**RebuiltPartsViewModel.cs**

The ViewModel remains largely the same but is annotated for clarity:

csharp

CopyEdit

namespace BCES.Models.Parts

{

public class RebuiltPartsViewModel

{

// Primary key for rebuilt part

public string RebuiltStockNum { get; set; }

// Search or filtering keyword

public string Keyword { get; set; }

// List of associated buses

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

}

public class BusesModel

{

// Unique ID of the bus

public int ListId { get; set; }

// Keyword associated with the bus

public string Keyword { get; set; }

// Comma-separated bus descriptions

public string? Description { get; set; }

}

}

**2. TestRebuiltPartController.cs**

The controller directly calls SQL queries using **Dapper**. Exception handling is added.

csharp

CopyEdit

using BCES.Models.Parts;

using Dapper;

using Microsoft.AspNetCore.Mvc;

using System.Data;

namespace BCES.Controllers

{

public class TestRebuiltPartController : Controller

{

private readonly IDbConnection \_dbContext;

public TestRebuiltPartController(IDbConnection dbContext)

{

\_dbContext = dbContext;

}

// GET: Retrieve all rebuilt parts

public IActionResult Index()

{

try

{

// Query for fetching rebuilt parts and their associated buses

var query = @"

SELECT rbm.RebuiltStockNum, rbm.Keyword,

lb.ListId, STRING\_AGG(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

GROUP BY rbm.RebuiltStockNum, rbm.Keyword";

var rebuiltParts = \_dbContext.Query<RebuiltPartsViewModel, BusesModel, RebuiltPartsViewModel>(

query,

(part, bus) =>

{

part.Buses.Add(bus);

return part;

},

splitOn: "ListId"

).GroupBy(r => r.RebuiltStockNum)

.Select(g =>

{

var groupedPart = g.First();

groupedPart.Buses = g.SelectMany(p => p.Buses).ToList();

return groupedPart;

}).ToList();

return View(rebuiltParts);

}

catch (Exception ex)

{

ViewBag.Error = "Error fetching rebuilt parts.";

return View(new List<RebuiltPartsViewModel>());

}

}

// POST: Create a new rebuilt part

[HttpPost]

public IActionResult Create(RebuiltPartsViewModel model)

{

try

{

if (ModelState.IsValid)

{

var query = @"

INSERT INTO SBCES.RbMasterlist (RebuiltStockNum, Keyword)

VALUES (@RebuiltStockNum, @Keyword)";

\_dbContext.Execute(query, new

{

model.RebuiltStockNum,

model.Keyword

});

return RedirectToAction("Index");

}

return View(model);

}

catch (Exception ex)

{

ViewBag.Error = "Error creating rebuilt part.";

return View(model);

}

}

// POST: Update an existing rebuilt part

[HttpPost]

public IActionResult Update(RebuiltPartsViewModel model)

{

try

{

if (ModelState.IsValid)

{

var query = @"

UPDATE SBCES.RbMasterlist

SET Keyword = @Keyword

WHERE RebuiltStockNum = @RebuiltStockNum";

\_dbContext.Execute(query, new

{

model.RebuiltStockNum,

model.Keyword

});

return RedirectToAction("Index");

}

return View(model);

}

catch (Exception ex)

{

ViewBag.Error = "Error updating rebuilt part.";

return View(model);

}

}

// POST: Delete an existing rebuilt part

[HttpPost]

public IActionResult Delete(string rebuiltStockNum)

{

try

{

var query = @"

DELETE FROM SBCES.RbMasterlist

WHERE RebuiltStockNum = @RebuiltStockNum";

\_dbContext.Execute(query, new { RebuiltStockNum = rebuiltStockNum });

return RedirectToAction("Index");

}

catch (Exception ex)

{

ViewBag.Error = "Error deleting rebuilt part.";

return RedirectToAction("Index");

}

}

}

}

**3. TestRebuiltPartIndex.cshtml**

This is the Razor view file for displaying the grids, tabs, and functionality using **Telerik UI components**.

html

CopyEdit

@model IEnumerable<BCES.Models.Parts.RebuiltPartsViewModel>

<h1>Rebuilt Parts</h1>

<div>

@\* Header button to create a new rebuilt part \*@

<button type="button" onclick="window.location.href='@Url.Action("Create", "TestRebuiltPart")'">Create New Part</button>

</div>

@(Html.Kendo().TabStrip()

