In-class Lab 05

ASP.NET Core MVC

1 Beginning the lab

1. Create a new project. The target framework should be .NET Core 2.0. Select `File` ▶ `New` ▶ `Project` ▶ `Visual C#` ▶ `Web`. Select `ASP.NET Core Web Application`. Name the application `Razor` and save it in your `/aspnetcore/projects` directory. See figure 1. Click `OK`.

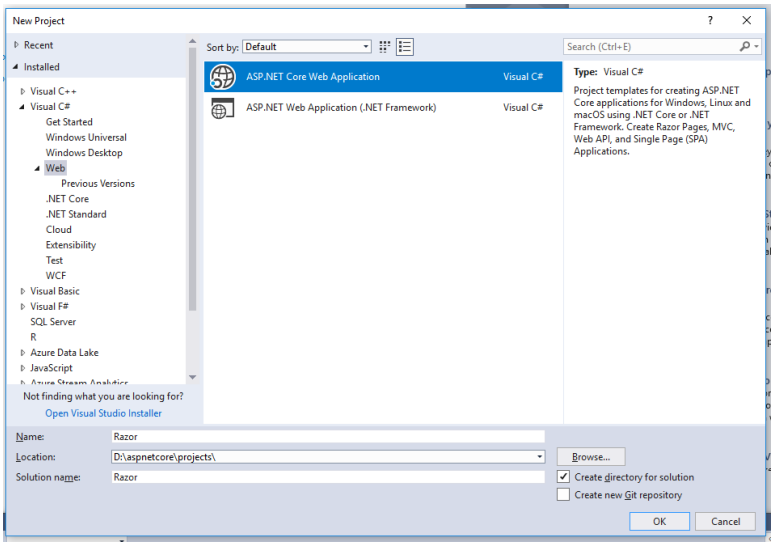

Figure 1: Crate a new ASP.NET Core project named `Razor`

2. Select the `Empty` template with **No Authentication**. See figure 2. Click `OK`.

3. Edit the `Startup.cs` file as shown in listing 1.

Listing 1: Editing class Startup.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Builder;
6  using Microsoft.AspNetCore.Hosting;
7  using Microsoft.AspNetCore.Http;
8  using Microsoft.Extensions.DependencyInjection;
9
10 namespace Razor
11 {
12     public class Startup
13     {
14         public void ConfigureServices(IServiceCollection services)
15         {
16             services.AddMvc();
```

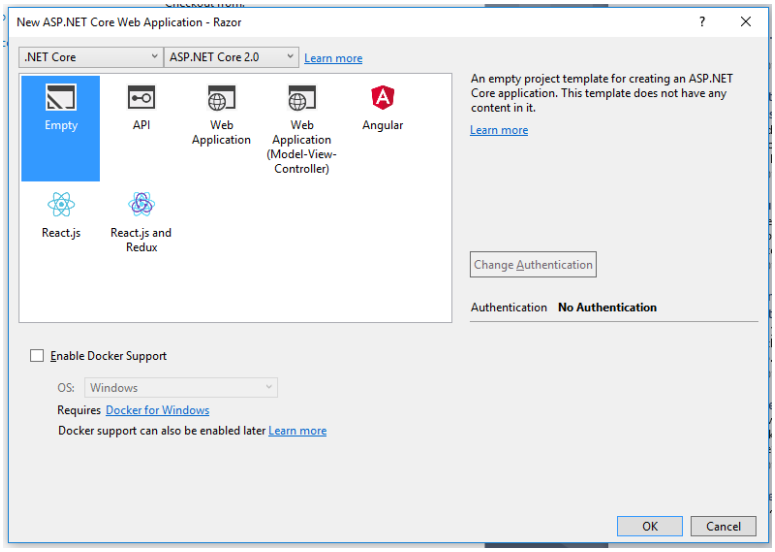

Figure 2: Select the **empty** template

