I am developing an HR Saas system using Blazor WebApp. I am using MudBlazor for designing the UI. I have designed an employee search razor page where I used MudAutocomplete components to display the list of employee types and departments. Now, when I navigate to another page then comeback to the employee search page, it triggers the OnInitialized() event where it calls the web service to retrieve the data from the backend data. Tell me how to preserve the data for these MudAutocomplete components so that when navigating away from the page, it gets save and navigating back will restore that saved data and map it to these components.

I have put an EditForm component in my Employee.razor page as follow:

<EditForm Model="@employee" OnValidSubmit="OnValidSubmit">

<MudPaper Class="pa-4 mud-height-full">

<MudText Typo="Typo.subtitle2">Validation Summary</MudText>

@if (\_success)

{

<MudText Color="Color.Success">Success</MudText>

}

else

{

<MudText Color="@Color.Error">

<ValidationSummary />

</MudText>

}

</MudPaper>

</EditForm>

However, upon running the application, the following error occurred:

The child content element 'CellTemplate' of component 'TemplateColumn' uses the same parameter name ('context') as enclosing child content element 'ChildContent' of component 'EditForm'. Specify the parameter name like: '<CellTemplate Context="another\_name"> to resolve the ambiguity

Tell me how to fix this issue.

I have added a Label attribute to my EmployeeDTO class but an error has occured during compilation. The error says: The type of namespace "LabelAttribute" could not be found (are you missing a using directive or an assembly reference. Note that I have added the following using directive in the C# code behind:

using System.ComponentModel.DataAnnotations;

Tell me how to fix this issue.

Provide a sample data validation using an EditForm that contains 3 fields such as Employee Name, Date of Birth, and Email Address where Employee Name is required, Date of Birth must be in dd/MM/yyyy format, and Email Address should be valid email address and in correct format. The validation summary must be displayed at the top section of the razor page and will only be shown if there are validation error.

I have a property in my Employee class called OfficialEmail. I set the [EmailAddress] data annotation attribute. However, when I run the application, it throws data validation error that says: The Official Email field is not a valid e-mail address, though this property is not mandatory where there is no [Required] attribute set.

Provide a complete sample of saving employee details changes from the Blazor page user interface to the backend database using clean architecture, with the Application and Infrastructure layers and using EF Core.

I am using a clean architecture pattern for my Blazor WebApp HR application. I have an entity called “EmergencyContact” with the following properties:

public int AutoId { get; set; }

public string ContactPerson { get; set; } = null!;

public string RelationCode { get; set; } = null!;

public string? Relation { get; set; } = null;

public string MobileNo { get; set; } = null!;

public string? LandlineNo { get; set; } = null;

public string? Address { get; set; } = null;

I am writing a code to save data changes from this entity to the backend database in SQL Server. I have class called EmergencyContactDto with the same properties as the “EmergencyContact” entity. I created 2 list collections: one for the EmergencyContact entity and the other one for the EmergencyContactDto. Now, write a code that select all items from List< EmergencyContact> that does not exist in List< EmergencyContactDto >

I have a DTO called EmployeeDTO with the following properties:

public int EmployeeId { get; set; }

public string FirstName { get; set; } = null!;

public string? MiddleName { get; set; }

public string LastName { get; set; } = null!;

I need to check if DTO has changed and if so, proceed with saving the data to the database using EF Core clean architecture. Tell me how to check from the Application layer if DTO has been changed.

I am designing a Blazor WebApp HRMS system using MudBlazor UI Framework. I want to create a form where all the fields are bound to a DTO using the bind-value and For attributes. Now, in my DTO class, there are several required fields that I want to highlight in the data entry form so that users would be able to know which fields are mandatory. This can be an asterisk character that is shown beside the field label or whatever best practices in the software development industry. Below is property listing of the DTO class.

public int EmployeeId { get; set; }

[Required(ErrorMessage = "First Name is required")]

[Label("First Name")]

[StringLength(50, ErrorMessage = "First Name length can't be more than 50 characters.")]

public string FirstName { get; set; } = null!;

[Display(Name = "Middle Name")]

