

Exercise 15 : Razor Pages

Objective

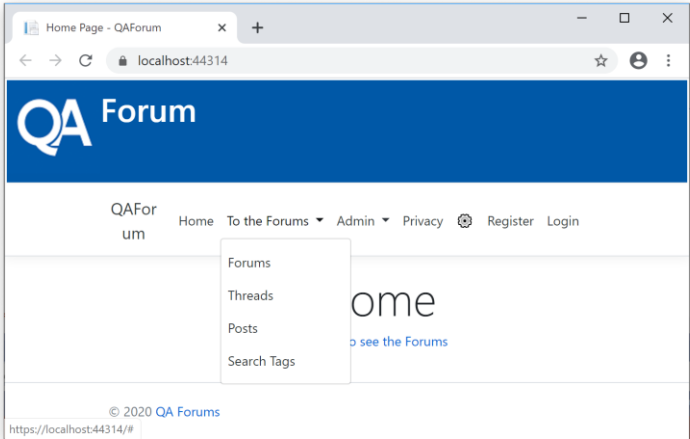
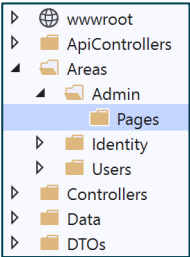
In this exercise you will add new functionality to your application using Razor Pages.

This exercise will take around 20 minutes.

Referenced material

This exercise is based on material from the chapter "Razor Pages".

Creating a Razor Page with a Post Handler

•	Open the 'Begin' solution in Visual Studio and compile (Shift Ctrl+B).
•	<p>The 'Begin' solution is almost identical to the 'End' solution from the previous lab, but we've tidied up the navigation bar a bit, since it was getting very full. Take a look in <code>_Layout.cshtml</code> and see how we've used the Bootstrap dropdown-toggle class to create a dropdown, and the dropdown-menu class to contain the items that are in the dropdown.</p> <p>Run the program and see how the menu looks now:</p> 
•	<p>In this lab, we are going to create some Administration pages for our forum. Let's create an Area to contain those pages.</p> <p>Right-click on the Areas folder, and add a folder called Admin. Then, right-click on this Admin folder, and add a sub-folder called Pages.</p> 

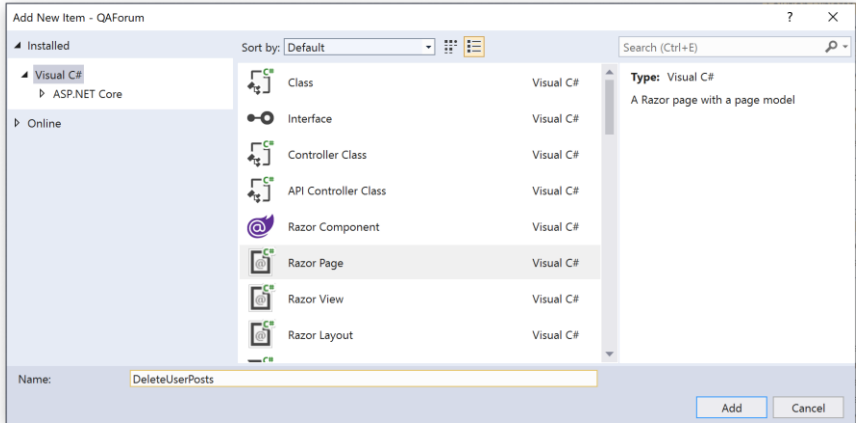
- Normally, in a Pages folder, we would create a Shared folder, and within that we would put a `_Layout.cshtml` file.

For our project, we want our Razor pages to share their layout with our MVC views, so we will use the `_Layout.cshtml` file in the `/Views/Shared` folder, which is the location normally used for MVC views.

Copy (don't move!) the following two files from `/Areas/Users/Views` into `/Areas/Admin/Posts`:

 - `_ViewStart.cshtml` (to set the location of the layout file)
 - `_ViewImports.cshtml` (to allow us to use tag helpers)
- Right-click on the Pages folder, and select Add, New Item.

Add an item of type Razor Page. Call the new page "DeleteUserPosts"


- In this new page, we are going to allow a facility for administrators to delete all the posts by a specific user.

Open the file `DeleteUserPosts.cshtml.cs` (the Page Model, which will be nested under the Page within the Solution Explorer).

