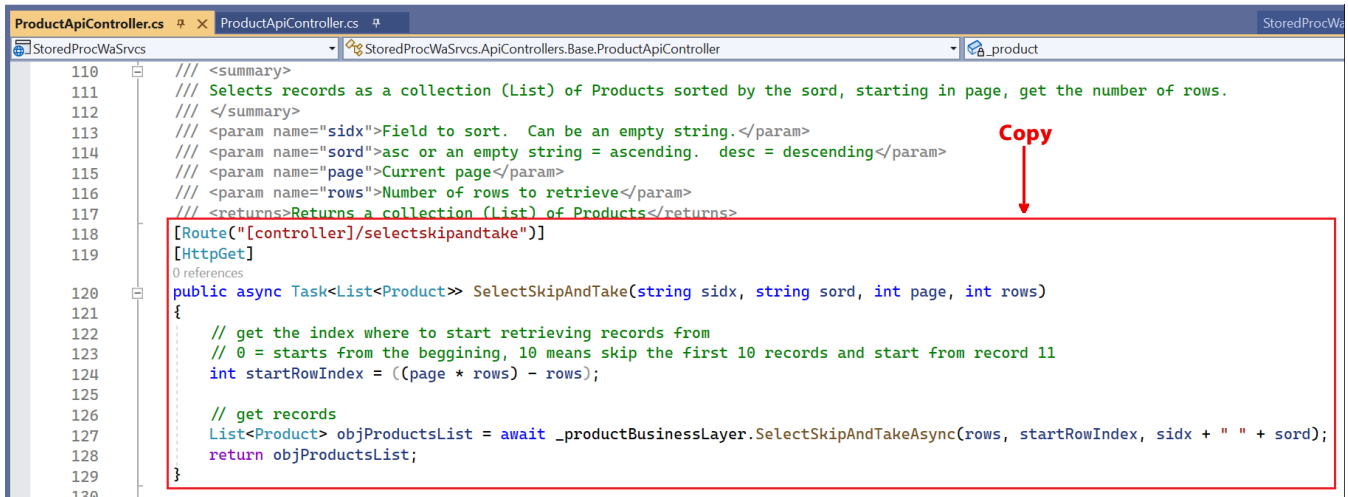
```

5 namespace StoredProcWa.Controllers
6 {
7     /// <summary>
8     /// You can put additional ProductController code in this class.
9     /// This file will not be overwritten.
10    /// </summary>
11
12    public partial class ProductController
13    {
14        0 references
15        public IActionResult MyCustomView(...)
16
17
18        0 references
19        public async Task<IActionResult> MyGridData(string sidx, string sord, int page, int rows)
20        {
21            // get the total number of records
22            string responseBody1 = await Functions.HttpClientGetAsync("ProductApi/GetRecordCount/");
23            int totalRecords = JsonConvert.DeserializeObject<int>(responseBody1);
24
25            // get the index where to start retrieving records from and the total number of pages
26            (int startRowIndex, int totalPages) = Functions.GetStartRowIndexAndTotalPages(page, rows, totalRecords);
27
28            // get records
29            List<Product> objProductsList = null;
30            string responseBody2 = await Functions.HttpClientGetAsync("ProductApi/MyCustomSelectSkipAndTake/?rows=" + rows + "&startRowIndex=" + startRowIndex + "&totalPages=" + totalPages);
31
32            // make sure responseBody2 is not empty before deserialization
33            if (!string.IsNullOrEmpty(responseBody2))
34                objProductsList = JsonConvert.DeserializeObject<List<Product>>(responseBody2);
35
36            // return json data for use by the jqgrid
37            return this.GetJsonData4MyCustomGrid(objProductsList, totalPages, page, totalRecords);
38        }
39    }

```

38. **Create a new Web API method.** In the *Web API Project (StoredProcWaSrvs)*, open the *ProductApiController.cs* under the *Controllers\Base* folder and then copy the *SelectSkipAndTake* method to the *ProductApiController.cs* directly under the *Controllers* folder. Rename the *Route*, *Method Names* adding "**MyCustom**".

