

Exercise 15 : Controllers And Actions

Objective

In this exercise you will build controllers for three key entities: Forums, Threads and Posts. You will write actions that return views for five scenarios: List, Details, Create, Edit and Delete. Finally you will provide simple views to be returned by each of your action results.

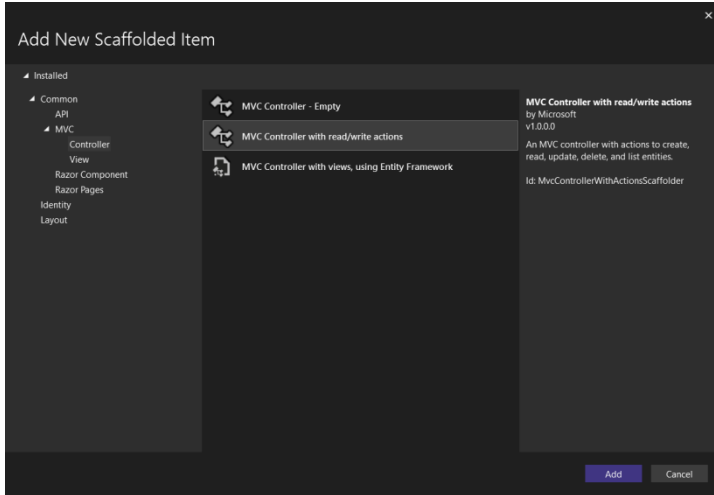
This exercise will take around 60minutes.

Referenced material

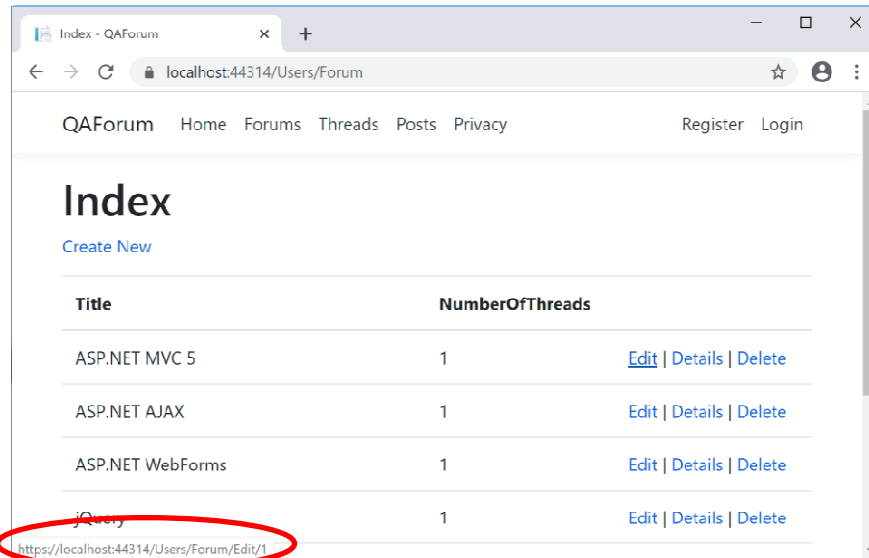
This exercise is based on material from the chapter "Controllers And Actions".

Actions – Forum Controller

1	<p>Open the 'Begin' solution in Visual Studio and compile (Shift Ctrl+B).</p> <p>Run the application and click the links for Forms, Posts and Threads. For reasons of time constraint some of this application has been pre-built but we will alter it to focus specifically on the topics from the chapter.</p> <p>Firstly you may want to investigate some of the code that is already there.</p> <p>Look in the Models folder to see our basic model classes for Forum, Thread and Post.</p> <p>Look in the ViewModels folder and see the ViewModels that are tailored for the specific needs of our application. For example the ForumViewModel has additional code to calculate the NumberOfThreads each Forum has. Also code that our app will use get the data it needs which is encapsulated by these ViewModels.</p> <p>Next look in the appsettings.json file where you can see a connection string pointing at a database which has already been created. You could try to use the information in this connectionString, in the Server Explorer window, to create a connection to this DB and have a closer look at it. Ask if you need help with this.</p> <p>In the EF folder you will see the ForumDbContext class defined (nb. It inherits the Entityframework “brain” DbContext) which has properties that will expose Forums, Threads and Posts for our app to use.</p> <p>All of the above is not specific to MVC and might be seen in any type of application that wants to use the Entity Framework to access a database.</p> <p>Next in the Program.cs file you can see the code where our ForumDbContext is being configured to be provided to our app via DI. You can also see that it is being wired to use Sql Server and specifically pointed to the connectionString in the appsettings.json file.</p> <p>In the Controllers folder, have a look at the ForumController (or ThreadController or PostController). You can see they use constructor DI to request a ForumDbContext be provided to them for DB access. Note the code in the Index</p>
---	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>action to use the context and our ViewModel to get the data and pass it to the returned Index View.</p> <p>Lastly, in the Views/Forum folder have a look at the Razor code in Index.cshtml. Note at the top the @model is typed appropriately. See the code looping over the data passed in via the model and wrapped in standard HTML . This will then be returned to the browser to be rendered.</p> <p>Ok, now let's make some changes to practise some of this for ourselves.</p>
2	Delete the existing ForumController in the Controllers folder
3	<p>Add a new ForumController to the same folder. We are going to use the "Scaffolding" process to create a template, which will save us a little bit of time:</p> <ul style="list-style-type: none"> • In Solution Explorer, right click on the Controllers folder • Choose Add, Controller • Choose MVC Controller With Read/Write Actions  <ul style="list-style-type: none"> • Click Add • Enter the name, ForumController • Click Add to add the new controller
4	<p>Take a moment to look through the resulting file: ForumController.cs</p> <p>Several read actions have been added to the class including Index and Details. There are also several write actions.</p>
5	<p>Re-add the code that was in the previous ForumController:</p> <pre> Public class ForumController : Controller { Private readonly ForumDbContext context; Public ForumController(ForumDbContext context) { this.context = context; } // GET: ForumController </pre>

