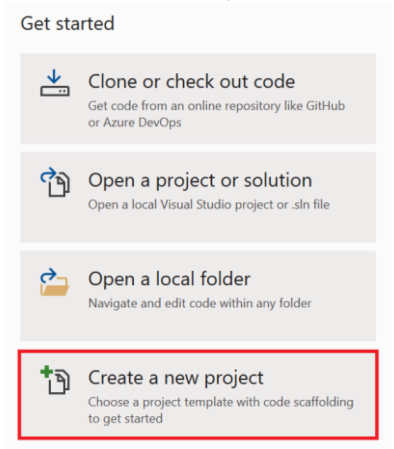
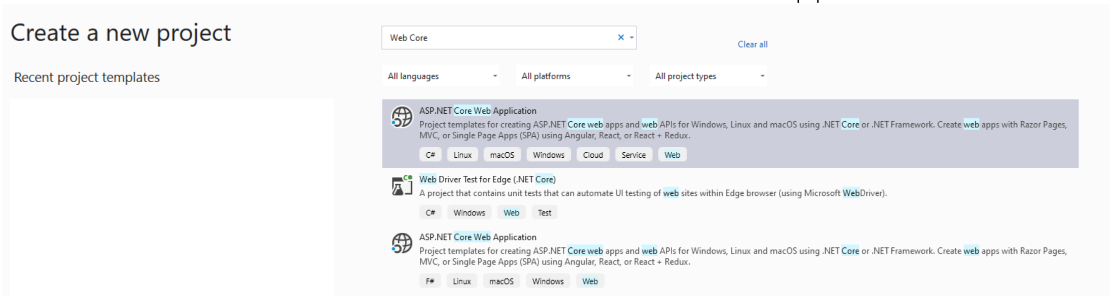
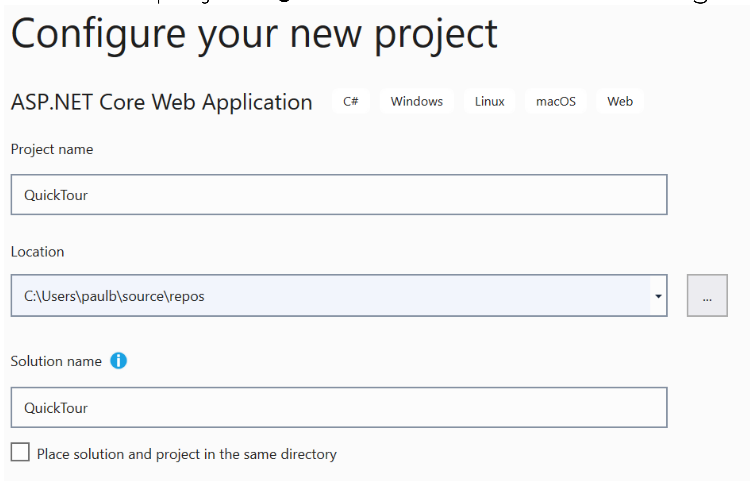
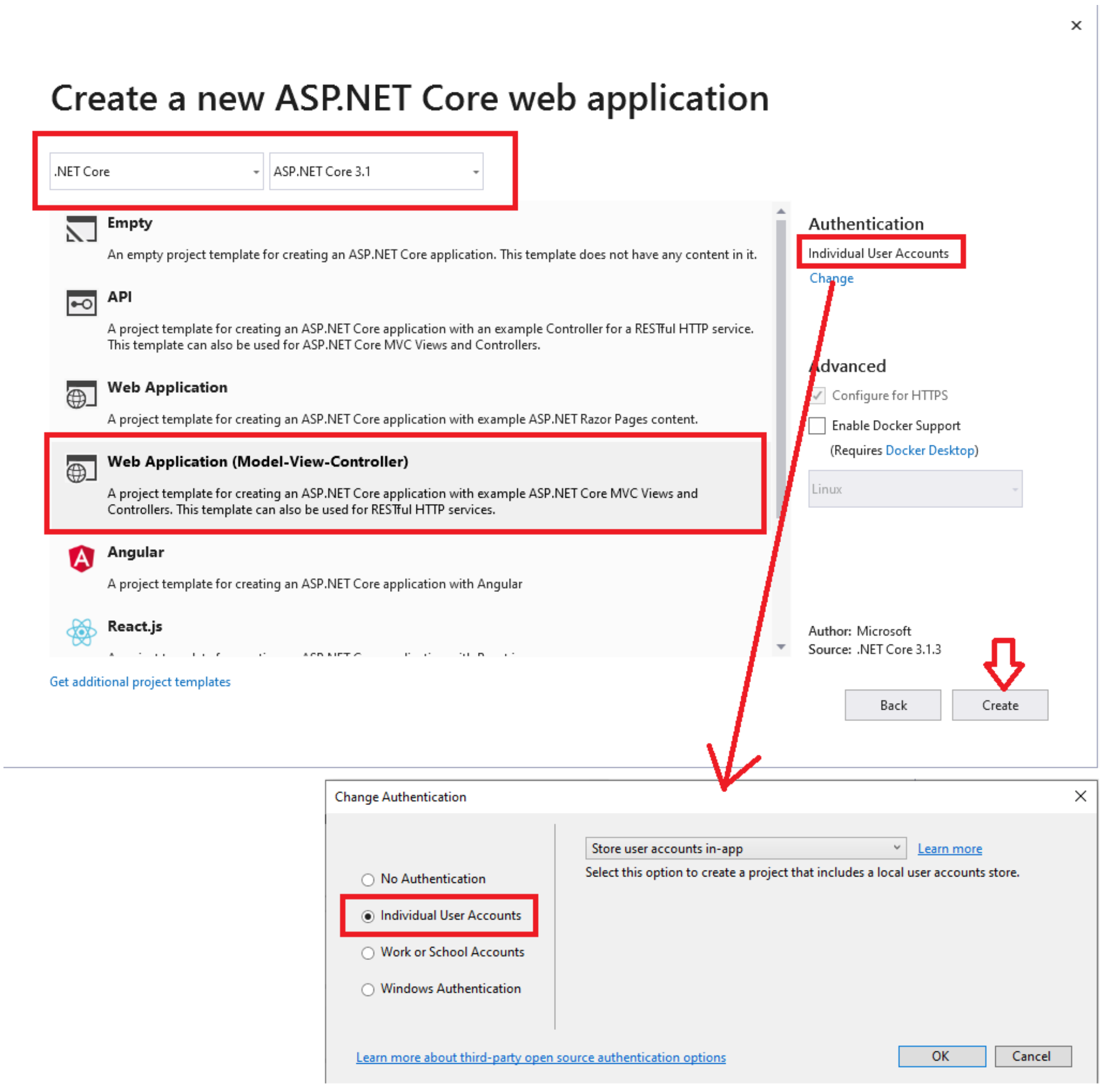
Exercise 3 : A Quick Tour around MVC Core

