

## Exercise3: A Quick Tour around MVC Core

### Objective

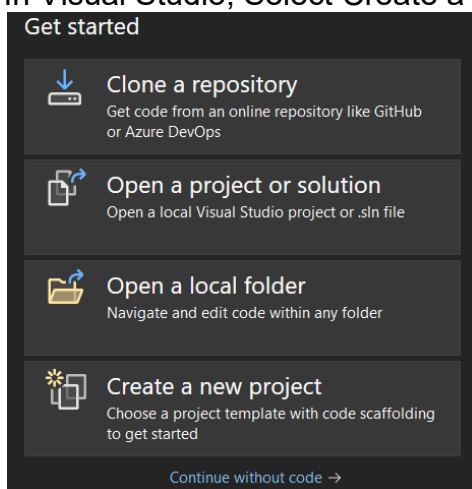
In this exercise we give you a quick tour around the fundamentals of MVC Core

This exercise will take around 60minutes.

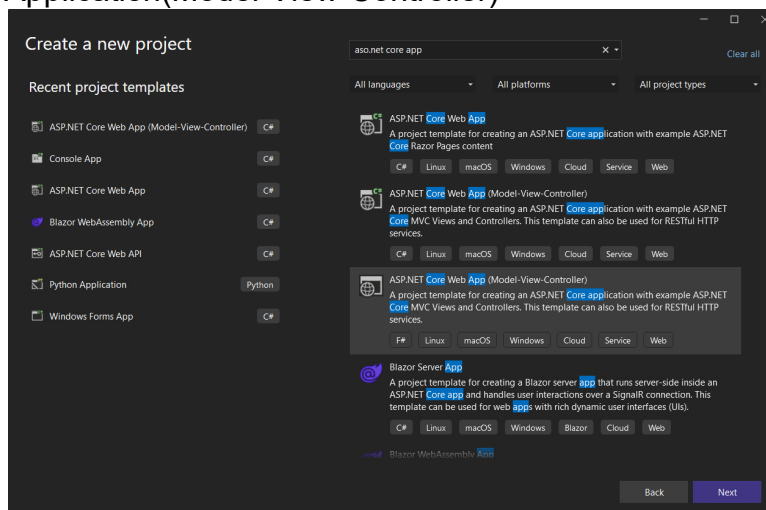
#### 1 Create a new Web Application:

We will create one that is very similar to the one which we will use in most of the labs.

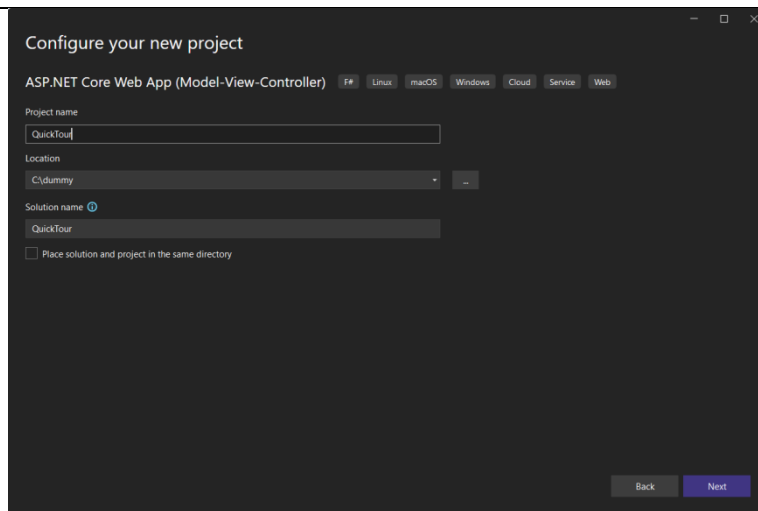
- In Visual Studio, Select Create a new project



