In-class Lab 06

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 `WorkingWithVisualStudio` and save it in your `/aspnetcore/projects` directory. See figure 1. Click `OK`.

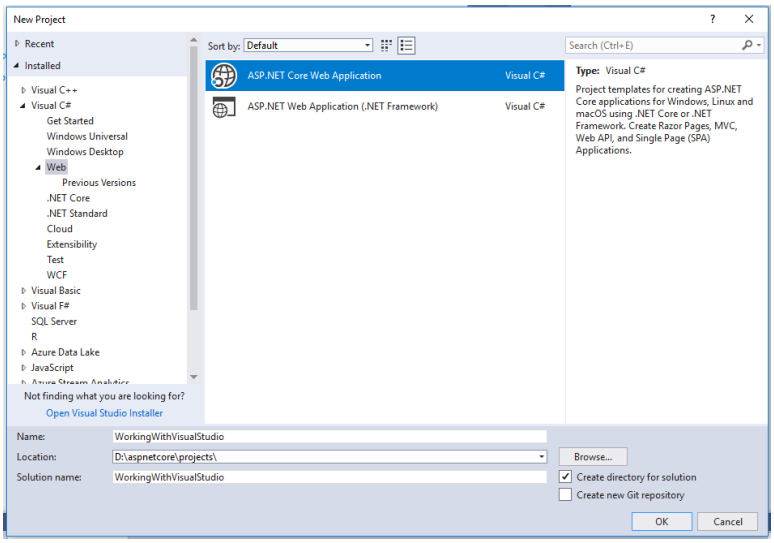

Figure 1: Crate a new ASP.NET Core project named `WorkingWithVisualStudio`

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 WorkingWithVisualStudio
11 {
12     public class Startup
13     {
14         public void ConfigureServices(IServiceCollection services)
15         {
```

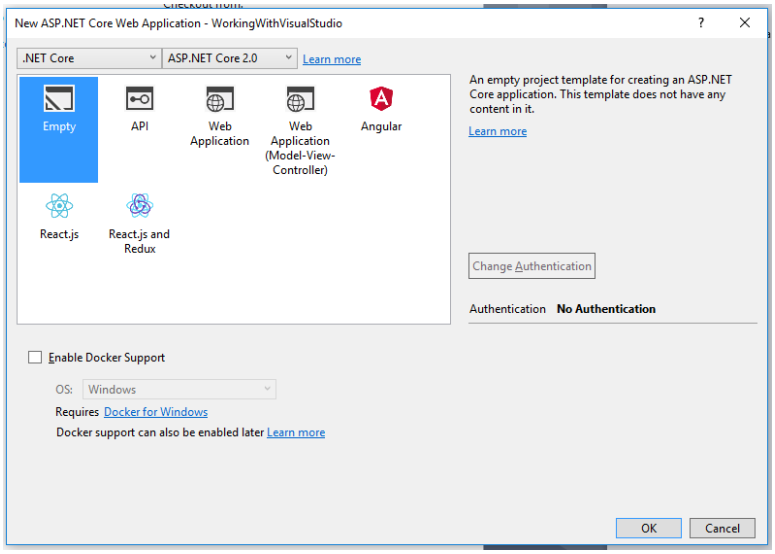

Figure 2: Select the **empty** template

```
16              services.AddMvc();
17          }
18          public void Configure(IApplicationBuilder app, IHostingEnvironment env)
19          {
20              app.UseMvcWithDefaultRoute();
21          }
22      }
23  }
```

4. Add a `Models` folder to the project. Right click the `WorkingWithVisualStudio` Project folder and select `Add` ▶ `New Folder`. Name the New Folder `Models`. See figure 3.

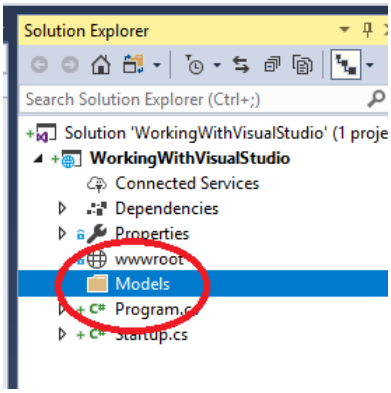

Figure 3: Adding the `Models` folder

5. Create a new class in `Models` by right clicking the folder and selecting `Add` ▶ `Class`. Select `Class`, name the class `Product.cs`, and click `Add`. See figure ??.

6. Edit the `Product` class file as shown in listing 2.

Listing 2: Class `Product`

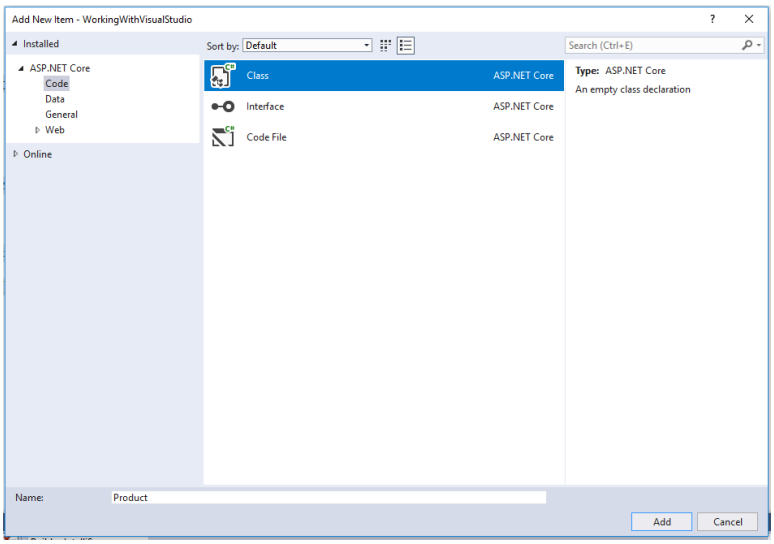

Figure 4: Adding a new class file to `Models` named `Product`

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

7. Create a new class in `Models` by right clicking the folder and selecting `Add` ▶ `Class`. Select `Class`, name the class `SimpleRepository.cs`, and click `Add`. See figure 5.

8. Edit the `SimpleRepository` class file as shown in listing 3.

Listing 3: Class `SimpleRepository`

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace WorkingWithVisualStudio.Models
7  {
8      public class SimpleRepository
9      {
10         private static SimpleRepository sharedRepository = new SimpleRepository();
11         private Dictionary<string, Product> products
12             = new Dictionary<string, Product>();
13
14         public static SimpleRepository SharedRepository => sharedRepository;
15
16         public SimpleRepository()
17         {
18             var initialItems = new[] {
19                 new Product { Name = "Kayak", Price = 275M },
```

