Sure! If you want to use **Option 1: Use Only the Controller Mapping**, we can simplify the implementation by moving all the mapping logic to the controller. This approach centralizes the mapping in the controller, making the JavaScript code simpler and more secure.

Below is the complete implementation with all the files:

**1. Common Controller (CommonController.cs)**

This controller handles the AutoComplete data fetching logic. It includes the **table-to-column mapping** and dynamically constructs SQL queries.

csharp

Copy

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Data.SqlClient;

using Dapper;

namespace YourNamespace.Controllers

{

public class CommonController : Controller

{

private readonly string \_connectionString;

public CommonController(IConfiguration configuration)

{

\_connectionString = configuration.GetConnectionString("DefaultConnection");

}

[HttpGet]

public IActionResult GetAutoCompleteData(string tableName, string columnName, string searchText)

{

// Validate inputs to prevent SQL injection

if (string.IsNullOrEmpty(tableName) || string.IsNullOrEmpty(columnName) || string.IsNullOrEmpty(searchText))

{

return BadRequest("Invalid parameters.");

}

// Map grid column names to SQL column names

var columnMapping = new Dictionary<string, string>

{

{ "TaskDescription", "TaskDesc" }, // Grid column: TaskDescription, SQL column: TaskDesc

{ "OrigSupplierName", "SupplierName" },

{ "KeyWord", "Keyword" },

{ "Description", "PartDescription" }

};

// Map grid table names to database table names

var tableMapping = new Dictionary<string, string>

{

{ "TaskDescription", "Tasks" }, // Grid column: TaskDescription, Database table: Tasks

{ "OrigSupplierName", "Suppliers" },

{ "KeyWord", "Keywords" },

{ "Description", "Descriptions" }

};

if (!columnMapping.ContainsKey(columnName) || !tableMapping.ContainsKey(columnName))

{

return BadRequest("Invalid column or table name.");

}

var sqlColumnName = columnMapping[columnName];

var sqlTableName = tableMapping[columnName];

// Example: Use Dapper to fetch data from the database

using (var connection = new SqlConnection(\_connectionString))

{

connection.Open();

// Dynamically construct the SQL query

var sql = $@"

SELECT DISTINCT {sqlColumnName}

FROM {sqlTableName}

WHERE {sqlColumnName} LIKE @SearchText + '%'";

var results = connection.Query<string>(sql, new { SearchText = searchText });

return Json(results);

}

}

}

}

**2. Common Editor Template (AutoCompleteEditor.cshtml)**

This is the reusable editor template for the AutoComplete feature. It dynamically passes the column name to the CommonController.

Place this file in the Views/Shared/EditorTemplates folder.

html

Copy

@model string

@(Html.Kendo().AutoComplete()

.Name(ViewData.TemplateInfo.HtmlFieldPrefix) // Dynamic name based on the field

.DataTextField("Text")

.Filter("contains")

.Placeholder("Type to search...")

.DataSource(source =>

{

source.Read(read =>

{

read.Action("GetAutoCompleteData", "Common") // Common controller action

.Data("onAdditionalData"); // Pass additional data (e.g., column name)

})

.ServerFiltering(true);

})

)

<script>

function onAdditionalData() {

// Get the column name dynamically

var columnName = "@ViewData.ModelMetadata.PropertyName";

return {

columnName: columnName,

searchText: $("#" + columnName).data("kendoAutoComplete").value()

};

}

</script>

Run HTML

**3. Grid Definitions**

**a. LabourTask.cshtml**

html

Copy

@model IEnumerable<BCES.Models.Admin.TaskModel>

@{

ViewData["Title"] = "Task Description";

}

<h6>Task Details</h6>

@(Html.Kendo().Grid<BCES.Models.Admin.TaskModel>()

.Name("TaskGrid")

.Columns(columns =>

{

columns.Bound(p => p.TaskId).Title("Task Id").Hidden(true);

columns.Bound(p => p.TaskDescription).Title("Task Description")

