**Updated Implementation**

**1. RebuiltPartsViewModel**

Ensure the Buses property is always a List<BusesModel>.

csharp

CopyEdit

public class RebuiltPartsViewModel

{

public string RebuiltStockNum { get; set; } // Primary Key

public string Keyword { get; set; } // Search or filtering keyword

// Always represent Buses as a List<BusesModel>

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

}

**2. Update VehicleSeriesEditor (MultiSelect Editor Template)**

Update the editor template to dynamically extract only the ListId values (List<int>) from the Buses property for the MultiSelect control.

html

CopyEdit

@model List<BCES.Models.Parts.BusesModel>

@(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

.Value(Model.Select(b => b.ListId)) // Bind only the ListId values

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

)

**Explanation:**

1. .Value(Model.Select(b => b.ListId)): This extracts the ListId values from the List<BusesModel> for the MultiSelect.
2. The BindTo still provides the full list of available buses (List<BusesModel>).

**3. Update GetRebuiltPartsView in Controller**

Make sure the Buses property is returned as a List<BusesModel>. Each part's Buses field should contain both ListId and Description.

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>(); // Initialize the buses list

rebuiltPartsDict.Add(rebuiltPart.RebuiltStockNum, rebuiltPart);

}

if (bus != null)

{

rebuiltPart.Buses.Add(bus);

}

return rebuiltPart;

},

splitOn: "ListId"

);

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

}

catch (Exception ex)

{

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

}

}

**4. Update the Update Action in Controller**

When the user submits changes to the grid, the Buses field will contain a List<BusesModel> with both ListId and Description. Convert this to a list of ListIds before updating the database.

csharp

CopyEdit

[HttpPost]

public async Task<JsonResult> Update([DataSourceRequest] DataSourceRequest request, RebuiltPartsViewModel model)

{

if (!ModelState.IsValid)

{

return Json(new DataSourceResult { Errors = ModelState });

}

try

{

// Update the RebuiltPartsViewModel data

var updatePartQuery = @"

UPDATE SBCES.RbMasterlist

SET Keyword = @Keyword

WHERE RebuiltStockNum = @RebuiltStockNum";

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

// Update the associated BusesModel data (clear and re-insert for simplicity)

var deleteBusesQuery = @"

DELETE FROM SBCES.RBLISTOFBUSES

WHERE RebuiltStockNum = @RebuiltStockNum";

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

if (model.Buses != null && model.Buses.Any())

{

var insertBusesQuery = @"

INSERT INTO SBCES.RBLISTOFBUSES (RebuiltStockNum, ListId)

VALUES (@RebuiltStockNum, @ListId)";

foreach (var bus in model.Buses)

{

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

}

}

return Json(new[] { model }.ToDataSourceResult(request));

}

catch (Exception ex)

{

return Json(new DataSourceResult { Errors = "Error occurred while updating the rebuilt part." });

}

}

**5. Update the Grid in Index.cshtml**

The grid remains mostly the same, but make sure the editor template (VehicleSeriesEditor) is correctly applied and bound to the Buses field.

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<BCES.Models.Parts.BusesModel>()); // Default empty list

})

)

)

**Key Fixes Summary:**