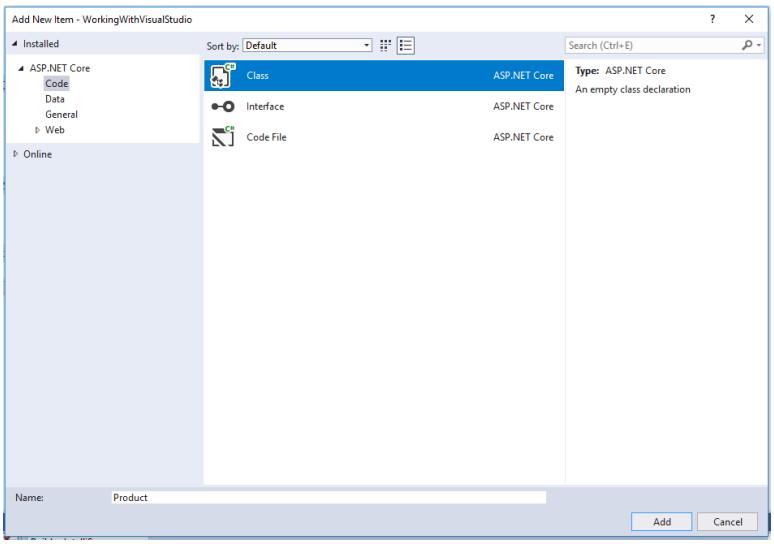

Figure 5: Adding a new class file to `Models` named `SimpleRepository`

```
20                  new Product { Name = "Lifejacket", Price = 48.95M },
21                  new Product { Name = "Soccer ball", Price = 19.50M },
22                  new Product { Name = "Corner flag", Price = 34.95M }
23              };
24              foreach (var p in initialItems)
25              {
26                  AddProduct(p);
27              }
28          }
29
30          public IEnumerable<Product> Products => products.Values;
31
32          public void AddProduct(Product p) => products.Add(p.Name, p);
33      }
34  }
```

9. Add a `Controllers` folder to the project. Right click the `WorkingWithVisualStudio` Project folder and select `Add` ▶ `New Folder`. Name the new folder `Controllers`. See figure ??.

10. Create a new class in `Controllers` by right clicking the folder and selecting `Add` ▶ `Controller`. To add the scaffolding select `Controller` ▶ `MVC Controller - Empty` ▶ `Add`. See figure 7. Name the controller `HomeController` and click `Add`.

11. Edit the `HomeController` class file as shown in listing 4.

Listing 4: Editing the `HomeController`

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Threading.Tasks;
5   using Microsoft.AspNetCore.Mvc;
6   using WorkingWithVisualStudio.Models;
7
8   namespace WorkingWithVisualStudio.Controllers
9   {
10      public class HomeController : Controller
11      {
12
13          public IActionResult Index()
```

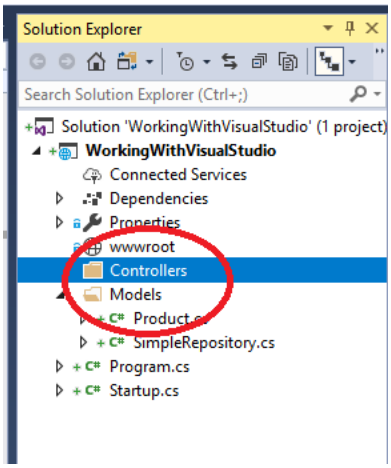

Figure 6: Adding a `Controllers` folder

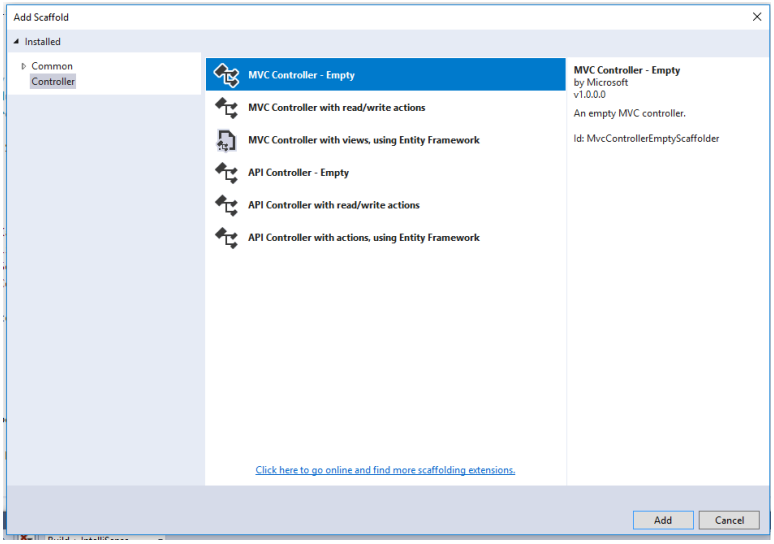

Figure 7: Adding a controller

