AJAX CRUD Operation With jQuery DataTables In ASP.NET MVC 5 For Beginners

* [](https://www.c-sharpcorner.com/members/ehsan-sajjad)

* [Ehsan Sajjad](https://www.c-sharpcorner.com/members/ehsan-sajjad)

[](https://www.c-sharpcorner.com/members/ehsan-sajjad)

[Ehsan Sajjad](https://www.c-sharpcorner.com/members/ehsan-sajjad)

[@Ehsan\_Sajjad](http://twitter.com/Ehsan_Sajjad)

**2835.2k1.7m**

Ehsan Sajjad is a Microsoft Certified Professional, Microsoft Certified C# Specialist and he is also among the...

Following  **10**Followers  **57**

+Follow

* Sep 03, 2018

* 242.3k

* 33

* 14

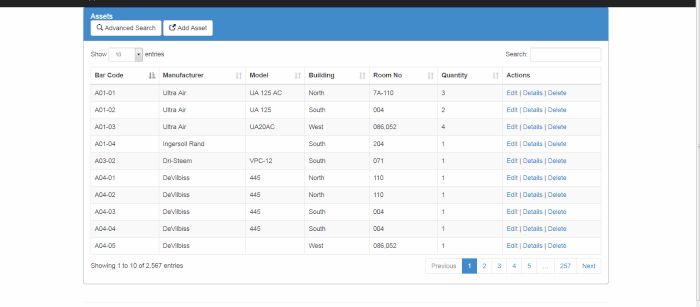
[**GridAjaxCRUDMVC.zip**](javascript:void(0);)

[Download Free .NET Java APIs for Office Documents](https://www.e-iceblue.com/download/free-products.html/?aff_id=100)

**Introduction**

I frequently come across questions on StackOverflow in which the questioners are able to create CRUD operations for their entity which is a normal form post, but they struggle implementing the CRUD via AJAX so that page does not reload completely for better User Experience.

In this post, I will be addressing this problem and we will learn how we can implement CRUD using jQuery DataTables in one page without any full page reloads.The end result will look something like:



**Background**

I have been writing last few posts about the usage of jQuery DataTables and it can be leveraged to build a Grid View like functionality in ASP.NET MVC. We saw how we can install it using NuGet Package Manager and implement a simple Grid using the plugin which provides us the essential features like Sorting, Searching and Paginations, then we saw how we can implement the paging, filtering and ordering; then we proceeded to see how advanced search can be added for more better user experience.

If someone wants to have refresher or is interested to read the previous posts, they can refer here.

* [Beginners Guide for Creating GridView in ASP.NET MVC 5](https://www.c-sharpcorner.com/article/beginners-guide-to-create-gridview-in-asp-net-mvc/)
* [Grid View with Server Side Filtering, Sorting and Paging in ASP.NET MVC 5](https://www.c-sharpcorner.com/article/grid-view-with-server-side-filtering-sorting-and-paging-in-asp-net-mvc-5/)
* [Grid View With Server Side Advanced Search Using jQuery DataTables In ASP.NET MVC 5](https://www.c-sharpcorner.com/article/grid-view-with-server-side-advanced-search-using-jquery-datatables-in-asp-net-mv/)

In this post, we will learn how we can add create, update and delete operations support in the Grid that we already implemented using jQuery DataTables.The end result will look something like:

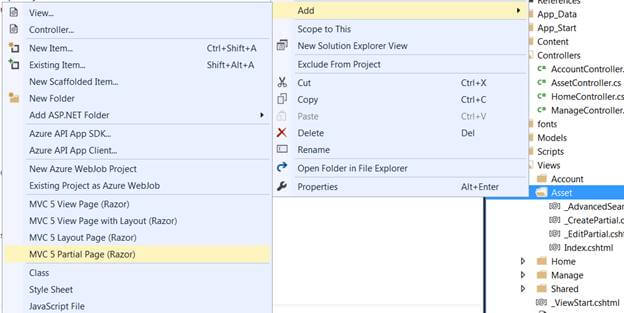
**Database Creation Script**

First of all, we will run the script for database and tables creation that would be essential for this post. Following is the script for that.

1. **CREATE** **DATABASE** [AdvancedSearchGridExampleMVC]
2. GO
3. **CREATE** **TABLE** [dbo].[FacilitySites] ([FacilitySiteID] UNIQUEIDENTIFIER NOT NULL,
4. [FacilityName]   NVARCHAR (**MAX**)   NULL,
5. [IsActive]       **BIT**              NOT NULL,
6. [CreatedBy]      UNIQUEIDENTIFIER NOT NULL,
7. [CreatedAt]      DATETIME         NOT NULL,
8. [ModifiedBy]     UNIQUEIDENTIFIER NULL,
9. [ModifiedAt]     DATETIME         NULL,
10. [IsDeleted]      **BIT**              NOT NULL
11. );
12. GO
14. **CREATE** **TABLE** [dbo].[Assets] (
15. [AssetID]                   UNIQUEIDENTIFIER NOT NULL,
16. [Barcode]                   NVARCHAR (**MAX**)   NULL,
17. [SerialNumber]              NVARCHAR (**MAX**)   NULL,
18. [PMGuide]                   NVARCHAR (**MAX**)   NULL,
19. [AstID]                     NVARCHAR (**MAX**)   NOT NULL,
20. [ChildAsset]                NVARCHAR (**MAX**)   NULL,
21. [GeneralAssetDescription]   NVARCHAR (**MAX**)   NULL,
22. [SecondaryAssetDescription] NVARCHAR (**MAX**)   NULL,
23. [Quantity]                  **INT**              NOT NULL,
24. [Manufacturer]              NVARCHAR (**MAX**)   NULL,
25. [ModelNumber]               NVARCHAR (**MAX**)   NULL,
26. [Building]                  NVARCHAR (**MAX**)   NULL,
27. [Floor]                     NVARCHAR (**MAX**)   NULL,
28. [Corridor]                  NVARCHAR (**MAX**)   NULL,
29. [RoomNo]                    NVARCHAR (**MAX**)   NULL,
30. [MERNo]                     NVARCHAR (**MAX**)   NULL,
31. [EquipSystem]               NVARCHAR (**MAX**)   NULL,
32. [Comments]                  NVARCHAR (**MAX**)   NULL,
33. [Issued]                    **BIT**              NOT NULL,
34. [FacilitySiteID]            UNIQUEIDENTIFIER NOT NULL
35. );
37. GO
39. **CREATE** NONCLUSTERED **INDEX** [IX\_FacilitySiteID]
40. **ON** [dbo].[Assets]([FacilitySiteID] **ASC**);
42. GO
43. **ALTER** **TABLE** [dbo].[Assets]
44. **ADD** **CONSTRAINT** [PK\_dbo.Assets] **PRIMARY** **KEY** CLUSTERED ([AssetID] **ASC**);
46. GO
48. **ALTER** **TABLE** [dbo].[Assets]
49. **ADD** **CONSTRAINT** [FK\_dbo.Assets\_dbo.FacilitySites\_FacilitySiteID] **FOREIGN** **KEY** ([FacilitySiteID]) \_
50. **REFERENCES** [dbo].[FacilitySites] ([FacilitySiteID]) **ON** **DELETE** **CASCADE**;
52. GO

