



## ASP.NET Core Activity #3

### Part 1 Display Data in a Table

#### Goal:

In this activity, you use .NET MVC Views to:

- Write Razor syntax on a webpage.
- Generate sample data.
- Create a data table in a SQL database.
- Query the database using SQL statements.
- Pass data to a Razor form and display the contents in a table.

#### What is Razor?

Razor is a Microsoft language used to generate HTML code on a .NET server. Similar technology exists in PHP and Java. A webpage contains a mix of Razor code and HTML code. The Razor code is processed by the server before being displayed to the browser. In the example to the right, each line that begins with the @ symbol is Razor code.

```
Index.cshtml > ProductsController.cs
@model IEnumerable<HelperSample.Models.Product>

<h2>Products</h2>

<ul>
    @foreach (var product in Model) {
        <li>
            <span class="producttitle">
                @product.Name
            </span>

            <span class="description">
                @product.Description
            </span>

            <span class="price">
                @if (product.UnitPrice == 0) {
                    <span>FREE!</span>
                }
                else{
                    @String.Format("{0:C2}", product.UnitPrice)
                }
            </span>
        </li>
    }
</ul>
```

Create a list of objects and display it in a table.

This activity is a continuation of the Activity #1.

1. Open the **Products Controller** and the Products **Index View** to review what we accomplished in previous tutorials.

The Products Controller managed a few methods as shown here.

```
6 using Microsoft.AspNetCore.Mvc;
7
8 namespace ASPCoreFirstApp.Controllers
9 {
10     [Route("")]
11     public class ProductsController : Controller
12     {
13         [Route("")]
14         public IActionResult Index()
15         {
16             return View();
17         }
18
19         [Route("")]
20         public IActionResult Welcome()
21         {
22             ViewBag.name = "Shad";
23             ViewBag.secretNumber = 13;
24             return View();
25         }
26     }
27 }
```

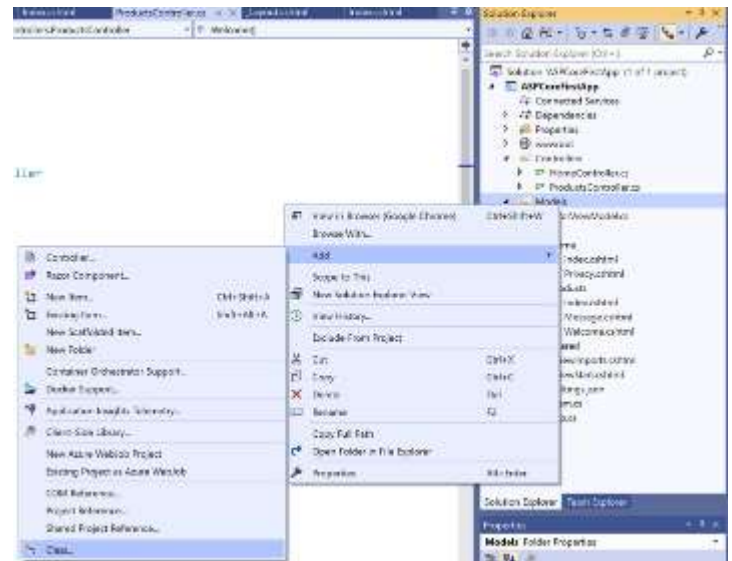
The Products **Index View** promised greater things to come. We will add a feature to the app where a list of products will be shown.

```
1
2
3 For more information on enabling MVC for empty projects, visit http://go.microsoft.com/fwlink/?
4 LinkID=397860
5
6 <h1>Products Index Page</h1>
7 <p>We will display a list of products here very soon.</p>
```

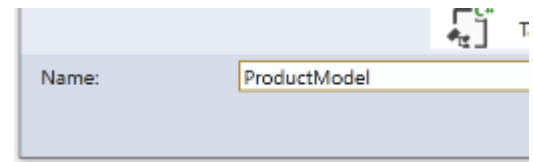
## Showing a Product List

In this section, we will create a list of products, store them in a database and display them on a webpage.

