o resolve the issue where rebuiltPartsViewModel is null in the AddRebuiltPartsView method and ensure proper model binding for both create and update actions, follow these steps:

**1. Adjust the Controller Methods**

Remove the separate listOfBusIds parameter and rely entirely on the ListOfBusIds property within RebuiltPartsViewModel.

**Updated AddRebuiltPartsView Method:**

csharp

Copy

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> AddRebuiltPartsView(

[DataSourceRequest] DataSourceRequest request,

[FromForm] RebuiltPartsViewModel rebuiltPartsViewModel) // Removed separate listOfBusIds parameter

{

try

{

if (rebuiltPartsViewModel == null)

{

return BadRequest("Invalid data.");

}

// 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}");

}

}

**Updated UpdateRebuiltPartsView Method:**

csharp

Copy

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> UpdateRebuiltPartsView(

[DataSourceRequest] DataSourceRequest request,

[FromForm] RebuiltPartsViewModel model)

{

try

{

if (model == null)

{

return BadRequest("Invalid data.");

}

var updatePartSql = @"

UPDATE SBCES.RbMasterlist

SET

LastModifiedBy = UPPER(ISNULL(@LastModifiedBy, ' ')),

DateEntered = GETDATE(),

JobNumber = UPPER(ISNULL(@JobNumber, ' ')),

CoreCharge = ISNULL(@CoreCharge, ' '),

CorePartNum = ISNULL(@CorePartNum, ' '),

Keyword = UPPER(ISNULL(@Keyword, ' ')),

DetailedDesc = UPPER(ISNULL(@DetailedDesc, ' ')),

MmsStockCode = ISNULL(@MmsStockCode, ' '),

SopNumber = ISNULL(@SopNumber, ' '),

BuyNewCost = ISNULL(@BuyNewCost, ''),

RemanCost = ISNULL(@RemanCost, ''),

ExternalCost = ISNULL(@ExternalCost, ''),

Active = @IsActive

WHERE RebuiltStockNum = @RebuiltStockNum";

await \_dbConnection.ExecuteAsync(updatePartSql, new

{

model.LastModifiedBy,

model.JobNumber,

model.CoreCharge,

model.CorePartNum,

model.Keyword,

model.DetailedDesc,

model.MmsStockCode,

model.SopNumber,

model.BuyNewCost,

model.RemanCost,

ExternalCost = "", // Adjust as needed

IsActive = model.IsActive ? 1 : 0,

model.RebuiltStockNum

});

// Update buses

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

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

if (model.ListOfBusIds?.Any() == 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}");

}

}

**2. Fix the MultiSelect Configuration**

Ensure the Kendo MultiSelect in the view is bound to RebuiltPartsViewModel.ListOfBusIds with the correct name attribute.

**Updated VehicleSeriesEditor.cshtml:**

html

Copy

@model List<int?>

@(

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

.Name("ListOfBusIds") // Explicitly set the name to match the ViewModel property

.HtmlAttributes(new { id = "ListOfBusIds" })

.DataTextField("Description")

.DataValueField("ListId")

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

.Filter("contains")

.DataSource(source =>

{

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

})

)

Run HTML

**3. Ensure Form Fields Match ViewModel Properties**

In your form, ensure all input fields have name attributes that exactly match the properties of RebuiltPartsViewModel. For example:

html

Copy

<input type="text" name="RebuiltStockNum" />

<input type="text" name="MmsStockCode" />

<!-- Other fields -->

Run HTML

**4. Test the Form Submission**

Use browser developer tools to inspect the network request and verify:

* All form fields (including ListOfBusIds) are present.
* The Content-Type header is application/x-www-form-urlencoded or multipart/form-data.

**5. Handle Model Binding Gracefully**

Add model state validation to catch binding issues:

csharp

Copy

if (!ModelState.IsValid)

{

var errors = ModelState.Values.SelectMany(v => v.Errors).Select(e => e.ErrorMessage);

return BadRequest(new { errors });

}

**6. Adjust the ViewModel (if necessary)**

Ensure ListOfBusIds is initialized and matches the form data type:

csharp

Copy

public class RebuiltPartsViewModel

{

public RebuiltPartsViewModel()

{

ListOfBus = new List<BusesModel>();

ListOfBusIds = new List<int?>();

}

// Other properties

public List<int?> ListOfBusIds { get; set; }

}

**Explanation**

* **Model Binding:** The [FromForm] attribute binds form data to the RebuiltPartsViewModel. By ensuring all form fields match the ViewModel's property names, the model binder can construct the ViewModel correctly.
* **MultiSelect Fix:** Explicitly setting the Name("ListOfBusIds") in the Kendo MultiSelect ensures the selected values are posted as a comma-separated string, which the model binder maps to ListOfBusIds.
* **Removing Redundant Parameters:** Using a single RebuiltPartsViewModel parameter instead of separate parameters avoids confusion and ensures all data is bound correctly.