To achieve your goal of calling the SaveToExcel function from ExportController in the RebuiltPartsController and using the returned MemoryStream for email attachment, we need to make the following changes:

1. **Modify SaveToExcel in ExportController to return a MemoryStream**.
2. **Call SaveToExcel from RebuiltPartsController and use the returned MemoryStream for email attachment**.

Below are the **full updated files** with the necessary changes:

**1. Updated ExportController.cs**

Modify the SaveToExcel method to return a MemoryStream instead of a FileResult.

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;

namespace BCES.Controllers.Parts

{

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;

}

[HttpGet]

public async Task<MemoryStream> SaveToExcel([FromQuery] string rebuiltPartNum)

{

try

{

// Fetch data from existing controllers

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 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 memoryStream;

}

}

catch (Exception ex)

{

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

throw; // Re-throw the exception to handle it in the calling method

}

}

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);

// Handle the result based on its type

if (result is Task taskResult)

{

// Await the task to get the actual result

await taskResult;

// Use reflection to get the Result property of the Task

var resultProperty = taskResult.GetType().GetProperty("Result");

if (resultProperty != null)

{

var actionResult = resultProperty.GetValue(taskResult);

if (actionResult is JsonResult jsonResult)

{

// Check if the JsonResult.Value is a DataSourceResult

if (jsonResult.Value is DataSourceResult dataSourceResult)

{

// Return the Data property as IEnumerable<T>

return (IEnumerable<T>)dataSourceResult.Data;

}

else

{

throw new InvalidOperationException($"Action {actionName} returned an unexpected result type: {jsonResult.Value.GetType().Name}.");

}

}

else if (actionResult is IActionResult)

{

throw new InvalidOperationException($"Action {actionName} returned an unexpected result type: {actionResult.GetType().Name}.");

}

}

}

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

}

catch (Exception ex)

{

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

throw;

}

}

}

}

**2. Updated RebuiltPartsController.cs**

Call the SaveToExcel method from ExportController and use the returned MemoryStream for email attachment.

csharp

Copy

using Dapper;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Data;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using BCES.Models.Parts;

using MailKit.Net.Smtp;

using MimeKit;

using MailKit.Security;

using OfficeOpenXml;

using Microsoft.Extensions.DependencyInjection;

namespace BCES.Controllers.Parts

{

public class RebuiltPartsController : BaseController

{

private readonly DapperContext \_db;

private readonly IDbConnection \_dbConnection;

private readonly IHttpContextAccessor \_httpContextAccessor;

private readonly IServiceProvider \_serviceProvider;

public RebuiltPartsController(

DapperContext dapper,

IHttpContextAccessor httpContextAccessor,

IServiceProvider serviceProvider) : base(dapper, httpContextAccessor)

{

\_db = dapper;

\_dbConnection = \_db.CreateConnection();

\_httpContextAccessor = httpContextAccessor;

\_serviceProvider = serviceProvider;

}

[HttpPost]

public async Task<IActionResult> SendEmail(string to, string subject, string body, string rebuiltPartNum)

{

try

{

// Resolve the ExportController from the service provider

var exportController = \_serviceProvider.GetRequiredService<ExportController>();

// Generate the Excel file in memory by calling SaveToExcel

var memoryStream = await exportController.SaveToExcel(rebuiltPartNum);

// Send the email with the attached Excel file

var message = new MimeMessage();

message.From.Add(new MailboxAddress("Your Name", "your-email@example.com"));

message.To.Add(new MailboxAddress("", to));

message.Subject = subject;

message.Body = new TextPart("plain") { Text = body };

var attachment = new MimePart("application", "vnd.openxmlformats-officedocument.spreadsheetml.sheet")

{

Content = new MimeContent(memoryStream),

ContentDisposition = new ContentDisposition(ContentDisposition.Attachment),

ContentTransferEncoding = ContentEncoding.Base64,

FileName = $"{rebuiltPartNum}\_RebuiltParts\_Estimate.xlsx"

};

var multipart = new Multipart("mixed");

multipart.Add(message.Body);

multipart.Add(attachment);

message.Body = multipart;

using (var client = new SmtpClient())

{

await client.ConnectAsync("smtp.example.com", 587, SecureSocketOptions.StartTls);

await client.AuthenticateAsync("your-email@example.com", "your-email-password");

await client.SendAsync(message);

await client.DisconnectAsync(true);

}

return Json(new { success = true });

}

catch (Exception ex)

{

// Log the exception

Console.WriteLine($"Error sending email: {ex.Message}");

return Json(new { success = false, message = ex.Message });

}

}

}

}

**Key Changes**

1. **SaveToExcel in ExportController**:
   * Now returns a MemoryStream instead of a FileResult.
2. **SendEmail in RebuiltPartsController**:
   * Calls SaveToExcel from ExportController using dependency injection.
   * Uses the returned MemoryStream to attach the Excel file to the email.
3. **Dependency Injection**:
   * The ExportController is resolved using IServiceProvider in RebuiltPartsController.

**Why These Changes?**

* **Reusability**: The SaveToExcel method is now reusable and can be called from any controller.
* **Memory Efficiency**: The Excel file is generated in memory and attached directly to the email without saving it to disk.
* **Separation of Concerns**: The ExportController handles Excel file generation, while the RebuiltPartsController handles email sending.

**Folder Structure**

Copy

Controllers/

├── ExportController.cs

├── RebuiltPartsController.cs

This implementation ensures that the Excel file is generated in memory, attached to the email, and no unnecessary files are saved to disk. Let me know if you need further assistance!