The issue arises because the routeValues parameter is not being passed correctly to the action method. Additionally, some of the controller actions expect a DataSourceRequest parameter, which is not being provided. To fix this, we need to properly construct the parameters for each action method and ensure that the rebuiltPartNum is passed correctly.

Here’s the updated solution:

**Step 1: Update ExportController.cs**

We will modify the GetDataFromController method to handle both parameterless actions and actions that require parameters (e.g., rebuiltPartNum or DataSourceRequest).

**Updated ExportController.cs**

csharp

Copy

using Microsoft.AspNetCore.Mvc;

using OfficeOpenXml;

using System.Collections.Generic;

using System.IO;

using System.Threading.Tasks;

using BCES.Models.Parts;

using Microsoft.Extensions.Logging;

using BCES.Data;

using Microsoft.AspNetCore.Http;

using System;

using Kendo.Mvc.UI;

using Microsoft.Extensions.DependencyInjection;

public class ExportController : Controller

{

private readonly ILogger<ExportController> \_logger;

private readonly DapperContext \_dapperContext;

private readonly IHttpContextAccessor \_httpContextAccessor;

public ExportController(

ILogger<ExportController> logger,

DapperContext dapperContext,

IHttpContextAccessor httpContextAccessor)

{

\_logger = logger;

\_dapperContext = dapperContext;

\_httpContextAccessor = httpContextAccessor;

}

[HttpPost]

public async Task<IActionResult> SaveToExcel(string rebuiltPartNum)

{

try

{

// Fetch data from existing controllers

var mainGridData = await GetDataFromController<RebuiltPartsViewModel>("RebuiltParts", "GetRebuiltPartsView");

var labourDetails = await GetDataFromController<LabourDetailsRebuiltPartsViewModel>("LabourDetails", "GetLabourDetails", new { rebuiltPartNum });

var labourHourSummary = await GetDataFromController<LabourDetailsRebuiltPartsViewModel>("LabourHourSummaryInRbParts", "GetLabourHourSummaryInRbParts", new { rebuiltPartNum });

var materialCostSummary = await GetDataFromController<MaterialCostSummaryInRbViewModel>("MaterialCostSummaryInRbParts", "GetMaterialCostSummaryInRbParts", new { rebuiltPartNum });

var scParts = await GetDataFromController<StockCodedPartsViewModel>("ScPartsUsedInRbParts", "GetScPartsUsedInRbParts", new { rebuiltPartNum });

var nscParts = await GetDataFromController<NscPartsUsedViewModel>("NscPartsUsedInRbParts", "GetNscPartsUsedInRbParts", new { rebuiltPartNum });

var rbParts = await GetDataFromController<RebuiltPartsViewModel>("RbPartsUsedInRbParts", "GetRbPartsUsedInRbParts", new { rebuiltPartNum });

var totalCost = await GetDataFromController<TotalCostInRbModel>("TotalCostInRbParts", "GetTotalCostInRbParts", new { rebuiltPartNum });

// Create Excel package

using (var package = new ExcelPackage())

{

// Add main grid data

var mainGridSheet = package.Workbook.Worksheets.Add("Main Grid");

mainGridSheet.Cells.LoadFromCollection(mainGridData, true);

// Add subgrid data

var labourDetailsSheet = package.Workbook.Worksheets.Add("Labour Details");

labourDetailsSheet.Cells.LoadFromCollection(labourDetails, true);

var labourHourSummarySheet = package.Workbook.Worksheets.Add("Labour Hour Summary");

labourHourSummarySheet.Cells.LoadFromCollection(labourHourSummary, true);

var materialCostSummarySheet = package.Workbook.Worksheets.Add("Material Cost Summary");

materialCostSummarySheet.Cells.LoadFromCollection(materialCostSummary, true);

var scPartsSheet = package.Workbook.Worksheets.Add("SC Parts");

scPartsSheet.Cells.LoadFromCollection(scParts, true);

var nscPartsSheet = package.Workbook.Worksheets.Add("NSC Parts");

nscPartsSheet.Cells.LoadFromCollection(nscParts, true);

var rbPartsSheet = package.Workbook.Worksheets.Add("RB Parts");

rbPartsSheet.Cells.LoadFromCollection(rbParts, true);

var totalCostSheet = package.Workbook.Worksheets.Add("Total Cost");

totalCostSheet.Cells.LoadFromCollection(totalCost, true);

