Here’s how you can implement the requirements step by step.

**1. ViewModel: LabourDetailsRebuiltPartsViewModel.cs**

This ViewModel represents the structure of the table you provided.

namespace YourNamespace.Models

{

public class LabourDetailsRebuiltPartsViewModel

{

public string LabourType { get; set; }

public string CC { get; set; }

public string Task { get; set; }

public int Usage { get; set; }

public double Time { get; set; }

public double WrenchTime { get; set; }

public DateTime DateRevised { get; set; }

}

}

**2. Partial View: LabourDetailsRebuiltParts.cshtml**

The partial view defines the Kendo UI Grid.

@model IEnumerable<YourNamespace.Models.LabourDetailsRebuiltPartsViewModel>

@{

ViewData["Title"] = "Labour Details Rebuilt Parts";

}

<h4>Labour Details Rebuilt Parts</h4>

@(Html.Kendo().Grid<LabourDetailsRebuiltPartsViewModel>()

.Name("LabourRebuiltPartGrid")

.Columns(columns =>

{

columns.Bound(c => c.LabourType).Title("Labour Type").Filterable(ftb => ftb.Cell(cell => cell.Operator("contains")));

columns.Bound(c => c.CC).Title("CC").Filterable(ftb => ftb.Cell(cell => cell.Operator("contains")));

columns.Bound(c => c.Task).Title("Task").Filterable(ftb => ftb.Cell(cell => cell.Operator("contains")));

columns.Bound(c => c.Usage).Title("Usage").Filterable(ftb => ftb.Cell(cell => cell.Operator("gte")));

columns.Bound(c => c.Time).Title("Time").Format("{0:N2}").Filterable(ftb => ftb.Cell(cell => cell.Operator("gte")));

columns.Bound(c => c.WrenchTime).Title("Wrench Time").Format("{0:N2}").Filterable(ftb => ftb.Cell(cell => cell.Operator("gte")));

columns.Bound(c => c.DateRevised).Title("Date Revised").Format("{0:yyyy-MM-dd hh:mm tt}");

columns.Command(command => command.Edit()).Title("Actions");

})

.ToolBar(toolbar => toolbar.Create())

.Editable(editable => editable.Mode(GridEditMode.InLine))

.Pageable()

.Sortable()

.Filterable(ftb => ftb.Mode(GridFilterMode.Row))

.Scrollable()

.DataSource(dataSource => dataSource

.Ajax()

.Model(model =>

{

model.Id(c => c.LabourType); // Assuming LabourType is a unique identifier

model.Field(c => c.LabourType);

})

.Read(read => read.Action("GetLabourDetails", "LabourDetails"))

.Create(create => create.Action("CreateLabourDetail", "LabourDetails"))

.Update(update => update.Action("UpdateLabourDetail", "LabourDetails"))

.Destroy(delete => delete.Action("DeleteLabourDetail", "LabourDetails"))

)

)

**3. Controller: LabourDetailsController.cs**

The controller handles CRUD operations.

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Kendo.Mvc.Extensions;

using Kendo.Mvc.UI;

using YourNamespace.Models;

namespace YourNamespace.Controllers

{

public class LabourDetailsController : Controller

{

private static List<LabourDetailsRebuiltPartsViewModel> \_data = new List<LabourDetailsRebuiltPartsViewModel>

{

new LabourDetailsRebuiltPartsViewModel

{

LabourType = "MECHANICAL",

CC = "05H2",

Task = "COMPONENT CLEANER",

Usage = 10,

Time = 2.0,

WrenchTime = 1.40,

DateRevised = DateTime.Parse("2024-11-01 15:45")

},

new LabourDetailsRebuiltPartsViewModel

{

LabourType = "COACH TECHNICIAN",

CC = "05H2",

Task = "CONTROL INSPECTION",

Usage = 100,

Time = 4.0,

WrenchTime = 3.05,

DateRevised = DateTime.Parse("2020-06-25 13:45")

}

};

[HttpGet]

public IActionResult GetLabourDetails([DataSourceRequest] DataSourceRequest request)

{

return Json(\_data.ToDataSourceResult(request));

}

[HttpPost]

public IActionResult CreateLabourDetail([DataSourceRequest] DataSourceRequest request, LabourDetailsRebuiltPartsViewModel model)

{

if (ModelState.IsValid)

{

model.DateRevised = DateTime.Now; // Example: auto-update the revised date

\_data.Add(model);

}

return Json(new[] { model }.ToDataSourceResult(request, ModelState));

}

[HttpPost]

public IActionResult UpdateLabourDetail([DataSourceRequest] DataSourceRequest request, LabourDetailsRebuiltPartsViewModel model)

{

if (ModelState.IsValid)

{

var existing = \_data.FirstOrDefault(m => m.LabourType == model.LabourType);

if (existing != null)

{

existing.CC = model.CC;

existing.Task = model.Task;

existing.Usage = model.Usage;

existing.Time = model.Time;

existing.WrenchTime = model.WrenchTime;

existing.DateRevised = model.DateRevised;

}

}

return Json(new[] { model }.ToDataSourceResult(request, ModelState));

}

[HttpPost]

public IActionResult DeleteLabourDetail([DataSourceRequest] DataSourceRequest request, LabourDetailsRebuiltPartsViewModel model)

{

var existing = \_data.FirstOrDefault(m => m.LabourType == model.LabourType);

if (existing != null)

{

\_data.Remove(existing);

}

return Json(new[] { model }.ToDataSourceResult(request, ModelState));

}

}

}

**4. Notes**

* **Namespace and Paths**: Update namespaces and controller paths as per your project structure.
* **CRUD Logic**: This example uses an in-memory list \_data. Replace this with your database interaction logic if needed.
* **Validation**: Add validation attributes to the LabourDetailsRebuiltPartsViewModel if required.

This setup creates a functional grid with Create, Edit, and Delete operations for your Labour details.

@{

ViewData["Title"] = "Labour Details Index";

}

<h2>Labour Details</h2>

<div>

@await Html.PartialAsync("LabourDetailsRebuiltParts")

</div>