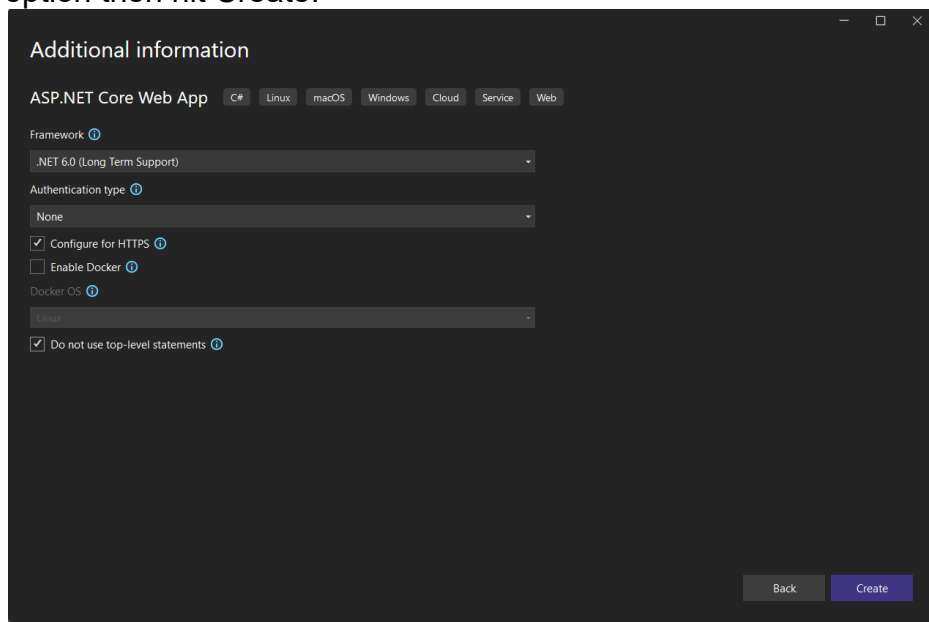
- Search for “asp.net core app” and select ASP.NET Core Web Application(Model-View-Controller)



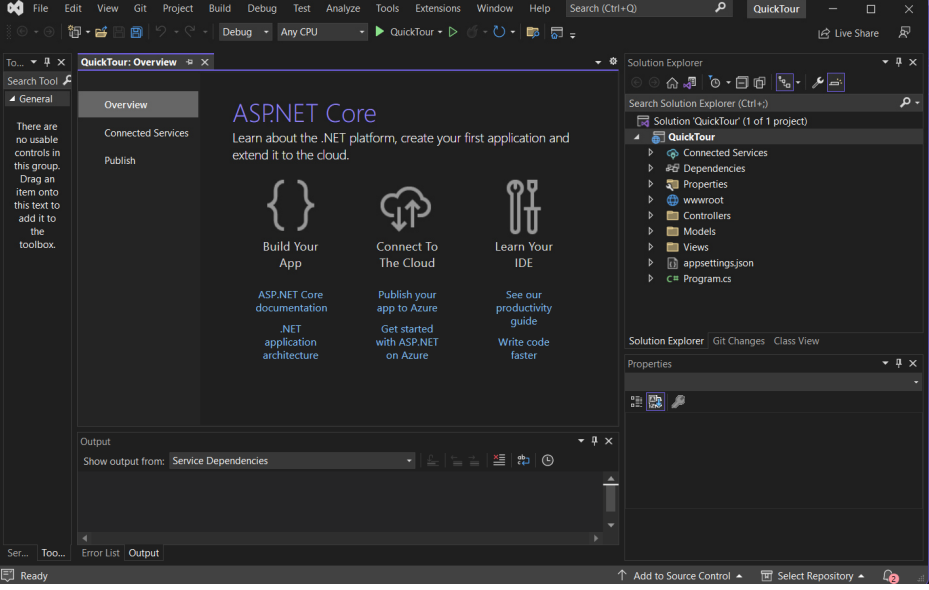
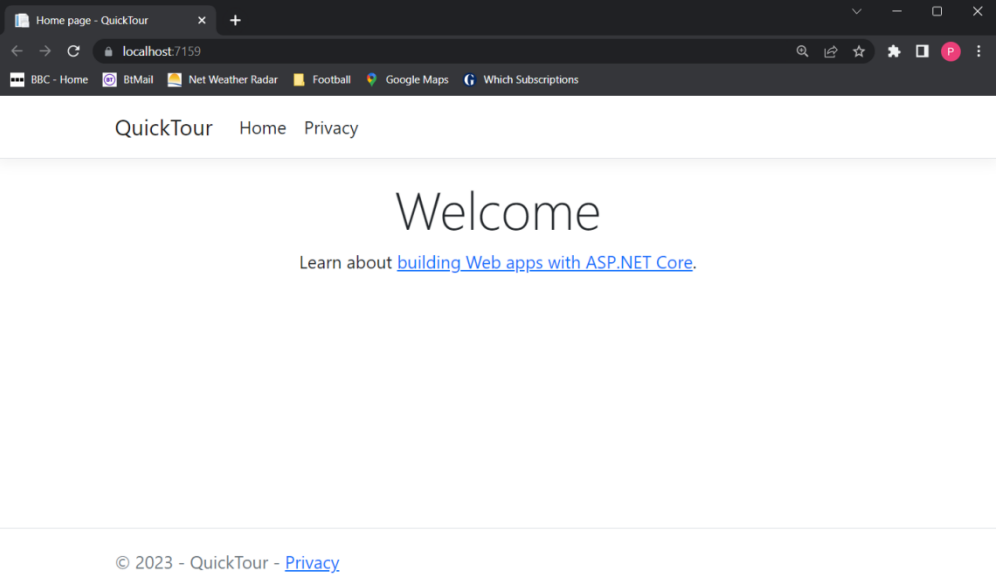
- Name the project **QuickTour**. Leave other settings as is:

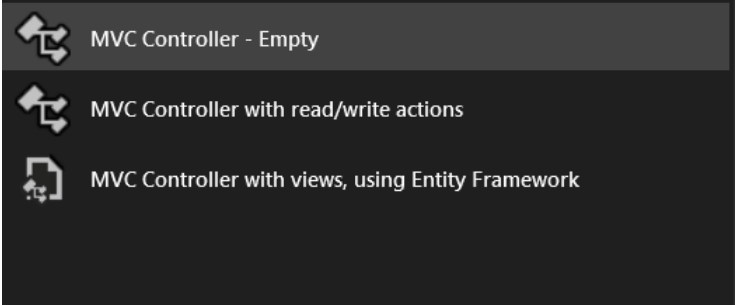


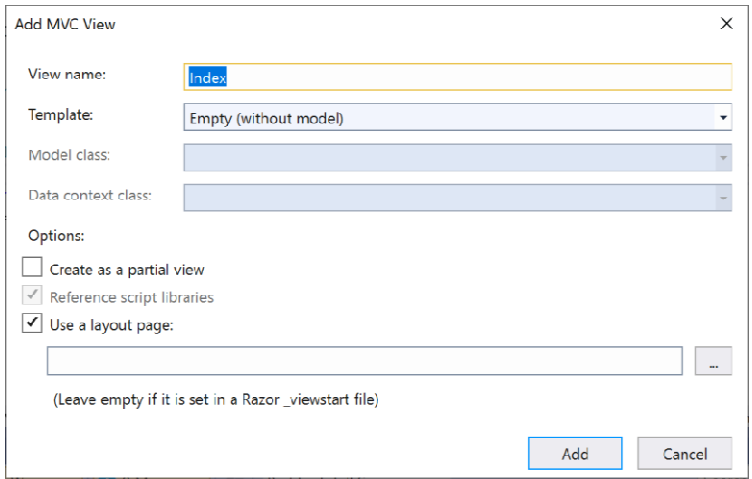
- Select Next
- In the Additional Information dialog make sure “.net 6.0 (Long term support)” is chosen as the framework and put a tick in the “Do not use top level statements” option then hit Create.