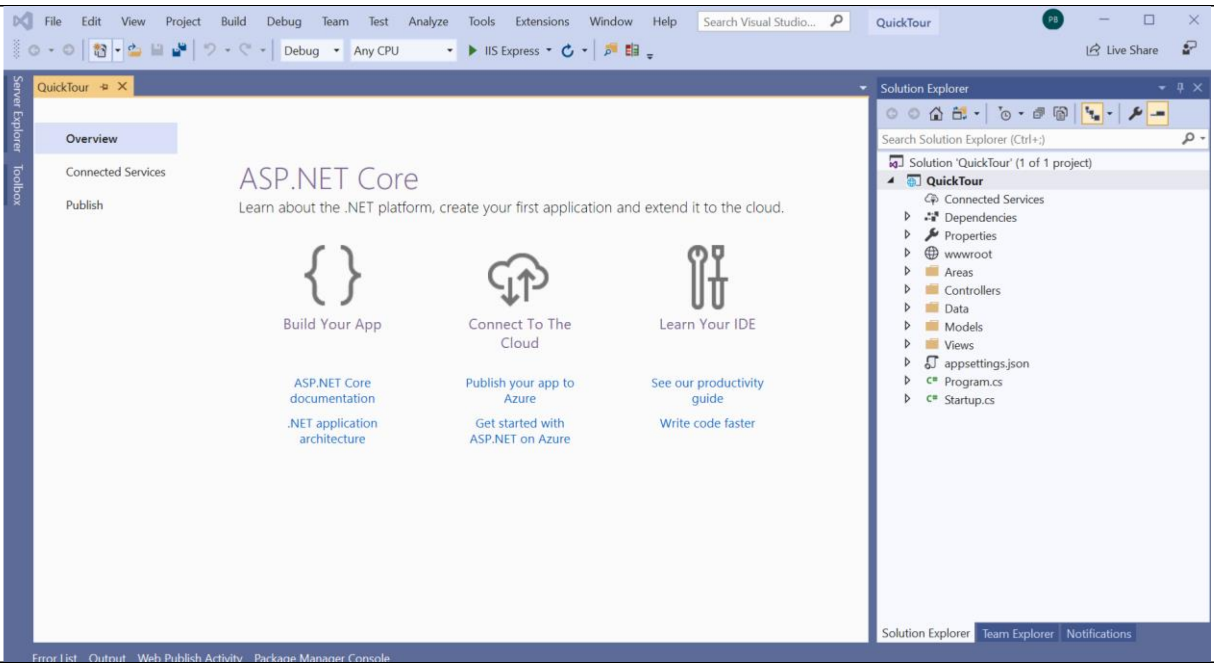
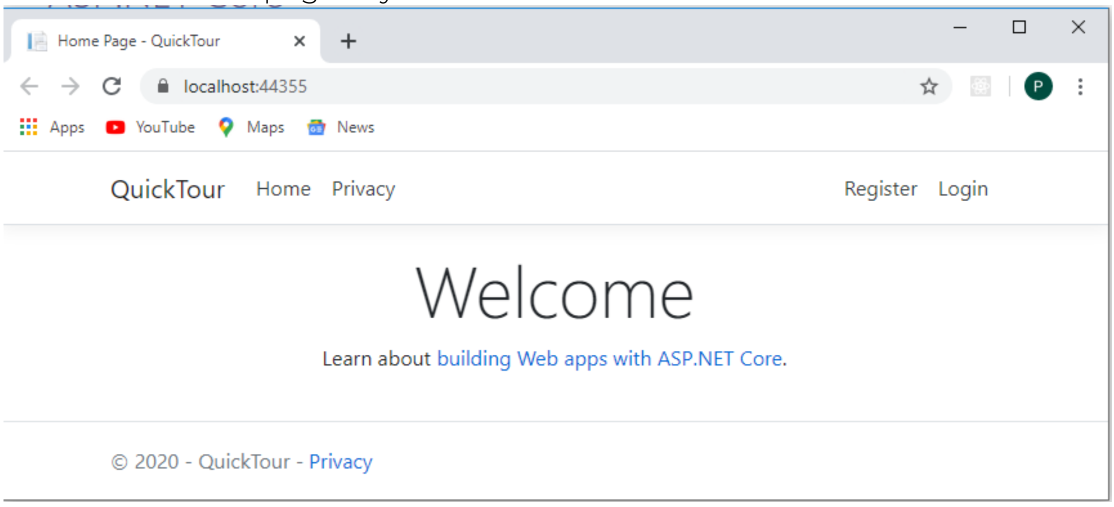
Objective