```
14          => View(SimpleRepository.SharedRepository.Products);
15      }
16  }
```

12. In your `WorkingWithVisualStudio` project, create a new folder named `Views`. *Within* that folder create a subfolder named `Home`. See figure 8. 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:**.

13. Edit the `Index.cshtml` file as shown in listing 5.

Listing 5: Editing the `Index` view

```
1  @model IEnumerable<WorkingWithVisualStudio.Models.Product>
2  @{
3      Layout = null;
4  }
5
```

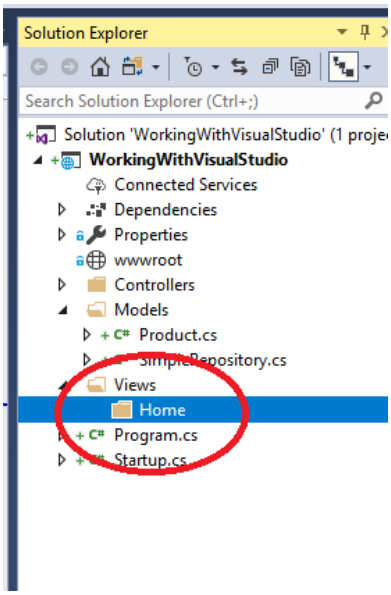

Figure 8: Creating `/Views/Home/`

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>WorkingWithVisualStudio</title>
</head>
<body>
    <table>
        <thead>
            <tr><td>Name</td><td>Price</td></tr>
        </thead>
        <tbody>
            @foreach (var p in Model)
            {
                <tr>
                    <td>@p.Name</td>
                    <td>@p.Price</td>
                </tr>
            }
        </tbody>
    </table>
</body>
</html>
```

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

2 Managing software packages

15. Display the nuGet Package Manager by cliccking `Tools` ▶ `NuGet Package Manager` ▶ `Manage NuGet Packages ...`. See figure 9.

16. Examine the `Installed` tab. Figure 10 shows my NuGet Package Manager screen. Your may be different.

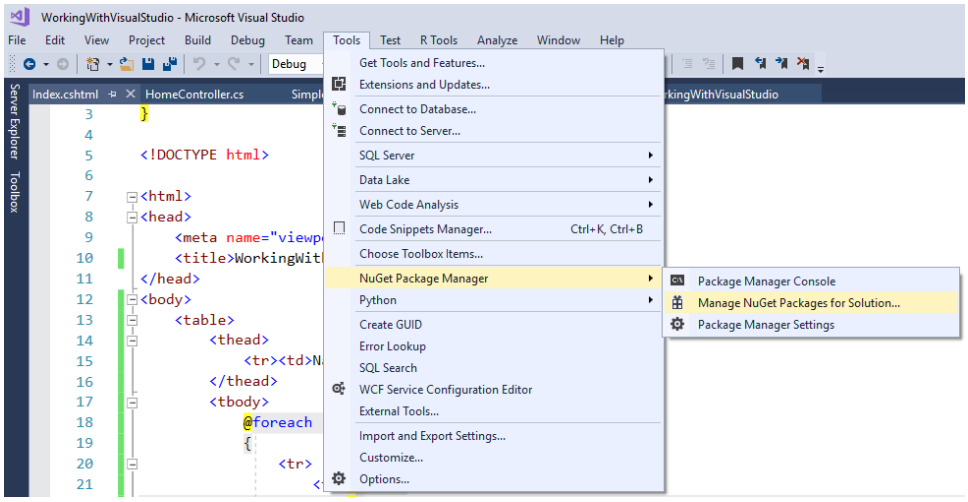

Figure 9: Accessing the NuGet Package Manager

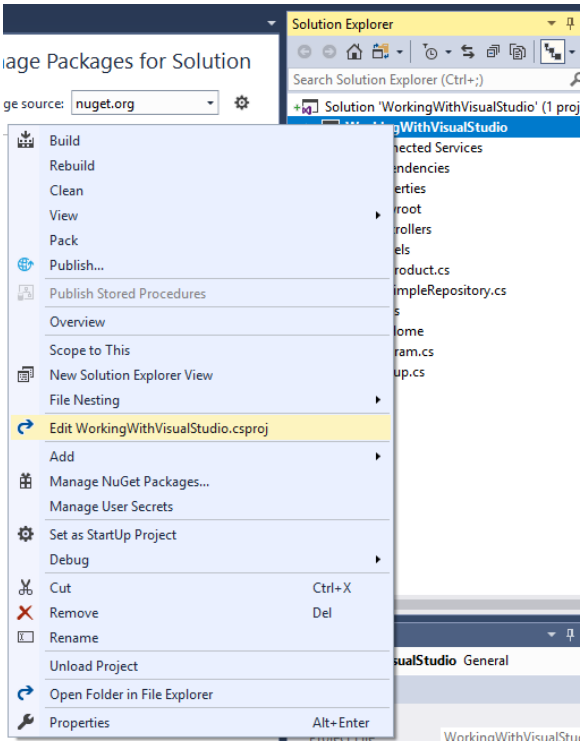

Figure 10: NuGet Package Manager, Installed tab

17. To edit the `WorkingWithVisualStudio.csproj` file, right click the WorkingWithVisualStudio project item and select `Edit WorkingWiithVisualStudio.csproj`. See figure 11. This is an XML file which you should examine and understand.