.Name("partsTabStrip")

.Items(items =>

{

// Tab 1: Rebuilt Parts

items.Add()

.Text("Rebuilt Parts")

.Content(@<text>

@(Html.Kendo().Grid(Model)

.Name("rebuiltPartsGrid")

.Columns(columns =>

{

columns.Bound(p => p.RebuiltStockNum).Title("Rebuilt Stock #");

columns.Bound(p => p.Keyword).Title("Keyword");

columns.Bound(p => p.Buses)

.ClientTemplate("#= Buses.map(function(bus) { return bus.Description; }).join(', ') #")

.Title("Buses");

columns.Command(command =>

{

command.Custom("Update").Click("onUpdate");

command.Custom("Delete").Click("onDelete");

}).Title("Actions");

})

.ToolBar(toolbar =>

{

toolbar.Create().Text("Add New Part");

})

.Editable(editable => editable.Mode(GridEditMode.PopUp))

.Pageable()

.Sortable()

.Filterable()

)

</text>);

// Tab 2: Archived Parts

items.Add()

.Text("Archived Parts")

.Content(@<text>

@(Html.Kendo().Grid(Model)

.Name("archivedPartsGrid")

.Columns(columns =>

{

columns.Bound(p => p.RebuiltStockNum).Title("Rebuilt Stock #");

columns.Bound(p => p.Keyword).Title("Keyword");

columns.Bound(p => p.Buses)

.ClientTemplate("#= Buses.map(function(bus) { return bus.Description; }).join(', ') #")

.Title("Buses");

columns.Command(command =>

{

command.Custom("Update").Click("onUpdate");

command.Custom("Delete").Click("onDelete");

}).Title("Actions");

})

.ToolBar(toolbar =>

{

toolbar.Create().Text("Add New Part");

})

.Editable(editable => editable.Mode(GridEditMode.PopUp))

.Pageable()

.Sortable()

.Filterable()

)

</text>);

})

)

**Summary of Key Features**

1. **Controller**: Handles CRUD operations using **Dapper** and direct SQL queries via \_dbContext.
2. **View**: Uses Telerik's TabStrip and Grid for displaying and managing data with proper templates.
3. **Buses in Grid**: Buses are displayed as a comma-separated list using a ClientTemplate in the Telerik Grid.
4. **Error Handling**: Included in the controller for all operations.

This setup ensures that your application adheres to clean coding practices and utilizes Telerik components effectively! Let me know if you'd like further refinements.

**You said:**

Use edit mode as inline also provide all the required javascipts, including on page load

**ChatGPT said:**



Below is an updated implementation of the TestRebuiltPartIndex.cshtml file using **inline editing mode** for the Telerik Grid. I'll also provide the necessary JavaScript for handling CRUD operations and include an **on-page load script**.

**TestRebuiltPartIndex.cshtml**

This Razor view uses **inline editing** for the Telerik Grid. I'll include all necessary JavaScript for handling actions like creating, updating, and deleting rows.

html

CopyEdit

@model IEnumerable<BCES.Models.Parts.RebuiltPartsViewModel>

<h1>Rebuilt Parts</h1>

<div>

@\* Header button to create a new rebuilt part \*@

<button type="button" id="btnCreate" class="k-button k-primary">

Create New Part

</button>

</div>

<script>

$(document).ready(function () {

// On page load, bind events

$("#btnCreate").click(function () {

$("#rebuiltPartsGrid").data("kendoGrid").addRow();

});

});

</script>

@(Html.Kendo().TabStrip()

.Name("partsTabStrip")

.Items(items =>

{

// Tab 1: Rebuilt Parts

items.Add()

.Text("Rebuilt Parts")

.Content(@<text>

@(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);

columns.Bound(p => p.Buses)

.ClientTemplate("#= Buses.map(function(bus) { return bus.Description; }).join(', ') #")

.Title("Buses").Width(300);

columns.Command(command =>

{

command.Edit(); // Inline Edit Command

command.Custom("Delete").Click("onDelete"); // 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()

.Model(model =>

{

model.Id(p => p.RebuiltStockNum); // Primary Key

model.Field(p => p.RebuiltStockNum).Editable(false); // Make PK non-editable

})