```
17          }
18          public void Configure(IApplicationBuilder app, IHostingEnvironment env)
19          {
20              if (env.IsDevelopment())
21              {
22                  app.UseDeveloperExceptionPage();
23              }
24              app.UseMvcWithDefaultRoute();
25          }
26      }
27  }
```

4. Add a `Models` folder to the project. Right click the `Razor` Project folder and select `Add` ▶ `New Folder`. See figure 3.
5. Name the new folder `Models`. See figure 4.
6. Create a new class in `Models` by right clicking the folder and selecting `Add` ▶ `Class`. See figure 5.
7. Select `Class`, name the class `Product.cs`, and click `Add`. See figure 6.
8. Edit the `Product` class file as shown in listing 2.

Listing 2: Class `Product`

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5
6   namespace Razor.Models
7   {
8       public class Product
9       {
10          public int ProductID { get; set; }
11          public string Name { get; set; }
12          public string Description { get; set; }
13          public decimal Price { get; set; }
14          public string Category { set; get; }
15      }
```

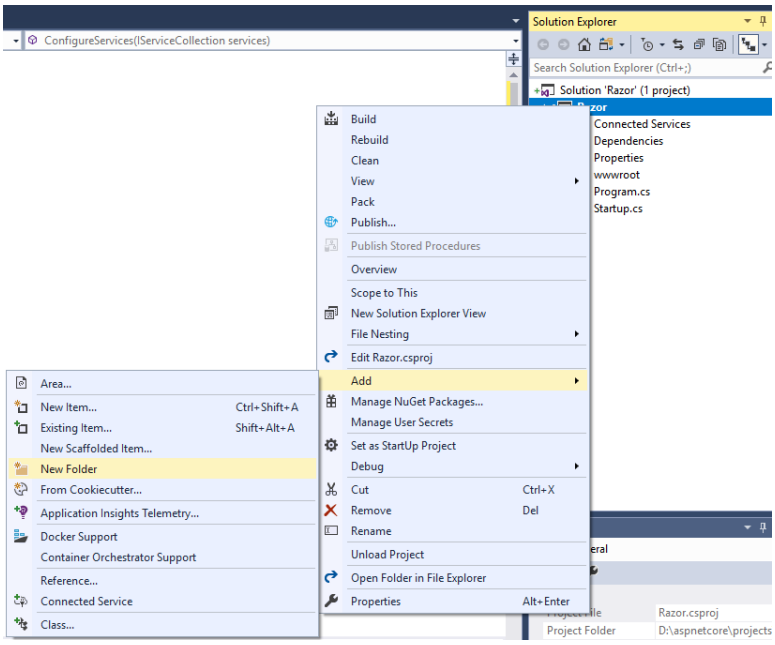

Figure 3: Adding the `Models` folder

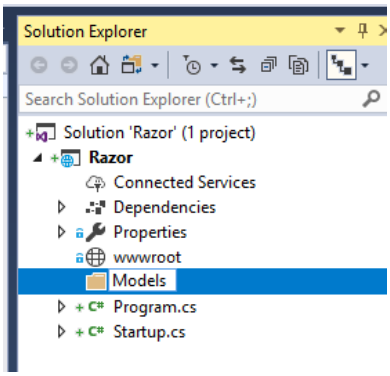

Figure 4: Naming the the folder `Models`

```
16 }
```

9. Add a `Controllers` folder to the project. Right click the `Razor` Project folder and select `Add` ► `New Folder`. Name the new folder `Controllers`. See figure 7.

10. Create a new class in `Controllers` by right clicking the folder and selecting `Add` ► `Controller`. To add the scffolding select `Controller` ► `MVC Controller – Empty` ► `Add`. See figure 8. Name the controller `HomeController` and click `Add`.