You can find the script in attached source code in file name dbScript.sql which also contains sample data to get started quickly. Running it will create the DB and will add some sample data in the tables as well.

**Create/Insert Operation**

For the insertion part, first of all, we need to create a Partial View in *Views >> Asset* by right clicking the Asset folder and navigate to Add >> MVC 5 Partial Page (Razor) like the below screenshot.  
  
  
  
Open the Container View in which asset rows are being rendered which *\_AssetsPartial.cshtml*located in*Views >> Asset* directory and add the HTML for the Add Asset button which will open up a popup for inserting a new asset row in database,

1. <button type="button" **class**="btn btn-default btn-md" data-toggle="modal" data-url="@Url.Action("Create","Asset")" id="btnCreateAsset">
2. <span **class**="glyphicon glyphicon-new-window" aria-hidden="true"></span> Add Asset
3. </button>

Now, open the Index View of Asset which is *Views >> Asset >> index.csthtml*and in scripts section add the following jQuery code for the button click event which we just created above, it will call the create get action and will display the partial view in bootstrap modal to user, the code for which is,

1. $("#btnCreateAsset").on("click", **function** () {
3. **var** url = $(**this**).data("url");
5. $.get(url, **function** (data) {
6. $('#createAssetContainer').html(data);
8. $('#createAssetModal').modal('show');
9. });
11. });

We also need to add bootstrap modal with container div in which we will be loading create partial view of Asset, so we will add the following html at the end of Index.cshtml view,

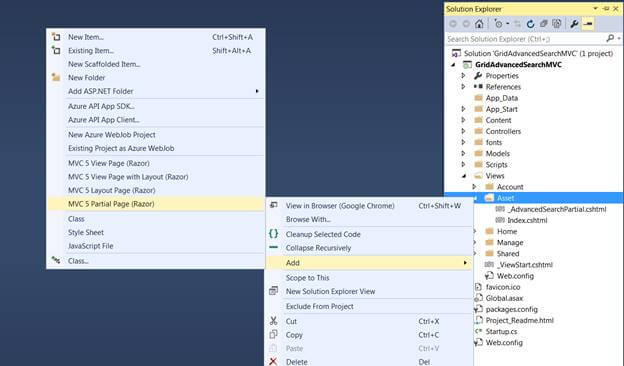
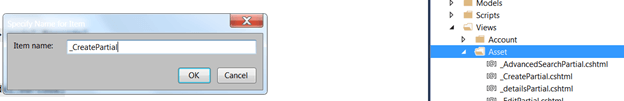
1. <div **class**="modal fade" id="createAssetModal" tabindex="-1" role="dialog" aria-labelledby="CreateAssetModal" aria-hidden="true" data-backdrop="static">
2. <div id="createAssetContainer">
3. </div>
4. </div>

**Defining Add GET Action**

Add action method for Asset Creation get request in the Asset controller file located at *Controllers >> AssetController.cs* which would be called by the above jQuery code, inside the action method we are populating the ViewModel and passing the instance back to PartialView as normally we do in mvc based application,

1. // GET: Asset/Create
2. **public** ActionResult Create()
3. {
4. **var** model = **new** AssetViewModel();
5. model.FacilitySitesSelectList = GetFacilitiySitesSelectList();
6. **return** View("\_CreatePartial", model);
7. }

**Add Partial View Creation**

Add a new Partial view in project in Asset Views for Create form, for that right click the Asset folder under Views in the Solution Explorer  
  
  
  
Enter the name of partial view to be \_CreatePartial in textbox or whatever name you think would be better in your case.  
  
 **Writing Add Partial View Code**

Now we will write the partial view that we created in previous step \_CreatePatial, add the following code in that View,