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

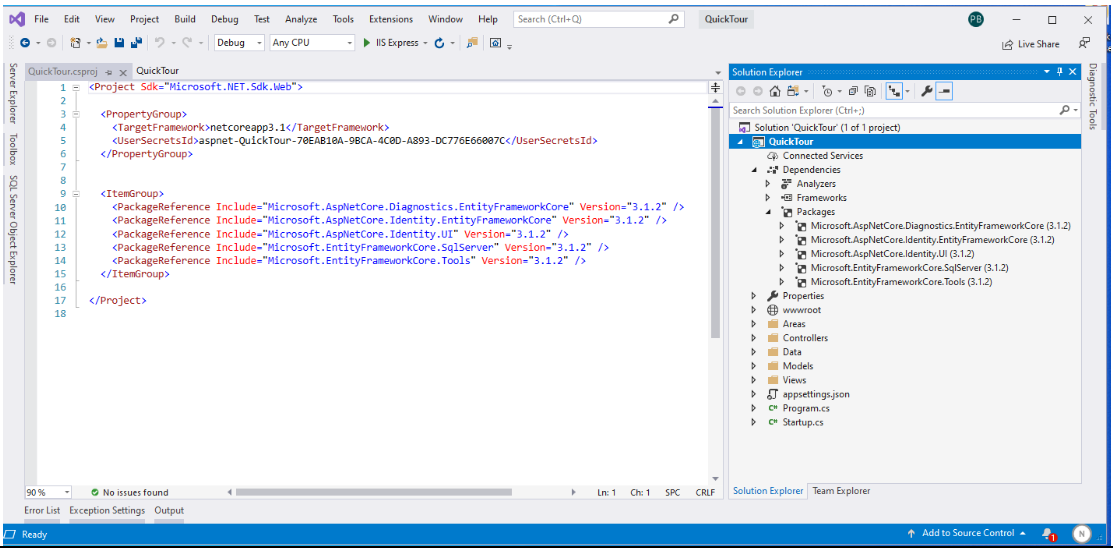
This exercise will take around 60 minutes.

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  Get started Clone or check out code Get code from an online repository like GitHub or Azure DevOps Open a project or solution Open a local Visual Studio project or .sln file Open a local folder Navigate and edit code within any folder Create a new project Choose a project template with code scaffolding to get started Search for “Web Core” and select ASP.NET Core Web Application  Create a new project Recent project templates Web Core Clear all All languages · All platforms · All project types ASP.NET Core Web Application Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux. C# · Linux · macOS · Windows · Cloud · Service · Web Web Driver Test for Edge (.NET Core) A project that contains unit tests that can automate UI testing of web sites within Edge browser (using Microsoft WebDriver). C# · Windows · Web · Test ASP.NET Core Web Application Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux. F# · Linux · macOS · Windows · Web Name the project **QuickTour.** Leave other settings as is:  Configure your new project ASP.NET Core Web Application · C# · Windows · Linux · macOS · Web Project name: QuickTour Location: C:\Users\paulb\source\repos Solution name: QuickTour ☐ Place solution and project in the same directory Select Create

	Select Web Application (Model-View-Controller) and change the Authentication to Individual User Accounts  **Create a new ASP.NET Core web application** .NET Core / ASP.NET Core 3.1 **Empty** An empty project template for creating an ASP.NET Core application. This template does not have any content in it. **API** A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers. **Web Application** A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content. **Web Application (Model-View-Controller)** A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services. **Angular** A project template for creating an ASP.NET Core application with Angular **React.js** Get additional project templates. Authentication: Individual User Accounts (Change). Advanced: ✓ Configure for HTTPS; ☐ Enable Docker Support (Requires Docker Desktop); Linux. Author: Microsoft. Source: .NET Core 3.1.3. Back / Create. Change Authentication: ○ No Authentication; ● Individual User Accounts; ○ Work or School Accounts; ○ Windows Authentication. Store user accounts in-app (Learn more). Select this option to create a project that includes a local user accounts store. Learn more about third-party open source authentication options. OK / Cancel. Visual Studio creates a new MVC project, based on the default MVC project template, which includes support for individual user accounts.

	 File Edit View Project Build Debug Team Test Analyze Tools Extensions Window Help Search Visual Studio... QuickTour Debug Any CPU IIS Express Live Share Server Explorer Toolbox QuickTour Overview Connected Services Publish ASP.NET Core Learn about the .NET platform, create your first application and extend it to the cloud. Build Your App Connect To The Cloud Learn Your IDE ASP.NET Core documentation .NET application architecture Publish your app to Azure Get started with ASP.NET on Azure See our productivity guide Write code faster Solution Explorer Search Solution Explorer (Ctrl+;) Solution 'QuickTour' (1 of 1 project) QuickTour Connected Services Dependencies Properties wwwroot Areas Controllers Data Models Views appsettings.json Program.cs Startup.cs Solution Explorer Team Explorer Notifications Error List Output Web Publish Activity Package Manager Console
2	To make sure it's a runnable website, press F5. 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:  Home Page - QuickTour localhost:44355 Apps YouTube Maps News QuickTour Home Privacy Register Login Welcome Learn about building Web apps with ASP.NET Core. © 2020 - QuickTour - Privacy Close the browser

3 Expand the Dependencies > Packages folder. Also, right-click the project > Edit Project File. Note that the (meta) packages listed here match those in the Packages folder



QuickTour.csproj

```xml
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <UserSecretsId>aspnet-QuickTour-70EAB10A-9BCA-4C0D-A893-DC776E66007C</UserSecretsId>
  </PropertyGroup>


  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore" Version="3.1.2" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="3.1.2" />
    <PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="3.1.2" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="3.1.2" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="3.1.2" />
  </ItemGroup>

</Project>
```

Solution Explorer

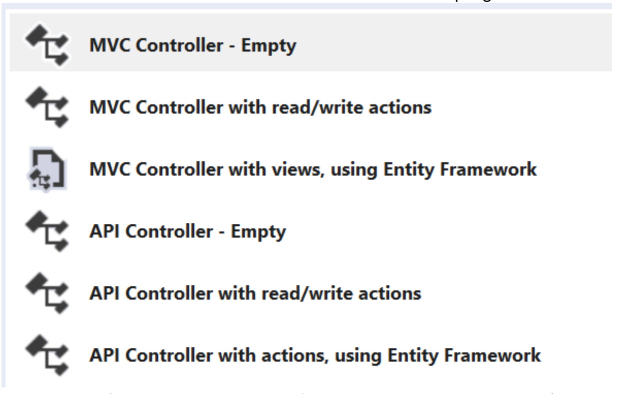
```
Solution 'QuickTour' (1 of 1 project)
  QuickTour
    Connected Services
    Dependencies
      Analyzers
      Frameworks
      Packages
        Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore (3.1.2)
        Microsoft.AspNetCore.Identity.EntityFrameworkCore (3.1.2)
        Microsoft.AspNetCore.Identity.UI (3.1.2)
        Microsoft.EntityFrameworkCore.SqlServer (3.1.2)
        Microsoft.EntityFrameworkCore.Tools (3.1.2)
    Properties
    wwwroot
    Areas
    Controllers
    Data
    Models
    Views
    appsettings.json
    Program.cs
    Startup.cs
```