11. Edit the `HomeController` class file as shown in listing 3.

Listing 3: Editing the `HomeController`

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
```

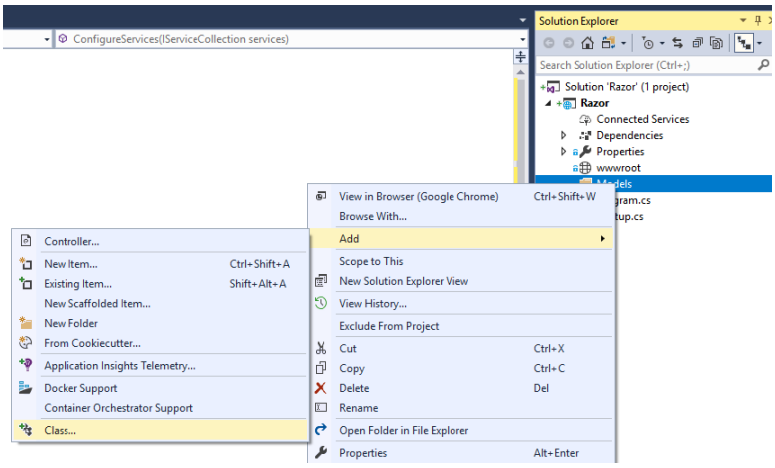

Figure 5: Adding a new class file to `Models`

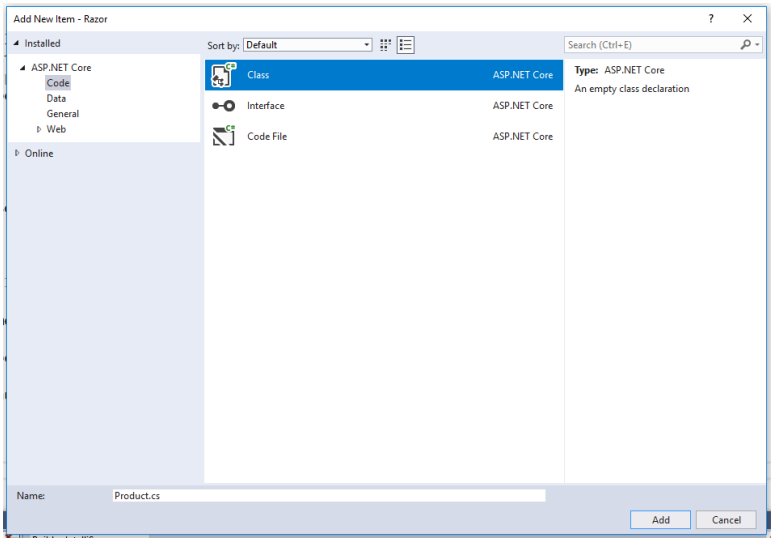

Figure 6: Adding class `Product` to `Models`

```
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6      using Razor.Models;
7
8  namespace Razor.Controllers
9  {
10     public class HomeController : Controller
11     {
12         public ViewResult Index()
13         {
14             Product myProduct = new Product
15             {
16                 ProductID = 1,
17                 Name = "Kayak",
18                 Description = "A boat for one person",
19                 Category = "Watersports",
20                 Price = 275M
21             };
```

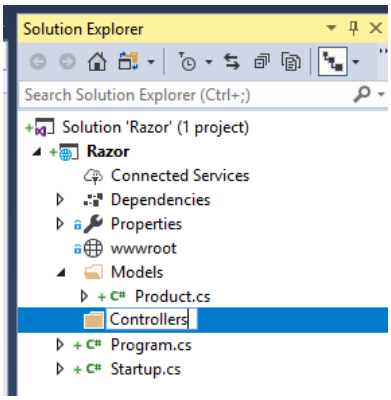

Figure 7: Adding a `Controllers` folder

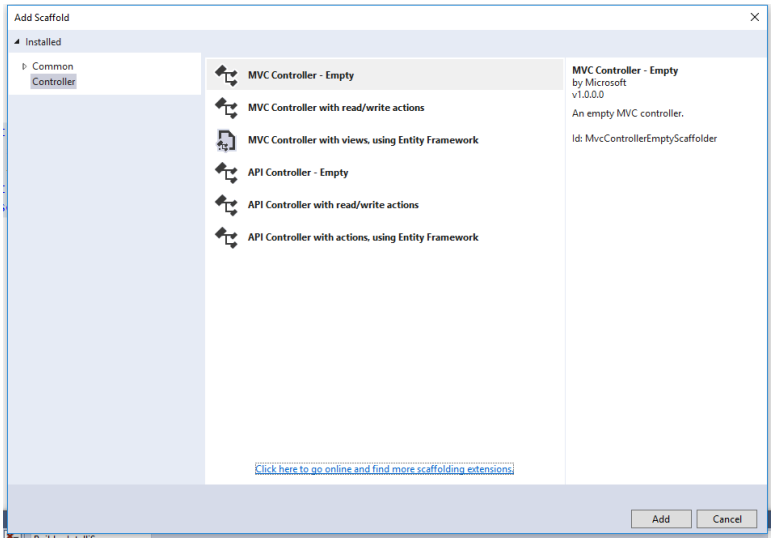

Figure 8: Adding a controller