	<pre> public ActionResult Index() { return View(ForumViewModel.FromForums(context.Forums)); } </pre>
6	<p>Right-click on the Index action, and choose Go To View. This will open the view that was created previously.</p> <p>Scroll to the bottom of the view. You will find three calls to @Html.ActionLink – one for Edit, one for Details, and the third one for Delete.</p> <p>At the moment, each of these ActionLink calls has its RouteValues parameter set to an empty anonymous object, but a comment indicates that we should amend this empty object to include the primary key of the item.</p> <p>Let's do that:</p> <pre> @Html.ActionLink("Edit", "Edit", new { id = item.ForumId }) @Html.ActionLink("Details", "Details", new { id = item.ForumId }) @Html.ActionLink("Delete", "Delete", new { id = item.ForumId }) </pre> <p>(In the ForumViewModel class, did you wonder why we included the ForumId property that would not be shown? It seemed a bit counter-intuitive to include it if we weren't intending to show it. But now it becomes clear – we included it because the view needs it, not to display to the user but to be able to link to the next action.)</p> <p>It's worth pointing out here that, if you choose a model type for your view that's part of a DbContext, the scaffolder will automatically do this step for you. It's only because of our choice to use view-models which are not stored in entity framework that we need to do this step manually.</p>
7	<p>Also, at the top of the file, change the text in the <h1>:</p> <pre> <h1>Welcome to the Forums</h1> <p> <a asp-action="Create">Create New </p> </pre>
8	<p>Press F5, and when your program starts, go to the Forums page.</p> <p>Hover the mouse over the Edit, Details and Delete links next to each forum. Depending on the web browser that you are using, you should see the URL that the link is going to take you to – often in the bottom left corner:</p>



As you move between links for each of the different forums, notice how the final part of the URL changes to match the id of each forum.

(There is no point clicking on these links yet – since we haven't written the code for them, they won't work.)

Stop the program.

9 Let's build a Details action next, that shows the details of each forum.

For some data entities, there will be many properties – too many to include them all in a list view, but we might want a lot more included in a details view than the list view. If this were the case, we'd need to make a new view-model that included the relevant properties.

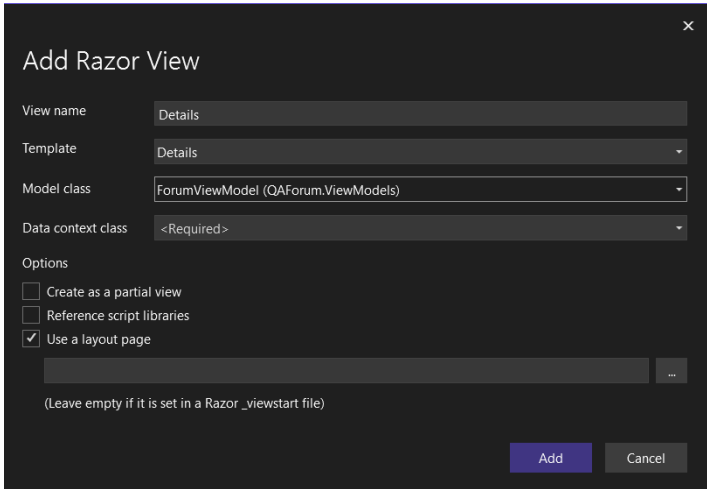
But the Forum entity is very simple, and the list view already includes all the details we need. Therefore, we can use the existing view-model. (There is a school of thought that every view should have its own view-model. This is a valid argument – sharing view-models might make it harder to maintain the program if one view were to change its required data. But for these labs we're going to share view-models where the underlying data for two views is the same.)