.Read(read => read.Action("GetRebuiltParts", "TestRebuiltPart")) // Read Action

.Create(create => create.Action("Create", "TestRebuiltPart")) // Create Action

.Update(update => update.Action("Update", "TestRebuiltPart")) // Update Action

.Destroy(destroy => destroy.Action("Delete", "TestRebuiltPart")) // Delete Action

)

)

</text>);

// Tab 2: Archived Parts

items.Add()

.Text("Archived Parts")

.Content(@<text>

@(Html.Kendo().Grid<BCES.Models.Parts.RebuiltPartsViewModel>()

.Name("archivedPartsGrid")

.Columns(columns =>

{

columns.Bound(p => p.RebuiltStockNum).Title("Rebuilt Stock #").Width(150).Editable(false);

columns.Bound(p => p.Keyword).Title("Keyword").Width(200);

columns.Bound(p => p.Buses)

.ClientTemplate("#= Buses.map(function(bus) { return bus.Description; }).join(', ') #")

.Title("Buses").Width(300);

columns.Command(command =>

{

command.Edit(); // Inline Edit Command

command.Custom("Delete").Click("onDelete"); // 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()

.Model(model =>

{

model.Id(p => p.RebuiltStockNum); // Primary Key

model.Field(p => p.RebuiltStockNum).Editable(false); // Make PK non-editable

})

.Read(read => read.Action("GetArchivedParts", "TestRebuiltPart")) // Read Action

.Create(create => create.Action("Create", "TestRebuiltPart")) // Create Action

.Update(update => update.Action("Update", "TestRebuiltPart")) // Update Action

.Destroy(destroy => destroy.Action("Delete", "TestRebuiltPart")) // Delete Action

)

)

</text>);

})

)

<script>

// JavaScript: Delete event

function onDelete(e) {

e.preventDefault();

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

var dataItem = grid.dataItem($(e.currentTarget).closest("tr"));

if (confirm("Are you sure you want to delete this record?")) {

$.ajax({

url: "@Url.Action("Delete", "TestRebuiltPart")",

type: "POST",

data: { rebuiltStockNum: dataItem.RebuiltStockNum },

success: function () {

grid.dataSource.read();

alert("Rebuilt part deleted successfully.");

},

error: function () {

alert("Error occurred while deleting the rebuilt part.");

}

});

}

}

// JavaScript: On page load logic

$(document).ready(function () {

// Logic to initialize and set up tabs or other elements, if needed

console.log("Page loaded and ready.");

});

</script>

**Explanation of Changes**

1. **Inline Editing Mode**:
   * The Editable method is set to .Mode(GridEditMode.InLine) to enable inline editing.
   * Columns like RebuiltStockNum are marked as non-editable because it's the primary key.
2. **Create Button**:
   * A header button is provided (btnCreate) that allows the user to add a new row to the grid. This triggers the addRow() method of the grid.
3. **JavaScript Functions**:
   * **onDelete**: Custom logic for handling the deletion of a record. It uses an AJAX call to delete the item server-side and reloads the grid.
   * **Page Load**: A simple $(document).ready() script to initialize and bind events when the page is loaded.
4. **Server-Side Actions**:
   * The DataSource specifies Read, Create, Update, and Delete actions to be handled by the TestRebuiltPartController. These map to GetRebuiltParts, Create, Update, and Delete actions in the controller.
5. **Archived Parts Grid**:
   * A similar grid setup is provided for the "Archived Parts" tab.

**4. Updated TestRebuiltPartController.cs**

Here's the corresponding controller with actions for **CRUD operations**.

csharp

CopyEdit

[HttpGet]

public IActionResult GetRebuiltParts()

{

try

{

var query = @"

SELECT rbm.RebuiltStockNum, rbm.Keyword,

lb.ListId, STRING\_AGG(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

GROUP BY rbm.RebuiltStockNum, rbm.Keyword";

var result = \_dbContext.Query<RebuiltPartsViewModel>(query).ToList();

return Json(result);

}

catch (Exception ex)

{

return BadRequest("Error fetching rebuilt parts.");

}

}

[HttpPost]

public IActionResult Create([FromBody] RebuiltPartsViewModel model)

{

try

{

var query = "INSERT INTO SBCES.RbMasterlist (RebuiltStockNum, Keyword) VALUES (@RebuiltStockNum, @Keyword)";

\_dbContext.Execute(query, model);

return Json(new { success = true });

}

catch (Exception ex)

{

return BadRequest("Error creating part.");

}

}

// Similar logic for Update and Delete