```
22              return View(myProduct);
23          }
24  }
```

12. In your `Razor` project, create a new folder named `Views`. *Within* that folder create a subfolder named `Home`. See figure 9.

13. In
    `Views`
    `Home`
    , create a view. Right click on `Home` and select `Add` ► `View`. Name the view `Index` and deselect the **Use a layout page:**. See figure 10.

14. Edit the `Index.cshtml` file as shown in listing 4.

Listing 4: Editing the `Index` view

```
1  @{
2      Layout = null;
```

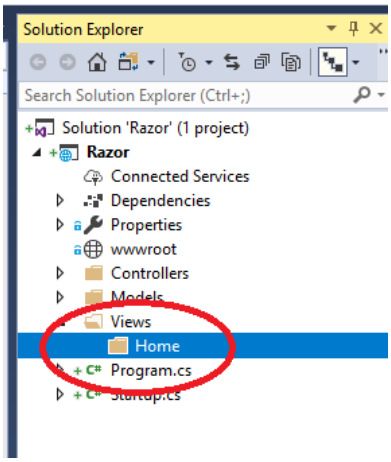

Figure 9: Creating `/Views/Home/`

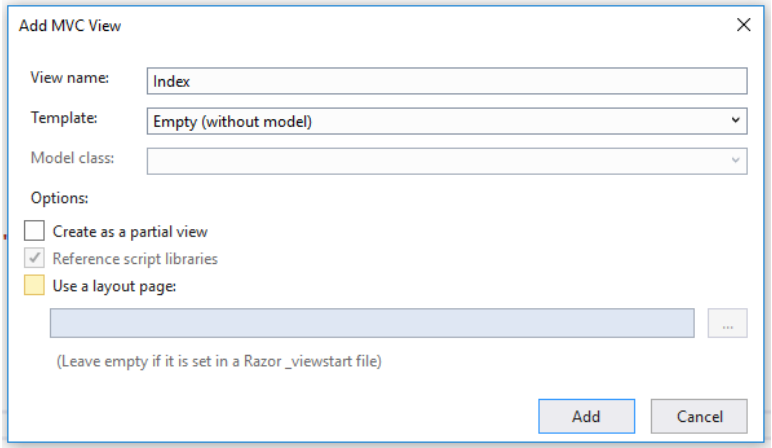

Figure 10: Adding a view named `Index`

```
3  }
4
5  <!DOCTYPE html>
6
7  <html>
8  <head>
9      <meta name="viewport" content="width=device-width" />
10     <title>Index</title>
11 </head>
12 <body>
13     <h1>Content will go here</h1>
14 </body>
15 </html>
```

15. Start your application without debugging. Correct your errors, if any.

2 Working with the Model object

16. Edit `Index.cshtml` as in listing 5. Start without debugging. After you exmine the result, close the browser window.

Listing 5: Adding a model to a view