In this file, we put properties that the page can bind to, which represent the user data. We will also arrange for a database context to be injected:

```

public class DeleteUserPostsModel : PageModel
{
    private readonly ForumDbContext context;

    [BindProperty]
    [Required]
    [Display(Name = "User Name")]
    public string UserName { get; set; }

    public string ErrorText { get; set; }
    public string Message { get; set; }

    public DeleteUserPostsModel(ForumDbContext context)
    {
        this.context = context;
    }
}

```

```

        public void OnGet()
        {
        }
    }

```

- Next, we will write the page. In the file DeleteUserPosts.cshtml, write the following Razor code:

```

@page
@model QAForum.Areas.Admin.Pages.DeleteUserPostsModel

@{
    ViewData["Title"] = "Delete by User";
}

<h1>Delete User's Posts</h1>

<p class="text-danger">@Model.ErrorMessage</p>
<p class="text-success">@Model.Message</p>

<h4>Enter Details of User</h4>
<div class="row">
    <div class="col-md-4">
        <form method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="UserName" class="control-label"></label>
                <input asp-for="UserName" class="form-control" />
                <span asp-validation-for="UserName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Delete Posts"
                    class="btn btn-danger" />
            </div>
        </form>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}

```

- When the user submits this form, the data will be posted back to the page model. So, add the following method to the page model:

```

public void OnPost()
{
    try
    {
        if (ModelState.IsValid)
        {
            var postsToDelete =
                context.Posts.Where(p => p.UserName == UserName);
            var count = postsToDelete.Count();

            context.RemoveRange(postsToDelete);
            context.SaveChanges();

            Message = $"{count} posts deleted";
        }
    }
}

```

	<pre> } catch { ErrorText = "Sorry, something went wrong"; } } </pre> <p>Note that the method returns void – this is equivalent of it returning an IActionResult by using the line <code>return Page()</code> – it simply results in the page being displayed in all circumstances.</p>
<ul style="list-style-type: none"> • 	<p>Next, we are going to add a link to the new page onto the redesigned navigation bar.</p> <p>Open <code>/Views/Shared/_Layout.cshtml</code>, remove the placeholder in the admin section, and add the following line in its place:</p> <pre> <li class="dropdown"> Admin <div class="dropdown-menu" id="admin-dropdown"> Coming Soon Delete User Posts </div> </pre>
<ul style="list-style-type: none"> • 	<p>Finally, test your work! Run the application, navigate to the new page, and delete all the posts by the user <code>moderator@testing.com</code>. You should find that 2 posts are deleted, assuming you haven't added or deleted posts by that user since seeding the database.</p> <p>Also, see what happens if you click the Delete Posts button without entering a name. The [Required] attribute on the page model ensures that the page is not submitted without data, in the same way that we have already seen with MVC views.</p>

Creating a Razor Page with Get Parameters

In the previous section, we saw how to create a page with an `OnGet()` method that displayed a form, and an `OnPost()` method that handled the form submission. This is the most common pattern to use for Razor pages.

In the next section, we will look at an alternative pattern, where the `OnGet()` method is the only handler. We will make use of query string parameters to retrieve data from a form. This will allow the user to bookmark or share a page, including the page's parameters.

•	Add another Razor Page to the Admin/Pages folder. When prompted, name the new page "PostsByDate".
•	Add the following properties and constructor-injected context field to the new

page model:

```
public class PostsByDateModel : PageModel
{
    private readonly ForumDbContext context;

    public int PostCount { get; set; }
    public string Description { get; set; }

    [BindProperty(SupportsGet = true)]
    [Display(Name = "Start Date")]
    public DateTime? StartDate { get; set; }

    [BindProperty(SupportsGet = true)]
    [Display(Name = "End Date")]
    public DateTime? EndDate { get; set; }

    public PostsByDateModel(ForumDbContext context)
    {
        this.context = context;
    }

    public void OnGet()
    {
    }
}
```

Notice how the two date properties are not only marked with [BindProperty], but additionally, the SupportsGet=true parameter has been set. It is less common for get requests to include form data, so this needs to be enabled explicitly. Additionally, they are both set to be nullable.

- Now, add the following code, to find (and describe) the number of posts:

```
public void OnGet()
{
    Description = GetDescription();
    PostCount = GetPostCount();
}

private string GetDescription()
{
    switch (StartDate, EndDate)
    {
        case (null, null):
            return "Number of posts from all time";
        case (_, null):
            return "Number of posts from " +
                StartDate.Value.ToShortDateString();
        case (null, _):
            return "Number of posts up until " +
                EndDate.Value.ToShortDateString();
        default:
            return $"Number of posts between" +
                $" {StartDate.Value.ToShortDateString()} " +
                $" and {EndDate.Value.ToShortDateString()}";
    }
}
```

```

private int GetPostCount()
{
    IQueryable<Post> posts = context.Posts;
    if (StartDate != null)
    {
        posts = posts.Where(p => p.PostDateTime >= StartDate);
    }
    if (EndDate != null)
    {
        // Add one day to the date to take into account that the
        // post date time may include a "time" element. We need
        // to include times up to (but not including) midnight on
        // the following day
        posts = posts.Where
            (p => p.PostDateTime < EndDate.Value.AddDays(1));
    }
    return posts.Count();
}

