The error occurs because the JsonResult.Value is of type Kendo.Mvc.UI.DataSourceResult, not IEnumerable<T>. The DataSourceResult contains the data in its Data property, which is of type IEnumerable<T>.

To fix this issue, we need to extract the Data property from the DataSourceResult and return it as IEnumerable<T>.

**Step 1: Update the GetDataFromController Method**

We will update the GetDataFromController method to handle DataSourceResult and extract the Data property.

**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)

{

// 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;

}

}

**Step 2: Explanation of Changes**

1. **Handle DataSourceResult**:
   * If the JsonResult.Value is a DataSourceResult, we extract its Data property and return it as IEnumerable<T>.
2. **Error Handling**:
   * If the JsonResult.Value is not a DataSourceResult, we throw an exception with a descriptive message.

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

Ensure that the action method in the target controller (e.g., LabourDetailsController) returns a JsonResult with a DataSourceResult.

**Example: LabourDetailsController**

csharp

Copy

public class LabourDetailsController : BaseController

{

private readonly DapperContext \_db;

private readonly IDbConnection \_dbConnection;

public LabourDetailsController(DapperContext dapper, IHttpContextAccessor httpContextAccessor)

: base(dapper, httpContextAccessor)

{

\_db = dapper;

\_dbConnection = \_db.CreateConnection();

}

[HttpGet]

public async Task<JsonResult> GetLabourDetails([DataSourceRequest] DataSourceRequest request, [FromQuery] string rebuiltPartNum)

{

var data = await GetLabourDetailsData(rebuiltPartNum);

return Json(data.ToDataSourceResult(request));

}

private async Task<IEnumerable<LabourDetailsRebuiltPartsViewModel>> GetLabourDetailsData(string rebuiltPartNum)

{

// Fetch data from the database

var query = "SELECT \* FROM LabourDetails WHERE RebuiltPartNum = @RebuiltPartNum";

return await \_dbConnection.QueryAsync<LabourDetailsRebuiltPartsViewModel>(query, new { RebuiltPartNum = rebuiltPartNum });

}

}

**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 JsonResult.Value**:
   * Add a breakpoint in the GetDataFromController method and inspect the JsonResult.Value to ensure it is a DataSourceResult.
2. **Check the Action Method**:
   * Add a breakpoint in the target action method (e.g., GetLabourDetails) to ensure it is being called with the correct parameters.
3. **Check the Return Value**:
   * Ensure that the action method returns a valid JsonResult with a DataSourceResult.