1. @model GridAjaxCRUDMVC.Models.AssetViewModel
2. @{
3. Layout = **null**;
4. }
6. <div **class**="modal-dialog">
7. <div **class**="modal-content">
8. <div **class**="modal-header">
9. <button type="button" **class**="close" data-dismiss="modal" aria-hidden="true">×</button>
10. <h4 **class**="modal-title">Add Asset</h4>
11. </div>
12. @using (Ajax.BeginForm("Create", "Asset", **null**, **new** AjaxOptions { HttpMethod = "Post", OnSuccess = "CreateAssetSuccess" }, **new** {  @**class** = "form-horizontal", role = "form" }))
13. {
14. <div **class**="modal-body">
15. <div **class**="form-horizontal">
16. @Html.ValidationSummary(**true**, "", **new** { @**class** = "text-danger" })
17. @Html.HiddenFor(model => model.AssetID, **new** { Value = Guid.NewGuid() })
18. <div **class**="form-group">
19. @Html.LabelFor(model => model.Barcode, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
20. <div **class**="col-md-10">
21. @Html.EditorFor(model => model.Barcode, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
22. @Html.ValidationMessageFor(model => model.Barcode, "", **new** { @**class** = "text-danger" })
23. </div>
24. </div>
26. <div **class**="form-group">
27. @Html.LabelFor(model => model.SerialNumber, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
28. <div **class**="col-md-10">
29. @Html.EditorFor(model => model.SerialNumber, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
30. @Html.ValidationMessageFor(model => model.SerialNumber, "", **new** { @**class** = "text-danger" })
31. </div>
32. </div>
34. <div **class**="form-group">
35. @Html.LabelFor(model => model.FacilitySiteID, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
36. <div **class**="col-md-10">
37. @Html.DropDownListFor(model => model.FacilitySiteID, Model.FacilitySitesSelectList, "Select One", **new** { htmlAttributes = **new** { @**class** = "form-control" } })
38. @Html.ValidationMessageFor(model => model.FacilitySiteID, "", **new** { @**class** = "text-danger" })
39. </div>
40. </div>
42. <div **class**="form-group">
43. @Html.LabelFor(model => model.PMGuide, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
44. <div **class**="col-md-10">
45. @Html.EditorFor(model => model.PMGuide, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
46. @Html.ValidationMessageFor(model => model.PMGuide, "", **new** { @**class** = "text-danger" })
47. </div>
48. </div>
50. <div **class**="form-group">
51. @Html.LabelFor(model => model.AstID, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
52. <div **class**="col-md-10">
53. @Html.EditorFor(model => model.AstID, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
54. @Html.ValidationMessageFor(model => model.AstID, "", **new** { @**class** = "text-danger" })
55. </div>
56. </div>
58. <div **class**="form-group">
59. @Html.LabelFor(model => model.ChildAsset, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
60. <div **class**="col-md-10">
61. @Html.EditorFor(model => model.ChildAsset, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
62. @Html.ValidationMessageFor(model => model.ChildAsset, "", **new** { @**class** = "text-danger" })
63. </div>
64. </div>
66. <div **class**="form-group">
67. @Html.LabelFor(model => model.GeneralAssetDescription, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
68. <div **class**="col-md-10">
69. @Html.EditorFor(model => model.GeneralAssetDescription, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
70. @Html.ValidationMessageFor(model => model.GeneralAssetDescription, "", **new** { @**class** = "text-danger" })
71. </div>
72. </div>
74. <div **class**="form-group">
75. @Html.LabelFor(model => model.SecondaryAssetDescription, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
76. <div **class**="col-md-10">
77. @Html.EditorFor(model => model.SecondaryAssetDescription, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
78. @Html.ValidationMessageFor(model => model.SecondaryAssetDescription, "", **new** { @**class** = "text-danger" })
79. </div>
80. </div>
82. <div **class**="form-group">
83. @Html.LabelFor(model => model.Quantity, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
84. <div **class**="col-md-10">
85. @Html.EditorFor(model => model.Quantity, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
86. @Html.ValidationMessageFor(model => model.Quantity, "", **new** { @**class** = "text-danger" })
87. </div>
88. </div>
90. <div **class**="form-group">
91. @Html.LabelFor(model => model.Manufacturer, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
92. <div **class**="col-md-10">
93. @Html.EditorFor(model => model.Manufacturer, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
94. @Html.ValidationMessageFor(model => model.Manufacturer, "", **new** { @**class** = "text-danger" })
95. </div>
96. </div>
98. <div **class**="form-group">
99. @Html.LabelFor(model => model.ModelNumber, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
100. <div **class**="col-md-10">
101. @Html.EditorFor(model => model.ModelNumber, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
102. @Html.ValidationMessageFor(model => model.ModelNumber, "", **new** { @**class** = "text-danger" })
103. </div>
104. </div>
106. <div **class**="form-group">
107. @Html.LabelFor(model => model.Building, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
108. <div **class**="col-md-10">
109. @Html.EditorFor(model => model.Building, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
110. @Html.ValidationMessageFor(model => model.Building, "", **new** { @**class** = "text-danger" })
111. </div>
112. </div>
114. <div **class**="form-group">
115. @Html.LabelFor(model => model.Floor, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
116. <div **class**="col-md-10">
117. @Html.EditorFor(model => model.Floor, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
118. @Html.ValidationMessageFor(model => model.Floor, "", **new** { @**class** = "text-danger" })
119. </div>
120. </div>
122. <div **class**="form-group">
123. @Html.LabelFor(model => model.Corridor, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
124. <div **class**="col-md-10">
125. @Html.EditorFor(model => model.Corridor, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
126. @Html.ValidationMessageFor(model => model.Corridor, "", **new** { @**class** = "text-danger" })
127. </div>
128. </div>
130. <div **class**="form-group">
131. @Html.LabelFor(model => model.RoomNo, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
132. <div **class**="col-md-10">
133. @Html.EditorFor(model => model.RoomNo, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
134. @Html.ValidationMessageFor(model => model.RoomNo, "", **new** { @**class** = "text-danger" })
135. </div>
136. </div>
138. <div **class**="form-group">
139. @Html.LabelFor(model => model.MERNo, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
140. <div **class**="col-md-10">
141. @Html.EditorFor(model => model.MERNo, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
142. @Html.ValidationMessageFor(model => model.MERNo, "", **new** { @**class** = "text-danger" })
143. </div>
144. </div>
146. <div **class**="form-group">
147. @Html.LabelFor(model => model.EquipSystem, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
148. <div **class**="col-md-10">
149. @Html.EditorFor(model => model.EquipSystem, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
150. @Html.ValidationMessageFor(model => model.EquipSystem, "", **new** { @**class** = "text-danger" })
151. </div>
152. </div>
154. <div **class**="form-group">
155. @Html.LabelFor(model => model.Comments, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
156. <div **class**="col-md-10">
157. @Html.EditorFor(model => model.Comments, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
158. @Html.ValidationMessageFor(model => model.Comments, "", **new** { @**class** = "text-danger" })
159. </div>
160. </div>
162. <div **class**="form-group">
163. @Html.LabelFor(model => model.Issued, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
164. <div **class**="col-md-10">
165. <div **class**="checkbox">
166. @Html.EditorFor(model => model.Issued)
167. @Html.ValidationMessageFor(model => model.Issued, "", **new** { @**class** = "text-danger" })
168. </div>
169. </div>
170. </div>

173. </div>
174. </div>
175. <div **class**="modal-footer">
176. <div **class**="form-group">
177. <div **class**="col-md-offset-2 col-md-10">
178. <button type="button" **class**="btn btn-default" data-dismiss="modal">Close</button>
179. <input type="submit" **class**="btn btn-primary" value="Save" />
180. </div>
181. </div>
182. </div>
183. }
184. </div>
185. </div>

**Writing Add/Create POST Action**

The important thing to note in the view code is the Ajax.BeginForm helper portion as the class Ajax should be enough to understand that it will post the model back to controller action via Ajax, which means that whole will not refresh i.e. no full postback will happen in terms on asp.net web forms.