Visual Studio creates a new MVC project, based on the default MVC project template.

	
2	<p>Just to make sure it's a runnable website, press F5.</p> <p>You may be asked to accept a certificate the very first time you run an MVC application in development mode. If so, you should accept the certificate. Then, you will see the home page of your web browser:</p> 
3	Close the browser
4	<p>To get started we are going to need some data.</p> <p>To flesh it out a bit we will imagine we're building an on-line Forum, so let's use that as the topic.</p> <p>Right-click the Models folder and add a C# class called Forum.cs.</p>

	<p>Populate it like this:</p> <pre>Public class Forum {     Public int ForumId{ get; set; }     Public string? Title { get; set; } }</pre> <p>Note we have added an Id because we intend to store this in a database eventually.</p> <p>The recommendation is to use &lt;className&gt;Id as it fits in with EntityFramework conventions somewhat better than just 'Id'</p>
5	<p>Right-click on the Controllers folder and add a ForumController.</p> <p>Select the MVC Controller – Empty</p>  <p>Name the controller 'ForumController'</p>
6	<p>You should now see the (default) method in the Controller called Index(). Public methods in a controller are called Actions – this is the Index Action.</p> <p>Edit the Index code:</p> <pre>Public IActionResult Index() {     Forum forum = new Forum { ForumId = 1, Title = "First Forum" };     ViewBag.Forum = forum;     Return View(); }</pre> <p><b>You will need to select Ctrl+dot to resolve the namespaces.</b> We typically won't mention this step in future instructions because it should just become a habit that you do this.</p>
7	<p>Expand the Views folder and note that it just contains subfolders Home and Shared</p>
8	<p>Place the cursor in the Index() method, right-click on View &gt; Add View, and accept all the default options as shown below:</p>

	 <p>Note that under the Views folder we now have a subfolder 'Forum' (because we are in the <i>ForumController</i>) and a file Index.cshtml (because we are in the Index() method).</p>
9	<p>Open the file Index.cshtml. It will contain a mix of C# and HTML. The "Razor" markup syntax allows you to mix C# and HTML together in this way, and the Razor engine turns this file into pure HTML (by running the C#) before sending it to the user's web browser.</p> <p>Let's display details of the forum, by adding this line to the end of the file:</p> <pre>&lt;h1&gt;Index&lt;/h1&gt; &lt;p&gt;Forum with ID @ViewBag.Forum.ForumId is @ViewBag.Forum.Title&lt;/p&gt;</pre>
10	<p>F5. When the starter page appears, append /Forum/Index in the address bar. You should get this (note that the port number is random):</p>
11	<p>Let's set the default route such that our Forum page appears on app start.</p> <p>Go to Program.cs</p> <p>Go right to the end of the file and modify 'Home' to 'Forum'</p> <pre>app.MapControllerRoute(     name: "default",     pattern: "{controller=Forum}/{action=Index}/{id?}");</pre>
12	<p>F5 and our Forum page should appear.</p>
13	<p>You may have noted that, whenever you write code to access the members of the ViewBag, you don't get any intellisense.</p> <p>The ViewBag is dynamically typed. This means you can add whatever data you want to it. But the downside is that there is no intellisense, and no compiler checking that what you've done is valid. If you make a small typo, you won't know about it until you actually run the program.</p> <p>Because of this, it's rare to use the ViewBag to pass anything other than the</p>