(You might ask what's the point of the details view if it only shows the same details as the list view, and that's a very valid question. It's really for you to practice.)

10 Amend the Details method in the Forum controller as follows:

```
public ActionResult Details(int id)
{
    var forum = context.Forums.Single(f => f.ForumId == id);
    return View(ForumViewModel.FromForum(forum));
}
```

Then, right-click on the Details action, and select Add View. Select Razor View add click Add. Change the template to Details, and change the model class to

	<p>ForumViewModel:</p>  <p>Click Add</p>
11	<p>A new file, Details.cshtml, is created.</p> <p>At the bottom of the page is a call the <code>Html.ActionLink</code>, which requires you to add the id in (the same way we did in the Index view):</p> <pre data-bbox="319 987 1466 1151"><div> @Html.ActionLink("Edit", "Edit", new { id = Model.ForumId }) <a asp-action="Index">Back to List </div></pre> <p>Also, modify the <code><h4></code> element towards the top of the page:</p> <pre data-bbox="319 1229 1466 1415"><h1>Details</h1> <div> <h4>Forum</h4> <hr/></pre>
12	<p>Run your program. Go to the Forum page, then click the Details link next to one of the forums. Check it shows the details of that forum.</p> <p>When you're satisfied that this code works, stop the program.</p>
13	<p>Next, we'll work on creating a new forum.</p> <p>Once again, we start by considering the view-model that we want to use. The existing view-model is not suitable, because it contains a property called <code>NumberOfThreads</code>. Users will not be expected to provide this value when they add a forum, nor when they edit it.</p> <p>What's more, the <code>ForumId</code> is not going to be entered by the user. It will be automatically assigned by the database. So this should not be part of the view-model for create a forum.</p>

Let's create a new view-model, in the ViewModels folder:

```
public class ForumWriteViewModel
{
    public string? Title { get; set; }

    public static ForumWriteViewModel FromForum(Forum forum)
    {
        return new ForumWriteViewModel
        {
            Title = forum.Title,
        };
    }
}
```

- 14 When the user clicks the Create link at the top of the Forum page, it will send a GET request to the server. In response, the server needs to send a blank page to the web browser.

To do this, right-click in the Create method (either of the overloads will work fine), and select Add View. Use the Create template, with the ForumWriteViewModel as the model. Make sure that Reference Script Libraries is checked, too. (We will need this to be checked for a later lab – you should ensure it's checked every time you add a Create or Edit view, at least for now.)

The screenshot shows the 'Add Razor View' dialog box. The 'View name' field contains 'Create'. The 'Template' dropdown menu is open, showing 'Create' selected. The 'Model class' dropdown menu is open, showing 'ForumWriteViewModel (QAForum.ViewModels)' selected. The 'Data context class' dropdown menu is open, showing '<Required>' selected. Under the 'Options' section, the 'Create as a partial view' checkbox is unchecked, the 'Reference script libraries' checkbox is checked, and the 'Use a layout page' checkbox is checked. The 'Add' button is highlighted.

Click Add, and the view will be created. Change the contents of the <h4> at the top:

```
<h1>Create</h1>
```

```
<h4>Forum</h4>
```

15	<p>In the view, you will see this line:</p> <pre><form asp-action="Create"></pre> <p>Although it's not obvious just from looking at it, the HTML which is generated by this includes an action of POST:</p> <pre><form action="/Forum/Create" method="post"></pre> <p>So when the user clicks the Create button, the data they entered will be handled by the second overload of Create in the controller, the one which is decorated with the [HttpPost] attribute.</p> <p>Change the parameter of that method to be a ForumWriteViewModel, and then add code to create a Forum and add it to the database:</p> <pre>public ActionResult Create(ForumWriteViewModel viewModel) { try { Forum forum = new Forum { Title = viewModel.Title }; context.Forums.Add(forum); context.SaveChanges(); return RedirectToAction(nameof(Index)); } catch { return View(viewModel); } }</pre> <p>Note that, after successfully adding the new forum to the database, we use the RedirectToAction helper method to generate a RedirectResult. This will send a response to the web browser (with response code 302, meaning Found) telling it that the next page it needs is the Index action. The browser will then make a new request, asking for the Index action from the server.</p>
16	Test your work, and make sure you can add forums.
17	Editing data is similar. And our ForumWriteViewModel is perfect for the job, since the data the user enters when creating a forum is the same as the data they enter when editing a forum. (The Forum Id is needed in order to update a forum, but this is passed from one view to the next through the URL, and does not need to be part of the view-model.)
18	<p>In the controller, when the user chooses to edit a forum, we need to retrieve the current details of that forum from the database and present them to the user. This is different to creating a forum. So, the first Edit action needs to include this extra step:</p> <pre>public ActionResult Edit(int id) {</pre>