```

• The code for the page is shown here. Copy it into the page file:

```

@page
@model QAForum.Areas.Admin.Pages.PostsByDateModel

@{
    ViewData["Title"] = "Count Posts by Date";
}

<h1>Count Posts</h1>

<h4>Count Posts by Date</h4>

<div class="col-md-10">
    <div class="row">
        @Model.Description - @Model.PostCount
    </div>

    <hr />

    <form method="get">
        <div class="row">
            <div class="form-group d-flex flex-column justify-content-end">
                <label asp-for="StartDate" class="control-label"></label>
                <input asp-for="StartDate" type="date"
                    asp-format="{0:yyyy-MM-dd}" class="form-control" />
            </div>
            <div class="form-group d-flex flex-column justify-content-end">
                <label asp-for="EndDate" class="control-label"></label>
                <input asp-for="EndDate" type="date"
                    asp-format="{0:yyyy-MM-dd}" class="form-control" />
            </div>

            <div class="form-group d-flex flex-column justify-content-end">
                <input type="submit" value="Count Posts"
                    class="btn btn-primary" />
            </div>
        </div>
        <div class="row">
            <div asp-validation-summary="All" class="text-danger"></div>
        </div>
    </form>
</div>

```

	<pre>@section Scripts { @{await Html.RenderPartialAsync("_ValidationScriptsPartial");} }</pre> <p>Notice the <code><form method="get"></code> tag – setting the method to Get here means that the two dates will be passed in the query string. As such, it will be possible for the user to bookmark or share a page, including its date parameters – something which would make sense for this page, but not for the previous example that we completed earlier in the lab.</p> <p>The use of <code>asp-format="{0:yyyy-MM-dd}"</code> forces the date to be represented in the ISO format, which is a requirement for <code>type="date"</code> to work correctly and show a date editor (the default, if we didn't do this, is a date/time editor).</p>
•	<p>Add a link to the navigation bar in <code>_Layout.cshtml</code> so that users can easily access this page:</p> <pre><li class="dropdown"> Admin <div class="dropdown-menu" id="admin-dropdown"> Delete User Posts Count Posts </div> </pre>
•	<p>Test the new page. You may need to look at the list of posts first, to see the range of dates, to be able to find some sensible date ranges to test.</p>
•	<p>One more thing to do. We should add some validation which prevents the start date from being after the end date.</p> <p>The built-in validation attributes can't do this for us, because (with the exception of the [Compare] attribute), they only look at individual properties, and we need to compare two properties.</p> <p>We would normally do that by implementing the <code>IValidatableObject</code> interface in the page model. But a bug in the Razor page validation logic¹ means that <code>IValidatableObject</code> can be implemented in classes which the page model binds to, but not in the page model itself.</p> <p>So, instead, we will use a nested class, and implement the interface there.</p>
•	<p>Add the following nested class within the page model class:</p> <pre>public class PostsByDateModel : PageModel</pre>

¹ See this Github issue – <https://github.com/dotnet/aspnetcore/issues/4895> – for more details about this bug. At the time of writing, a post by a Microsoft engineer states that it won't be fixed until at least the .Net 5 beta.

	<pre> { public class StartEndDate : IValidatableObject { [Display(Name = "Start Date")] public DateTime? StartDate { get; set; } [Display(Name = "End Date")] public DateTime? EndDate { get; set; } public IEnumerable<ValidationResult> Validate (ValidationContext validationContext) { if (StartDate > EndDate) { yield return new ValidationResult ("Start date must not be after end date"); } } } } </pre> <p>Using “yield return” in the Validate() method is a handy way of returning an IEnumerable<> that only contains a single item.</p>
<ul style="list-style-type: none"> Also, within the page model class, remove the StartDate and EndDate properties, and replace them with a property of the new StartEndDate nested type: 	<pre> [BindProperty(SupportsGet = true)] [Display(Name = "Start Date")] public DateTime? StartDate { get; set; } [BindProperty(SupportsGet = true)] [Display(Name = "End Date")] public DateTime? EndDate { get; set; } [BindProperty(SupportsGet = true)] public StartEndDate Dates { get; set; } </pre> <p>The [BindProperty(SupportsGet = true)] attribute now needs to be applied to the Dates property to ensure that the members of Dates – StartDate and EndDate – can be bound on a Get request.</p>
<ul style="list-style-type: none"> In both the page and the page model, there will now be many errors because they reference the two properties, StartDate and EndDate, which you have just deleted. Make the following changes in as many places as necessary, in the Razor page and also in the page model: <ul style="list-style-type: none"> StartDate – change to Dates.StartDate EndDate – change to Dates.EndDate 	
<ul style="list-style-type: none"> Test your changes. If you now try to select a start date before the end date, an error should be shown. All other functionality should still work as before. 	

Challenge – Adding More Razor Pages

Although our Razor pages provide some basic administration functions, they

fall a little short of what might be required.

How would you make the following changes?

The Posts By Date page should show a list of posts, rather than just the number of posts

The Delete User Posts page should show a list of posts that are going to be deleted, and then get further confirmation, before actually deleting the posts

Hint – you will need a new Razor page to implement this, and you will need to think about how the two pages link together

The `<form>` tag-helper enables you to write something like this:
`<form method="post" asp-page="NextPage">`

If you have time, you may like to try to implement one or both of these changes.

Congratulations, you have now created some Razor Pages!