18. To add a `bower.json` file, follow the following instructions. Microsoft has removed the Bower Configuration File template from Visual Studio and the source for Bower packages has changed. To create the Bower file required by the example projects: Right-click on the project item in the Solution Explorer window, select `Add` ▶ `New Item`, select JSON File from the ASP.NET Core/General category and

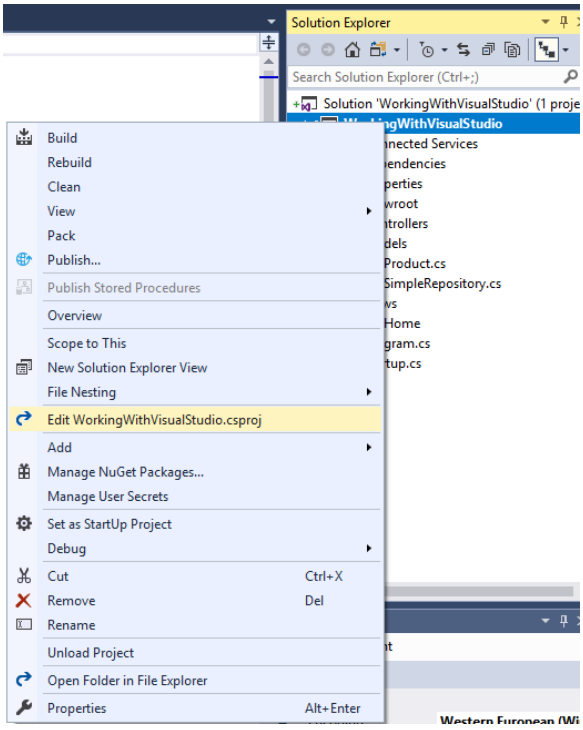

Figure 11: Editing the `csproj` file

set the file name to `.bowerrc` (note the letter r appears twice and the file name begins with a dot (.)), and click the `Add` button. See figure 12.

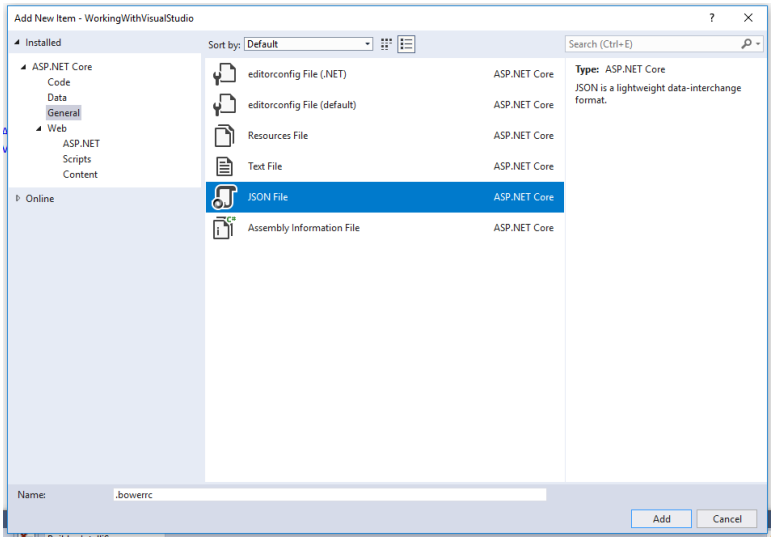

Figure 12: Adding `.bowerrc`

19. Set the contents of the .bowerrc file as follows in listing 6.

Listing 6: Contents of `bowerrc`

```
{
    "directory": "wwwroot/lib",
    "registry": "https://registry.bower.io"
}
```

20. Save the changes. (This is important - you must save the changes to the .bowerrc file before proceeding)

21. Right-click on the project item and create another JSON file, this time called `bower.json`. See figure 13. Click `Add`. Set the contents of the file as shown in listing 7.

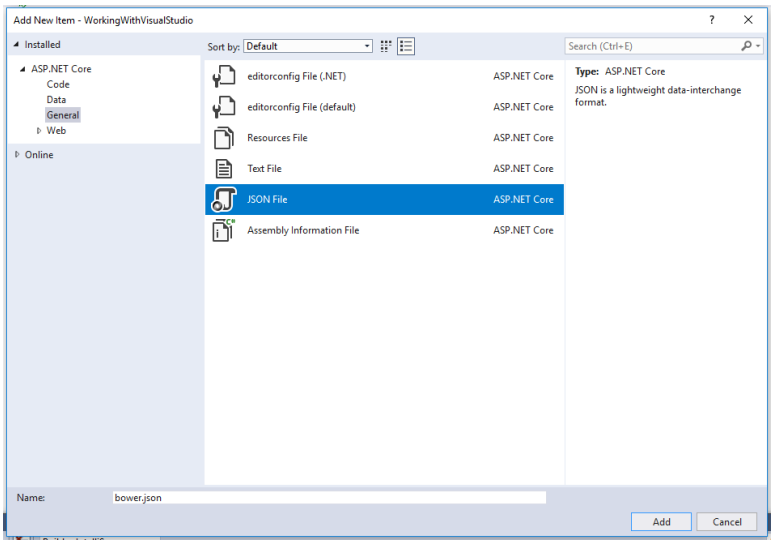

Figure 13: Adding `bower.json`

Listing 7: Contents of `bower.json`

```
{
    {
        "name": "asp.net",
        "private": true,
        "dependencies": {
        "bootstrap": "4.1.*"
    }
}
```

22. Save the changes to the `bower.json` file. Close and re-open the project and Visual Studio will download the Bootstrap package. You can check to see of Bootstrap installed by looking at the installed packages in your Nuget Package Manager.