// Save the Excel file

var filePath = Path.Combine(Path.GetTempPath(), $"{rebuiltPartNum}\_Estimate.xlsx");

package.SaveAs(new FileInfo(filePath));

// Return the file for download

var fileBytes = await System.IO.File.ReadAllBytesAsync(filePath);

return File(fileBytes, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", $"{rebuiltPartNum}\_Estimate.xlsx");

}

}

catch (Exception ex)

{

\_logger.LogError(ex, "An error occurred while generating the Excel file.");

return StatusCode(500, "An error occurred while generating the Excel file.");

}

}

private async Task<IEnumerable<T>> GetDataFromController<T>(string controllerName, string actionName, object routeValues = null)

{

try

{

// Resolve the controller type

var controllerType = Type.GetType($"BCES.Controllers.Parts.{controllerName}Controller");

if (controllerType == null)

{

throw new InvalidOperationException($"Controller {controllerName} not found.");

}

// Create an instance of the controller using dependency injection

var controller = ActivatorUtilities.CreateInstance(HttpContext.RequestServices, controllerType);

// Get the action method

var methodInfo = controllerType.GetMethod(actionName);

if (methodInfo == null)

{

throw new InvalidOperationException($"Action {actionName} not found in controller {controllerName}.");

}

// Prepare parameters for the action method

var parameters = methodInfo.GetParameters();

var arguments = new object[parameters.Length];

for (int i = 0; i < parameters.Length; i++)

{

var parameter = parameters[i];

if (parameter.ParameterType == typeof(DataSourceRequest))

{

// Create a default DataSourceRequest for actions that require it

arguments[i] = new DataSourceRequest();

}

else if (routeValues != null && parameter.Name == "rebuiltPartNum")

{

// Pass the rebuiltPartNum parameter

arguments[i] = ((dynamic)routeValues).rebuiltPartNum;

}

else

{

// Pass null for other parameters (if applicable)

arguments[i] = null;

}

}

// Invoke the action method with the prepared arguments

var result = methodInfo.Invoke(controller, arguments) as Task<IActionResult>;

if (result == null)

{

throw new InvalidOperationException($"Action {actionName} did not return a valid result.");

}

// Await the result

var actionResult = await result;

if (actionResult is JsonResult jsonResult)

{

return (IEnumerable<T>)jsonResult.Value;

}

throw new InvalidOperationException($"Action {actionName} did not return a JSON result.");

}

catch (Exception ex)

{

\_logger.LogError(ex, $"An error occurred while fetching data from {controllerName}.{actionName}.");

throw;

}

}

}

**Step 2: Explanation of Changes**

1. **Parameter Handling**:
   * The GetDataFromController method now dynamically constructs the arguments for the action method based on its parameters.
   * If the action requires a DataSourceRequest, it creates a default one.
   * If the action requires a rebuiltPartNum, it extracts it from the routeValues object.
2. **Dynamic Invocation**:
   * The methodInfo.Invoke method is called with the correct arguments, ensuring that the action method receives the required parameters.
3. **Error Handling**:
   * Added error handling to log issues when fetching data from other controllers.

**Step 3: Test the Solution**

1. Ensure the ExportController is registered and accessible.
2. Click the "Save to Excel" button in the Index.cshtml view.
3. Verify that the Excel file is generated and contains the expected data for the selected rebuiltPartNum.

**Step 4: Example Output**

For a rebuiltPartNum value of 12345, the Excel file will contain:

* A "Main Grid" sheet with the main grid data.
* A "Labour Details" sheet with labour details for rebuiltPartNum = 12345.
* A "Labour Hour Summary" sheet with labour hour summary for rebuiltPartNum = 12345.
* Other sheets with corresponding data for rebuiltPartNum = 12345.