.EditorTemplateName("AutoCompleteEditor"); // Use the common editor template

columns.Command(command =>

{

if (@ViewBag.RoleId == 1)

{

command.Edit();

}

if (@ViewBag.RoleId == 1)

{

command.Destroy();

}

}).Width(200);

})

.ToolBar(toolbar =>

{

if (@ViewBag.RoleId == 1)

{

toolbar.Create();

}

})

.Editable(editable => editable.Mode(GridEditMode.InLine))

.Pageable()

.Sortable()

.Filterable(ftb => ftb.Mode(GridFilterMode.Row))

.DataSource(dataSource => dataSource

.Ajax()

.Model(model =>

{

model.Id(u => u.TaskId);

model.Field(c => c.TaskDescription);

})

.Read(read => read.Url(Url.Action("ReadTasks", "Task")).Type(HttpVerbs.Get))

.Create(create => create.Url(Url.Action("CreateTask", "Task")).Type(HttpVerbs.Post))

.Update(update => create.Url(Url.Action("UpdateTask", "Task")).Type(HttpVerbs.Post))

.Destroy(delete => create.Url(Url.Action("DeleteTask", "Task")).Type(HttpVerbs.Post))

)

)

Run HTML

**b. NonStockCodedParts.cshtml**

html

Copy

@model IEnumerable<BCES.Models.Parts.NscPartsUsedViewModel>

@{

ViewData["Title"] = "Non Stock Coded Parts";

}

<h4>Non Stock Coded Parts</h4>

@(Html.Kendo().Grid<BCES.Models.Parts.NscPartsUsedViewModel>()

.Name("NscPartsUsedGrid")

.Columns(columns =>

{

columns.Bound(p => p.OrigSuppNum).Title("Supplier Number")

.EditorTemplateName("AutoCompleteEditor"); // Use the common editor template

columns.Bound(p => p.OrigSupplierName).Title("Supplier Name")

.EditorTemplateName("AutoCompleteEditor"); // Use the common editor template

columns.Bound(p => p.KeyWord).Title("Keyword")

.EditorTemplateName("AutoCompleteEditor"); // Use the common editor template

columns.Bound(p => p.Description).Title("Description")

.EditorTemplateName("AutoCompleteEditor"); // Use the common editor template

columns.Bound(p => p.CoreCost).Title("Per Unit Cost")

.Format("{0:C2}").EditorTemplateName("Currency");

columns.Command(command =>

{

if (@ViewBag.RoleId == 1)

{

command.Edit();

}

if (@ViewBag.RoleId == 1)

{

command.Destroy();

}

}).Width(200);

})

.ToolBar(toolbar =>

{

if (@ViewBag.RoleId == 1)

{

toolbar.Create();

}

})

.Editable(editable => editable.Mode(GridEditMode.InLine))

.Pageable()

.Sortable()

.Filterable(ftb => ftb.Mode(GridFilterMode.Row))

.DataSource(dataSource => dataSource

.Ajax()

.Model(model =>

{

model.Id(u => u.NscPartsUsedId);

})

.Read(read => read.Url(Url.Action("GetNscPartsUsedView", "NscPartsUsed")).Type(HttpVerbs.Get))

.Create(create => create.Url(Url.Action("CreateNscPartsUsed", "NscPartsUsed")).Type(HttpVerbs.Post))

.Update(update => create.Url(Url.Action("UpdateNscPartsUsed", "NscPartsUsed")).Type(HttpVerbs.Post))

.Destroy(delete => create.Url(Url.Action("DeleteNscPartsUsed", "NscPartsUsed")).Type(HttpVerbs.Post))

)

)

Run HTML

**Summary**

1. **Common Controller**:
   * Handles the AutoComplete data fetching logic.
   * Includes mappings for grid column names to SQL column and table names.
2. **Common Editor Template**:
   * Dynamically passes the column name to the controller.
3. **Grid Definitions**:
   * Use the AutoCompleteEditor template for columns that require AutoComplete.

This approach ensures that the AutoComplete feature is secure, reusable, and centralized in the controller. Let me know if you need further assistance!