    If it's not installed, select `Browse`, serch for Bootstrap, select `Project`, and click `Install`. You will have to click through thee install dialog boxes. See figure 14.

23. Start without debugging. Then, change `Index.cshtml` as shown in listing 8. Save the changes to the file. Refresh the page in the browser window. Did you see the changes?

Listing 8: Edits to `Index.cshtnl`

```
<body>
<h3>Products</h3>
    <table>
        <thead>
            <tr><td>Name</td><td>Price</td></tr>
        </thead>
```

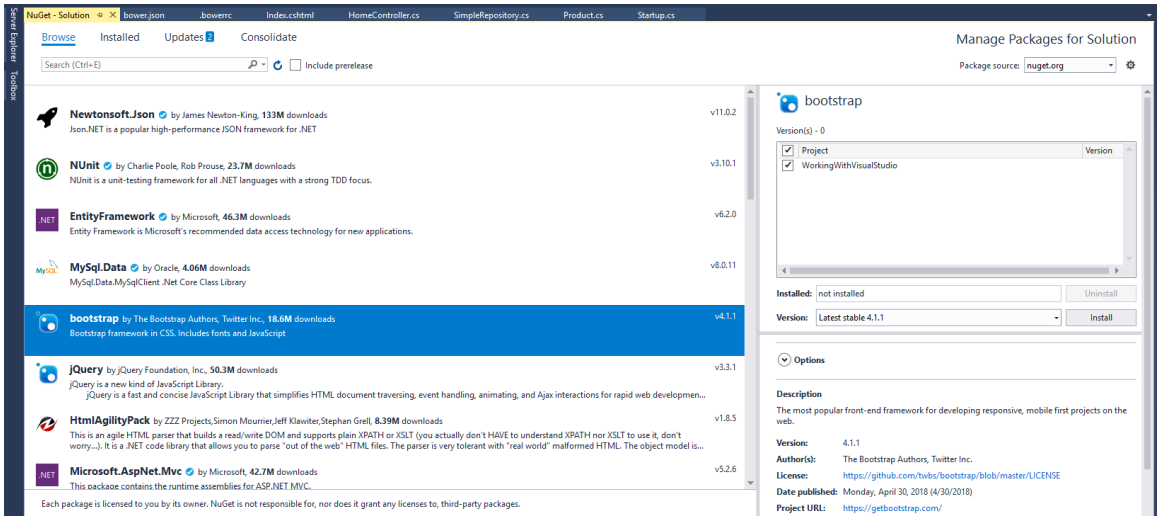

Figure 14: Manually installing Bootstrap

```
7           <tbody>
8               @foreach (var p in Model)
9               {
10                  <tr>
11                  <td> @p.Name </td>
12                  <td> @($"{p.Price:C2}") </td>
13                  </tr>
14                  }
15          </tbody>
16      </table>
17  </body>
```

24. Edit `HomeController.cs` as shown in listing 9. Save the file and then start without debugging. What happened?

Listing 9: Edits to `HomeController.cs`

```
1  public class HomeController : Controller
2  {
3      public IActionResult Index() => View(SimpleRepository.SharedRepository.Products
4      .Where(p => p.Price < 50));
5  }
```

3 Debugging

25. In order to exercise debugging, we first need to introduce an error. Add this line: **`products.Add("Error", null);`** to the `SimpleRepository.cs` model as the last line in the constructor. This attempts to add a null reference to the `Products` dictionary, which will produce an error at run time. Then, attempt to start without debugging. Close the browser window.

26. Add the following line to the `Configure()` method in `Startup.cs`: **`app.UseDeveloperExceptionPage();`**. See listing 10. Start without debugging. You will see the following error page.

```
An unhandled exception occurred while processing the request.
NullReferenceException: Object reference not set to an instance of an object.
WorkingWithVisualStudio.Cotrollers.HomeController+<>c.<Index>b__0_0(Product p)
```

```
    in HomeController.cs, line 13
```

Listing 10: Editing `Startup.cs`

```
1           public void Configure(IApplicationBuilder app, IHostingEnvironment env)
2           {
3               app.UseDeveloperExceptionPage();
4               app.UseMvcWithDefaultRoute();
5           }
6       }
7   }
```

27. Select `Debug` ▶ `Windows` ▶ `Exception Settings` from the Visual Studio menu bar and ensure that `Common Language Runtime Exceptions` is selected. See figure 15.

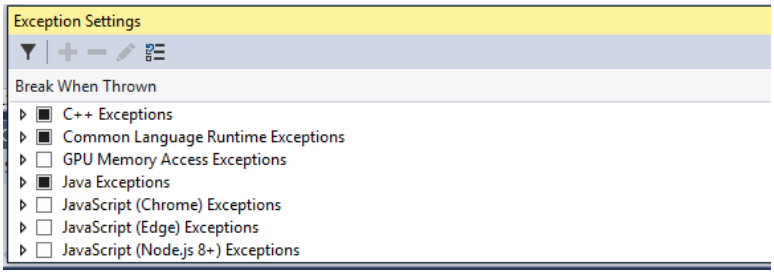

Figure 15: Common Language Runtime Exceptions

28. Set a breakpoint in the class file `SimpleRepository.cs` on the line reading `public void AddProduct(Product p) => products.Add(p.Name, p);`. Right click that line and select tyBreakpoint ▶ Insert Breakpoint. See figure 16.