1. Create a Products Model class.
  - a. **Right-click** in the Models folder, choose **Add > Class**.



- b. Name the class **ProductModel**.



2. Add some properties to define a product. Notice that I choose three different data types: **integer**, **string**, and **decimal**. Create a parameterized constructor method.

```
5
6 namespace ASPCoreFirstApp.Models
7 {
8     public class ProductModel
9     {
10         public int Id { get; set; }
11         public string Name { get; set; }
12         public decimal Price { get; set; }
13         public string Description { get; set; }
14
15         public ProductModel(int id, string name, decimal price, string description)
16         {
17             Id = id;
18             Name = name;
19             Price = price;
20             Description = description;
21         }
22     }
23 }
24
```

3. Add some hardcoded data into a **List of products** inside the Products controller. Pass the list of data to the View as shown here. Later, we will replace this hardcoded data with information from a database.

```

namespace ASPCoreFirstApp.Controllers
{
    public class ProductsController : Controller
    {
        public IActionResult Index()
        {
            List<ProductModel> productList = new List<ProductModel>();

            productList.Add(new ProductModel(1, "Mouse Pad", 5.99m, "A square piece of plastic to make mousing easier"));
            productList.Add(new ProductModel(2, "Web Cam", 45.50m, "Enables you to attend more Zoom meetings."));
            productList.Add(new ProductModel(3, "4 TB USB Hard Drive", 130.00m, "Back up all of your work. You won't regret it."));
            productList.Add(new ProductModel(4, "Wireless Mouse", 15.99m, "Notebook mice really don't work very well. Do they?"));

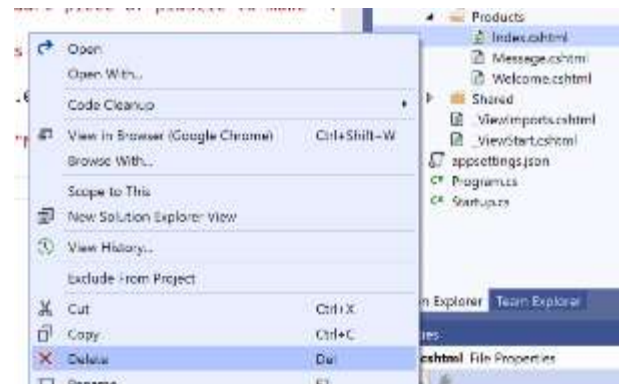
            return View(productList);
        }
    }
}

```

Invent some product instances and add them to the list.

The productList is passed to the index view as a parameter.

4. Delete the existing **Index** file inside the **Views/Products** folder. We will replace it with a new version shortly.



5. Right-click inside the **Index** method and choose **Add View...**



6. Use the following configuration to create the view: **name** = index, **Template** = list, **Model** = product.

The generated code should show a <table> element with a table head <thead>, table header elements <th>, table rows <tr>, and table data elements <td>.

Add Razor View

View name: Index

Template: List

Model class: ProductModel (ASPCoreFirstApp.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

Let's highlight some important parts of the page.

```
Index.cshtml | ProductModel.cs | Welcome.cshtml | Message.cshtml | Program.cs | Startup.cs
1  @model IEnumerable<ASPCoreFirstApp.Models.ProductModel>
2
3  <p>
4      <a asp-action="Create">Create New</a>
5  </p>
6  <table class="table">
7      <thead>
8          <tr>
9              <th>
10                 @Html.DisplayNameFor(model => model.Id)
11             </th>
12             <th>
13                 @Html.DisplayNameFor(model => model.Name)
14             </th>
15             <th>
16                 @Html.DisplayNameFor(model => model.Price)
17             </th>
18             <th>
19                 @Html.DisplayNameFor(model => model.Description)
20             </th>
21             <th></th>
22         </tr>
23     </thead>
24     <tbody>
25         @foreach (var item in Model) {
26             <tr>
27                 <td>
28                     @Html.DisplayFor(modelItem => item.Id)
29                 </td>
30                 <td>
31                     @Html.DisplayFor(modelItem => item.Name)
32                 </td>
33                 <td>
34                     @Html.DisplayFor(modelItem => item.Price)
35                 </td>
36                 <td>
37                     @Html.DisplayFor(modelItem => item.Description)
38                 </td>
```

@model

This tells the page what data type was passed to it from the controller. IEnumerable is an interface for the list<> data type.