The post action is simply mapping ViewModel object to the Model and then saving it in the repository,

1. // POST: Asset/Create
2. [HttpPost]
3. **public** async Task<ActionResult> Create(AssetViewModel assetVM)
4. {
5. **if** (!ModelState.IsValid)
6. **return** View("\_CreatePartial", assetVM);
8. Asset asset = MaptoModel(assetVM);
10. DbContext.Assets.Add(asset);
11. **var** task = DbContext.SaveChangesAsync();
12. await task;
14. **if** (task.Exception != **null**)
15. {
16. ModelState.AddModelError("", "Unable to add the Asset");
17. **return** View("\_CreatePartial", assetVM);
18. }
20. **return** Content("success");
21. }

We are trying to save the newly entered asset in the database and tracking if it is saved successfully or not and if it saves successfully we are returning a string message “success” back to client side and in the success call back of Ajax Form we will be checking if operation was successful or not to do UI changes according to that.

**Implementing Ajax Callback Function**

We can see that in BeginForm helper method parameters we are specifying JavaScript function to be called when the Ajax Form successfully returns back from server i.e. OnSuccess = "CreateAssetSuccess"

Now let’s go to the index.cshtml view of Asset and define the success callback implementation,

1. **function** CreateAssetSuccess(data) {
3. **if** (data != "success") {
4. $('#createAssetContainer').html(data);
5. **return**;
6. }
7. $('#createAssetModal').modal('hide');
8. $('#createAssetContainer').html("");
9. assetListVM.refresh();
11. }

What we are doing here is if the operation is not successful we are updating the client side html to notify the user about the failure of creation and if the insert goes successful we are closing the modal popup and refreshing the Grid to display the up to date information.

**Edit/Update Operation**

Until now we should be able to run the application and successfully add new assets in the database via Ajax using the bootstrap modal which we created above, now let’s move to the update part of the Asset.

We will be adding another column in the datatable columns collection which will contain hyper link that will navigate to Edit View, but as we are using Ajax and bootstrap modal, we will be doing updates as well same way, so no redirect will be involved. Let’s get started.

Open the Index.csthml file and add a new column in the columns collection in jQuery datatable initialization, after updating the columns array our code would look like,

1. "columns": [
2. { "title": "Bar Code", "data": "BarCode", "searchable": **true** },
3. { "title": "Manufacturer", "data": "Manufacturer", "searchable": **true** },
4. { "title": "Model", "data": "ModelNumber", "searchable": **true** },
5. { "title": "Building", "data": "Building", "searchable": **true** },
6. { "title": "Room No", "data": "RoomNo" },
7. { "title": "Quantity", "data": "Quantity" },
8. {
9. "title": "Actions",
10. "data": "AssetID",
11. "searchable": **false**,
12. "sortable": **false**,
13. "render": **function** (data, type, full, meta) {
14. **return** '<a href="@Url.Action("Edit","Asset")?id=' + data + '" class="editAsset">Edit</a>';
15. }
16. }
17. ]

We are setting the header of new column to display Actions as title and we would need to disable the searching and sorting for this column, as it is for edit operation and it does not makes sense to enable sorting and searching on this column. Next we are defining the render method of the column and we are generating anchor link html which could call the Edit action of Asset controller and will pass the current asset id to pull the information of it and display in the Edit View.

**Defining Edit/Update GET Action**

After doing the datatable js changes, now we need to create a get action method which will pull the asset record from the database and will display it for editing to the user in a popup.

Let’s implement the Edit get action of it,

1. **public** ActionResult Edit(Guid id)
2. {
3. **var** asset = DbContext.Assets.FirstOrDefault(x => x.AssetID == id);
5. AssetViewModel assetViewModel = MapToViewModel(asset);
7. **if** (Request.IsAjaxRequest())
8. **return** PartialView("\_EditPartial",assetViewModel);
9. **return** View(assetViewModel);
10. }

The action simply retrieves the row from the database and after converting it to ViewModel passes it to back to partial view to rendered or returned back to the client side for processing, as no post backs would happen it will generate the html and will send the html back in response of ajax call which client side will handle and decide where to put that html.

**Handling Action Links Events on Client Side using JQuery**

From the column render function you can see that there is class applied on anchor link called EditAsset, it is defined because jQuery event handler will be applied to the anchor link and Ajax call will be sent to server, let’s define the event handler for that in Index View:

1. $('#assets-data-table').on("click", ".editAsset", **function** (event) {
3. event.preventDefault();
5. **var** url = $(**this**).attr("href");
7. $.get(url, **function** (data) {
8. $('#editAssetContainer').html(data);
10. $('#editAssetModal').modal('show');
11. });
13. });

**Addition of Bootstrap Modal**

Now add the bootstrap modal html in the Index View which will be placeholder for loading the Edit View, define it just after create bootstrap modal html,

1. <div **class**="modal fade" id="editAssetModal" tabindex="-1" role="dialog" aria-labelledby="EditAssetModal" aria-hidden="true" data-backdrop="static">
2. <div id="editAssetContainer">
3. </div>
4. </div>

**Edit/Update Partial View Creation**

Create another new partial view following the same steps which we did for adding \_CreatePartial.cshtml which was for Create Asset, So add a new Partial View in the Asset folder in Views with name \_EditPartial.cshtml and add the following code in it,

