The issue occurs because the DataSourceRequest object is being passed to the action method, but the action method is not returning a valid result. This can happen if the action method is not properly handling the DataSourceRequest or if the method signature does not match the expected parameters.

To resolve this issue, we need to ensure that:

1. The DataSourceRequest is correctly passed to the action method.
2. The action method returns a valid JsonResult or Task<IActionResult>.

Let’s debug and fix the issue step by step.

**Step 1: Debugging the Issue**

1. **Check the Action Method Signature**:
   * Ensure that the action method in the target controller (e.g., RebuiltPartsController) has the correct signature.
   * For example, if the action method expects a DataSourceRequest, it should look like this:

csharp

Copy

public async Task<ActionResult> GetRebuiltPartsView([DataSourceRequest] DataSourceRequest request)

{

var data = await GetRebuiltPartsData();

return Json(data.ToDataSourceResult(request));

}

1. **Verify the Return Type**:
   * Ensure that the action method returns a JsonResult or Task<IActionResult>.
2. **Check the arguments Array**:
   * Ensure that the arguments array is correctly populated with the required parameters.

**Step 2: Fix the GetDataFromController Method**

We need to ensure that the arguments array is correctly populated and that the action method is invoked properly.

**Updated GetDataFromController Method**

csharp

Copy

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);

if (result is Task<IActionResult> taskResult)

{

var actionResult = await taskResult;

if (actionResult is JsonResult jsonResult)

{

return (IEnumerable<T>)jsonResult.Value;

}

}

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;

}

}

**Step 3: Verify the Action Method in the Target Controller**

Ensure that the action method in the target controller (e.g., RebuiltPartsController) is implemented correctly.

**Example: RebuiltPartsController**

csharp

Copy

public class RebuiltPartsController : BaseController

{

private readonly DapperContext \_db;

private readonly IDbConnection \_dbConnection;

public RebuiltPartsController(DapperContext dapper, IHttpContextAccessor httpContextAccessor)

: base(dapper, httpContextAccessor)

{

\_db = dapper;

\_dbConnection = \_db.CreateConnection();

}

[HttpGet]

public async Task<ActionResult> GetRebuiltPartsView([DataSourceRequest] DataSourceRequest request)

{

var data = await GetRebuiltPartsData();

return Json(data.ToDataSourceResult(request));

}

private async Task<IEnumerable<RebuiltPartsViewModel>> GetRebuiltPartsData()

{

// Fetch data from the database

var query = "SELECT \* FROM RebuiltParts";

return await \_dbConnection.QueryAsync<RebuiltPartsViewModel>(query);

}

}

**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 contains the expected data for the selected rebuiltPartNum.

**Step 5: Debugging Tips**

If the issue persists:

1. **Check the arguments Array**:
   * Add a breakpoint in the GetDataFromController method and inspect the arguments array to ensure it is correctly populated.
2. **Check the Action Method**:
   * Add a breakpoint in the target action method (e.g., GetRebuiltPartsView) to ensure it is being called with the correct parameters.
3. **Check the Return Value**:
   * Ensure that the action method returns a valid JsonResult or Task<IActionResult>.