Tutorial 07-
CRUD operations

Objectives

- CRUD UI for ToDo entity
- Forms and validation
- Authorize attribute
- Add new menu item
- Custom CSS

In this tutorial we will create CRUD operations for a simple todo application.

Before we start, we need to create a Blazor WebAssembly project with ASP.NET and Identity enabled.

Add new menu item

Let us create a new folder in Pages and call it Todos. Next, we need to create a new menu item that we will use to access todos.

NavMenu.razor | ItemComponent.razor | MyComponent.razor | Counter.razor

```
<div class="top-row pl-4 navbar navbar-dark">
    <a class="navbar-brand" href="">HostedWithIdentity</a>
    <button class="navbar-toggler" @onclick="ToggleNavMenu">
        <span class="navbar-toggler-icon"></span>
    </button>
</div>

<div class="@NavMenuCssClass" @onclick="ToggleNavMenu">
    <ul class="nav flex-column">
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </li>
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="counter">
                <span class="oi oi-plus" aria-hidden="true"></span> Counter
            </NavLink>
        </li>
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="todos">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Todos
            </NavLink>
        </li>
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="fetchdata">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
            </NavLink>
        </li>
    </ul>
</div>
```

File location: Solution > Pages > Shared > NavMenu.razor

CRUD UI for ToDo entity

Todo model

Create Todo model class in Shared project

```
namespace HostedWithIdentity.Shared
{
    0 references
    public class Todo
    {
        [Key]
        0 references
        public Guid Id { get; set; }

        [Required]
        0 references
        public string Name { get; set; }
        0 references
        public string Description { get; set; }
    }
}
```

Authorization

When we want to ensure the user is authenticate we need to use Authorize attribute. For this attribute to work we need to add some namespaces. Instead of repeating the namespaces in each file we can add them to _Imports.razor file so all components can use them.

Add the following namespaces to the global _Imports.razor for the [Authorize] attribute

- @using Microsoft.AspNetCore.Authorization
- @using Microsoft.AspNetCore.Components.WebAssembly.Authentication

Storage

We need to be able to store our todos somewhere, at least temporarily. Probably, the easiest way to do this is to create a data structure and store todos in memory. For constant time look up, we can use a hash-based data structure like dictionary. To have some data to work with, we can create a few items and add them to our data structure.

Now, we can start manipulating with this data structure in our CRUD pages.

```
Delete.razor   Read.razor   App.razor   Index.razor   Create.razor   Edit.razor   _Impor
@code
{
    public static Dictionary<Guid, Todo> Todos = new Dictionary<Guid, Todo>();
    protected override void OnInitialized()
    {
        var todo1 = new Todo()
        {
            Id = Guid.NewGuid(),
            Name = "Grosery shopping",
            Description = "Buy some vegetables and milk"
        };

        var todo2 = new Todo()
        {
            Id = Guid.NewGuid(),
            Name = "Pay bills",
            Description = "Pay for this month's rent and electricity"
        };

        var todo3 = new Todo()
        {
            Id = Guid.NewGuid(),
            Name = "Clean up the room",
            Description = "Vacum the floor and wipe the table"
        };

        Todos.Add(todo1.Id, todo1);
        Todos.Add(todo2.Id, todo2);
        Todos.Add(todo3.Id, todo3);
    }
}
```

List Todos page

Go to Todos folder and create a new component called Index.razor. This component will contain the code for displaying (listing) all todos.

```
or        Delete.razor        Read.razor        App.razor        Index.razor
@page "/todos"

@attribute [Authorize]

<h1>List todos</h1>

<div class="table-responsive">
    <table class="table table-hover">
        <thead class="thead-dark">
            <tr>
                <th>Name</th>
                <th>Read</th>
                <th>Edit</th>
                <th>Delete</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var todo in App.Todos)
            {
            <tr>
                <td>
                    @todo.Value.Name
                </td>
                <td>
                    <a href="/todos/details/@todo.Key">Details</a>
                </td>
                <td>
                    <a href="/todos/edit/@todo.Key">Edit</a>
                </td>
                <td>
                    <a href="/todos/delete/@todo.Key">Delete</a>
                </td>
            </tr>
            }
        </tbody>
    </table>
</div>

<p>
    <a class="btn btn-primary" href="/todos/create">Create</a>
</p>

@code {}
```