1. @model TA\_UM.ViewModels.AssetViewModel
2. @{
3. Layout = **null**;
4. }
5. <div **class**="modal-dialog">
6. <div **class**="modal-content">
7. <div **class**="modal-header">
8. <button type="button" **class**="close" data-dismiss="modal" aria-hidden="true">×</button>
9. <h4 **class**="modal-title">Edit Asset</h4>
10. </div>
11. @using (Ajax.BeginForm("Edit", "Asset", **null**, **new** AjaxOptions { HttpMethod="Post", OnSuccess = "UpdateAssetSuccess" }, **new** { @**class** = "form-horizontal", role = "form" }))
12. {
13. <div **class**="modal-body">
14. <div **class**="form-horizontal">
15. @Html.ValidationSummary(**true**, "", **new** { @**class** = "text-danger" })
16. @Html.HiddenFor(model => model.AssetID)
18. <div **class**="form-group">
19. @Html.LabelFor(model => model.Barcode, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
20. <div **class**="col-md-10">
21. @Html.EditorFor(model => model.Barcode, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
22. @Html.ValidationMessageFor(model => model.Barcode, "", **new** { @**class** = "text-danger" })
23. </div>
24. </div>
25. <div **class**="form-group">
26. @Html.LabelFor(model => model.SerialNumber, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
27. <div **class**="col-md-10">
28. @Html.EditorFor(model => model.SerialNumber, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
29. @Html.ValidationMessageFor(model => model.SerialNumber, "", **new** { @**class** = "text-danger" })
30. </div>
31. </div>
32. <div **class**="form-group">
33. @Html.LabelFor(model => model.FacilitySiteID, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
34. <div **class**="col-md-10">
35. @Html.DropDownListFor(model => model.FacilitySiteID,Model.FacilitySitesSelectList, "Select One", **new** { htmlAttributes = **new** { @**class** = "form-control" } })
36. @Html.ValidationMessageFor(model => model.FacilitySiteID, "", **new** { @**class** = "text-danger" })
37. </div>
38. </div>
40. <div **class**="form-group">
41. @Html.LabelFor(model => model.PMGuide, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
42. <div **class**="col-md-10">
43. @Html.EditorFor(model => model.PMGuide, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
44. @Html.ValidationMessageFor(model => model.PMGuide, "", **new** { @**class** = "text-danger" })
45. </div>
46. </div>
48. <div **class**="form-group">
49. @Html.LabelFor(model => model.AstID, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
50. <div **class**="col-md-10">
51. @Html.EditorFor(model => model.AstID, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
52. @Html.ValidationMessageFor(model => model.AstID, "", **new** { @**class** = "text-danger" })
53. </div>
54. </div>
56. <div **class**="form-group">
57. @Html.LabelFor(model => model.ChildAsset, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
58. <div **class**="col-md-10">
59. @Html.EditorFor(model => model.ChildAsset, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
60. @Html.ValidationMessageFor(model => model.ChildAsset, "", **new** { @**class** = "text-danger" })
61. </div>
62. </div>
63. <div **class**="form-group">
64. @Html.LabelFor(model => model.GeneralAssetDescription, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
65. <div **class**="col-md-10">
66. @Html.EditorFor(model => model.GeneralAssetDescription, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
67. @Html.ValidationMessageFor(model => model.GeneralAssetDescription, "", **new** { @**class** = "text-danger" })
68. </div>
69. </div>
71. <div **class**="form-group">
72. @Html.LabelFor(model => model.SecondaryAssetDescription, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
73. <div **class**="col-md-10">
74. @Html.EditorFor(model => model.SecondaryAssetDescription, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
75. @Html.ValidationMessageFor(model => model.SecondaryAssetDescription, "", **new** { @**class** = "text-danger" })
76. </div>
77. </div>
79. <div **class**="form-group">
80. @Html.LabelFor(model => model.Quantity, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
81. <div **class**="col-md-10">
82. @Html.EditorFor(model => model.Quantity, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
83. @Html.ValidationMessageFor(model => model.Quantity, "", **new** { @**class** = "text-danger" })
84. </div>
85. </div>
86. <div **class**="form-group">
87. @Html.LabelFor(model => model.Manufacturer, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
88. <div **class**="col-md-10">
89. @Html.EditorFor(model => model.Manufacturer, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
90. @Html.ValidationMessageFor(model => model.Manufacturer, "", **new** { @**class** = "text-danger" })
91. </div>
92. </div>
94. <div **class**="form-group">
95. @Html.LabelFor(model => model.ModelNumber, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
96. <div **class**="col-md-10">
97. @Html.EditorFor(model => model.ModelNumber, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
98. @Html.ValidationMessageFor(model => model.ModelNumber, "", **new** { @**class** = "text-danger" })
99. </div>
100. </div>
102. <div **class**="form-group">
103. @Html.LabelFor(model => model.Building, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
104. <div **class**="col-md-10">
105. @Html.EditorFor(model => model.Building, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
106. @Html.ValidationMessageFor(model => model.Building, "", **new** { @**class** = "text-danger" })
107. </div>
108. </div>
110. <div **class**="form-group">
111. @Html.LabelFor(model => model.Floor, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
112. <div **class**="col-md-10">
113. @Html.EditorFor(model => model.Floor, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
114. @Html.ValidationMessageFor(model => model.Floor, "", **new** { @**class** = "text-danger" })
115. </div>
116. </div>
117. <div **class**="form-group">
118. @Html.LabelFor(model => model.Corridor, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
119. <div **class**="col-md-10">
120. @Html.EditorFor(model => model.Corridor, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
121. @Html.ValidationMessageFor(model => model.Corridor, "", **new** { @**class** = "text-danger" })
122. </div>
123. </div>
125. <div **class**="form-group">
126. @Html.LabelFor(model => model.RoomNo, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
127. <div **class**="col-md-10">
128. @Html.EditorFor(model => model.RoomNo, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
129. @Html.ValidationMessageFor(model => model.RoomNo, "", **new** { @**class** = "text-danger" })
130. </div>
131. </div>
133. <div **class**="form-group">
134. @Html.LabelFor(model => model.MERNo, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
135. <div **class**="col-md-10">
136. @Html.EditorFor(model => model.MERNo, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
137. @Html.ValidationMessageFor(model => model.MERNo, "", **new** { @**class** = "text-danger" })
138. </div>
139. </div>
141. <div **class**="form-group">
142. @Html.LabelFor(model => model.EquipSystem, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
143. <div **class**="col-md-10">
144. @Html.EditorFor(model => model.EquipSystem, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
145. @Html.ValidationMessageFor(model => model.EquipSystem, "", **new** { @**class** = "text-danger" })
146. </div>
147. </div>
148. <div **class**="form-group">
149. @Html.LabelFor(model => model.Comments, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
150. <div **class**="col-md-10">
151. @Html.EditorFor(model => model.Comments, **new** { htmlAttributes = **new** { @**class** = "form-control" } })
152. @Html.ValidationMessageFor(model => model.Comments, "", **new** { @**class** = "text-danger" })
153. </div>
154. </div>
156. <div **class**="form-group">
157. @Html.LabelFor(model => model.Issued, htmlAttributes: **new** { @**class** = "control-label col-md-2" })
158. <div **class**="col-md-10">
159. <div **class**="checkbox">
160. @Html.EditorFor(model => model.Issued)
161. @Html.ValidationMessageFor(model => model.Issued, "", **new** { @**class** = "text-danger" })
162. </div>
163. </div>
164. </div>

167. </div>
168. </div>
169. <div **class**="modal-footer">
170. <div **class**="form-group">
171. <div **class**="col-md-offset-2 col-md-10">
172. <button type="button" **class**="btn btn-default" data-dismiss="modal">Close</button>
173. <input type="submit" **class**="btn btn-primary" value="Save changes"/>
174. </div>
175. </div>
176. </div>
177. }
178. </div>
179. </div>

The Edit View is also pretty same as we had for Insert except it would be posting to different action which would be responsible for handling the updates of a particular Asset row.