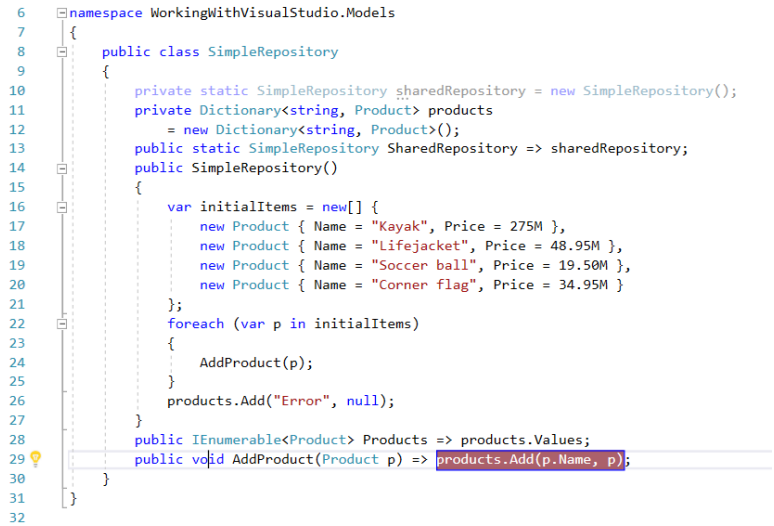

Figure 16: Inserting a breakpoint

29. Start debugging. NOTE: This means start *with* debugging. When execution reaches the breakpoint, execution will stop and you can examine the variable value by hovering over the breakpoint in Visual Studio. See figure 17. To continue, click `Continuue` (see figure 18) until you reach the point in the execution where the error occurs.

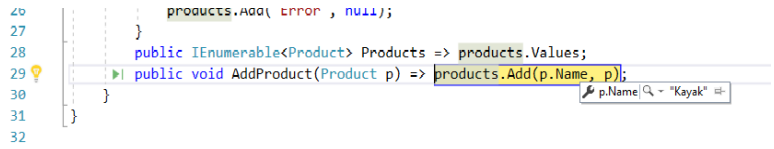

Figure 17: Variable values

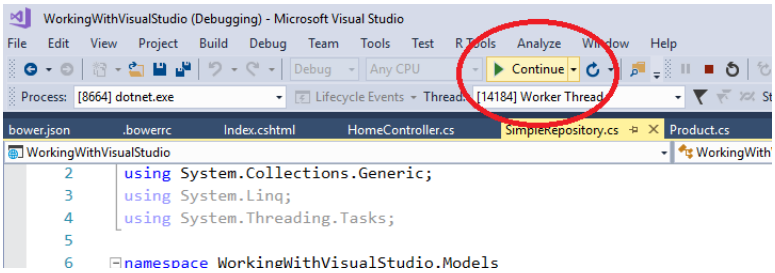

Figure 18: Continue

30. When debugging, you can select `Debug` ► `Windows` ► `Locals` to open a window showing more information about the status of local variables. See figure 19.

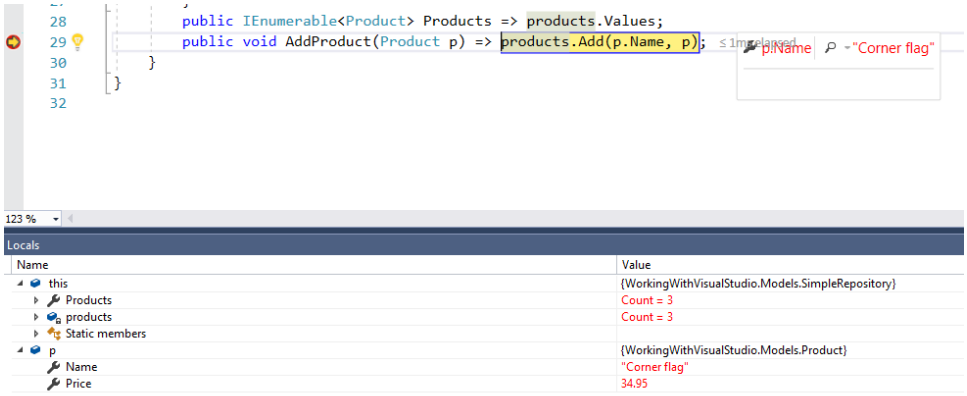

Figure 19: Locals

31. At this point, you can correct the error by deleting the line of code that attempts to set the reference variable to NULL. Alternatively, you can use the null conditional operator to check for null references. Use the second approach by altering the `Index()` method i `HomeController.cs` as shown in listing 11. Make this change and then start witout debugging.

Listing 11: Using the null conditional operator

```
1  public class HomeController : Controller
2      {
3          public IActionResult Index() => View(SimpleRepository.SharedRepository.Products
4          .Where(p => p?.Price < 50));
5      }
```

4 Using browser link

32. Edit `Startup.cs` by adding **`app.UseBrowserLink();`**, as shown in listing 12.

Listing 12: Using browser link

```
1  public void Configure(IApplicationBuilder app, IHostingEnvironment env)
2  {
3      app.UseDeveloperExceptionPage();
4      app.UseBrowserLink();
5      app.UseMvcWithDefaultRoute();
6  }
```