[StringLength(50, ErrorMessage = "Middle Name length can't be more than 50 characters.")]

public string? MiddleName { get; set; }

[Required(ErrorMessage = "Last Name is required")]

[Display(Name = "Last Name")]

[StringLength(50, ErrorMessage = "Last Name length can't be more than 50 characters.")]

public string LastName { get; set; } = null!;

[Required(ErrorMessage = "Position is required")]

[Display(Name = "Position")]

[StringLength(100, ErrorMessage = "Position length can't be more than 100 characters.")]

public string Position { get; set; } = null!;

[Display(Name = "Date of Birth")]

[DataType(DataType.Date)]

public DateTime? DOB { get; set; }

public string NationalityCode { get; set; } = null!;

[Required(ErrorMessage = "Nationality is required")]

[Display(Name = "Nationality")]

public string NationalityDesc { get; set; } = null!;

public string ReligionCode { get; set; } = null!;

[Required(ErrorMessage = "Religion is required")]

[Display(Name = "Religion")]

public string ReligionDesc { get; set; } = null!;

public string GenderCode { get; set; } = null!;

[Required(ErrorMessage = "Gender is required")]

[Display(Name = "Gender")]

public string GenderDesc { get; set; } = null!;

public string MaritalStatusCode { get; set; } = null!;

[Required(ErrorMessage = "Marital Status is required")]

[Display(Name = "Marital Status")]

public string MaritalStatusDesc { get; set; } = null!;

public string? Salutation { get; set; }

[Display(Name = "Salutation")]

public string? SalutationDesc { get; set; }

I have an entity class with the following properties. Tell me how to get the maximum value of the EmployeeNo property.

public int EmployeeNo { get; set; }

public string FirstName { get; set; } = null!;

public string? MiddleName { get; set; }

public string LastName { get; set; } = null!;

public DateTime? HireDate { get; set; }

Provide a complete sample code using Clean Architecture pattern for adding new employee record to the backend SQL Server database. Note that I am using Blazor Server App in the front end, then Application and Infrastructure layer as well as EF Core. Below is the Employee entity class with some basic properties:

public class Employee

{

public int EmployeeId { get; set; }

public string FirstName { get; set; } = null!;

public string? MiddleName { get; set; }

public string LastName { get; set; } = null!;

public string Position { get; set; } = null!;

public DateTime? DOB { get; set; }

public string Religion { get; set; } = null!;

public string MaritalStatus { get; set; } = null!;

public DateTime HireDate { get; set; }

}

I have a Delete button which is used to delete the employee record from the database. I’s using

a MudDialog component to display a confirmation box that ask the user to confirm the deletion. However, there is compile time error that says: “The type or namespace name could not be found (are you missing a using directive or an assembly reference?” Tell me how to fix this issue.

Below is the razor code:

<**MudButton** **Variant**="Variant.Filled" **StartIcon**="@Icons.Material.Filled.Delete"

**Size**="Size.Small" **Color**="Color.Error" **Disabled**="@\_isEditMode" **OnClick**=" DeleteConfirmationAsync ">

Delete

</**MudButton**>

private Task DeleteConfirmationAsync()

{

var parameters = new DialogParameters<DialogTemplateExample\_Dialog>

{

{ x => x.ContentText, "Do you really want to delete these records? This process cannot be undone." },

{ x => x.ButtonText, "Delete" },

{ x => x.Color, Color.Error }

};

var options = new DialogOptions() { CloseButton = true, MaxWidth = MaxWidth.Small};

return DialogService.ShowAsync<DialogTemplateExample\_Dialog>("Delete", parameters, options);

}

Write a sample code that will show a dialog box asking the user with the following questions: “Do you really want to delete these records? This process cannot be undone.”. Then, the user needs to click the Delete button to proceed. Otherwise, click the Cancel button to cancel the process. User MudDialog as a razor component which allows reusability throughout my HRMS application.