```
1  @model Razor.Models.Product
2  @{
3      Layout = null;
4  }
5
6  <!DOCTYPE html>
7
8  <html>
9  <head>
10     <meta name="viewport" content="width=device-width" />
11     <title>Index</title>
12 </head>
13 <body>
14     <h1>Content will go here</h1>
15     @Model.Name
16 </body>
17 </html>
```

17. In order to aviodd using the fully qualified object name to access object properties, add a View Inports file. Right click the `views` folder and select `New` ▶ `New Item`. Select `Web` ▶ `Razor View Imports` template. Name the file `_ViewImports.cshtml` — see figure 11. Click `Add`.

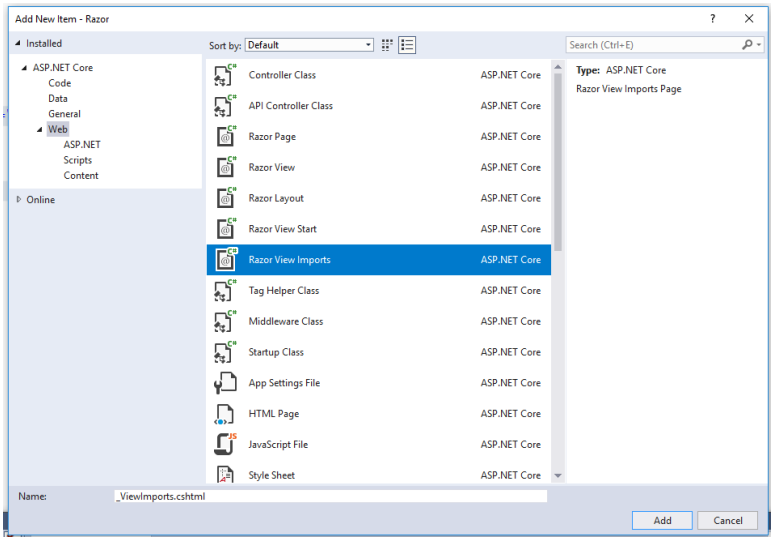

Figure 11: Adding `_ViewImports.cshtml`

18. Add **`@using Razor.Models`** to the `_ViewImports.cshtml` file. Edit the `Index.cshtml` View file by changing the top line to **`@model Product`** Stat wiithout debugging and exmine the result.

19. To add a View Layout file, first create a `/Views/Shared/` folder by right clicking the `Views` folder and selecting `Add` ▶ `New Folder`. Name the folder `Shared`. See figure ??.

20. Add a `_BasicLayout.cshtml` file to the `/Views/Shared/` folder by right clicking the `Shared` folder and selecting `Add` ▶ `New Item`. Selet `Web` ▶ `Razor Layout` and name the file `_BasicLayout.cshtml`. See figure 13.

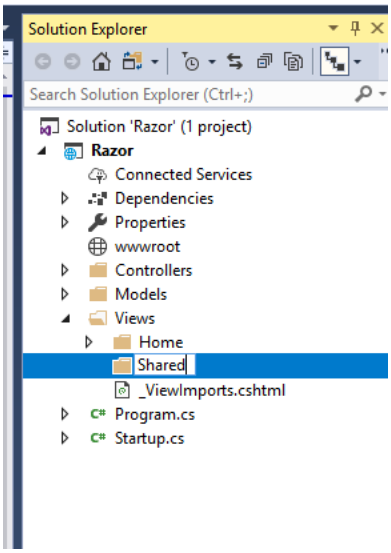

Figure 12: Creating the `/Views/Shared/` folder

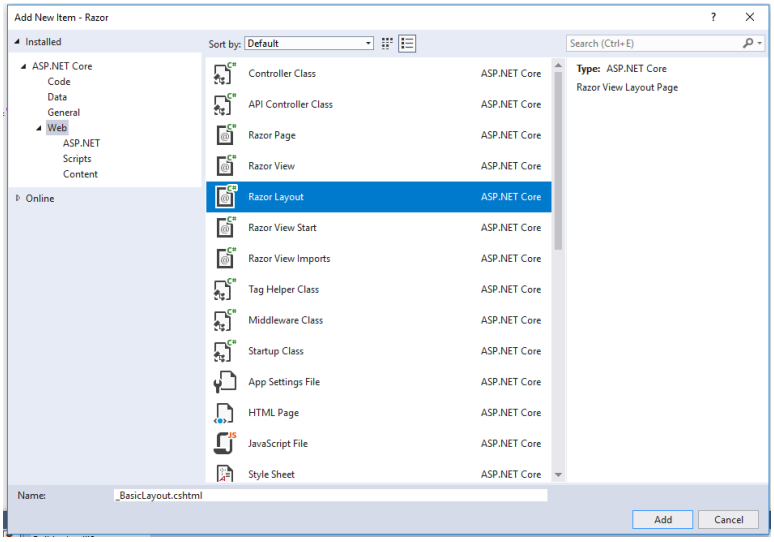

Figure 13: Adding `_BasicLayout.cshtml`

21. Edit the contents of `_BasicLayout.cshtml` so that it looks like listing 6.

Listing 6: Contents of `_BasicLayout.cshtml`

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    <style>
        #mainDiv {
            padding:  20px;
            border:  solid medium black;
            font-size:  20pt
        }
```

```
12      </style>
13  </head>
14  <body>
15      <h1>Product Information</h1>
16      <div id="mainDiv">
17          @RenderBody()
18      </div>
19  </body>
20  </html>
```