**Implementing Edit/Update Post Action**

Now let’s implement the post action of Edit,

1. [HttpPost]
2. **public** async Task<ActionResult> Edit(AssetViewModel assetVM)
3. {
5. assetVM.FacilitySitesSelectList = GetFacilitiySitesSelectList(assetVM.FacilitySiteID);
6. **if** (!ModelState.IsValid)
7. {
8. Response.StatusCode = (**int**)HttpStatusCode.BadRequest;
9. **return** View(Request.IsAjaxRequest() ? "\_EditPartial" : "Edit", assetVM);
10. }
12. Asset asset = MaptoModel(assetVM);
14. DbContext.Assets.Attach(asset);
15. DbContext.Entry(asset).State = EntityState.Modified;
16. **var** task = DbContext.SaveChangesAsync();
17. await task;
19. **if** (task.Exception != **null**)
20. {
21. ModelState.AddModelError("", "Unable to update the Asset");
22. Response.StatusCode = (**int**)HttpStatusCode.BadRequest;
23. **return** View(Request.IsAjaxRequest() ? "\_EditPartial" : "Edit", assetVM);
24. }
26. **if**(Request.IsAjaxRequest())
27. {
28. **return** Content("success");
29. }
31. **return** RedirectToAction("Index");
33. }

**Handling Update Ajax Success CallBack**

In Index.cshtml implement the OnSuccess callback function which will be called when the asset gets updated successfully, in the call back function modal would be closed and the form will be cleared so that if user opens for editing again the fresh html would be fetched and updated information will be displayed in View and of course datatable will also be refreshed to display the latest updates:

1. /\*\*\*\* Edit Asset Ajax Form CallBack \*\*\*\*\*\*\*\*/
3. **function** UpdateAssetSuccess(data) {
5. **if** (data != "success") {
6. $('#editAssetContainer').html(data);
7. **return**;
8. }
9. $('#editAssetModal').modal('hide');
10. $('#editAssetContainer').html("");
11. assetListVM.refresh();
13. }

The same approach will be followed for the details and delete action, let’s see the delete portion, first of all open the Index view and let’s add the details and delete action hyperlinks in the render function where we defined the edit link above,

1. "render": **function** (data, type, full, meta) {
2. **return** '<a href="@Url.Action("Edit","Asset")?id=' + data + '" class="editAsset">Edit</a> | <a href="@Url.Action("Details","Asset")?id=' + data + '">Details</a> | <a href="@Url.Action("Delete","Asset")?id=' + data + '">Delete</a>';
3. }

Now the action column will contain three hyperlinks for Edit, Details and Delete of Asset.

**Retrieve and Delete Operation**

At this stage we should be able to see the insert and update functionality working correctly, now we will move to the deletion part to see how we implement the Deletion part for Assets. For doing that, we will need to add class to the hyperlinks which we are generated in the render function for the column in which links will be appearing, let’s do that first, we need to define the render property for the last column in columns array of the DataTables initialization code, and we will define the how the column value should be rendered,

1. "render": **function** (data, type, full, meta) {
2. **return** '<a href="@Url.Action("Edit","Asset")?id=' + data + '" class="editAsset">Edit</a> | <a href="@Url.Action("Details","Asset")?id=' + data + '" class="detailsAsset">Details</a> | <a href="@Url.Action("Delete","Asset")?id=' + data + '" class="deleteAsset">Delete</a>';
3. }

We have added the detailsAsset and deleteAsset classes to the respective anchor tags so that we can bind the events using jQuery and do some logic to display details or delete particular asset.

After doing the above step, now we will be writing the events to handle the click of these two hyperlinks. We will have to write the following code to achieve it,

1. $('#assets-data-table').on("click", ".detailsAsset", **function** (event) {
3. event.preventDefault();
5. **var** url = $(**this**).attr("href");
7. $.get(url, **function** (data) {
8. $('#detailsAssetContainer').html(data);
10. $('#detailsAssetModal').modal('show');
11. });

Now as the handler for details tag is placed, let’s move to the view part and write the needed razor code in the respective view. So, Create a new partial view named **\_**detailsPartial which will be responsible for displaying the details of the particular asset selected. For that again Right click the Asset folder in Views directory in Solution Explorer and do the same steps as previously and create the partial with the name mentioned above and add the following code in it.