	<pre> var forum = context.Forums.Single(f => f.ForumId == id); return View(ForumWriteViewModel.FromForum(forum)); } </pre>
19	Create a view for editing forums. Use the Edit template, and set the model class to ForumWriteViewModel. Once again, change the contents of the <h4> to simply the word "Forum".
20	<p>Finally, add code to the HttpPost version of the Edit method to edit the data and save it back to the database. Don't forget to change the parameters!</p> <pre> public ActionResult Edit(int id, ForumWriteViewModel viewModel) { try { var forum = context.Forums.Single(f => f.ForumId == id); forum.Title = viewModel.Title; context.SaveChanges(); return RedirectToAction(nameof(Index)); } catch { return View(viewModel); } } </pre> <p>Then run the program, and test that you can edit the name of the forums</p>
21	<p>The next step is to get the Delete action to work.</p> <p>When the user clicks the Delete button, they will see a confirmation screen, which is quite similar to the Details screen. Because of this, we can use the same view-model as was used for the Details action</p>
22	<p>Edit the Get version of the Delete method to retrieve the data:</p> <pre> public ActionResult Delete(int id) { var forum = context.Forums.Single(f => f.ForumId == id); return View(ForumViewModel.FromForum(forum)); } </pre>
23	Add a view. Use the Delete template, and set the model to ForumViewModel. As before, change the contents of the <h4> at the top of the page.
24	<p>Write the code to delete a forum in the [HttpPost] Delete method:</p> <pre> public ActionResult Delete(int id, IFormCollection collection) { var forum = context.Forums.Single(f => f.ForumId == id); try { context.Forums.Remove(forum); context.SaveChanges(); } } </pre>

	<pre> return RedirectToAction(nameof(Index)); } catch { return View(ForumViewModel.FromForum(forum)); } } </pre> <p>Note that, for this method, we leave the <code>IFormCollection</code> parameter in place. We don't need to use it at all – the only data we need is the id. But the <code>Get</code> method also needs an id, so we have to leave the second parameter in place to ensure that the two methods are sufficiently different to meet the C# rules about method overloading.</p>
25	Test the delete method by deleting the forum you created earlier in the lab. It should delete without any problems.
26	<p>But now, try to delete one of the pre-existing forums. What happens? Do you know why?</p> <p>Read on, and we'll explain it</p>
27	<p>When you try to delete one of the existing forums, an exception occurs. We don't know what the details of that exception are (yet), but this exception results in control going into the "catch" block in the <code>Delete</code> action.</p> <p>And that catch block... well, it simply re-displays the same view again, with no indication to the user that anything has gone wrong (apart from the fact that the button didn't seem to do anything).</p>
28	<p>Let's see if we can improve things. Change the catch block in the <code>Delete</code> action as follows:</p> <pre> catch(Exception e) { ViewBag.ErrorText = e.GetBaseException().Message; return View(ForumViewModel.FromForum(forum)); } </pre>
29	<p>Right-click on the <code>Delete</code> action, and select <code>Go To View</code>. Modify the view by adding a line to show the error message, as shown below:</p> <pre> <h1>Delete</h1> <pclass="text-danger">@ViewBag.ErrorText</p> <h3>Are you sure you want to delete this?</h3> </pre>
30	<p>Now, try to delete one of the pre-existing forums again.</p> <p>You should now see an error message! It's not a great error message – it's not very user-friendly, and it's also not good practice to show exception messages to the user in general in case they contain sensitive data. But it's enough for us to work out what's going on. It shows that there's a foreign key violation – we can't</p>

	<p>delete this forum, because the forum contains threads. We'd need to delete the threads it contains first (and also delete any posts in those threads).</p> <p>(This behaviour was configured in the ForumDbContext class, in the OnModelCreating method. The rules got copied into the initial migration when the database was created and then applied to the database. It is now SQL Server's responsibility to ensure that these rules are not violated.)</p> <p>We'll leave you to think about how this error message could be improved. But for now, it's enough that we can at least see the error message.</p>
31	<p>That's as much as we have time for for now but if you have time and want more practise you could try going through the same process (from task 2) for threads and posts.</p>