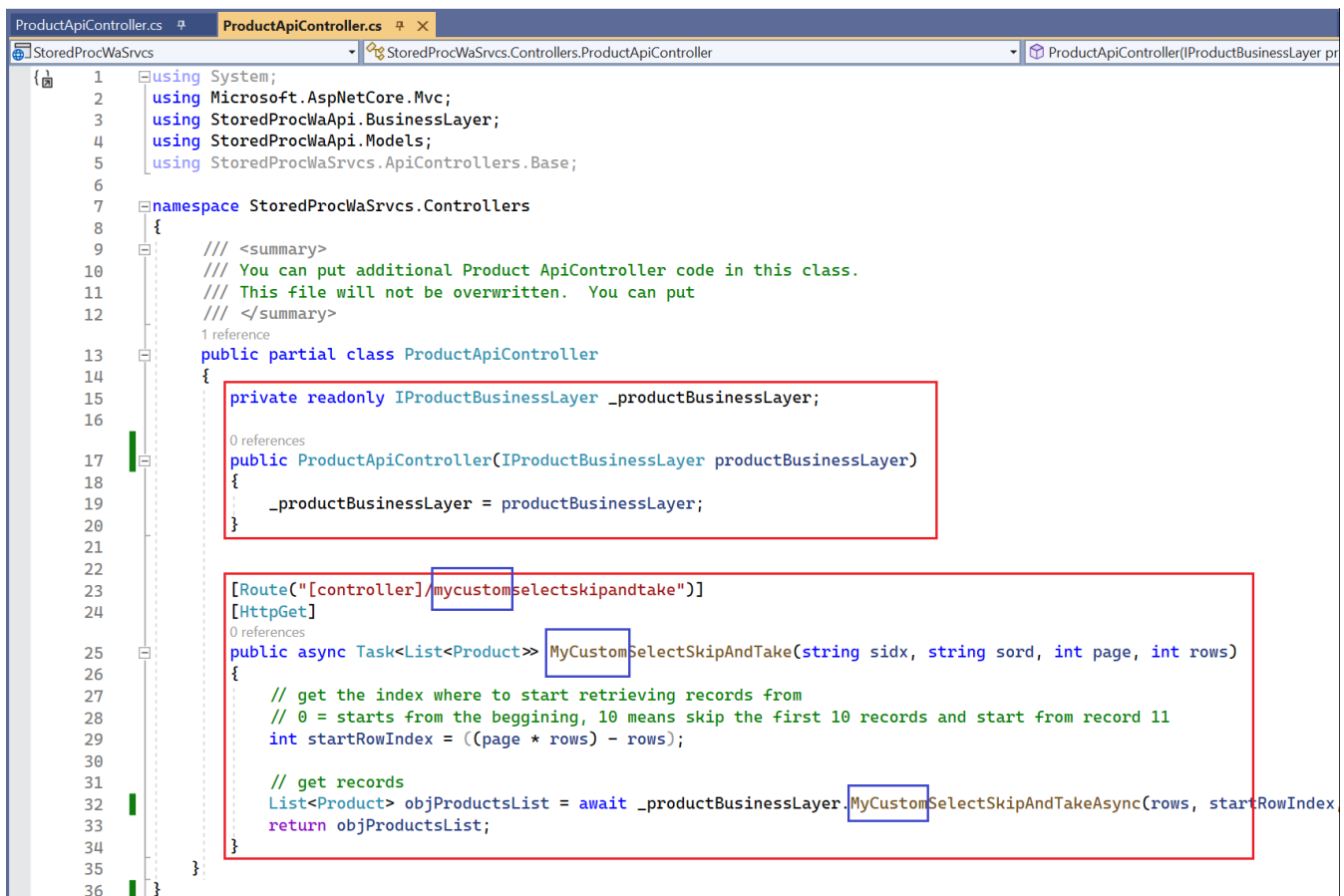
Add a *readonly* variable: *\_productBusinessLayer*. Add a *Constructor* injecting the *IProductBusinessLayer* to it.



```

110     /// <summary>
111     /// Selects records as a collection (List) of Products sorted by the sord, starting in page, get the number of rows.
112     </summary>
113     /// <param name="sidx">Field to sort. Can be an empty string.</param>
114     /// <param name="sord">asc or an empty string = ascending. desc = descending</param>
115     /// <param name="page">Current page</param>
116     /// <param name="rows">Number of rows to retrieve</param>
117     /// <returns>Returns a collection (List) of Products</returns>
118     [Route("[controller]/selectskipandtake")]
119     [HttpGet]
120     public async Task<List<Product>> SelectSkipAndTake(string sidx, string sord, int page, int rows)
121     {
122         // get the index where to start retrieving records from
123         // 0 = starts from the beggining, 10 means skip the first 10 records and start from record 11
124         int startRowIndex = ((page * rows) - rows);
125
126         // get records
127         List<Product> objProductsList = await _productBusinessLayer.SelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);
128         return objProductsList;
129     }
130