1. @model GridAdvancedSearchMVC.Models.AssetViewModel
3. <div **class**="modal-dialog">
4. <div **class**="modal-content">
5. <div **class**="modal-header">
6. <button type="button" **class**="close" data-dismiss="modal" aria-hidden="true">×</button>
7. <h4 **class**="modal-title">Asset Details</h4>
8. <hr/>
9. </div>
11. <div **class**="modal-body">
12. <dl **class**="dl-horizontal">
13. <dt>
14. @Html.DisplayNameFor(model => model.Barcode)
15. </dt>
17. <dd>
18. @Html.DisplayFor(model => model.Barcode)
19. </dd>
21. <dt>
22. @Html.DisplayNameFor(model => model.SerialNumber)
23. </dt>
25. <dd>
26. @Html.DisplayFor(model => model.SerialNumber)
27. </dd>
29. <dt>
30. @Html.DisplayNameFor(model => model.FacilitySite)
31. </dt>
33. <dd>
34. @Html.DisplayFor(model => model.FacilitySite)
35. </dd>
37. <dt>
38. @Html.DisplayNameFor(model => model.PMGuide)
39. </dt>
41. <dd>
42. @Html.DisplayFor(model => model.PMGuide)
43. </dd>
45. <dt>
46. @Html.DisplayNameFor(model => model.AstID)
47. </dt>
49. <dd>
50. @Html.DisplayFor(model => model.AstID)
51. </dd>
53. <dt>
54. @Html.DisplayNameFor(model => model.ChildAsset)
55. </dt>
57. <dd>
58. @Html.DisplayFor(model => model.ChildAsset)
59. </dd>
61. <dt>
62. @Html.DisplayNameFor(model => model.GeneralAssetDescription)
63. </dt>
65. <dd>
66. @Html.DisplayFor(model => model.GeneralAssetDescription)
67. </dd>
69. <dt>
70. @Html.DisplayNameFor(model => model.SecondaryAssetDescription)
71. </dt>
73. <dd>
74. @Html.DisplayFor(model => model.SecondaryAssetDescription)
75. </dd>
77. <dt>
78. @Html.DisplayNameFor(model => model.Quantity)
79. </dt>
81. <dd>
82. @Html.DisplayFor(model => model.Quantity)
83. </dd>
85. <dt>
86. @Html.DisplayNameFor(model => model.Manufacturer)
87. </dt>
89. <dd>
90. @Html.DisplayFor(model => model.Manufacturer)
91. </dd>
93. <dt>
94. @Html.DisplayNameFor(model => model.ModelNumber)
95. </dt>
97. <dd>
98. @Html.DisplayFor(model => model.ModelNumber)
99. </dd>
101. <dt>
102. @Html.DisplayNameFor(model => model.Building)
103. </dt>
105. <dd>
106. @Html.DisplayFor(model => model.Building)
107. </dd>
109. <dt>
110. @Html.DisplayNameFor(model => model.Floor)
111. </dt>
113. <dd>
114. @Html.DisplayFor(model => model.Floor)
115. </dd>
117. <dt>
118. @Html.DisplayNameFor(model => model.Corridor)
119. </dt>
121. <dd>
122. @Html.DisplayFor(model => model.Corridor)
123. </dd>
125. <dt>
126. @Html.DisplayNameFor(model => model.RoomNo)
127. </dt>
129. <dd>
130. @Html.DisplayFor(model => model.RoomNo)
131. </dd>
133. <dt>
134. @Html.DisplayNameFor(model => model.MERNo)
135. </dt>
137. <dd>
138. @Html.DisplayFor(model => model.MERNo)
139. </dd>
141. <dt>
142. @Html.DisplayNameFor(model => model.EquipSystem)
143. </dt>
145. <dd>
146. @Html.DisplayFor(model => model.EquipSystem)
147. </dd>
149. <dt>
150. @Html.DisplayNameFor(model => model.Comments)
151. </dt>
153. <dd>
154. @Html.DisplayFor(model => model.Comments)
155. </dd>
157. <dt>
158. @Html.DisplayNameFor(model => model.Issued)
159. </dt>
161. <dd>
162. @Html.DisplayFor(model => model.Issued)
163. </dd>
165. </dl>
167. </div>
169. </div>
170. </div>

**Details Get Action Implementation**

Now it’s the time to define the controller action code which will retrieve the asset information from the data source and will pass it back to view to be displayed to the user. Here is the code for controller action:

1. **public** async Task<ActionResult> Details(Guid id)
2. {
3. **var** asset = await DbContext.Assets.FirstOrDefaultAsync(x => x.AssetID == id);
4. **var** assetVM = MapToViewModel(asset);
6. **if**(Request.IsAjaxRequest())
7. **return** PartialView("\_detailsPartial", assetVM);
9. **return** View(assetVM);
10. }

**Implementing Delete Operation**

We will add the GET action method for delete which will get the particular asset from the repository and will display the details in popup, where user would be able to delete it or cancel it. Here is the code for the action method:

1. **public** ActionResult Delete(Guid id)
2. {
3. **var** asset = DbContext.Assets.FirstOrDefault(x => x.AssetID == id);
5. AssetViewModel assetViewModel = MapToViewModel(asset);
7. **if** (Request.IsAjaxRequest())
8. **return** PartialView("\_DeletePartial", assetViewModel);
9. **return** View(assetViewModel);
10. }

**Delete Partial View Addition**

We will now add another partial view in the solution for delete part, so navigate to the Views >> Asset folder in the Solution Explorer and from the context menu which appears by right clicking the Asset folder, add a new View using the Option Add >> MVC 5 Partial Page (Razor) and name the partial view to \_DeletePartial and add the following code to it,

