**Step 1: Fix the Excel File Generation**

We need to ensure that the Excel file is generated correctly and saved in the proper format. Here's the updated SaveToExcel method:

**Updated SaveToExcel Method**

csharp

Copy

[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 to a memory stream

var memoryStream = new MemoryStream();

package.SaveAs(memoryStream);

memoryStream.Position = 0; // Reset the stream position to the beginning

// Return the file for download

return File(memoryStream, "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.");

}

}

**Step 2: Key Fixes in the Code**

1. **Memory Stream Handling**:
   * The MemoryStream is reset to position 0 after saving the Excel file. This ensures that the file is read from the beginning when returned.
2. **Correct MIME Type**:
   * The MIME type application/vnd.openxmlformats-officedocument.spreadsheetml.sheet is used for .xlsx files.
3. **Proper Data Loading**:
   * The LoadFromCollection method is used to load data into the Excel sheets. Ensure that the data is not null or invalid.

**Step 3: Verify the Data**

Ensure that the data being passed to the LoadFromCollection method is valid and does not contain any problematic characters or formatting.

**Example: Check for Null or Invalid Data**

csharp

Copy

if (mainGridData == null || !mainGridData.Any())

{

throw new InvalidOperationException("Main grid data is null or empty.");

}

**Step 4: 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 opens without corruption.

**Step 5: Debugging Tips**

If the issue persists:

1. **Check the Data**:
   * Add breakpoints and inspect the data being passed to the LoadFromCollection method to ensure it is valid.
2. **Check the Memory Stream**:
   * Verify that the MemoryStream is correctly populated and reset to position 0.
3. **Test with Simple Data**:
   * Create a simple Excel file with hardcoded data to ensure the issue is not with the data itself.

**Example: Simple Excel File Generation**

csharp

Copy

[HttpPost]

public IActionResult TestExcel()

{

using (var package = new ExcelPackage())

{

var sheet = package.Workbook.Worksheets.Add("Test Sheet");

sheet.Cells["A1"].Value = "Hello";

sheet.Cells["B1"].Value = "World";

var memoryStream = new MemoryStream();

package.SaveAs(memoryStream);

memoryStream.Position = 0;

return File(memoryStream, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", "Test.xlsx");

}

}

**Summary**

* The SaveToExcel method now correctly handles the MemoryStream and ensures the file is not corrupted.
* Proper MIME type and file format are used.
* Error handling and logging are included for robustness.
* Test with simple data to isolate the issue if the problem persists.