**Fix: Use a Dynamic Name for the MultiSelect**

Update the VehicleSeriesEditor.cshtml to use the correct dynamic name for the MultiSelect. Instead of hardcoding .Name("Buses"), dynamically use the name generated by the Kendo Grid for the Buses field.

**Updated VehicleSeriesEditor.cshtml**

html

CopyEdit

@model List<int> <!-- Expect a list of ListId values -->

@(Html.Kendo().MultiSelect()

.Name(ViewData.TemplateInfo.HtmlFieldPrefix) <!-- Dynamically set the name -->

.DataTextField("Description") // Display Description in the MultiSelect dropdown

.DataValueField("ListId") // Bind to ListId for the values

.BindTo((IEnumerable)ViewData["BusesList"]) // Provide the full list of available buses

)

**Explanation of the Fix**

1. **Dynamic Name Using ViewData.TemplateInfo.HtmlFieldPrefix**:
   * HtmlFieldPrefix is a Razor helper that dynamically sets the name of the editor based on the grid's context.
   * For example:
     + For the first row: Buses[0]
     + For the second row: Buses[1]
   * This ensures that the editor's name matches the field in the RebuiltPartsViewModel.
2. **Model Expectation**:
   * The model for the editor (List<int>) represents the selected ListId values for the buses. These values will be sent back to the server when the row is updated.
3. **Binding to Full List of Buses**:
   * The BindTo method uses ViewData["BusesList"] to provide the available options for the MultiSelect dropdown.

**Updated Controller Logic**

**Pass Full Buses List to ViewData**

The controller must provide the full list of buses (ListId and Description) to the ViewData so the MultiSelect has options to display.

csharp

CopyEdit

public async Task<IActionResult> Index()

{

// Query to fetch all available buses for the MultiSelect

var query = "SELECT ListId, Description FROM SBCES.LISTOFBUSES";

// Fetch the list of buses

var busesList = await \_dbConnection.QueryAsync<BusesModel>(query);

// Pass the buses list to ViewData for the MultiSelect

ViewData["BusesList"] = busesList;

// Return the view

return View();

}

**Update GetRebuiltPartsView to Return Selected Bus IDs**

The grid expects the Buses field to contain a list of selected ListId values for each row. Update the query to retrieve and format this data.

csharp

CopyEdit

public async Task<JsonResult> GetRebuiltPartsView([DataSourceRequest] DataSourceRequest request)

{

try

{

var query = @"

SELECT

rbm.RebuiltStockNum,

rbm.Keyword,

rbl.ListId,

lb.Description

FROM SBCES.RbMasterlist rbm

LEFT JOIN SBCES.RBLISTOFBUSES rbl ON rbm.RebuiltStockNum = rbl.RebuiltStockNum

LEFT JOIN SBCES.LISTOFBUSES lb ON rbl.ListId = lb.ListId";

var rebuiltPartsDict = new Dictionary<string, RebuiltPartsViewModel>();

var data = await \_dbConnection.QueryAsync<RebuiltPartsViewModel, BusesModel, RebuiltPartsViewModel>(

query,

(part, bus) =>

{

if (!rebuiltPartsDict.TryGetValue(part.RebuiltStockNum, out var rebuiltPart))

{

rebuiltPart = part;

rebuiltPart.Buses = new List<BusesModel>();

rebuiltPartsDict.Add(rebuiltPart.RebuiltStockNum, rebuiltPart);

}

if (bus != null)

{

rebuiltPart.Buses.Add(bus);

}

return rebuiltPart;

},

splitOn: "ListId"

);

// Convert the Buses list to a list of ListId values for proper MultiSelect binding

foreach (var part in rebuiltPartsDict.Values)

{

part.Buses = part.Buses.Select(b => new BusesModel { ListId = b.ListId, Description = b.Description }).ToList();

}

return Json(rebuiltPartsDict.Values.ToDataSourceResult(request));

}

catch (Exception ex)

{

return Json(new DataSourceResult { Errors = "Error occurred while fetching rebuilt parts." });

}

}

**Updated Grid in Index.cshtml**

Ensure the grid's Buses column uses the correct EditorTemplateName.

html

CopyEdit

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

.Name("RebuiltPartsGrid")

.Columns(columns =>

{

columns.Bound(p => p.RebuiltStockNum).Title("Rebuilt Stock #").Width(150).Editable(false); // PK, not editable

columns.Bound(p => p.Keyword).Title("Keyword").Width(200);

// MultiSelect column for Buses

columns.Bound(p => p.Buses)

.ClientTemplate("#= Buses.map(function(bus) { return bus.Description; }).join(', ') #") // Display descriptions

.EditorTemplateName("VehicleSeriesEditor") // Use custom MultiSelect editor

.Title("Buses").Width(300);

columns.Command(command =>

{

command.Edit(); // Inline Edit Command

command.Destroy(); // Delete button

}).Title("Actions").Width(150);

})

.ToolBar(toolbar => toolbar.Create()) // Create button

.Editable(editable => editable.Mode(GridEditMode.InLine)) // Inline editing

.Pageable()

.Sortable()

.Filterable()

.DataSource(dataSource => dataSource

.Ajax()

.Read(read => read.Url(Url.Action("GetRebuiltPartsView", "RebuiltParts")).Type(HttpVerbs.Get)) // Read data

.Create(create => create.Url(Url.Action("Create", "RebuiltParts")).Type(HttpVerbs.Post)) // Create action

.Update(update => update.Url(Url.Action("Update", "RebuiltParts")).Type(HttpVerbs.Post)) // Update action

.Destroy(destroy => destroy.Url(Url.Action("Delete", "RebuiltParts")).Type(HttpVerbs.Post)) // Delete action

.Model(model =>

{

model.Id(p => p.RebuiltStockNum); // Set the primary key

model.Field(p => p.Buses).DefaultValue(new List<int>()); // Default empty list for MultiSelect

})

)

)

**Key Fixes**

1. **Dynamic Name for MultiSelect**:
   * @Html.Kendo().MultiSelect().Name(ViewData.TemplateInfo.HtmlFieldPrefix) ensures the Name matches the dynamically generated field name.
2. **Pass Correct BusesList**:
   * Ensure ViewData["BusesList"] contains a list of all buses with ListId and Description.
3. **Proper MultiSelect Value Binding**:
   * The Buses field in the grid now binds to a list of selected ListId values.
4. **ClientTemplate**:
   * Use ClientTemplate to display a comma-separated list of bus descriptions (Description) in the grid.