**RebuiltPartsViewModel Changes**

Update the RebuiltPartsViewModel to include a separate list for IDs. This will simplify the binding.

csharp

CopyEdit

public class RebuiltPartsViewModel

{

public string RebuiltStockNum { get; set; }

// Full list of bus objects (for display)

public List<BusesModel> ListOfBus { get; set; } = new List<BusesModel>();

// List of IDs for editor binding

public List<int> ListOfBusIds { get; set; } = new List<int>();

}

**2. GetRebuiltPartsData Update**

Ensure that ListOfBusIds is populated alongside ListOfBus.

csharp

CopyEdit

private async Task<IEnumerable<RebuiltPartsViewModel>> GetRebuiltPartsData()

{

try

{

var rebuiltPartsQuery = @"

SELECT

rbm.RebuiltStockNum,

rbl.listid AS ListId,

lb.description AS 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 rebuiltParts = await \_dbConnection.QueryAsync<RebuiltPartsViewModel, BusesModel, RebuiltPartsViewModel>(

rebuiltPartsQuery,

(rebuiltPart, bus) =>

{

if (bus != null)

{

rebuiltPart.ListOfBus.Add(bus);

rebuiltPart.ListOfBusIds.Add(bus.ListId); // Add IDs for editor

}

return rebuiltPart;

},

splitOn: "ListId"

);

var groupedData = rebuiltParts

.GroupBy(rp => rp.RebuiltStockNum)

.Select(group =>

{

var rebuiltPart = group.First();

rebuiltPart.ListOfBus = group.SelectMany(g => g.ListOfBus).Distinct().ToList();

rebuiltPart.ListOfBusIds = group.SelectMany(g => g.ListOfBusIds).Distinct().ToList();

return rebuiltPart;

})

.ToList();

return groupedData;

}

catch (Exception ex)

{

Console.WriteLine($"Error fetching rebuilt parts data: {ex.Message}");

return Enumerable.Empty<RebuiltPartsViewModel>();

}

}

**3. VehicleSeriesEditor.cshtml**

Update the editor template to bind the MultiSelect to ListOfBusIds. Remove the Name property as it will be automatically generated by the grid.

html

CopyEdit

@model List<int>

@(Html.Kendo().MultiSelectFor(m => m) // Bind to List<int>

.DataTextField("Description")

.DataValueField("ListId")

.Placeholder("Select vehicle series...")

.Filter("contains")

.DataSource(source =>

{

source.Read(read => read.Url(Url.Action("GetVehicleSeries", "RebuiltParts")).Type(HttpVerbs.Get));

})

)

**4. Index.cshtml Grid Update**

The grid needs to handle the ListOfBusIds for editing but still display ListOfBus as a comma-separated string.

html

CopyEdit

@(Html.Kendo().Grid<RebuiltPartsViewModel>()

.Name("RebuiltPartsGrid")

.Columns(columns =>

{

columns.Bound(c => c.RebuiltStockNum)

.Title("Rebuilt Stock Code")

.Filterable(ftb => ftb.Cell(cell => cell.Operator("contains").SuggestionOperator(FilterType.Contains)));

// Column for displaying comma-separated vehicle series

columns.Bound(c => c.ListOfBus)

.Title("Vehicle Series")

.ClientTemplate("# if (ListOfBus && ListOfBus.length > 0) { # #= ListOfBus.map(bus => bus.Description).join(', ') # # } else { # - # } #")

.EditorTemplateName("VehicleSeriesEditor");

// Update/Delete commands

columns.Command(command =>

{

command.Edit().Text("Update");

command.Destroy().Text("Delete");

}).Title("&nbsp;").Width(200);

})

.ToolBar(toolbar => toolbar.Create().Text("Add New Part"))

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

.Pageable()

.Sortable()

.Scrollable()

.Filterable()

.DataSource(dataSource => dataSource

.Ajax()

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

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

.Destroy(destroy => destroy.Url(Url.Action("DeleteRebuiltPart", "RebuiltParts")).Type(HttpVerbs.Post))

.Model(model =>

{

model.Id(p => p.RebuiltStockNum);

model.Field(p => p.ListOfBusIds).DefaultValue(new List<int>()); // Bind to ListOfBusIds

})

)

)

**5. Controller Update**

Update the UpdateRebuiltPart method to handle ListOfBusIds.

csharp

CopyEdit

[HttpPost]

public async Task<IActionResult> UpdateRebuiltPart(RebuiltPartsViewModel model)

{

try

{

// Update rebuilt part logic here...

// Handle updating the associated buses

var deleteOldBusesSql = "DELETE FROM SBCES.RBLISTOFBUSES WHERE RebuiltStockNum = @RebuiltStockNum";

await \_dbConnection.ExecuteAsync(deleteOldBusesSql, new { model.RebuiltStockNum });

if (model.ListOfBusIds?.Count > 0)

{

var insertNewBusesSql = @"

INSERT INTO SBCES.RBLISTOFBUSES (RebuiltStockNum, ListId)

VALUES (@RebuiltStockNum, @ListId)";

foreach (var listId in model.ListOfBusIds)

{

await \_dbConnection.ExecuteAsync(insertNewBusesSql, new { model.RebuiltStockNum, ListId = listId });

}

}

return Json(new { success = true });

}

catch (Exception ex)

{

return StatusCode(500, $"Internal server error: {ex.Message}");

}

}

**Explanation of Changes**

1. **Separated Display and Editor Data**:
   * ListOfBus is used for display (comma-separated descriptions).
   * ListOfBusIds is used for the MultiSelect editor.
2. **Proper Binding**:
   * The MultiSelect binds to ListOfBusIds (a List<int>), which avoids the Name conflict.
3. **Fixed JavaScript Error**:
   * Removed explicit Name property, allowing the grid to generate it dynamically.