	<p>occasional bit of ad hoc data to views. Instead, we use strongly-typed models, which give us all the type checking we are used to.</p> <p>We supply a strongly-typed model to the view by passing it as a parameter to the View method. So, change your code as follows:</p> <pre> public IActionResult Index() {     Forum forum = new Forum { ForumId = 1, Title = "First Forum" };     ViewBag.Forum = forum; // Remove this line     return View(forum); } </pre>
14	<p>Modify the index.cshtml file as follows:</p> <pre> @using QuickTour.Models @model Forum  @{     ViewData["Title"] = "Index"; }  &lt;h1&gt;Index&lt;/h1&gt;  &lt;p&gt;Forum with ID @Model.ForumId is @Model.Title&lt;/p&gt; </pre> <p>Press F5, and check this still does the same as before.</p> <p>Note how we use @model (with a lower-case 'm') to state what data type is used as the model for this view. Then, we use @Model (with a capital 'M') to access the model, this time with full intellisense (i.e. if you type @Model. you will get suggestions for what comes after the .)</p>
15	<p>We can ask Visual Studio to build a view for us, which will display the data in a model. Let's do that.</p> <p>Switch back to the ForumController, and right-click on the Index method. Choose Add View. This time chose Razor View (NOT Razor View – Empty)</p>  <p>Click Add</p> <p>Chose Template: Details, Model class: Forum (QuickTour.Models)</p>

Add Razor View

View name
Index

Template
Details

Model class
Forum (QuickTour.Models)

Options

☐ Create as a partial view

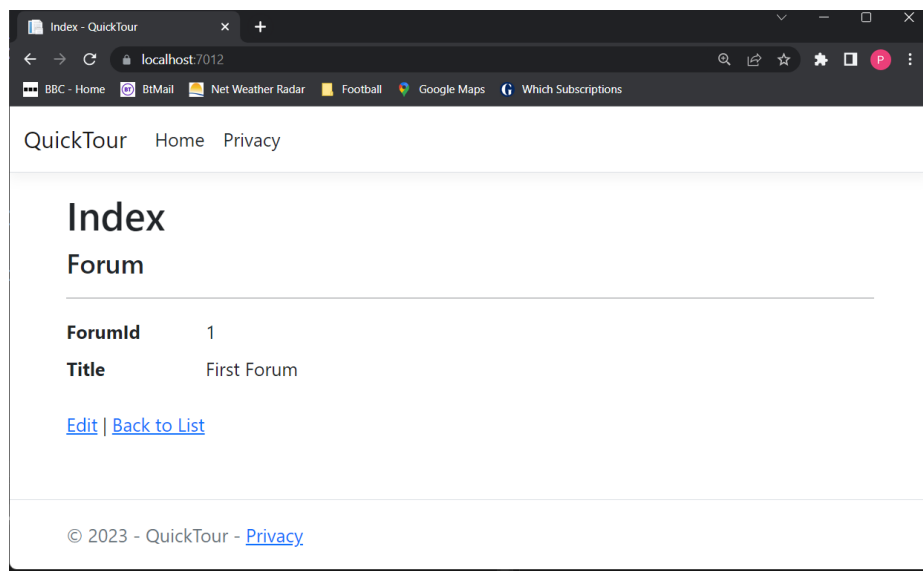
☒ Reference script libraries

☒ Use a layout page

(Leave empty if it is set in a Razor \_viewstart file)

Add
Cancel

You will be prompted to confirm that you want to replace the old view. Select Yes. Take a brief look at the Index.cshtml file that was generated, and note how it uses `@model` at the top of the file. Then press F5 to see what it looks like in your web browser:

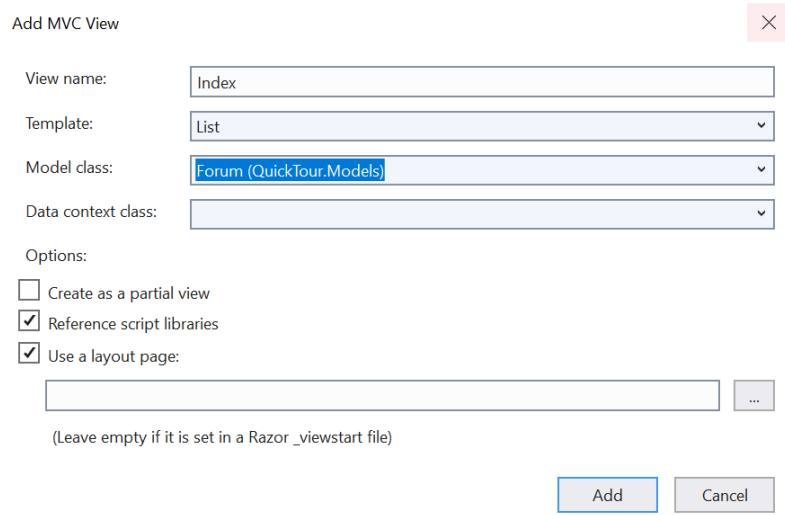


16 What if we need to display more than one Forum?

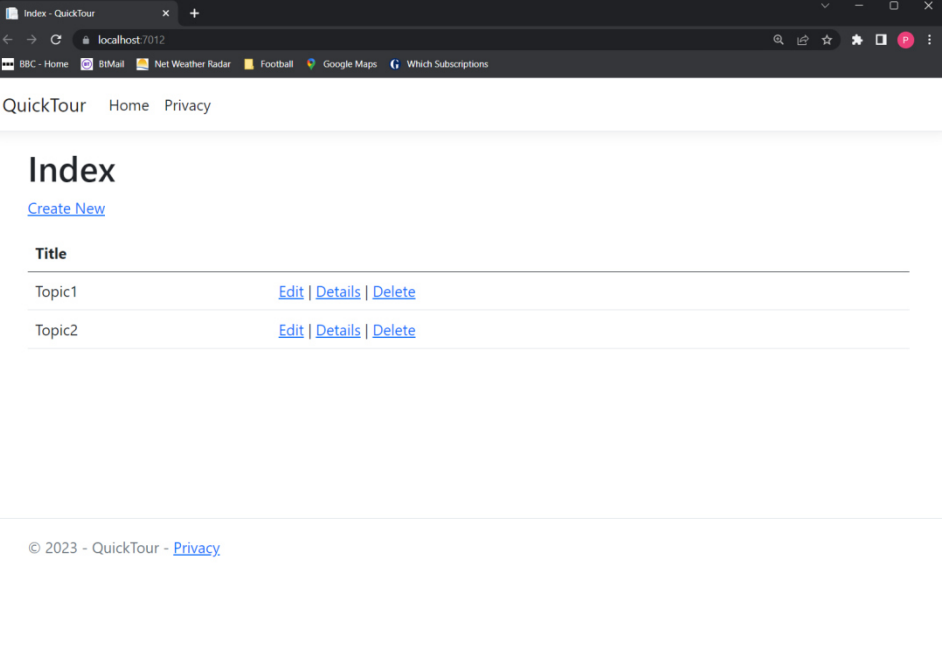
Edit the Index action code:

```
public IActionResult Index()
{
    // Get a list of Forums from the database
    IEnumerable<Forum> forums = new List<Forum>() {
        new Forum() { Title = "Topic1" },
        new Forum() { Title = "Topic2" }
    };
    return View(forums);
}
```

Press F5 now, and note that you get an error. This is because the model that's specified in the controller is not the same data type as the type expected by the view.

