To achieve your goal of **calling SaveToExcel** in the onEmailEstimateClick function, **opening the Kendo Window**, and **attaching the Excel file** generated by SaveToExcel, we need to make the following changes:

1. **Call SaveToExcel to Generate the Excel File**:
   * Use window.location.href to trigger the SaveToExcel action and generate the Excel file.
2. **Attach the Excel File to the Email**:
   * Once the Excel file is generated, pass its path or content to the SendEmail action for attachment.
3. **Open the Kendo Window**:
   * Open the Kendo Window after the Excel file is generated.

**Updated onEmailEstimateClick Function**

Here’s the updated onEmailEstimateClick function:

javascript

Copy

function onEmailEstimateClick(e) {

e.preventDefault();

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

var rebuiltPartNum = dataItem.RebuiltStockNum;

// Set the subject with the rebuilt number and today's date

var today = new Date();

var subject = `Rebuilt Part Estimates for ${rebuiltPartNum} - Date ${today.toLocaleDateString()} ${today.toLocaleTimeString()}`;

// Set the form values

$("#to").val("");

$("#subject").val(subject);

$("#body").val(`Dear Sir/Madam,\nPlease find attached here with Rebuilt Part Estimate for ${rebuiltPartNum}.\n\nThank you`);

// Generate the Excel file by calling SaveToExcel

window.location.href = '@Url.Action("SaveToExcel", "Export")' + '?rebuiltPartNum=' + rebuiltPartNum;

// Open the Kendo Window

var window = $("#emailEstimateWindow").data("kendoWindow");

if (window) {

window.center().open();

} else {

console.error("Kendo Window not found.");

}

}

**Updated ExportController.cs**

The SaveToExcel action in the ExportController should return the Excel file for download. No changes are needed here since it already returns the file.

csharp

Copy

[HttpGet]

public async Task<IActionResult> 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 the file for download

return File(memoryStream, "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet", $"{rebuiltPartNum}\_RebuiltParts\_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.");

}

}

**Updated SendEmail Action in RebuiltPartsController.cs**

The SendEmail action should accept the Excel file as an attachment. Here’s how you can modify it:

csharp

Copy

[HttpPost]

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

{

try

{

// Generate the Excel file by calling SaveToExcel

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

// Call SaveToExcel to generate the file

var excelResult = await SaveToExcel(rebuiltPartNum);

if (excelResult is FileContentResult fileContentResult)

{

// Save the file to the temporary location

await System.IO.File.WriteAllBytesAsync(filePath, fileContentResult.FileContents);

}

else

{

throw new Exception("Failed to generate the Excel file.");

}

// 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(System.IO.File.OpenRead(filePath)),

ContentDisposition = new ContentDisposition(ContentDisposition.Attachment),

ContentTransferEncoding = ContentEncoding.Base64,

FileName = Path.GetFileName(filePath)

};

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 (you can use a logging framework like Serilog or NLog)

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

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

}

}

**Key Changes**

1. **Call SaveToExcel in onEmailEstimateClick**:
   * Use window.location.href to trigger the SaveToExcel action and generate the Excel file.
2. **Attach the Excel File in SendEmail**:
   * The SendEmail action saves the Excel file to a temporary location and attaches it to the email.
3. **Open the Kendo Window**:
   * The Kendo Window is opened after the Excel file is generated.

**Folder Structure**

Copy

Controllers/

├── ExportController.cs

├── RebuiltPartsController.cs

Views/

└── Parts/

└── RebuiltParts/

├── Index.cshtml

└── \_EmailEstimatePopup.cshtml

**Summary of Changes**

| **File** | **Changes** |
| --- | --- |
| **JavaScript** | Added window.location.href to call SaveToExcel. |
| **ExportController.cs** | No changes (already returns the Excel file). |
| **RebuiltPartsController.cs** | Updated SendEmail to attach the Excel file. |

This implementation ensures that the Excel file is generated, attached to the email, and the Kendo Window is opened correctly. Let me know if you need further assistance!