I have the following code in \_Imports.razor:

@using System.Net.Http

@using System.Net.Http.Json

@using Microsoft.AspNetCore.Components.Authorization

@using Microsoft.AspNetCore.Components.Forms

@using Microsoft.AspNetCore.Components.Routing

@using Microsoft.AspNetCore.Components.Web

@using static Microsoft.AspNetCore.Components.Web.RenderMode

@using Microsoft.AspNetCore.Components.Web.Virtualization

@using Microsoft.JSInterop

@using KenHRApp.Web

@using KenHRApp.Web.Client

@using KenHRApp.Web.Components

@using KenHRApp.Application.DTOs

@using KenHRApp.Application.Interfaces

@using MudBlazor

@using MudBlazor.Services

@using KenHRApp.Web.Components.Shared

I create the ConfirmDialog.razor component inside KenHRApp.Web.Components.Shared

Folder, and have the following code:

@using MudBlazor

@namespace KenHRApp.Web.Components.Shared

<**MudDialog**>

<**DialogContent**>

<**MudText** **Typo**="Typo.h6">@ContentText</**MudText**>

</**DialogContent**>

<**DialogActions**>

<**MudButton** **Color**="Color.Error" **Variant**="Variant.Filled" **OnClick**="Confirm">

@ButtonText

</**MudButton**>

<**MudButton** **Color**="Color.Default" **Variant**="Variant.Text" **OnClick**="Cancel">

Cancel

</**MudButton**>

</**DialogActions**>

</**MudDialog**>

@code {

[CascadingParameter]

MudDialogInstance MudDialog { get; set; } = default!;

[Parameter] public string ContentText { get; set; } = "Are you sure?";

[Parameter] public string ButtonText { get; set; } = "Delete";

private void Confirm() => MudDialog.Close(DialogResult.Ok(true));

private void Cancel() => MudDialog.Cancel();

}

But still the following compile error happens within ConfirmDialog.razor. It seems that 'MudDialogInstance' is not recognized inside ConfirmDialog.razor

The type or namespace name 'MudDialogInstance' could not be found (are you missing a using directive or an assembly reference?)

Below is the content of the KenHRApp.Web project but still MudDialogInstance is not resolve correctly.

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

<TargetFramework>net8.0</TargetFramework>

<Nullable>enable</Nullable>

<ImplicitUsings>enable</ImplicitUsings>

<UserSecretsId>aspnet-KenHRApp.Web-0756599a-959e-49d1-979f-7029f94565e1</UserSecretsId>

</PropertyGroup>

<ItemGroup>

<ProjectReference Include="..\..\KenHRApp.Application\KenHRApp.Application.csproj" />

<ProjectReference Include="..\KenHRApp.Web.Client\KenHRApp.Web.Client.csproj" />

<PackageReference Include="Blazorise.Icons.FontAwesome" Version="1.8.0" />

<PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.Server" Version="8.0.17" />

<PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="8.0.17" />

<PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="8.0.17" />

<PackageReference Include="Microsoft.EntityFrameworkCore.InMemory" Version="8.0.17" />

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.17" />

<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.17" />

<PackageReference Include="MudBlazor" Version="8.8.0" />

</ItemGroup>

</Project>

I have added the following code, “@using MudBlazor.Dialog”, on top of ConfirmDialog.razor, but now, both MudBlazor.Dialog and MudDialogInstance are not recognized with the following error:

CS0234 The type or namespace name 'Dialog' does not exist in the namespace 'MudBlazor' (are you missing an assembly reference?)

I have a MudDataGrid that is used to display a list of departments. I set the following grid attibutes:

***Filterable****="true"* ***QuickFilter****="@\_quickFilter"*

Then, the *quickFilter method declaration is as follow:*

private Func<Element, bool> \_quickFilter => x =>

{

if (string.IsNullOrWhiteSpace(\_searchString))

return true;

if (x.Sign.Contains(\_searchString, StringComparison.OrdinalIgnoreCase))

return true;

if (x.Name.Contains(\_searchString, StringComparison.OrdinalIgnoreCase))

return true;

if ($"{x.Number} {x.Position} {x.Molar}".Contains(\_searchString))

return true;

return false;

};

When I build the solution, the following error occurred:

Error CS0718 'CategoryTypes.Element': static types cannot be used as type arguments. Tell me how to fix this issue. Note that I am using .NET 8.0 and MudBlazor version 8.8.0

I am using a **MudDataGrid** to display the list of departments. Below is the Blazor markup for the datagrid:

<**MudDataGrid** **T**="DepartmentDTO" **Items**="@\_departmentList" **Hover**="true" **Striped**="true" **Class**="department-master-grid"

**Filterable**="true" **QuickFilter**="@\_quickFilter" **Hideable**="true" PageSize="5"

**EditTrigger**="DataGridEditTrigger.Manual" **ReadOnly**="false" **EditMode**="DataGridEditMode.Form"

**StartedEditingItem**="@StartedEditingItem" **CanceledEditingItem**="@CanceledEditingItem" **CommittedItemChanges**="@CommittedItemChanges"

TableLayout="TableLayout.Fixed" **Bordered**="false" **Dense**="true" **ColumnResizeMode**="ResizeMode.Column" **HorizontalScrollbar**="true">

<**ToolBarContent**>

<**MudText** **Typo**="Typo.h6"></**MudText**>

<**MudSpacer** />

<**MudPaper** **Class**="d-flex justify-content-end m-0 p-0" **Elevation**="0">

<**MudTextField** @bind-Value="\_searchString" **Placeholder**="Enter search string" **Adornment**="Adornment.Start"

**Immediate**="true" **Style**="width:400px;"

**AdornmentIcon**="@Icons.Material.Filled.Search" **IconSize**="Size.Medium" **Class**="mt-0">

</**MudTextField**>

</**MudPaper**>

</**ToolBarContent**>

<**Columns**>

<**PropertyColumn** **Property**="x => x.DepartmentId" **Hidden** **Editable**="false" />

<**PropertyColumn** **Property**="x => x.DepartmentCode" **Title**="Department Code" **Class**="col-dept-code" />

<**PropertyColumn** **Property**="x => x.DepartmentName" **Title**="Department Name" **Class**="col-dept-name" />

<**PropertyColumn** **Property**="x => x.GroupCode" **Hidden** />

<**PropertyColumn** **Property**="x => x.GroupName">

<**EditTemplate**>

<**MudSelect** @bind-Value="context.Item.GroupCode" **Variant**="Variant.Text" **Margin**="Margin.Dense" **Label**="Group Name"

**Placeholder**="Please Select" **AdornmentIcon**="@Icons.Material.Filled.ArrowDropDown" **AdornmentColor**="Color.Default"

**HelperText**="Filter by employment status" **HelperTextOnFocus**="true" **Clearable**="true" **Immediate**="true" **Typo**="Typo.subtitle2">

@foreach (var group in \_groupList)

{

<**MudSelectItem** **Value**="group.UDCCode">@group.UDCDesc1</**MudSelectItem**>

}

</**MudSelect**>

</**EditTemplate**>

</**PropertyColumn**>

<**PropertyColumn** **Property**="x => x.Description" **Class**="col-description" **Required**="false" />

<**PropertyColumn** **Property**="x => x.ParentDepartmentId" **Hidden** **Editable**="false" **Required**="false" />

<**PropertyColumn** **Property**="x => x.ParentDepartmentName" **Title**="Parent Department" **Class**="col-parent-dept" **Hidden** **Editable**="false" **Required**="false" />

<**PropertyColumn** **Property**="x => x.SuperintendentEmpNo" **Hidden** **Required**="false" />

<**PropertyColumn** **Property**="x => x.SuperintendentName" **Title**="Department Head" **Class**="col-superintendent" **Required**="false" />

<**PropertyColumn** **Property**="x => x.ManagerEmpNo" **Hidden** **Required**="false" />

<**PropertyColumn** **Property**="x => x.ManagerName" **Title**="Department Manager" **Class**="col-manager" **Required**="false" />

<**PropertyColumn** **Property**="x => x.CreatedAt" **Title**="Created Date" **Class**="col-created-date" **Editable**="false">

<**CellTemplate**>

@context.Item.CreatedAt.ToString("dd-MMM-yyyy")