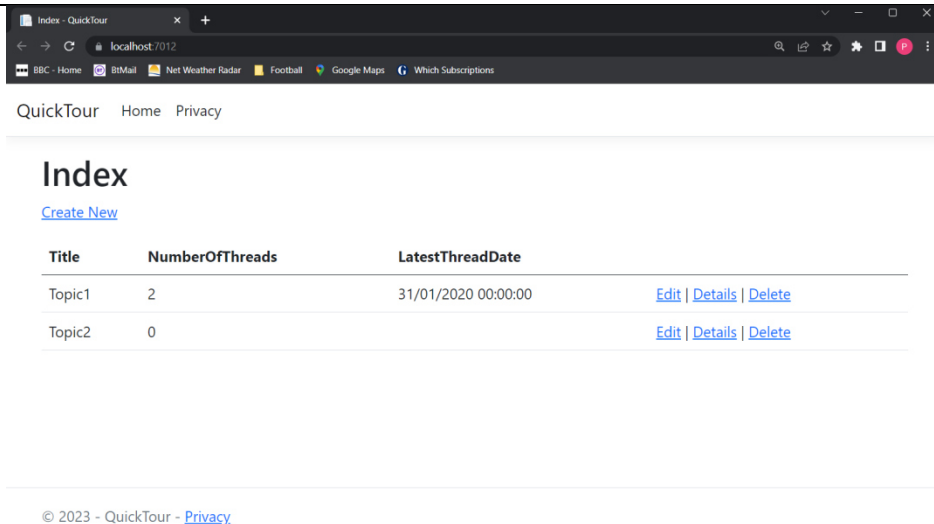
	The controller supplies an IEnumerable<Forum>, whereas the view expects a Forum.
17	<p>Place the cursor in the Index() method, and once again right-click on View &gt; Add View. Again add a Razor View but this time select a List template based on the Forum class:</p>  <p>As before, select Yes when asked to confirm you want to replace the existing view.</p>
18	<p>Ensure Index.cshtml is open. Take a look at the generated file. Look at the @model line at the top of the file. Notice that the List template generates a view that expects the same data type as the controller is supplying.</p> <p>It also contains a &lt;table&gt; (there are mixed views on whether it should use a table!), and within that table are two lines which display the Forum id:</p> <pre>@Html.DisplayNameFor(model =&gt;model.ForumId)</pre> <p>and</p> <pre>@Html.DisplayFor(modelItem =&gt;item.ForumId)</pre> <p>We don't really want it to show the database Id. We could simply delete these two lines, but if we produced other views such as Edit, Create, Details etc., again we would not want the database Id to be scaffolded so let's configure it so by default this is never scaffolded.</p> <p>Go to Models &gt; Forum.cs and add this attribute to suppress this property when scaffolding a view (and yes, resolve the namespace):</p> <pre>[ScaffoldColumn(false)] Public int ForumId{ get; set; }</pre> <p>Repeat the operation (from task 17) of creating a View (overwriting the current Index.cshtml). Note that the lines which show the Id are no longer included.</p>
19	F5. You should now get this:



	
20	<p>Let's make our data a little more interesting. Add a new class in the Models folder:</p> <pre>public class Thread {     public int ThreadId { get; set; }     public string? Title { get; set; }     public string? UserName { get; set; }     public DateTime DateCreated { get; set; }     public int ForumId { get; set; }     public virtual Forum? Forum { get; set; } }</pre> <p>And modify the Forum class so that it contains many Threads:</p> <pre>public class Forum {     [ScaffoldColumn(false)]     public int ForumId { get; set; }     public string? Title { get; set; }     public virtual ICollection&lt;Thread&gt;? Threads { get; set; } }</pre>
21	<p>Now that we have a bit more data, it's becoming even less appropriate for the controller to instantiate the data. Create a new class in the Models folder which represents a fake database. (Of course, we will explore real database connections a little later in the course.) Call the class MockForumContext. Add this method to the new class:</p> <pre>public class MockForumContext {     public IEnumerable&lt;Forum&gt; GetForums()     {         List&lt;Thread&gt; topic1Threads = new List&lt;Thread&gt;() {             new Thread() {                 Title = "Thread 1",                 UserName = "User 1",                 DateCreated = new DateTime(2023, 1, 28)             }, </pre>

