Here’s a cleaned-up version of the **Create**, **Update**, and **Get** methods:

**1. Get Data (Read)**

This method reads the HrsReqd column from the database, splits it into LabourHrs and LabourMins, and sends it to the view.

csharp

CopyEdit

[HttpGet]

[AllowAnonymous]

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

{

if (string.IsNullOrEmpty(rebuiltPartNum))

{

return BadRequest("RebuiltPartNum is required.");

}

try

{

var data = await GetLabourDetailsData(rebuiltPartNum, gridType);

return Json(data.ToDataSourceResult(request));

}

catch (Exception ex)

{

// Log the error

Console.WriteLine($"Error in GetLabourDetails: {ex.Message}");

return StatusCode(500, "An error occurred while fetching data.");

}

}

private async Task<IEnumerable<LabourDetailsRebuiltPartsViewModel>> GetLabourDetailsData(string partNum, string gridType)

{

try

{

var query = @"

SELECT

EL.Id,

EL.LinkNumber,

EL.LabourDefn,

EL.CostCentre,

EL.Task,

REPLACE(EL.Usage, ',', '.') AS [Usage],

EL.HrsReqd,

ES.RatePerHour,

EL.TimeAddition

FROM

SBCES.EmployeeLabour EL

JOIN SBCES.EmplClass EC ON EL.LabourDefn = UPPER(EC.LabourDefn)

JOIN SBCES.EmployeeSalaries ES ON EC.WageGroup = ES.WageGroup

WHERE

EL.LinkNumber = @RebuiltPartNumber";

var parameters = new { RebuiltPartNumber = partNum };

var result = await \_dbConnection.QueryAsync<LabourDetailsRebuiltPartsViewModel>(query, parameters);

// Split HrsReqd into LabourHrs and LabourMins

foreach (var item in result)

{

if (!string.IsNullOrEmpty(item.HrsReqd))

{

var parts = item.HrsReqd.Split(':');

item.LabourHrs = int.Parse(parts[0]);

item.LabourMins = int.Parse(parts[1]);

}

}

return result;

}

catch (Exception ex)

{

Console.WriteLine($"Error in GetLabourDetailsData: {ex.Message}");

throw; // Rethrow the exception for centralized handling

}

}

**2. Add Data (Create)**

This method converts LabourHrs and LabourMins into a HrsReqd string before saving it to the database.

csharp

CopyEdit

[HttpPost]

[AllowAnonymous]

public async Task<IActionResult> CreateLabourDetail(

[DataSourceRequest] DataSourceRequest request,

[Bind(Prefix = "models")] IEnumerable<LabourDetailsRebuiltPartsViewModel> models,

[FromQuery] string rebuiltPartNum)

{

if (!ModelState.IsValid)

return Json(models.ToDataSourceResult(request, ModelState));

using (var transaction = \_dbConnection.BeginTransaction())

{

try

{

foreach (var model in models)

{

model.HrsReqd = $"{model.LabourHrs}:{model.LabourMins:D2}";

string query = @"

INSERT INTO SBCES.EMPLOYEELABOUR (

LABOURDEFN, DATEENTERED, LINKNUMBER, TYPEID,

COSTCENTRE, TASK, LABOURTYPE, USAGE, HRSREQD,

ADJHRS, DATEREVISED, TIMEADDITION, REBUILTPARTNUM, LASTMODIFIEDBY

) VALUES (

@LabourDefn, @DateEntered, @LinkNumber, @TypeId,

@CostCentre, @Task, @LabourType, @Usage, @HrsReqd,

@AdjHrs, @DateRevised, @TimeAddition, @RebuiltPartNum, @LastModifiedBy

)";

var parameters = new

{

model.LabourDefn,

DateEntered = DateTime.Now,

LinkNumber = rebuiltPartNum,

TypeId = "RB",

model.CostCentre,

model.Task,

model.LabourType,

model.Usage,

model.HrsReqd,

model.AdjHrs,

DateRevised = DateTime.Now,

model.TimeAddition,

RebuiltPartNum = rebuiltPartNum,

LastModifiedBy = ViewBag.UserName

};

await \_dbConnection.ExecuteAsync(query, parameters, transaction);

}

transaction.Commit();

return Json(models.ToDataSourceResult(request));

}

catch (Exception ex)

{

transaction.Rollback();

Console.WriteLine($"Error in CreateLabourDetail: {ex.Message}");

ModelState.AddModelError("", "An error occurred while saving data.");

return Json(models.ToDataSourceResult(request, ModelState));

}

}

}

**3. Update Data**

This method updates the HrsReqd value and other fields in the database.

csharp

CopyEdit

[AcceptVerbs("Post")]

public IActionResult UpdateLabourDetail(

[DataSourceRequest] DataSourceRequest request,

[Bind(Prefix = "models")] IEnumerable<LabourDetailsRebuiltPartsViewModel> labourDetails)

{

if (labourDetails != null && ModelState.IsValid)

{

try

{

foreach (var model in labourDetails)

{

model.HrsReqd = $"{model.LabourHrs}:{model.LabourMins:D2}";

string query = @"

UPDATE SBCES.EMPLOYEELABOUR

SET

LABOURDEFN = @LabourDefn,

COSTCENTRE = @CostCentre,

TASK = @Task,

USAGE = @Usage,

HRSREQD = @HrsReqd,

REBUILTPARTNUM = @LinkNumber

WHERE Id = @Id";

var parameters = new

{

model.LabourDefn,

model.CostCentre,

model.Task,

model.Usage,

model.HrsReqd,

model.LinkNumber,

model.Id

};

\_dbConnection.Execute(query, parameters);

}

return Json(labourDetails.ToDataSourceResult(request));

}

catch (Exception ex)

{

Console.WriteLine($"Error in UpdateLabourDetail: {ex.Message}");

ModelState.AddModelError("", "An error occurred while updating data.");

return Json(labourDetails.ToDataSourceResult(request, ModelState));

}

}

return Json(labourDetails.ToDataSourceResult(request, ModelState));

}

**✅ Improvements Made**

✔️ Added consistent exception handling  
✔️ Cleaned up conversion logic  
✔️ Improved formatting and logging  
✔️ Applied async/await for better performance  
✔️ Centralized exception handling