22. Edit the contents of `Index.cshtml` so that it matches listing 7. Start without debugging and exmine the result. Close the browser window.

Listing 7: Ediiting `Index.cshtml`

```
1  @model Product
2
3  @{
4      Layout = "_BasicLayout";
5      ViewBag.Title = "Product Name";
6  }
7
8  Product Name: @Model.Name
```

23. To add a `_ViewStart.cshtml` file, right click the `Views` folder and select `Add` ▶ `New Item`. Select `Web` ▶ `Razor View Start` and name the file `_ViewStart.cshtml`. Name the file `_ViewStart.cshtml`. Click `Add`. See figure 14.

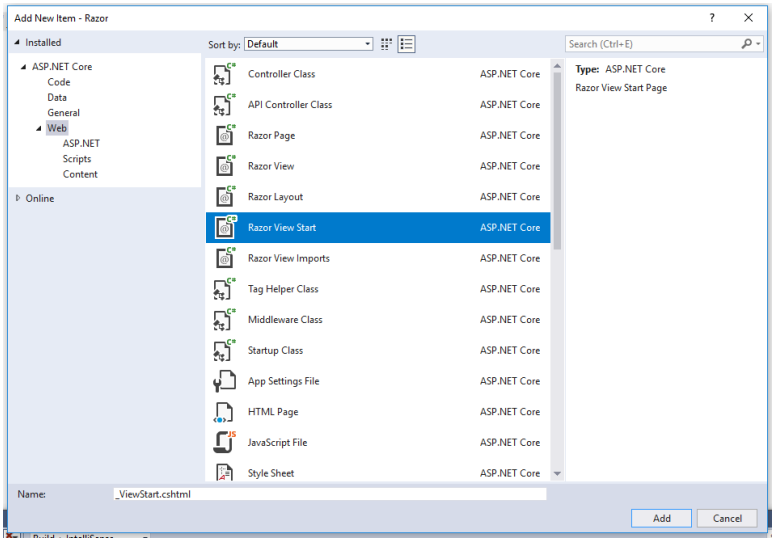

Figure 14: The `_ViewStart.cshtml` file

24. Edit the `_ViewStart.cshtml` file as shown in listing 8. Then, remove the line **`Layout = "_BasicLayout";`** from `Index.cshtml`. Start without debugging. When you are finished, close the browssr window.