33. Enable Browser Link by clicking on the Reload button for IIS Express. See figure 20.

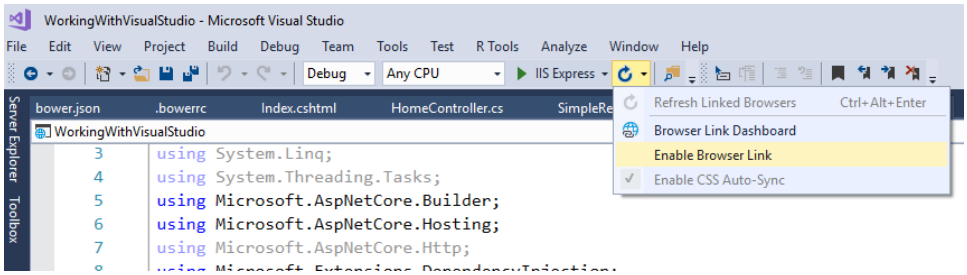

Figure 20: Enable Browser Link

34. Start without debugging and examine the HTML source code. You will see a line to the file similar to the following:

```
<!-- Visual Studio Browser Link -->
<script
    type="text/javascript"
    src="http://localhost:53601/d07d3c508f1d45609e13b799aea26668/browserLink"
    async="async"
    id="__browserLink_initializationData"
    data-requestId="963918e76c9146848c2e68a34431380a"
    data-requestMappingFromServer="False">
</script>
<!-- End Browser Link -->
```

35. Add this line to `Index.cshtml`: **`<p>Request Time:  @DateTime.Now.ToString("HH:mm:ss")</p>`**. The result should be similar to listing 14.

Listing 13: Using browser link

```
1  <body>
2      <h3>Products</h3>
3      <p>Request Time:  @DateTime.Now.ToString("HH:mm:ss")</p>
4      <table>
5          <thead>
6              <tr><td>Name</td><td>Price</td></tr>
7          </thead>
8          <tbody>
9              @foreach (var p in Model)
10             {
```

```
                <tr>
                    <td> @p.Name </td>
                    <td> @($"{p.Price:C2}") </td>
                </tr>
            }
        </tbody>
    </table>
</body>
```

36. Using the IIS Express drop down menu, select `Browse ith ...`, and select the broswers you wish to test with. See figure 21. Start without debugging. Your application should open in the browsers you have selected.

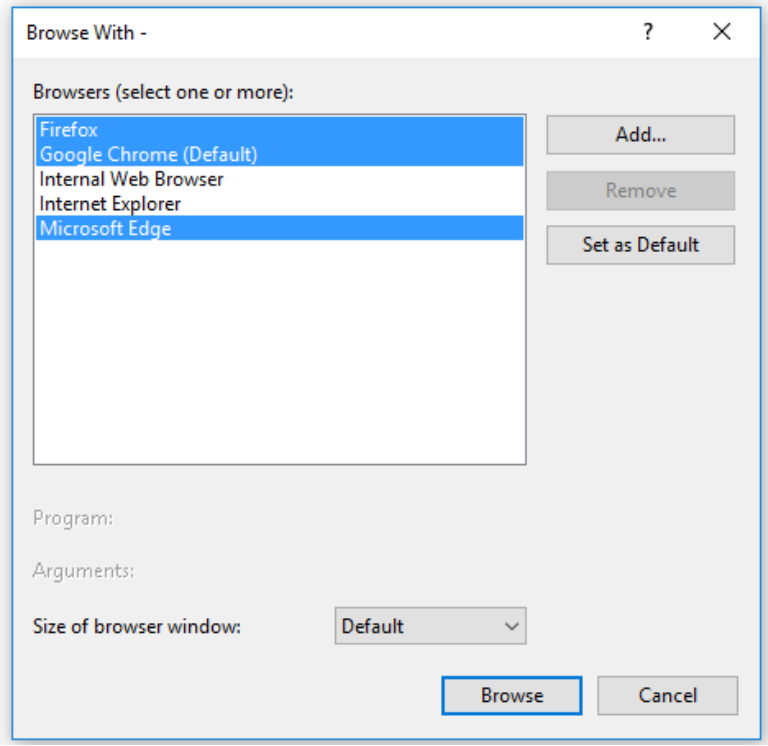

Figure 21: IIS Express Browse With selector

5 Adding static content

37. Edit `Startup.cs` by adding **`app.UseStaticFiles();`**, as shown in listing 14.

Listing 14: Adding static content

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    app.UseDeveloperExceptionPage();
    app.UseBrowserLink();
    app.UseStaticFiles();
    app.UseMvcWithDefaultRoute();
}
```

38. Create a new folder under `wwwroot` and name it `css`. See figure 22.

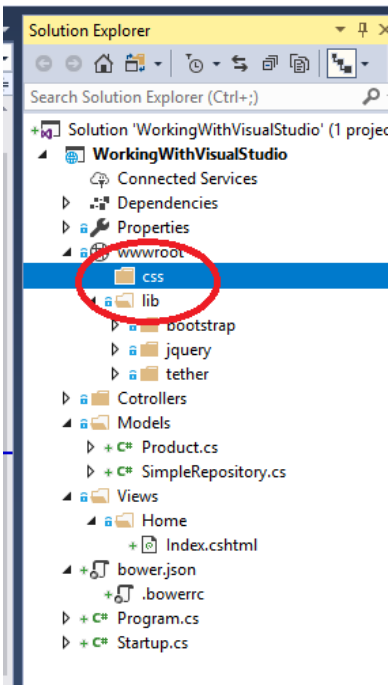

Figure 22: Adding a new folder, `/wwwroot/css/`

39. Add a CSS file named `first.css` by right clicking on the `css` folder and selecting `Add` ▶ `New Item`. Then, select `ASP.NET Core` ▶ `Web` ▶ `Content` ▶ `Style Sheet`. See figure ??.

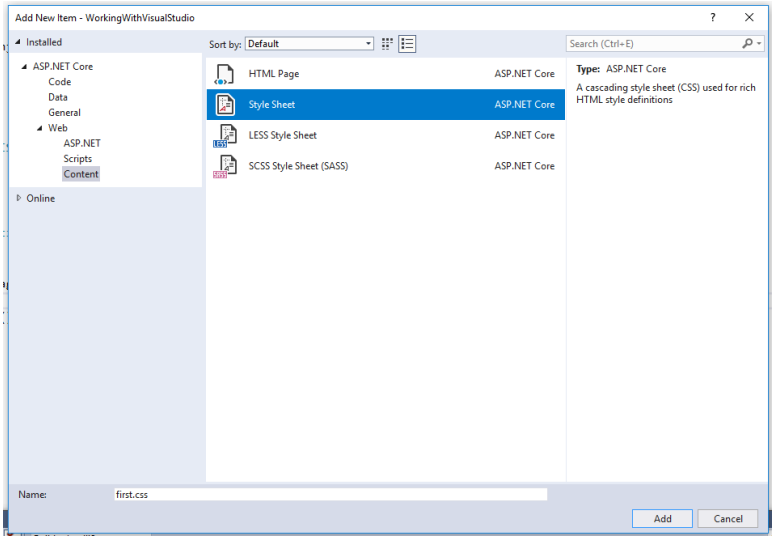

Figure 23:

40. Edit `first.css` as shown in listing 15.

Listing 15: Stylesheet `first.css`

