**Updated Grid Configuration in Index.cshtml**

Here is the updated Kendo Grid configuration using the RequestStart event to pass RebuiltStockNum dynamically.

csharp

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)));

columns.Bound(c => c.ListOfBus)

.Title("Vehicle Series")

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

columns.Command(command =>

{

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

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

}).Title("&nbsp;").Width(150);

})

.Editable(editable => editable.Mode(GridEditMode.InLine)) // Inline editing

.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); // Specify the primary key

model.Field(p => p.ListOfBusIds).DefaultValue(new List<int?>());

})

.Events(events => events.RequestStart("onRequestStart")) // Attach the RequestStart event

)

)

<script>

// Event triggered before any CRUD operation (Read, Update, Destroy)

function onRequestStart(e) {

// Check if the request type is Update or Destroy

if (e.type === "update" || e.type === "destroy") {

var grid = $("#RebuiltPartsGrid").data("kendoGrid");

var selectedRow = grid.select(); // Get the selected row

var dataItem = grid.dataItem(selectedRow); // Get the data for the selected row

if (dataItem && dataItem.RebuiltStockNum) {

// Append RebuiltStockNum to the request

e.sender.transport.options.update.data = { RebuiltStockNum: dataItem.RebuiltStockNum };

e.sender.transport.options.destroy.data = { RebuiltStockNum: dataItem.RebuiltStockNum };

}

}

}

</script>

**Explanation of the Configuration**

1. **Event Hook (RequestStart)**:
   * The RequestStart event is triggered before the grid sends a request to the server for any CRUD operation (e.g., Read, Update, Destroy).
   * The onRequestStart function is used to dynamically append the RebuiltStockNum of the selected row to the AJAX request.
2. **Dynamically Add Parameters**:
   * For Update and Destroy operations, the RebuiltStockNum is extracted from the selected row and added to the request payload via:

javascript

CopyEdit

e.sender.transport.options.update.data

e.sender.transport.options.destroy.data

1. **Inline Editing**:
   * Inline editing works seamlessly as the RebuiltStockNum for the currently edited row is passed dynamically.
2. **Transport Options**:
   * The e.sender.transport.options.update.data and e.sender.transport.options.destroy.data properties are used to add custom parameters (RebuiltStockNum) to the request.

**Updated Controller: RebuiltPartsController**

The controller actions for Update and Delete need to handle the RebuiltStockNum being passed in the request.

**Updated UpdateRebuiltPart and DeleteRebuiltPart Actions:**

csharp

CopyEdit

[HttpPost]

public async Task<IActionResult> UpdateRebuiltPart(string RebuiltStockNum, [FromBody] RebuiltPartsViewModel model)

{

try

{

if (string.IsNullOrEmpty(RebuiltStockNum) || model == null)

{

return BadRequest("Invalid data.");

}

// Update the RebuiltPart in the database

var updatePartSql = @"

UPDATE SBCES.RbMasterlist

SET ... -- Add fields to update here

WHERE RebuiltStockNum = @RebuiltStockNum";

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

// Update associated buses

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

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

}

}

return Json(new { success = true });

}

catch (Exception ex)

{

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

}

}

[HttpPost]

public async Task<IActionResult> DeleteRebuiltPart(string RebuiltStockNum)

{

try

{

if (string.IsNullOrEmpty(RebuiltStockNum))

{

return BadRequest("Invalid data.");

}

var deletePartSql = "DELETE FROM SBCES.RbMasterlist WHERE RebuiltStockNum = @RebuiltStockNum";

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

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

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

return Json(new { success = true });

}

catch (Exception ex)

{

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

}

}

**Benefits of Using Events**

1. **Dynamic Parameter Passing**:
   * The RequestStart event ensures that RebuiltStockNum is passed only when required (for Update and Destroy operations).
   * This avoids the need for hardcoding parameters in the grid's DataSource.
2. **Inline Editing Compatibility**:
   * The selected row's RebuiltStockNum is passed seamlessly during inline editing, making the grid highly interactive.
3. **Clean Separation**:
   * Using events keeps the grid configuration clean and reusable.

**Summary of Changes**

1. **onRequestStart Event**:
   * Dynamically appends RebuiltStockNum to the Update and Destroy requests.
2. **Controller Updates**:
   * Both UpdateRebuiltPart and DeleteRebuiltPart handle the RebuiltStockNum parameter from the request.
3. **Simplified Grid Configuration**:
   * No need to pass parameters through .Data() or Transport.