	<pre>         New Thread() {             Title = "Thread 2",             UserName = "User 2",             DateCreated = new DateTime(2023, 1, 31)         }     };      Return new List&lt;Forum&gt;() {         New Forum() { Title = "Topic1", Threads = topic1Threads },         New Forum() { Title = "Topic2", Threads = new List&lt;Thread&gt;() }     }; } </pre>
	<p>Modify the ForumController class to use this new method:</p> <pre> public IActionResult Index() {     MockForumContext mockForum = new MockForumContext();     IEnumerable&lt;Forum&gt; forums = mockForum.GetForums();     return View(forums); } </pre>
22	<p>We'd like to modify the view to include two extra columns:</p> <ul style="list-style-type: none"> <li>• The number of threads in the forum, and</li> <li>• The date of the latest thread in the forum</li> </ul> <p>Where should we put the code to find these data? We have several options:</p> <ul style="list-style-type: none"> <li>• In the view. The view has access to the whole of the model, and would be able to find these details, but it is generally considered bad practice to process data in the view. The data should be processed before it is given to the view</li> <li>• In the controller. The controller also has the capability to do this processing, but once again, it is not its job. The job of the controller is to process the incoming web request. It should not be directly responsible for processing data</li> <li>• The other existing class we could use is the model. But the model should be shared between all of the business, not just this view and indeed not just this web application. It would be wrong to put something into the model which has such a specific purpose</li> </ul> <p>Since none of these is appropriate, we are left with the final option, which is to create a new class. The class is a ViewModel – a model which is built specifically for the purpose of passing data to the view.</p> <p>This will have another benefit: we can remove the [ScaffoldColumn] attribute from the Models.Forum class. Although this attribute certainly makes sense for our view, and maybe even for other, related views we might create in the future, we can't guarantee with 100% certainty that this column won't need to be scaffolded in some view, somewhere, ever, either in this application or any other program we share our model with. Therefore, the attribute really does not belong on the model, and would fit much better on the ViewModel.</p>
23	<p>So, start by removing the [ScaffoldColumn] attribute from the Models.Forum class</p>

24	<p>Now, add a folder to the project, called ViewModels.</p> <p>In that folder, create a new class called ForumViewModel:</p> <pre data-bbox="304 324 1525 562"> public class ForumViewModel {     [ScaffoldColumn(false)]     public int ForumId { get; set; }     public string? Title { get; set; }     public int NumberOfThreads { get; set; }     public DateTime? LatestThreadDate { get; set; } }</pre> <p>Note how the class is customised to contain exactly the data we want to show, even including the fact that the LatestThreadDate is nullable just in case there are no threads.</p>
25	<p>Add two static methods to the ForumViewModel class, which are used to create the view-models:</p> <pre data-bbox="304 824 1525 1317"> public static ForumViewModel FromForum(Forum forum) {     return new ForumViewModel     {         ForumId = forum.ForumId,         Title = forum.Title,         NumberOfThreads = forum.Threads.Count,         // DefaultIfEmpty() and ?. ensure this gives null if 0 threads         LatestThreadDate = forum.Threads.DefaultIfEmpty().Max(t =&gt; t?.DateCreated)     }; }  public static IEnumerable&lt;ForumViewModel&gt; FromForums(     IEnumerable&lt;Forum&gt; forums) {     return forums.Select(f =&gt; FromForum(f)); }</pre>
26	<p>In the controller, change the line that returns the view so that the view receives a collection of view-models instead of models:</p> <pre data-bbox="304 1440 1525 1581"> public IActionResult Index() {     ...     return View(ForumViewModel.FromForums(forums)); }</pre>
27	<p>Finally, re-create the view. As before, right-click inside the Index action (method) and select Add View. This time, choose the List template, and set the Model to be ForumViewModel. Confirm that you're happy to overwrite the existing view.</p>
28	<p>Press F5, and check the results:</p>



Not bad! The important thing at this stage is to understand the separation of concerns, i.e. how each class has one and only one job to do. The model represents the data as it is (or will eventually be) stored in the database. The view-model represents the data that will be displayed. The view is responsible for actually creating that display, and the controller handles the incoming request, pulling together all the other bits as required.