</**CellTemplate**>

</**PropertyColumn**>

<**PropertyColumn** **Property**="x => x.UpdatedAt" **Title**="Updated Date" **Class**="col-updated-date" **Editable**="false">

<**CellTemplate**>

@context.Item.UpdatedAt?.ToString("dd-MMM-yyyy")

</**CellTemplate**>

</**PropertyColumn**>

<**PropertyColumn** **Property**="x => x.IsActiveDesc" **Title**="Is Active?" **Class**="col-active" **Required**="false">

<**CellTemplate** **Context**="row">

<**MudChip** **T**="string" **Size**=Size.Small **Color**="@(row.Item.IsActiveDesc == "Yes" ? Color.Success : Color.Error)">

@row.Item.IsActiveDesc

</**MudChip**>

</**CellTemplate**>

<**EditTemplate**>

<**MudSwitch** @bind-Value="context.Item.IsActive" **Color**="Color.Success" **UncheckedColor**="Color.Error" **Label**="Is Actives?" />

</**EditTemplate**>

</**PropertyColumn**>

<**TemplateColumn** **Hidden**="false" **CellClass**="d-flex justify-end">

<**CellTemplate** **Context**="row">

<**MudIconButton** **Color**="Color.Info" title="View and edit employee information" **Class**="show-btn fa-icon-btn"

**OnClick**="@row.Actions.StartEditingItemAsync">

<**ChildContent**>

<i class="fas fa-edit fa-1x"></i>

</**ChildContent**>

</**MudIconButton**>

<**MudIconButton** **Color**="Color.Error" title="Delete department record" **Class**="hide-btn fa-icon-btn" **OnClick**="@row.Actions.StartEditingItemAsync">

<**ChildContent**>

<i class="fas fa-trash-alt fa-lg"></i>

</**ChildContent**>

</**MudIconButton**>

</**CellTemplate**>

</**TemplateColumn**>

</**Columns**>

<**NoRecordsContent**>

<div class="no-data-box">

<**MudIcon** **Icon**="@Icons.Material.Filled.Info" **Class**="no-data-icon" />

<span>No data found</span>

</div>

</**NoRecordsContent**>

<**PagerContent**>

<**MudDataGridPager** **T**="DepartmentDTO" **PageSizeOptions**="new int[] {5,10,20,50,100,500}" />

</**PagerContent**>

</**MudDataGrid**>

Now, when I clicked the edit link, it shows a form but it displays the hidden properties. Tell me how to hide the hidden properties in the edit form.

I have a DTO class called RecruitmentBudgetDTO with the following properties:

public int BudgetId { get; set; }

public string DepartmentCode { get; set; } = null!;

[Required(ErrorMessage = "Department Name is required")]

[Display(Name = "Department Name")]

[StringLength(120, ErrorMessage = "Department Name can't be more than 120 characters.")]

public string DepartmentName { get; set; } = null!;

[Required(ErrorMessage = "Budget Description is required")]

[Display(Name = "Budget Description")]

[StringLength(200, ErrorMessage = "Budget Description can't be more than 200 characters.")]

public string BudgetDescription { get; set; } = null!;

[Required(ErrorMessage = "Head Count Budget is required")]

[Display(Name = "Head Count Budget")]

public int BudgetHeadCount { get; set; }

[Display(Name = "Active Employees")]

public int ActiveCount { get; set; }

I need to create a custom validation attribute that will throw a validation error message if the entered value in the ActiveCount property is greater than the value defined in the BudgetHeadCount property

I have added the following properties to the DTO object given above:

[Display(Name = "Exit Employees")]

public int ExitCount { get; set; }

[Display(Name = "Active Requisition")]

public int RequisitionCount { get; set; }

[Display(Name = "Net Gap")]

public int NetGapCount { get; set; }

Now, I need to calculate the value of NetGapCount field based on the following formula:

NetGapCount = BudgetHeadCount - ((ActiveCount + RequisitionCount) - ExitCount)

The calculation should be triggered when the value is changed in any of the following fields:

* BudgetHeadCount
* ActiveCount
* RequisitionCount