Create Todos page

Next we need a component for creating todos. Create a new component in Todos folder and call it Create.razor.

```
Create.razor   Todo.cs   NavMenu.razor   ItemComponent.razor   MyComponent.razor   Counte
1   @page "/todos/create"
2
3   @attribute [Authorize]
4   @inject HttpClient Http
5   @inject NavigationManager Navigation
6
7   <h3>Todos Create</h3>
8
9   <EditForm Model="Todo" OnValidSubmit="@HandleValidSubmit">
10      <DataAnnotationsValidator />
11
12      <div class="form-group">
13          <label class="control-label">Name </label>
14          <InputText @bind-Value="Todo.Name" class="form-control" />
15          <ValidationMessage For="@(() => Todo.Name)" />
16      </div>
17
18      <div class="form-group">
19          <label class="control-label">Description </label>
20          <InputText @bind-Value="Todo.Description" class="form-control" />
21          <ValidationMessage For="@(() => Todo.Description)" />
22      </div>
23
24      <button type="submit" class="btn btn-primary">
25          <i class="fas fa-save"></i> Create
26      </button>
27  </EditForm>
28
29  @code {
30      private Todo Todo { get; set; } = new Todo();
31
32      private async void HandleValidSubmit()
33      {
34
35      }
36  }
37
```

Edit Todos page

Next let's create a component for editing todos.

```
Delete.razor   Read.razor   App.razor   Index.razor   Create.razor   Edit.razor   _Imports.razor
@page "/todos/edit/{Id:guid}"

@attribute [Authorize]
@inject NavigationManager Navigation

<h3>Edit Todo</h3>


<EditForm Model="Todo" OnValidSubmit="@HandleValidSubmit">
    <DataAnnotationsValidator />

    <div class="form-group">
        <label class="control-label">Name </label>
        <InputText @bind-Value="Todo.Name" class="form-control" />
        <ValidationMessage For="@(() => Todo.Name)" />
    </div>

    <div class="form-group">
        <label class="control-label">Description </label>
        <InputText @bind-Value="Todo.Description" class="form-control" />
        <ValidationMessage For="@(() => Todo.Description)" />
    </div>

    <button type="submit" class="btn btn-primary">
        <i class="fas fa-save"></i> Update
    </button>
</EditForm>
```

```
@code {
    [Parameter] public Guid Id { get; set; }
    private Todo Todo { get; set; }

    protected override void OnInitialized()
    {
        Console.WriteLine("Parameter: " + Id.ToString());
        if (App.Todos.ContainsKey(Id))
            Todo = App.Todos[Id];
        else
            Console.WriteLine("Id does not exist.");
    }

    private void HandleValidSubmit()
    {
        Console.WriteLine("Form submitted!");
        Console.WriteLine("Todo name: " + Todo.Name);
        Console.WriteLine("Todo description: " + Todo.Description);

        App.Todos[Id] = Todo;
        Navigation.NavigateTo("/todos");
    }
}
```

Read Todos page

Now let's create a component for showing up details of individual todos.

```
Delete.razor   Read.razor   App.razor   Index.razor   Create.razor   Edit.razor

@page "/todos/details/{Id:guid}"

@attribute [Authorize]

<h1>Todo details</h1>

<h4>Title</h4>
<p>@Todo.Name</p>

<h4>Description</h4>
<p>@Todo.Description</p>

@code {
    [Parameter] public Guid Id { get; set; }
    private Todo Todo { get; set; }

    protected override void OnInitialized()
    {
        Console.WriteLine("Parameter: " + Id.ToString());
        if (App.Todos.ContainsKey(Id))
            Todo = App.Todos[Id];
        else
            Console.WriteLine("Id does not exist.");
    }
}
```

Delete Todos page

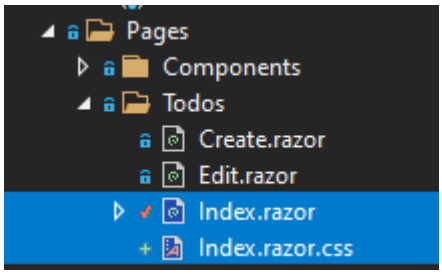
Finally let's create a page for deleting todos.