4 To get started we are going to need some data.
To flesh it out a bit we will imagine we're building an on-line Forum, so let's use that as the topic.
Right-click the Models folder and add a C# class called Forum.cs.
Populate it like this:

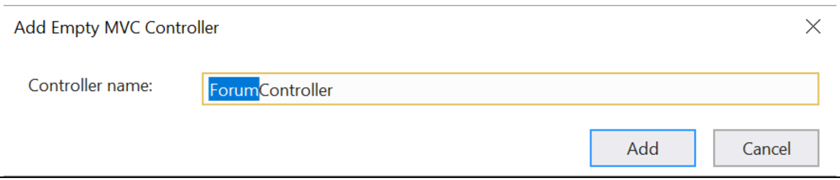
```csharp
    public class Forum
    {
        public int ForumId { get; set; }
        public string Title { get; set; }
    }
```

Note we have added an Id because we intend to store this in a database eventually.
The recommendation is to use <className>Id as it fits in with EntityFramework conventions somewhat better than just 'Id'

5 Right-click on the Controllers folder and add a ForumController.
Select the MVC Controller – Empty


MVC Controller - Empty
MVC Controller with read/write actions
MVC Controller with views, using Entity Framework
API Controller - Empty
API Controller with read/write actions
API Controller with actions, using Entity Framework

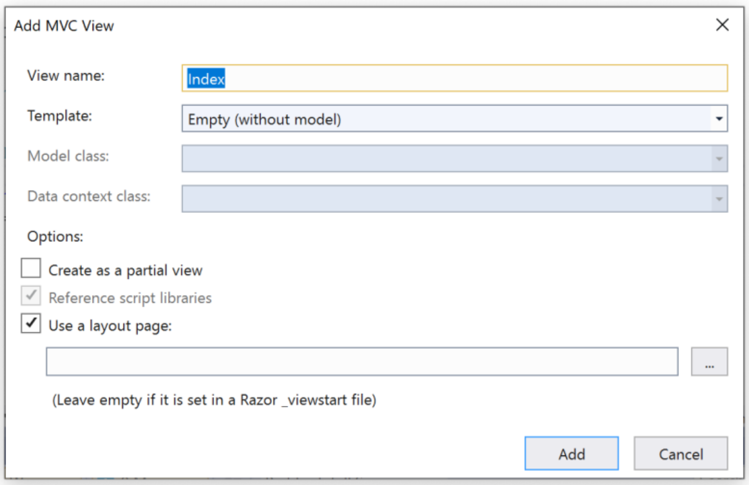
Name the controller 'ForumController'



Add Empty MVC Controller

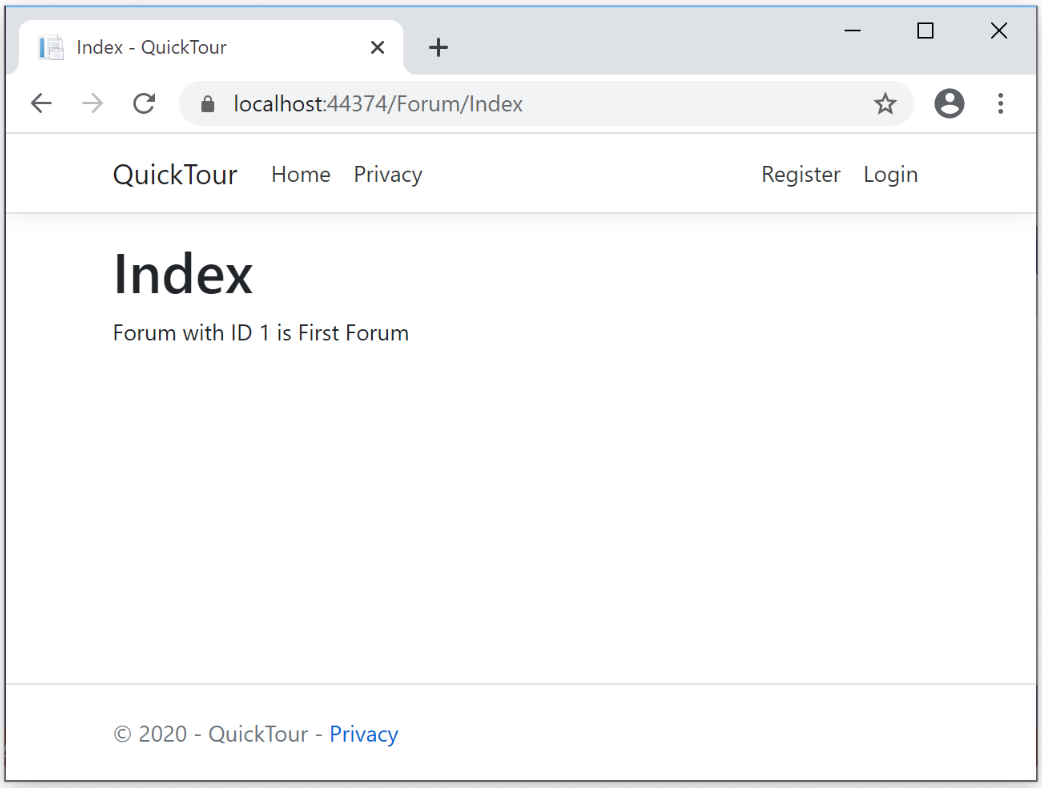
Controller name: ForumController

6	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.
	Edit the Index code:
	`public IActionResult Index()`
	`{`
	`    Forum forum = new Forum { ForumId = 1, Title = "First Forum" };`
	`    ViewBag.Forum = forum;`
	`    return View();`
	`}`
	You will need to select Ctrl+dot to resolve the namespaces.
	We typically won't mention this step in future instructions because it should just become a habit that you do this.