@Html

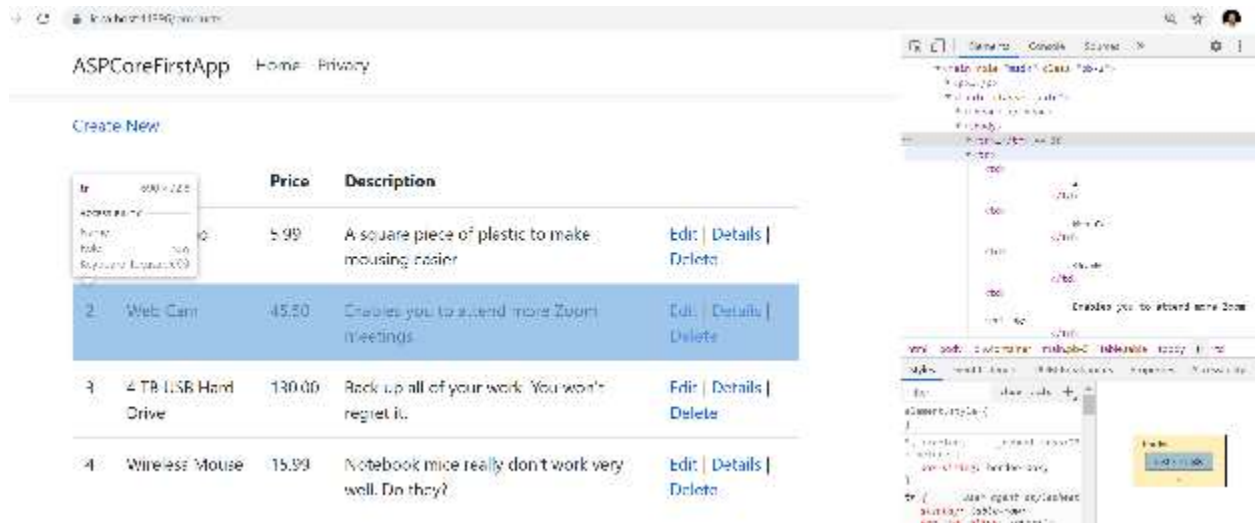
These html "helper" elements dynamically generate html tags in the rendered page.

=>

Arrow functions are not required, but help with how an item is formatted. See explanation below.



- Run the application and **inspect** the HTML elements in the browser. The following image shows the developer tools inspecting the page in Chrome. You can see that the **@Html helper tags** generated **standard HTML code** in the browser.
- Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.



The arrow functions, officially called **Lambda Expressions**, used in the Razor pages seem completely unnecessary at first. The following Razor code would work equally well as the code generated for us.

