**Update BusesModel to Make ListId Nullable**

Ensure the ListId property is nullable in the BusesModel class:

csharp

CopyEdit

public class BusesModel

{

public int? ListId { get; set; } // Nullable ListId

public string Description { get; set; }

}

**2. Handle Nulls in the GetRebuiltPartsData Method**

When populating ListOfBusIds in the GetRebuiltPartsData method, filter out or handle null ListId values to prevent them from causing issues.

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)

{

// Add the bus if it has a valid ListId

rebuiltPart.ListOfBus.Add(bus);

if (bus.ListId.HasValue)

{

rebuiltPart.ListOfBusIds.Add(bus.ListId.Value); // Only add non-null IDs

}

}

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>();

}

}

Here:

* Only non-null ListId values are added to ListOfBusIds.
* This prevents nulls from being passed to the MultiSelect.

**3. Update the MultiSelect Editor to Handle Nulls**

In VehicleSeriesEditor.cshtml, we can adjust the MultiSelect configuration to account for potential null values.

html

CopyEdit

@model List<int?> <!-- Update to List<int?> to allow nullable IDs -->

@(Html.Kendo().MultiSelectFor(m => m)

.DataTextField("Description")

.DataValueField("ListId")

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

.Filter("contains")

.DataSource(source =>

{

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

})

.Value(Model.Where(id => id.HasValue).Select(id => id.Value)) <!-- Filter out null IDs -->

)

**Key changes:**

1. The @model is updated to List<int?> to allow nullable IDs.
2. The Value() method filters out null IDs using Where(id => id.HasValue).Select(id => id.Value) before passing the values to the MultiSelect.

**4. Update the GetVehicleSeries Controller**

If ListId can be null in the database, ensure the GetVehicleSeries endpoint handles these gracefully.

csharp

CopyEdit

public async Task<IActionResult> GetVehicleSeries()

{

try

{

var sql = @"

SELECT

RBLISTOFBUSES.listid AS ListId,

LISTOFBUSES.description AS Description

FROM

SBCES.RBLISTOFBUSES

INNER JOIN

SBCES.LISTOFBUSES ON RBLISTOFBUSES.listid = LISTOFBUSES.listid";

var vehicleSeries = await \_dbConnection.QueryAsync<BusesModel>(sql);

// Ensure null ListId entries are removed

var validVehicleSeries = vehicleSeries.Where(vs => vs.ListId.HasValue);

return Json(validVehicleSeries);

}

catch (Exception ex)

{

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

}

}

Here:

* Where(vs => vs.ListId.HasValue) ensures only valid ListId values are returned to the client.

**5. Optional: Adjust Client Template for Display**

If null ListId values indicate invalid or unassigned entries, you may want to handle this scenario in the grid's display logic (e.g., ClientTemplate).

In Index.cshtml, update the ClientTemplate for the ListOfBus column:

html

CopyEdit

columns.Bound(c => c.ListOfBus)

.Title("Vehicle Series")

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

Here:

* filter(bus => bus.ListId) ensures that only buses with valid ListId values are displayed.
* This prevents the display of rows with invalid or null IDs.

**6. Ensure Proper Model Handling in the Update Action**

In the UpdateRebuiltPart controller, ensure null ListId values are handled correctly:

csharp

CopyEdit

[HttpPost]

public async Task<IActionResult> UpdateRebuiltPart(RebuiltPartsViewModel model)

{

try

{

// Update logic (update rebuilt part and its associated buses)

// Remove old buses

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

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

// Insert new buses (exclude null IDs)

if (model.ListOfBusIds?.Any(id => id.HasValue) == true)

{

var insertNewBusesSql = @"

INSERT INTO SBCES.RBLISTOFBUSES (RebuiltStockNum, ListId)

VALUES (@RebuiltStockNum, @ListId)";

foreach (var listId in model.ListOfBusIds.Where(id => id.HasValue))

{

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

}

}

return Json(new { success = true });

}

catch (Exception ex)

{

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

}

}

Here:

* Null ListId values are filtered out before inserting into the database using Where(id => id.HasValue).

**Summary of Changes**

1. **Nullable ListId Handling**:
   * ListId is made nullable in the BusesModel class.
   * Null ListId values are filtered out before binding to the MultiSelect.
2. **Editor Updates**:
   * VehicleSeriesEditor.cshtml is updated to bind List<int?> and filter out null IDs.
3. **Controller Updates**:
   * The GetVehicleSeries action ensures only valid ListId values are returned.
   * The UpdateRebuiltPart action filters out null IDs before database operations.
4. **Grid Updates**:
   * The ClientTemplate for ListOfBus skips null IDs when displaying descriptions.