7	Expand the Views folder and note that it just contains subfolders Home and Shared
8	Place the cursor in the Index() method, right-click on View > Add View, and accept all the default options as shown below:  Add MVC View View name: Index Template: Empty (without model) Model class: Data context class: 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 Note that under the Views folder we now have a subfolder 'Forum' (because we are in the *Forum*Controller) and a file Index.cshtml (because we are in the Index() method).

9	Open the file Index.cshtml. It contains 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.
	Let's display details of the forum, by adding this line to the end of the file:
	`<h1>Index</h1>`
	
	`<p>Forum with ID @ViewBag.Forum.ForumId is @ViewBag.Forum.Title</p>`

10 F5. When the starter page appears, append /Forum/Index in the address bar. You should get this (note that the port number is random):



Index - QuickTour

localhost:44374/Forum/Index

QuickTour Home Privacy Register Login

Index

Forum with ID 1 is First Forum

© 2020 - QuickTour - Privacy

11 Let’s set the default route such that our Forum page appears on app start.
Go to Startup.cs
Go right to the end of the file and modify ‘Home’ to ‘Forum’

```
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller=Forum}/{action=Index}/{id?}");
                endpoints.MapRazorPages();
            });
```

12 F5 and our Forum page should appear.

13

You may have noted that, whenever you write code to access the members of the ViewBag, you don't get any intellisense.

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.
Because of this, it's rare to use the ViewBag to pass anything other than the 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.

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:

```
        public IActionResult Index()
        {
            Forum forum = new Forum { ForumId = 1, Title = "First Forum" };
            ~~ViewBag.Forum = forum;~~ // Remove this line
            return View(forum);
        }
```

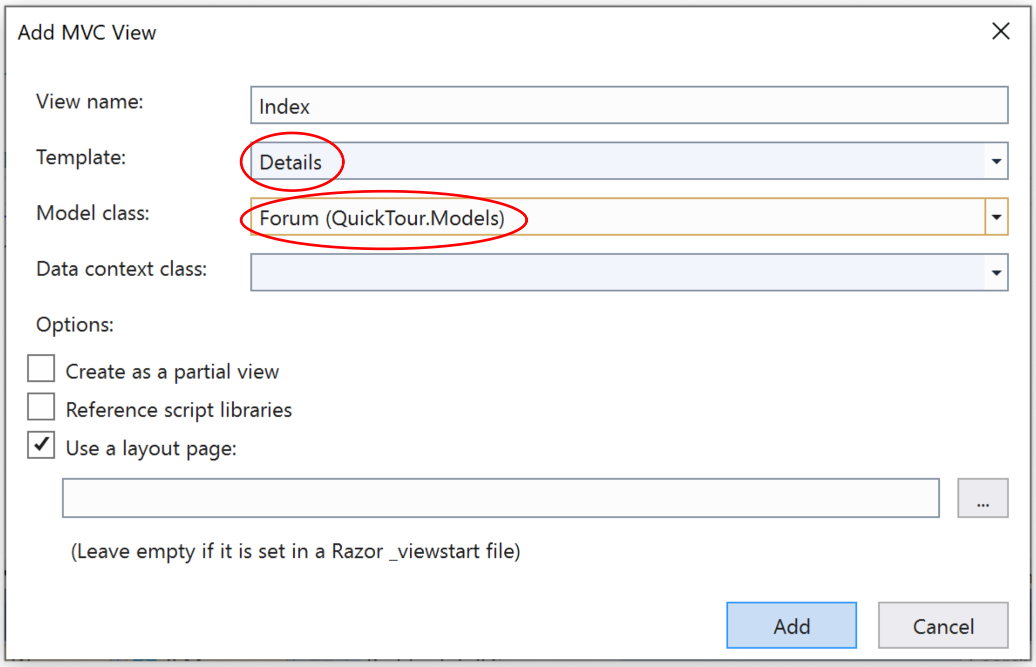
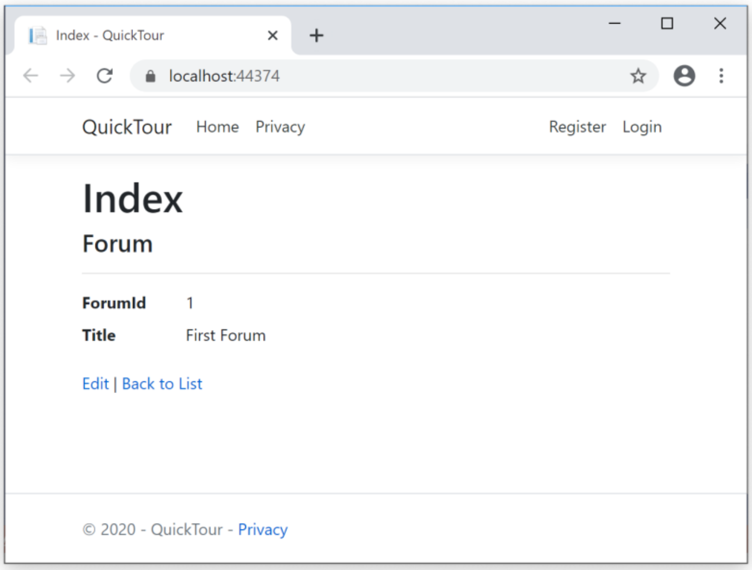
14

Modify the index.cshtml file as follows:

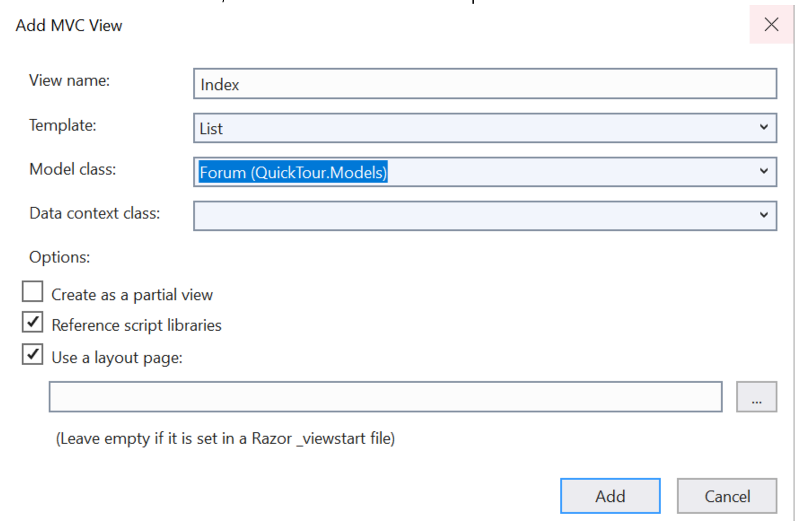
```
@using QuickTour.Models
@model Forum

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>Forum with ID @Model.ForumId is @Model.Title</p>
```

Press F5, and check this still does the same as before.

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 .)
I'm sure you will agree that this is a far better way of writing code!

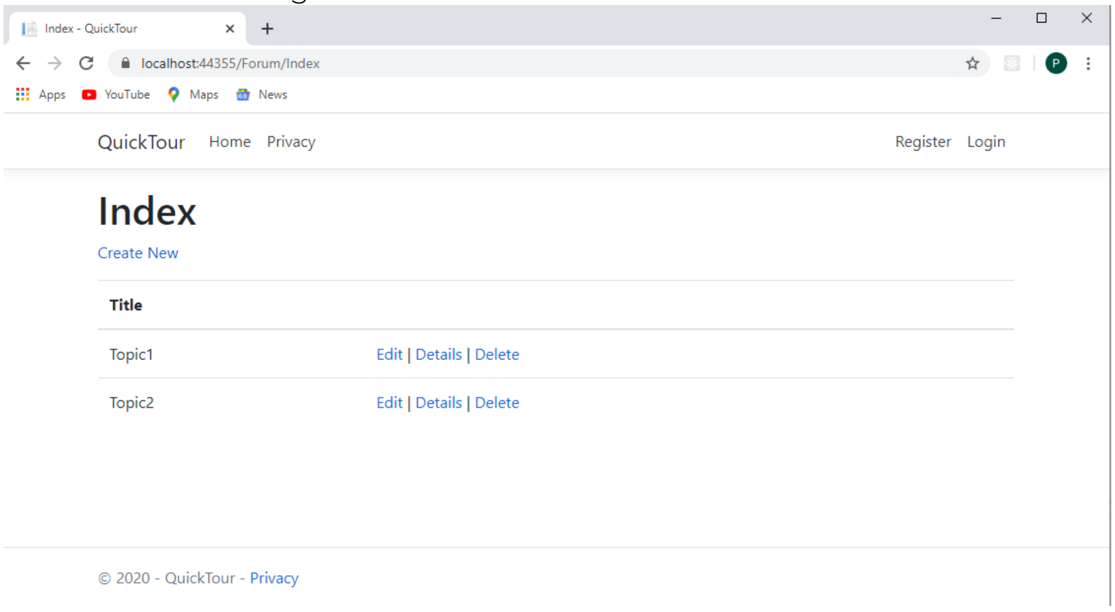
15	We can ask Visual Studio to build a view for us, which will display the data in a model. Let's do that. Switch back to the ForumController, and right-click on the Index method. Choose Add View. This time, modify the Template option to "Details", and then change the Model option to "Forum":  Add MVC View View name: Index Template: Details Model class: Forum (QuickTour.Models) Data context class: Options: ☐ Create as a partial view ☐ Reference script libraries ☑ Use a layout page: (Leave empty if it is set in a Razor _viewstart file) 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:  Index - QuickTour localhost:44374 Index Forum ForumId 1 Title First Forum © 2020 - QuickTour - Privacy

16	What if we need to display more than one Forum?
	Edit the Index 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	Place the cursor in the Index() method, and once again right-click on View > Add View. This time, select a List template based on the Forum class:  Add MVC View View name: Index Template: List Model class: Forum (QuickTour.Models) Data context class: 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 As before, select Yes when asked to confirm you want to replace the existing view.

18	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.
	It also contains a <table> (there are mixed views on whether it should use a table!), and within that table are two lines which display the Forum id:
	`@Html.DisplayNameFor(model => model.ForumId)`
	and
	`@Html.DisplayFor(modelItem => item.ForumId)`
	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.
	
	Go to Models > Forum.cs and add this attribute to suppress this property when scaffolding a view (and yes, resolve the namespace):
	`[ScaffoldColumn(false)]`
	`public int ForumId { get; set; }`
	Repeat the operation of creating a View (overwriting the current Index.cshtml). Note that the lines which show the Id are no longer included.

19 F5. You should now get this:



Index

Create New

Title	
Topic1	Edit | Details | Delete
Topic2	Edit | Details | Delete

© 2020 - QuickTour - Privacy

20 Let's make our data a little more interesting. Add a new class in the Models folder:

```
    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; }
    }
```

And modify the Forum class so that it contains many Threads:

```
    public class Forum
    {
        [ScaffoldColumn(false)]
        public int ForumId { get; set; }
        public string Title { get; set; }
        public virtual ICollection<Thread> Threads { get; set; }
    }
```

21 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:

```
    public class MockForumContext
    {
        public IEnumerable<Forum> GetForums()
        {
            List<Thread> topic1Threads = new List<Thread>() {
                new Thread() {
                    Title = "Thread 1",
                    UserName = "User 1",
                    DateCreated = new DateTime(2020, 1, 28)
                },
                new Thread() {
                    Title = "Thread 2",
                    UserName = "User 2",
                    DateCreated = new DateTime(2020, 1, 31)
                }
            };

            return new List<Forum>() {
                new Forum() { Title = "Topic1", Threads = topic1Threads },
                new Forum() { Title = "Topic2", Threads = new List<Thread>() }
            };
        }
    }
```

Modify the ForumController class to use this new method:

```
        public IActionResult Index()
        {
            MockForumContext mockForum = new MockForumContext();
            IEnumerable<Forum> forums = mockForum.GetForums();
            return View(forums);
        }
```

22	We'd like to modify the view to include two extra columns: The number of threads in the forum, and The date of the latest thread in the forum Where should we put the code to find these data? We have several options: 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 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 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. 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. 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.
23	So, start by removing the [ScaffoldColumn] attribute from the Models.Forum class

24	Now, add a folder to the project, called ViewModels.
	In that folder, create a new class called ForumViewModel:
	`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; }`
	`}`
	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.