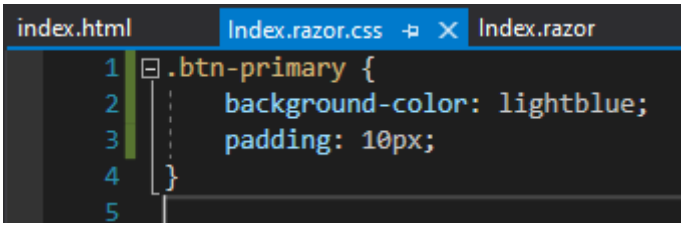
```
Delete.razor   Read.razor   App.razor   Index.razor   Create.razor   Edit.razor   _Imports.razor
@page "/todos/delete/{Id:guid}"

@attribute [Authorize]
@inject NavigationManager Navigation

<h1>Please confirm you want to delete the following todo?</h1>

<h4>Title</h4>
<p>@Todo.Name</p>

<h4>Description</h4>
<p>@Todo.Description</p>

<button class="btn btn-danger" @onclick="DeleteTodo">Delete</button>
```

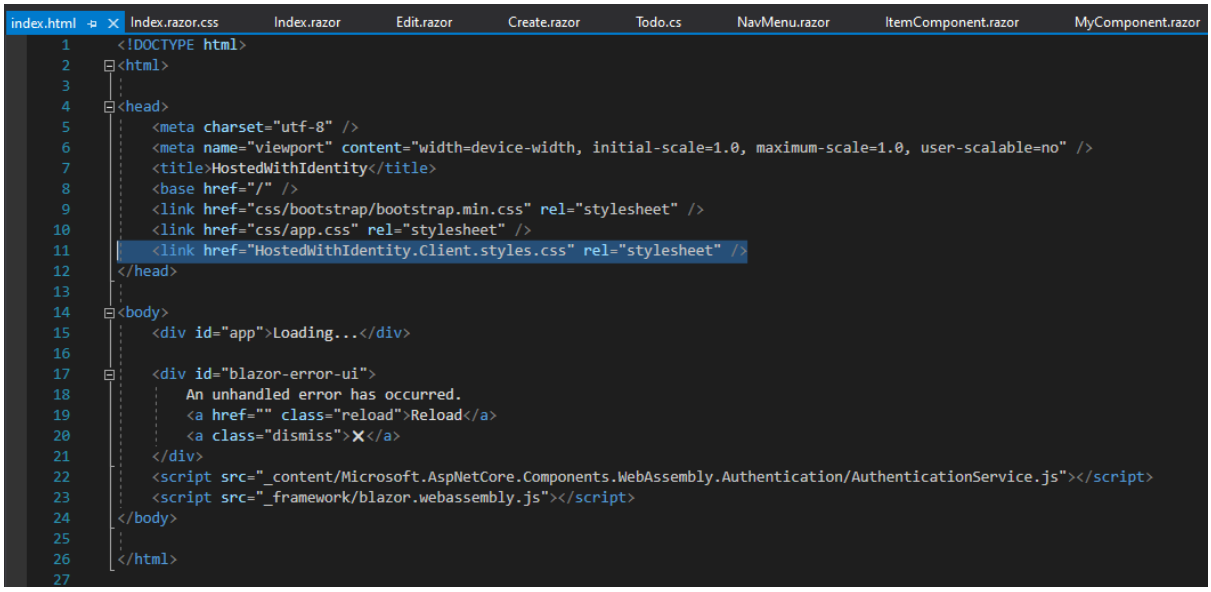
```
@code {
    [Parameter] public Guid Id { get; set; }
    private Todo Todo { get; set; }

    protected override void OnInitialized()
    {
        Console.WriteLine("Parameter: " + Id.ToString());
        if (App.Todos.ContainsKey(Id))
            Todo = App.Todos[Id];
        else
            Console.WriteLine("Id does not exist.");
    }

    private void DeleteTodo()
    {
        App.Todos.Remove(Id);
        Navigation.NavigateTo("/todos");
    }
}
```

CSS isolation

We can create CSS files that contains only the CSS that will be applied to a specific component.







If style is not showing up check if project was built and stylesheet is included as shown above (wwwroot > index.html).

Useful links

- https://docs.microsoft.com/en-us/aspnet/core/blazor/components/css-isolation?view=aspnetcore-5.0