Version 1: The generated code for the table	Version 2: Simpler code that you could alternatively use. This does not utilize lambda functions.
<pre> @foreach (var item in Model) {     &lt;tr&gt;         &lt;td&gt;             @Html.DisplayFor(modelItem =&gt; item.Id)         &lt;/td&gt;         &lt;td&gt;             @Html.DisplayFor(modelItem =&gt; item.Name)         &lt;/td&gt;         &lt;td&gt;             @Html.DisplayFor(modelItem =&gt; item.Price)         &lt;/td&gt;         &lt;td&gt;             @Html.DisplayFor(modelItem =&gt; item.Description)         &lt;/td&gt;         &lt;td&gt;             @Html.ActionLink("Edit", "Edit", new { /* id=item.Id */ })             @Html.ActionLink("Details", "Details", new { /* id=item.Id */ })             @Html.ActionLink("Delete", "Delete", new { /* id=item.Id */ })         &lt;/td&gt;     &lt;/tr&gt; } </pre>	<pre> &lt;tbody&gt;     @foreach (var item in Model) {         &lt;tr&gt;             &lt;td&gt;                 @item.Id             &lt;/td&gt;             &lt;td&gt;                 @item.Name             &lt;/td&gt;             &lt;td&gt;                 @item.Price             &lt;/td&gt;             &lt;td&gt;                 @item.Description             &lt;/td&gt;             &lt;td&gt;                 @Html.ActionLink("Edit", "Edit", new { /* id=item.Id */ })                 @Html.ActionLink("Details", "Details", new { /* id=item.Id */ })                 @Html.ActionLink("Delete", "Delete", new { /* id=item.Id */ })             &lt;/td&gt;         &lt;/tr&gt;     } </pre>

## The Purpose of the Lambda Expressions

Lambda expressions (a.k.a. arrow functions) help with formatting data. ASP.NET allows you to modify the display properties and format of each property inside a class. For example, try these modifications shown below in the **ProductModel** class. The **display name** is changed for each item and the **currency** format is applied to the price property. You will have to do some imports to use the notations.

```
13 references
public class ProductModel
{
    [DisplayName("Id number")]
    6 references
    public int Id { get; set; }
    [DisplayName("Product Name")]
    6 references
    public string Name { get; set; }

    [DataType(DataType.Currency)]
    [DisplayName("Cost to customer")]
    6 references
    public decimal Price { get; set; }
    [DisplayName("What you get...")]
    6 references
    public string Description { get; set; }
```

The resulting changes appear in the following picture. Notice the **column header names** have changed and the **price has a dollar sign**. The arrow functions in the View code make this formatting possible.

9. Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.

Id number	Product Name	Cost to customer	What you get...
1	Mouse Pad	\$5.99	A square piece of plastic to make mousing easier
2	Web Cam	\$45.50	Enables you to attend more Zoom meetings.
3	4 TB USB Hard Drive	\$130.00	Back up all of your work. You won't regret it.
4	Wireless Mouse	\$15.99	Notebook mice really don't work very well. Do they?
5	Practical Cotton Table	\$43.87	I tried to grab it but got bonbon all over it.



## Generate Fake Data

For an easy and humorous way to generate fake data, we are going to use a library called **Faker** or **Bogus**. Here is the GitHub resource to see the complete documentation.



