To resolve the issue where ListOfBusIds and ListOfBus are always null in the AddRebuiltPartsView method, even when buses are selected in the Kendo MultiSelect, you need to ensure proper model binding between the MultiSelect and the ASP.NET Core controller. Here's the solution:

**Step 1: Fix the Model Binding for MultiSelect**

The Kendo MultiSelect in your VehicleSeriesEditor.cshtml is likely not binding to the model correctly. Update the VehicleSeriesEditor.cshtml to ensure the MultiSelect's name attribute matches the model property ListOfBusIds:

**Updated VehicleSeriesEditor.cshtml:**

html

Copy

@model List<int?>

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

.Name("ListOfBusIds") <!-- Explicitly set the name to match the model property -->

.DataTextField("Description")

.DataValueField("ListId")

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

.Filter("contains")

.AutoBind(false)

.DataSource(source =>

{

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

})

)

Run HTML

**Step 2: Ensure Model Initialization**

In the RebuiltPartsViewModel, ensure ListOfBusIds is initialized to avoid null values:

csharp

Copy

public class RebuiltPartsViewModel

{

public RebuiltPartsViewModel()

{

ListOfBus = new List<BusesModel>();

ListOfBusIds = new List<int?>(); // Initialize here

}

public List<BusesModel> ListOfBus { get; set; }

public List<int?> ListOfBusIds { get; set; } // Ensure this is initialized

// ... other properties

}

**Step 3: Update the AddRebuiltPartsView Method**

Modify the AddRebuiltPartsView method to handle the incoming ListOfBusIds properly. The selected bus IDs will be sent as a comma-separated string, so parse them into a list of integers:

csharp

Copy

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> AddRebuiltPartsView(

[DataSourceRequest] DataSourceRequest request,

[FromForm] RebuiltPartsViewModel rebuiltPartsViewModel,

[FromForm] string listOfBusIds) // Add this parameter to capture the MultiSelect values

{

try

{

if (rebuiltPartsViewModel == null)

{

return BadRequest("Invalid data.");

}

// Parse the comma-separated string of selected bus IDs

if (!string.IsNullOrEmpty(listOfBusIds))

{

rebuiltPartsViewModel.ListOfBusIds = listOfBusIds

.Split(',')

.Select(id => int.TryParse(id, out var parsedId) ? parsedId : (int?)null)

.ToList();

}

// Ensure ListOfBus is initialized (if needed)

rebuiltPartsViewModel.ListOfBus ??= new List<BusesModel>();

// Insert the rebuilt part into the database

var insertPartSql = @"

INSERT INTO SBCES.RbMasterlist

(

RebuiltStockNum, MmsStockCode, Keyword, JobNumber, CorePartNum,

DetailedDesc, CoreCharge, EstimatedCost, BuyNewCost, RemanCost,

ExternalCost, LastModifiedBy, DateEntered, IsActive

)

VALUES

(

@RebuiltStockNum, @MmsStockCode, @Keyword, @JobNumber, @CorePartNum,

@DetailedDesc, @CoreCharge, @EstimatedCost, @BuyNewCost, @RemanCost,

@ExternalCost, @LastModifiedBy, GETDATE(), @IsActive

)";

await \_dbConnection.ExecuteAsync(insertPartSql, new

{

rebuiltPartsViewModel.RebuiltStockNum,

rebuiltPartsViewModel.MmsStockCode,

rebuiltPartsViewModel.Keyword,

rebuiltPartsViewModel.JobNumber,

rebuiltPartsViewModel.CorePartNum,

rebuiltPartsViewModel.DetailedDesc,

rebuiltPartsViewModel.CoreCharge,

rebuiltPartsViewModel.EstimatedCost,

rebuiltPartsViewModel.BuyNewCost,

rebuiltPartsViewModel.RemanCost,

ExternalCost = "", // Adjust as needed

LastModifiedBy = User.Identity.Name, // Or your logic

IsActive = rebuiltPartsViewModel.IsActive ? 1 : 0

});

// Insert associated buses

if (rebuiltPartsViewModel.ListOfBusIds?.Any() == true)

{

var insertBusesSql = @"

INSERT INTO SBCES.RbListOfBuses (RebuiltStockNum, ListId)

VALUES (@RebuiltStockNum, @ListId)";

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

{

await \_dbConnection.ExecuteAsync(insertBusesSql, new

{

rebuiltPartsViewModel.RebuiltStockNum,

ListId = listId.Value

});

}

}

return Json(new { success = true });

}

catch (Exception ex)

{

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

}

}

**Step 4: Update the Grid Configuration**

Ensure the grid's Create action maps the ListOfBusIds parameter correctly. In the grid configuration, explicitly include ListOfBusIds in the Create settings:

**In Index.cshtml:**

csharp

Copy

.Create(create => create

.Url(Url.Action("AddRebuiltPartsView", "RebuiltParts"))

.Data("function(data) { return { rebuiltPartsViewModel: data, listOfBusIds: data.ListOfBusIds }; }") // Pass ListOfBusIds

.Type(HttpVerbs.Post)

)

**Why This Works:**

1. **Model Binding Fix:**
   * The listOfBusIds parameter captures the selected values as a comma-separated string from the MultiSelect.
   * The values are parsed into a list of integers and assigned to rebuiltPartsViewModel.ListOfBusIds.
2. **MultiSelect Configuration:**
   * Setting .Name("ListOfBusIds") ensures the MultiSelect sends data with the correct key expected by the model binder.
3. **Grid Data Mapping:**
   * The Create action explicitly passes ListOfBusIds to the controller method.

**Additional Checks:**

1. **Network Request Inspection:**
   * Use the browser's developer tools to verify that the ListOfBusIds parameter is included in the POST request payload as a comma-separated string (e.g., 1,2,3).
2. **Database Schema:**
   * Ensure the RbListOfBuses table has the correct schema to store the RebuiltStockNum and ListId pairs.
3. **Null Handling:**
   * The code handles ListOfBusIds being null or empty gracefully.

With these changes, the selected bus IDs will now be correctly bound to ListOfBusIds in the AddRebuiltPartsView method.