```



```

1  using System;
2  using Microsoft.AspNetCore.Mvc;
3  using StoredProcWaApi.BusinessLayer;
4  using StoredProcWaApi.Models;
5  using StoredProcWaSrvcs.ApiControllers.Base;
6
7  namespace StoredProcWaSrvcs.Controllers
8  {
9      /// <summary>
10     /// You can put additional Product ApiController code in this class.
11     /// This file will not be overwritten. You can put
12     /// </summary>
13     public partial class ProductApiController
14     {
15         private readonly IProductBusinessLayer _productBusinessLayer;
16
17         public ProductApiController(IProductBusinessLayer productBusinessLayer)
18         {
19             _productBusinessLayer = productBusinessLayer;
20         }
21
22         [Route("[controller]/mycustomselectskipandtake")]
23         [HttpGet]
24         public async Task<List<Product>> MyCustomSelectSkipAndTake(string sidx, string sord, int page, int rows)
25         {
26             // get the index where to start retrieving records from
27             // 0 = starts from the beggining, 10 means skip the first 10 records and start from record 11
28             int startRowIndex = ((page * rows) - rows);
29
30             // get records
31             List<Product> objProductsList = await _productBusinessLayer.MyCustomSelectSkipAndTakeAsync(rows, startRowIndex);
32             return objProductsList;
33         }
34     }
35 }
36

```



39. In the *MyCustomView.cshtml* MVC View, change the *colNames* to *Supplier* and *Category* respectively. Also, remove the *SupplierID* and *CategoryID* and replace with *CompanyName* and *CategoryName* respectively as shown below.

```

14 <script type="text/javascript">
15     var urlAndMethod = '/Product/Delete/';
16
17     $(function () {
18         // set jqgrid properties
19         $('#list-grid').jqGrid({
20             url: '/Product/MyGridData/',
21             datatype: 'json',
22             mtype: 'GET',
23             colNames: ['Product ID', 'Product Name', 'Company Name', 'Category Name', 'Quantity Per Unit', 'Discontinued', '', ''],
24             colModel: [
25                 { name: 'ProductID', index: 'ProductID', align: 'right' },
26                 { name: 'ProductName', index: 'ProductName', align: 'left' },
27                 { name: 'CompanyName', index: 'CompanyName', align: 'right' },
28                 { name: 'CategoryName', index: 'CategoryName', align: 'right' },
29                 { name: 'QuantityPerUnit', index: 'QuantityPerUnit', align: 'left' },
30                 { name: 'Discontinued', index: 'Discontinued', align: 'center', formatter: 'checkbox' },
31                 { name: 'editoperation', index: 'editoperation', align: 'center', width: 40, sortable: false, title: false },
32                 { name: 'deleteoperation', index: 'deleteoperation', align: 'center', width: 40, sortable: false, title: false }
33             ],

```

40. Run the *Web Application* by pressing *F5* while in Visual Studio 2022. And then go to the *MyCustomView* MVC View.

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

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

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. Download example shown here at: <https://junnark.com/CustomProjectSamples/acg6mvc/StoredProcWa.zip>

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

End of tutorial.