25 Add two static methods to the ForumViewModel class, which are used to create the view-models:

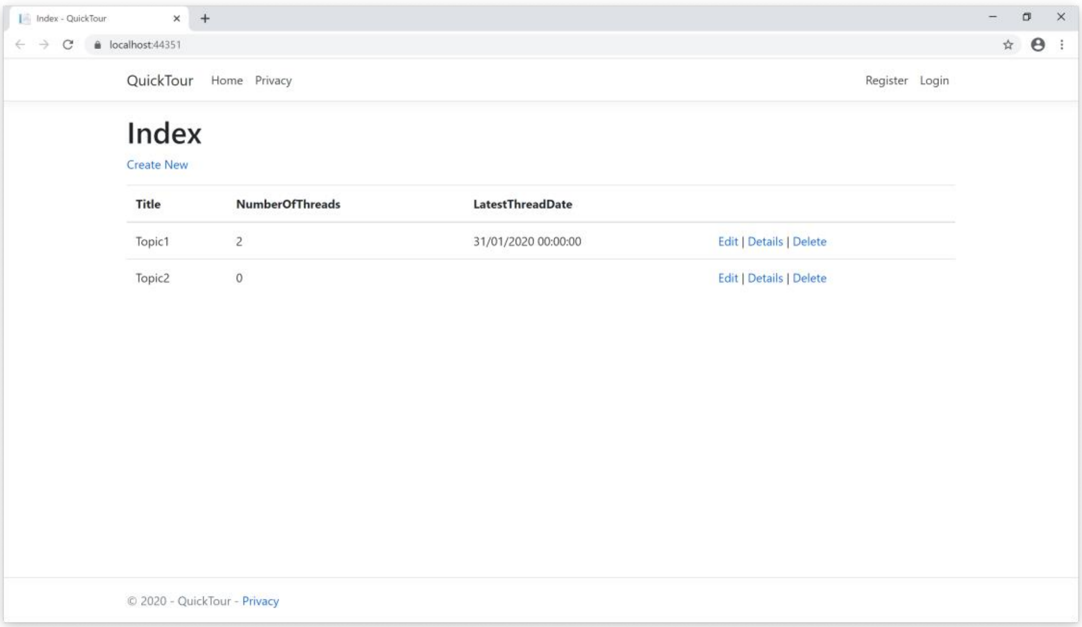
```
        public static ForumViewModel FromForum(Forum forum)
        {
            return new ForumViewModel
            {
                ForumId = forum.ForumId,
                Title = forum.Title,
                NumberOfThreads = forum.Threads.Count,
                // DefaultIfEmpty() and ?. ensure this gives null if 0 threads
                LatestThreadDate = forum.Threads.DefaultIfEmpty().Max(t =>
t?.DateCreated)
            };
        }

        public static IEnumerable<ForumViewModel> FromForums
                                                     (IEnumerable<Forum> forums)
        {
            return forums.Select(f => FromForum(f));
        }
```

26 In the controller, change the line that returns the view so that the view receives a collection of view-models instead of models:

```
        public IActionResult Index()
        {
            ...
            return View(ForumViewModel.FromForums(forums));
        }
```

27 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.

28 Press F5, and check the results:



Index

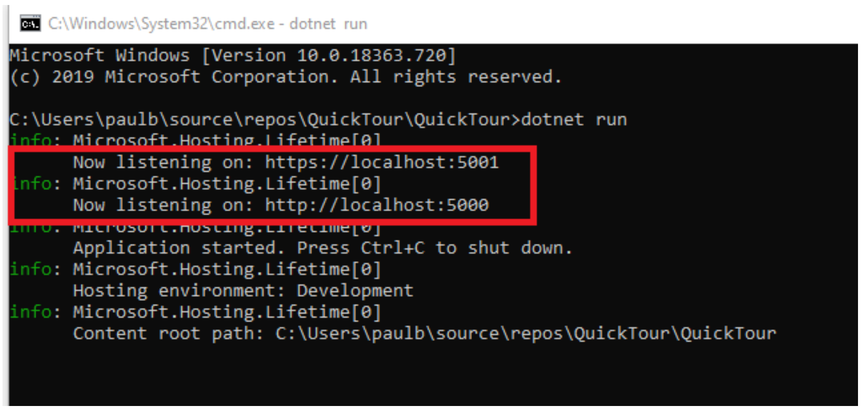
Create New

Title	NumberOfThreads	LatestThreadDate	
Topic1	2	31/01/2020 00:00:00	Edit | Details | Delete
Topic2	0		Edit | Details | Delete

© 2020 - QuickTour - Privacy

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.

29 Now let’s run this *completely* independently of IIS Express
Right-click the project and select “Open Folder in File Explorer” from the menu near the bottom
In the address bar, type ‘cmd’ and carriage-return
A command window appears at this folder
Enter dotnet run



```
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\paulb\source\repos\QuickTour\QuickTour>dotnet run
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\paulb\source\repos\QuickTour\QuickTour
```

It is now listening on the above ports.
Start a Web browser and navigate to either http://localhost:5000 or https://localhost:5001 (note that many modern web browsers are configured to give warnings if you visit sites beginning http:// without the “s” – you may need to skip past these warnings, often by clicking “Advanced”)
Ctrl-C to stop the application

30 Not only does your web page appear, but trace information also appears in the command window. We can add our own trace information. Add the following to the top of the ForumController class:

```
private readonly ILogger<ForumController> _logger;

public ForumController(ILogger<ForumController> logger)
{
    _logger = logger;
}
```

Then, add some logging code to the end of the Index method:

```
public IActionResult Index()
{
    _logger.LogInformation("In the Forums Index() method <=======");

    MockForumContext mockForum = new MockForumContext();
    IEnumerable<Forum> forums = mockForum.GetForums();

    _logger.LogDebug($"Number of forums: {forums.Count()}");
    return View(ForumViewModel.FromForums(forums));
}
```

Use “dotnet run” to start your web server, and then visit your web site in a web browser. Your trace information is displayed in the console window! Use Ctrl-C to stop the web server

31 But what about the debug information, showing the number of forums?
Open appsettings.json, and add the line highlighted below:

```
{
  "ConnectionStrings": {...},
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information",
      "QuickTour.Controllers.ForumController":  "Debug"
    }
  },
  "AllowedHosts": "*"
}
```

Once again, start the web server and visit your web site. Now you can see debug information in the console window too! Stop your web server.