1. @model GridAdvancedSearchMVC.Models.AssetViewModel
3. @{
4. ViewBag.Title = "Delete";
5. Layout = "~/Views/Shared/\_Layout.cshtml";
6. }
8. @model GridAdvancedSearchMVC.Models.AssetViewModel
10. @{
11. ViewBag.Title = "Delete";
12. Layout = "~/Views/Shared/\_Layout.cshtml";
13. }
15. <div **class**="modal-dialog">
16. <div **class**="modal-content">
17. <div **class**="modal-header">
18. <button type="button" **class**="close" data-dismiss="modal" aria-hidden="true">×</button>
19. <h4 **class**="modal-title">Delete Asset</h4>
20. <strong>Are you sure you want to **delete** **this**?</strong>
22. </div>
24. <div **class**="modal-body">
25. <dl **class**="dl-horizontal">
26. <dt>
27. @Html.DisplayNameFor(model => model.Barcode)
28. </dt>
30. <dd>
31. @Html.DisplayFor(model => model.Barcode)
32. </dd>
34. <dt>
35. @Html.DisplayNameFor(model => model.SerialNumber)
36. </dt>
38. <dd>
39. @Html.DisplayFor(model => model.SerialNumber)
40. </dd>
42. <dt>
43. @Html.DisplayNameFor(model => model.FacilitySite)
44. </dt>
46. <dd>
47. @Html.DisplayFor(model => model.FacilitySite)
48. </dd>
50. <dt>
51. @Html.DisplayNameFor(model => model.PMGuide)
52. </dt>
54. <dd>
55. @Html.DisplayFor(model => model.PMGuide)
56. </dd>
58. <dt>
59. @Html.DisplayNameFor(model => model.AstID)
60. </dt>
62. <dd>
63. @Html.DisplayFor(model => model.AstID)
64. </dd>
66. <dt>
67. @Html.DisplayNameFor(model => model.ChildAsset)
68. </dt>
70. <dd>
71. @Html.DisplayFor(model => model.ChildAsset)
72. </dd>
74. <dt>
75. @Html.DisplayNameFor(model => model.GeneralAssetDescription)
76. </dt>
78. <dd>
79. @Html.DisplayFor(model => model.GeneralAssetDescription)
80. </dd>
82. <dt>
83. @Html.DisplayNameFor(model => model.SecondaryAssetDescription)
84. </dt>
86. <dd>
87. @Html.DisplayFor(model => model.SecondaryAssetDescription)
88. </dd>
90. <dt>
91. @Html.DisplayNameFor(model => model.Quantity)
92. </dt>
94. <dd>
95. @Html.DisplayFor(model => model.Quantity)
96. </dd>
98. <dt>
99. @Html.DisplayNameFor(model => model.Manufacturer)
100. </dt>
102. <dd>
103. @Html.DisplayFor(model => model.Manufacturer)
104. </dd>
106. <dt>
107. @Html.DisplayNameFor(model => model.ModelNumber)
108. </dt>
110. <dd>
111. @Html.DisplayFor(model => model.ModelNumber)
112. </dd>
114. <dt>
115. @Html.DisplayNameFor(model => model.Building)
116. </dt>
118. <dd>
119. @Html.DisplayFor(model => model.Building)
120. </dd>
122. <dt>
123. @Html.DisplayNameFor(model => model.Floor)
124. </dt>
126. <dd>
127. @Html.DisplayFor(model => model.Floor)
128. </dd>
130. <dt>
131. @Html.DisplayNameFor(model => model.Corridor)
132. </dt>
134. <dd>
135. @Html.DisplayFor(model => model.Corridor)
136. </dd>
138. <dt>
139. @Html.DisplayNameFor(model => model.RoomNo)
140. </dt>
142. <dd>
143. @Html.DisplayFor(model => model.RoomNo)
144. </dd>
146. <dt>
147. @Html.DisplayNameFor(model => model.MERNo)
148. </dt>
150. <dd>
151. @Html.DisplayFor(model => model.MERNo)
152. </dd>
154. <dt>
155. @Html.DisplayNameFor(model => model.EquipSystem)
156. </dt>
158. <dd>
159. @Html.DisplayFor(model => model.EquipSystem)
160. </dd>
162. <dt>
163. @Html.DisplayNameFor(model => model.Comments)
164. </dt>
166. <dd>
167. @Html.DisplayFor(model => model.Comments)
168. </dd>
170. <dt>
171. @Html.DisplayNameFor(model => model.Issued)
172. </dt>
174. <dd>
175. @Html.DisplayFor(model => model.Issued)
176. </dd>
178. </dl>
180. @using (Ajax.BeginForm("Delete", "Asset", **null**, **new** AjaxOptions { HttpMethod = "Post", OnSuccess = "DeleteAssetSuccess" }, **new** { @**class** = "form-horizontal", role = "form" }))
181. {
183. <div **class**="form-actions no-color">
184. @Html.HiddenFor(x => x.AssetID)
185. <input type="submit" value="Delete" **class**="btn btn-default" /> |
186. @Html.ActionLink("Back to List", "Index",**null**,**new** { data\_dismiss = "modal" })
187. </div>
188. }
189. </div>
190. </div>
191. </div>

Now we need to again a container div for delete popup as well which will be holding the html returned by the partial view \_DeletePartial as we did for other three operations, so add the following html in the Index.cshtml view of Asset,

1. div **class**="modal fade" id="deleteAssetModal" tabindex="-1" role="dialog" aria-labelledby="DeleteAssetModal" aria-hidden="true" data-backdrop="static">
2. <div id="deleteAssetContainer">
3. </div>
4. </div>

**Handling Delete Link Event with JQuery**

We also need to implement the delete button click event which will be responsible for calling the action method asynchronously and will add the response html to the popup container and the popup container will be displayed to user, the code for which would be,

1. $('#assets-data-table').on("click", ".deleteAsset", **function** (event) {
3. event.preventDefault();
5. **var** url = $(**this**).attr("href");
7. $.get(url, **function** (data) {
8. $('#deleteAssetContainer').html(data);
10. $('#deleteAssetModal').modal('show');
11. });
13. });

**Handling DELETE Ajax POST Success Callback**

If you notice here as well we are using Ajax.BeginForm helper method to post the GUID of Asset row that needs to be deleted, and we have specified a JavaScript success callback, but we haven’t yet defined the function in the Index view, so let’s do that as well, here is the function definition,

1. **function** DeleteAssetSuccess(data) {
3. **if** (data != "success") {
4. $('#deleteAssetContainer').html(data);
5. **return**;
6. }
7. $('#deleteAssetModal').modal('hide');
8. $('#deleteAssetContainer').html("");
9. assetListVM.refresh();
11. }

This is also quite same as we did for other actions, we are making sure that if deletion was successful or not, we update the UI accordingly, that’s what the above code is doing, it closes the popup, clears the html of the container div and refreshes the datatable ViewModel to reflect the latest changes in the grid.

**Implementing Delete POST Action**

Lastly, we need to define the POST action for delete which will be responsible for deleting the row from the database table and will return the status of the action back to view in either case i.e. success or failure, let’s do that, here is the code for Delete post action,

1. [HttpPost, ActionName("Delete")]
2. **public** async Task<ActionResult> DeleteAsset(Guid AssetID)
3. {
4. **var** asset = **new** Asset { AssetID = AssetID };
5. DbContext.Assets.Attach(asset);
6. DbContext.Assets.Remove(asset);
8. **var** task = DbContext.SaveChangesAsync();
9. await task;
11. **if** (task.Exception != **null**)
12. {
13. ModelState.AddModelError("", "Unable to Delete the Asset");
14. Response.StatusCode = (**int**)HttpStatusCode.BadRequest;
15. AssetViewModel assetVM = MapToViewModel(asset);
16. **return** View(Request.IsAjaxRequest() ? "\_DeletePartial" : "Delete", assetVM);
17. }
19. **if** (Request.IsAjaxRequest())
20. {
21. **return** Content("success");
22. }
24. **return** RedirectToAction("Index");
26. }

Now run the application and you should be able to add update and delete rows via Ajax without navigating between the pages.