Given the project structure shown in your image, you can implement the required functionality by using the existing setup and making the necessary modifications. Since you already have models like `TaxRisk` and `TaxRiskYear`, I'll guide you on how to implement the functionality you requested (editing, deleting, and adding rows) in this structure using MudBlazor.

### Step 1: Define the Model

Ensure your models are defined properly. For example, your `TaxRisk` model might look like this:

```csharp

namespace TarrWebApp.Models

{

public class TaxRisk

{

public string Code { get; set; }

public string Name { get; set; }

public string ShortDescription { get; set; }

public string Description { get; set; }

public string Status { get; set; }

}

}

```

### Step 2: Add the `using` Directive

In your Razor components (e.g., `TaxRiskItemComponent.razor`), add the necessary `@using` directive:

```razor

@using TarrWebApp.Models

@using TarrWebApp.Services

```

### Step 3: Create the Service

If you don’t have a service yet for managing `TaxRisk` objects, create one in the `Services` folder:

```csharp

namespace TarrWebApp.Services

{

public class TaxRiskService

{

private List<TaxRisk> \_taxRisks = new List<TaxRisk>();

public List<TaxRisk> GetAllTaxRisks() => \_taxRisks;

public void AddTaxRisk(TaxRisk taxRisk)

{

\_taxRisks.Add(taxRisk);

}

public void UpdateTaxRisk(TaxRisk taxRisk)

{

var existing = \_taxRisks.FirstOrDefault(x => x.Code == taxRisk.Code);

if (existing != null)

{

existing.Name = taxRisk.Name;

existing.ShortDescription = taxRisk.ShortDescription;

existing.Description = taxRisk.Description;

existing.Status = taxRisk.Status;

}

}

public void DeleteTaxRisk(string code)

{

var taxRisk = \_taxRisks.FirstOrDefault(x => x.Code == code);

if (taxRisk != null)

{

\_taxRisks.Remove(taxRisk);

}

}

}

}

```

### Step 4: Register the Service

In `Program.cs`, register the service:

```csharp

builder.Services.AddSingleton<TaxRiskService>();

```

### Step 5: Implement the Razor Component

In `TaxRiskItemComponent.razor`, implement the logic using MudBlazor:

```razor

@page "/taxrisks"

@inject TaxRiskService TaxRiskService

<MudTable Items="taxRisks" Striped="true">

<HeaderContent>

<MudTh>Code</MudTh>

<MudTh>Name</MudTh>

<MudTh>Short Description</MudTh>

<MudTh>Description</MudTh>

<MudTh>Status</MudTh>

<MudTh>Actions</MudTh>

</HeaderContent>

<RowTemplate>

<MudTd DataLabel="Code">@context.Code</MudTd>

<MudTd DataLabel="Name">@context.Name</MudTd>

<MudTd DataLabel="Short Description">@context.ShortDescription</MudTd>

<MudTd DataLabel="Description">@context.Description</MudTd>

<MudTd DataLabel="Status">@context.Status</MudTd>

<MudTd DataLabel="Actions">

<MudButton Variant="Variant.Filled" Color="Color.Primary" OnClick="@(() => EditTaxRisk(context))">Edit</MudButton>

<MudButton Variant="Variant.Filled" Color="Color.Error" OnClick="@(() => DeleteTaxRisk(context.Code))">Delete</MudButton>

</MudTd>

</RowTemplate>

</MudTable>

<MudButton Variant="Variant.Filled" Color="Color.Success" OnClick="AddNewTaxRisk">New</MudButton>

<MudDialog @bind-Open="showDialog">

<DialogContent>

<MudTextField @bind-Value="selectedTaxRisk.Code" Label="Code" />

<MudTextField @bind-Value="selectedTaxRisk.Name" Label="Name" />

<MudTextField @bind-Value="selectedTaxRisk.ShortDescription" Label="Short Description" />

<MudTextField @bind-Value="selectedTaxRisk.Description" Label="Description" />

<MudTextField @bind-Value="selectedTaxRisk.Status" Label="Status" />

</DialogContent>

<DialogActions>

<MudButton Variant="Variant.Filled" Color="Color.Primary" OnClick="SaveTaxRisk">Save</MudButton>

<MudButton Variant="Variant.Filled" Color="Color.Secondary" OnClick="CancelEdit">Cancel</MudButton>

</DialogActions>

</MudDialog>

@code {

private List<TaxRisk> taxRisks;

private TaxRisk selectedTaxRisk = new TaxRisk();

private bool showDialog = false;

protected override void OnInitialized()

{

taxRisks = TaxRiskService.GetAllTaxRisks();

}

private void AddNewTaxRisk()

{

selectedTaxRisk = new TaxRisk();

showDialog = true;

}

private void EditTaxRisk(TaxRisk taxRisk)

{

selectedTaxRisk = taxRisk;

showDialog = true;

}

private void SaveTaxRisk()

{

if (taxRisks.Contains(selectedTaxRisk))

{

TaxRiskService.UpdateTaxRisk(selectedTaxRisk);

}

else

{

TaxRiskService.AddTaxRisk(selectedTaxRisk);

}

taxRisks = TaxRiskService.GetAllTaxRisks();

showDialog = false;

}

private void DeleteTaxRisk(string code)

{

TaxRiskService.DeleteTaxRisk(code);

taxRisks = TaxRiskService.GetAllTaxRisks();

}

private void CancelEdit()

{

showDialog = false;

}

}

```

### Step 6: Integrate in Navigation

If this component is part of your main app, make sure it's accessible by updating `NavMenu.razor` or the router.

For example, in `NavMenu.razor`:

```razor

<MudNavLink Href="taxrisks" Match="NavLinkMatch.All">

<MudIcon Icon="@Icons.Material.Filled.List"/> Tax Risks

</MudNavLink>

```

### Step 7: Test the Application

Run your application, and navigate to the page to manage `TaxRisk` records. You should be able to add, edit, and delete records using the MudBlazor table and dialog components.

This setup should work well with the structure you've shown in your project. If you encounter any issues, ensure all namespaces and service registrations are correct.