Listing 8: Editing the `_ViewStart.cshtml` file

```
1  @{
2      Layout = "_BasicLayout";
3  }
```

3 Using Razor expressions

25. Edit `Index.cshtml` to match listing 9. Start without debugging and examine the result.

Listing 9: Editing `Index.cshtml`

```
@model Product

@{
    ViewBag.Title = "Product Name";
}

<p>
    Product Name: @Model.Name
</p>
<p>
    Product Price:  @($"{Model.Price}")
</p>
```

26. Add this line of code to `HomeCotroller.cs`: `cd`**`ViewBag.StockLevel = 2;`** just above the `return View(myProduct);` statement.

27. Add this line of code to `Index.cshtm`: `cd`**`<p>Stock Level:  @ViewBag.StockLevel</p>`** as the last line of the file. Start without debugging and exsmine the result.

28. Edit `Index.cshtml` as shown in listing 10. Start without debugging and look View Page Source. What do you see? Close the browset window.

Listing 10: Adding attributes

```
@model Product

@{
    ViewBag.Title = "Product Name";
}
<div data-productid="@Model.ProductID" data-stocklevel="@ViewBag.StockLevel">
    <p>Product Name: @Model.Name</p>
    <p>Product Price: @($"{Model.Price:C2}")</p>
    <p>Stock Level: @ViewBag.StockLevel</p>
</div>
```

29. To illustrate how to use `switch` statements with Razor, edit `Index.cshtml` as shown in listing 11. Start without debugging and view the result.

Listing 11: Using a `switch` statement with Razor

```
@model Product

@{
    ViewBag.Title = "Product Name";
}
<div data-productid="@Model.ProductID" data-stocklevel="@ViewBag.StockLevel">
    <p>Product Name: @Model.Name</p>
    <p>Product Price: @($"{Model.Price:C2}")</p>
    <p>Stock Level:
        @switch (ViewBag.StockLevel)
        {
            case 0:
                @:Out of stock
                break;
            case 1:
            case 2:
            case 3:
                <b>Low Stock:  (@ViewBag.StockLevel)</b>
                break;
```

```
            default:
                @:  @ViewBag.Stocllevel in stock
                break;
        }
    </p>
</div>
```

30. To illustrate how to use `if-else` statements with Razor, edit `Index.cshtml` as shown in listing 12. Start without debugging and view the result.

Listing 12: Using a `if-else` statement with Razor

```
@model Product

@{
    ViewBag.Title = "Product Name";
}
<div data-productid="@Model.ProductID" data-stocklevel="@ViewBag.StockLevel">
    <p>Product Name: @Model.Name</p>
    <p>Product Price: @($"{Model.Price:C2}")</p>
    <p>Stock Level:
        @if (ViewBag.StockLevel == 0)
        {
            @:Out of stock
        }
        else if (ViewBag.StockLevel > 0 && ViewBag.StockLevel <= 3)
        {
            <b> Low Stock:  (@ViewBag.StockLevel) </b>
        }
        else
        {
            @:  @ViewBag.StockLevel in stock
        }
    </p>
</div>
```

31. To use an enumerating array wiith Razor, first edit `HomeController.cs` as shown in listing 13.

Listing 13: Edit to `HomeController.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Razor.Models;

namespace Razor.Controllers
{
    public class HomeController : Controller
    {
        public IActionResult Index()
        {
            Product[] array = {
                new Product {Name = "Kayak", Price = 275M},
                new Product {Name = "Lifejacket", Price = 48.95M},
                new Product {Name = "Soccer ball", Price = 19.50M},
                new Product {Name = "Corner flag", Price = 34.95M}
            };
            return View(array);
        }
    }
}
```

32. Then, edit `Index.cshtml` as shown in listing ??. Start without debugging and examine the result.

Listing 14: Final edit to `Index.cshtml`

```
@model Product[]

@{
    ViewBag.Title = "Product Name";
}

<table>
    <thead>
        <tr><th>Name</th><th>Price</th></tr>
    </thead>
    <tbody>
        @foreach (Product p in Model)
        {
            <tr>
                <td>@p.Name</td>
                <td>@($"{p.Price:C2}")</td>
            </tr>
        }
    </tbody>
</table>
```