It's not a bad idea to get into the habit of logging debug information in this way. It won't be displayed unless you turn debugging on for the particular class (or namespace) in appsettings.json – something which you can do when you need to debug a problem.

32 You might be looking at the logging code above, and wondering how it works. The ForumController class requires a constructor parameter which is a logger – but where does the logger come from?
The answer is dependency injection – something which we will investigate fully in the next chapter.
Look in the file Program.cs. You will see this line:

```
        Host.CreateDefaultBuilder(args)
```

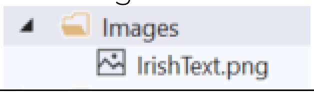
This is the line which allows MVC to create an ILogger to give to its controllers. You can alter the types of loggers as follows:

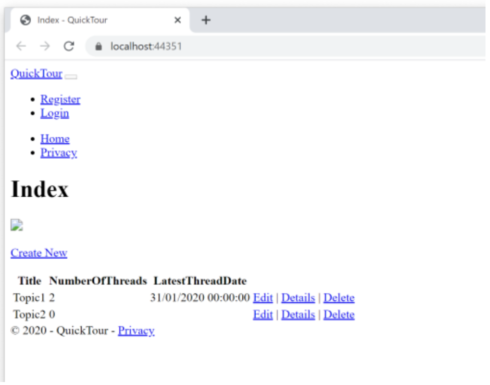
```
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureLogging(logging =>
                {
                    logging.ClearProviders();
                    logging.AddConsole();
                })
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
```

This has resulted in the default logging providers all being removed, and then only a single logging provider (the console logger) being added back again. We won't notice any difference when we run the program again, because we've only been using the console logger anyway.

33 You might've wondered why its running on the port that it is – open Properties/launchsettings.json and you will see the port settings

34 Create an Images folder in the project. In the Assets folder is a file; drag this file onto the Images folder.



35	Add this line just under the <h1> header of Views/Forum/Index.cshtml:
	`    <img src="/Images/IrishText.png" />`
	Run the website – the picture doesn’t appear
	Drag the images folder into wwwroot & run again. Now it does appear.
	Go to Startup.cs and comment out:
	`app.UseHttpsRedirection();`
	`//app.UseStaticFiles();`
	
	`app.UseRouting();`
	Run the site again, and force the browser to refresh all its cached files (in most web browsers you can do this using Ctrl-F5 – an alternative is to open in a different browser that hasn’t cached the information). You should get
	
	Index - QuickTour
	localhost:44351
	QuickTour
	• Register
	• Login
	• Home
	• Privacy
	Index
	Create New
	Title NumberOfThreads LatestThreadDate
	Topic1 2 31/01/2020 00:00:00 Edit | Details | Delete
	Topic2 0 Edit | Details | Delete
	© 2020 - QuickTour - Privacy
	This looks the way it does because the CSS has not been loaded, and neither has the image. Static files have to be explicitly enabled for download and these *only* come from under the wwwroot folder.
	Remove the comment-out

36	You might have wondered where the navigation bar at the top came from – look in Views/Shared/_Layout.cshtml and you should recognize e.g. the title, and contents of the <nav> tag.
37	How does it know to use this particular page? Open Views/_ViewStart.cshtml

38 On a big project, we are likely to have many controllers, view-models and views all related to each other but not particularly related to other groups of controllers/view-models and views (e.g. in an online store the customer/order/payment set will have little to do with the stockControl/ReOrder set or the Register/Login/ForgottenPassword set)

Open up the Areas folder – you will see that the starter has already separated the Identity management.

Right-click the Areas folder and Add > Area called Users. After this process completes, if you are shown a text file (ScaffoldingReadMe.txt), there is no need to follow the instructions, which may instruct you to use the app.UseMvc() method that is now considered to be out of date.

Instead, modify the Configure method in the Startup.cs file as below, to create a route that maps to the new area:

```
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute(
                    name: "default",
                    pattern: "{controller}/{action=Index}/{id?}");
                endpoints.MapControllerRoute(
                    name: "areas",
                    pattern:
                      "{area:exists=Users}/{controller=Forum}/{action=Index}/{id?}");
                endpoints.MapRazorPages();
            });
```

39 Expand the Areas/Users folder and you will see that it has produced a set of suitable folders in which to place related concepts.
Delete the Data folder.
The Models will be for the whole site, but we do need a ViewModels folder here so rename Areas/User/Models to Areas/Users/ViewModels

40 Move the Controllers/ForumController.cs file to Areas/Users/Controllers
Move the ViewModels/ForumViewModel.cs file to Areas/Users/ViewModels
Move the Views/Forum folder to Areas/Users/Views

Copy Views/_ViewStart.cshtml to Areas/Users/Views/_ViewStart.cshtml
Copy Views/_ViewImports.cshtml to Areas/Users/Views/_ViewImports.cshtml
(You can copy files by dragging them while holding the Ctrl key)

You can now delete the empty ViewModels folder at the project level

Edit Areas/Users/Views/_ViewStart.cshtml

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";  // Note the .cshtml suffix!
}
```

41 And lastly, attribute the ForumController to say it is in the Users area:

```
[Area("Users")]
public class ForumController : Controller
{
```

42 Run – it should run again, but now we have the ability to group related files together.
This is likely to be the pattern that you will need for other than small websites.