```
1  h3 {
2      font-size: 18pt;
```

```
3       font-family: sans-serif;
4   }
5
6   table, td {
7       border: 2px solid black;
8       border-collapse: collapse;
9       padding: 5px;
10      font-family: sans-serif;
11  }
```

41. In the same way, create another style sheet named `second.css` and edit it as shown in listing 16.

Listing 16: Stylesheet `second.css`

```
1   p {
2       font-family: sans-serif;
3       font-size: 10pt;
4       color: darkgreen;
5       background-color: antiquewhite;
6       border: 1px solid black;
7       padding: 2px;
8   }
```

42. In the same way you created `/wwwroot/css/`, create a new folder ib `wwwroot` named `/wwwroot/js/`.

43. Create a new file named `third.js` in `/wwwroot/js/`. The contents are shown in listing 17

Listing 17: Contents of `third.js`

```
1   document.addEventListener("DOMContentLoaded", function () {
2       var element = document.createElement("p");
3       element.textContent = "This␣is␣the␣element␣from␣the␣(modified)␣third.js␣file";
4       document.querySelector("body").appendChild(element);
5   });
```

44. Create a new file named `fourth.js` in `/wwwroot/js/`. The contents are shown in listing 18

Listing 18: Contents of `fourth.js`

```
1   document.addEventListener("DOMContentLoaded", function () {
2       var element = document.createElement("p");
3       element.textContent = "This␣is␣the␣element␣from␣the␣fourth.js␣file";
4       document.querySelector("body").appendChild(element);
5   });
```

45. Edit `Index.cshtml` to add the CSS and JavaScript files. The listing is shown in 19

Listing 19: Edits to `Index.cshtml`

```
1   <head>
2       <meta name="viewport" content="width=device-width" />
3       <title>WorkingWithVisualStudio</title>
4       <link rel="stylesheet" href="css/first.css" />
5       <link rel="stylesheet" href="css/second.css" />
6       <script src="js/third.js"></script>
7       <script src="js/fourth.js"></script>"
8   </head>
```

46. To install the Bundler and Minifer utility, from the file menu select `Tools` ► `Extensions and Updates` ► `Online`, and search for "bundler" (see figure **??**. Click `Download` and close the window. Then, save all your work, close and restart Visual Studio. This will allow the Bundler and Minifier extension to install. When the installation is finished, restart your application.

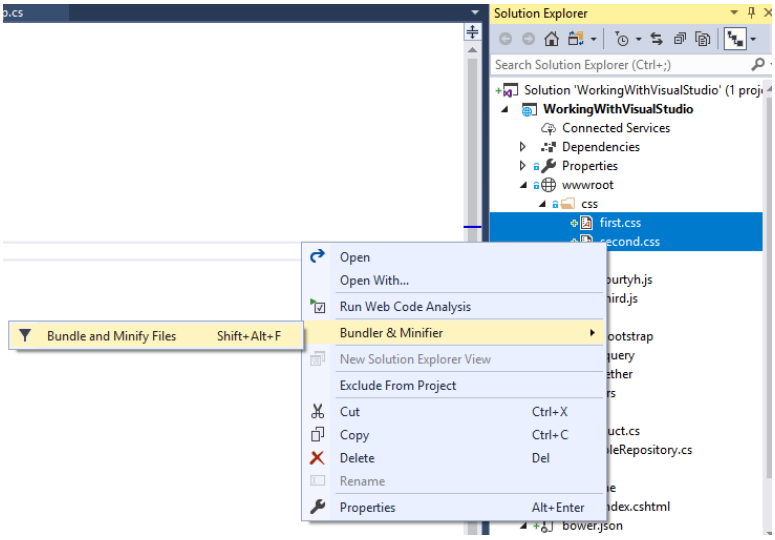

Figure 24: Bundle and Minify

47. Select *both* `first.css` and `second.css` by using CTL-click, right click, and select `Bundler & Minifier` ▶ `Bundle and Minify Files`. See figure 24.

48. Repeat this process for `third.js` and `fourth.js`.

49. Edit `Index.cshtml` to add the CSS and JavaScript files. The listing is shown in 20

Listing 20: Edits to `Index.cshtml`

```
1  <head>
2      <meta name="viewport" content="width=device-width" />
3      <title>Working with Visual Studio</title>
4      <link rel="stylesheet" href="css/bundle.min.css" />
5      <script src="js/bundle.min.js"></script>
6  </head>
```