The issue arises because the result object is not being correctly cast to Task<IActionResult>. This can happen if the action method returns a Task<JsonResult> or if the result object is not properly unwrapped.

To fix this, we need to handle the result object more carefully and ensure that we correctly await the task and extract the JsonResult.

**Step 1: Debugging the Issue**

1. **Check the Return Type of the Action Method**:
   * If the action method returns Task<JsonResult>, the result object will be a Task<JsonResult>, not a Task<IActionResult>.
   * We need to handle both cases (Task<JsonResult> and Task<IActionResult>).
2. **Unwrap the Task**:
   * If the result is a Task, we need to await it to get the actual result.

**Step 2: Update the GetDataFromController Method**

We will update the GetDataFromController method to handle both Task<JsonResult> and Task<IActionResult>.

**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);

// 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)

{

return (IEnumerable<T>)jsonResult.Value;

}

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;

}

}

**Step 3: Explanation of Changes**

1. **Handle Task Results**:
   * If the result is a Task, we await it and use reflection to access the Result property of the Task.
2. **Check for JsonResult**:
   * If the Result property is a JsonResult, we extract its Value and return it.
3. **Error Handling**:
   * If the Result property is not a JsonResult, we throw an exception with a descriptive message.

**Step 4: Verify the Action Method in the Target Controller**

Ensure that the action method in the target controller (e.g., RebuiltPartsController) returns a Task<JsonResult>.

**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<JsonResult> 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 5: 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 6: Debugging Tips**

If the issue persists:

1. **Check the Result Property**:
   * Add a breakpoint in the GetDataFromController method and inspect the Result property of the Task.
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.

**Summary**

* The GetDataFromController method now correctly handles Task<JsonResult> and Task<IActionResult>.