1. Modify the Products controller to the following code to **include 100 more products**.

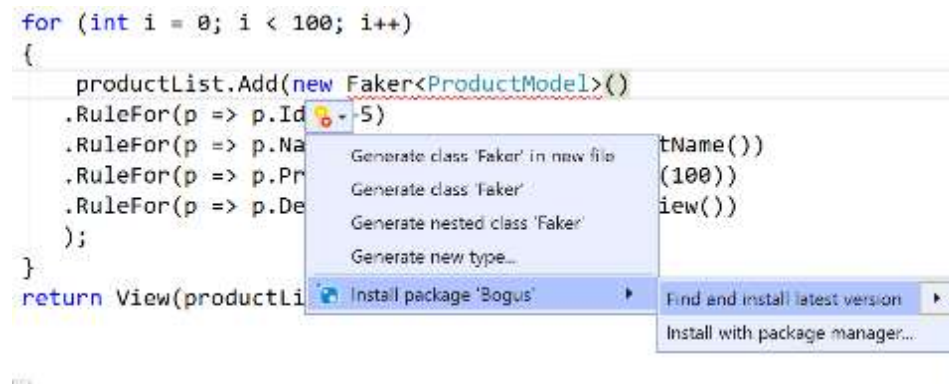
```
public class ProductsController : Controller
{
    Unreferenced
    public IActionResult Index()
    {
        List<ProductModel> productList = new List<ProductModel>();

        productList.Add(new ProductModel(1, "Mouse Pad", 5.99m, "A square piece of plastic to make mousing easier"));
        productList.Add(new ProductModel(2, "Web Cam", 45.50m, "Enables you to attend more Zoom meetings."));
        productList.Add(new ProductModel(3, "4 TB USB Hard Drive", 130.00m, "Back up all of your work. You won't regret it."));
        productList.Add(new ProductModel(4, "Wireless Mouse", 15.99m, "Notebook mice really don't work very well. Do they?"));

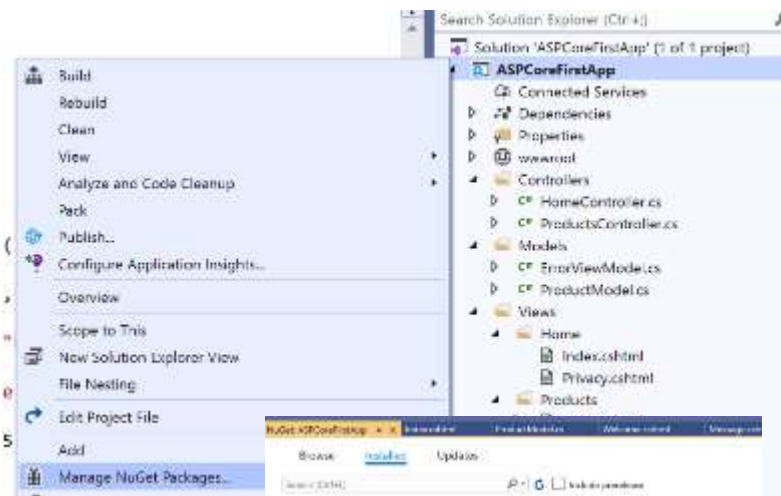
        for (int i = 0; i < 100; i++)
        {
            productList.Add(new Faker<ProductModel>()
                .RuleFor(p => p.Id, i+5)
                .RuleFor(p => p.Name, f => f.Commerce.ProductName())
                .RuleFor(p => p.Price, f => f.Random.Decimal(100))
                .RuleFor(p => p.Description, f => f.Rant.Review())
            );
        }

        return View(productList);
    }
}
```

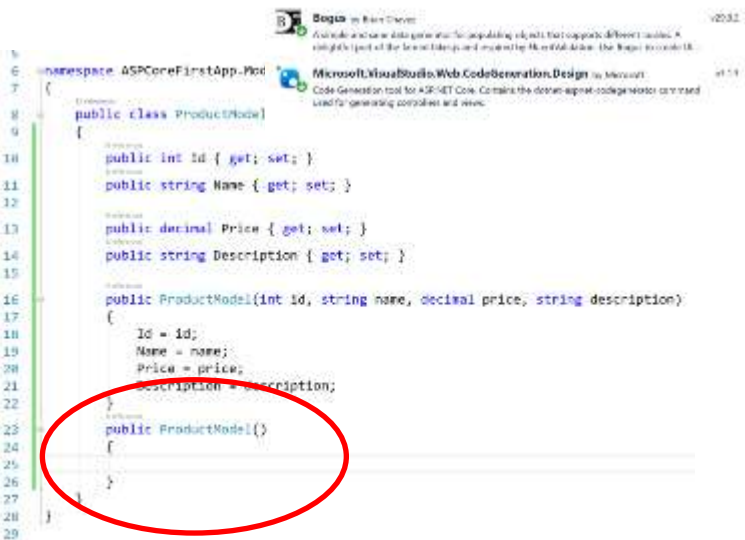
**Faker<>** is a function from the **Bogus** library, which is a dependency available via the **Nuget** package manager. You should be able to install Bogus the moment you type **Faker<ProductModel>**. Visual Studio intelli-sense will recognize the need for a dependency.



Once Bogus is added to the project, you can check on its status by right-clicking the solution title and choosing **Manage NuGet Packages**.



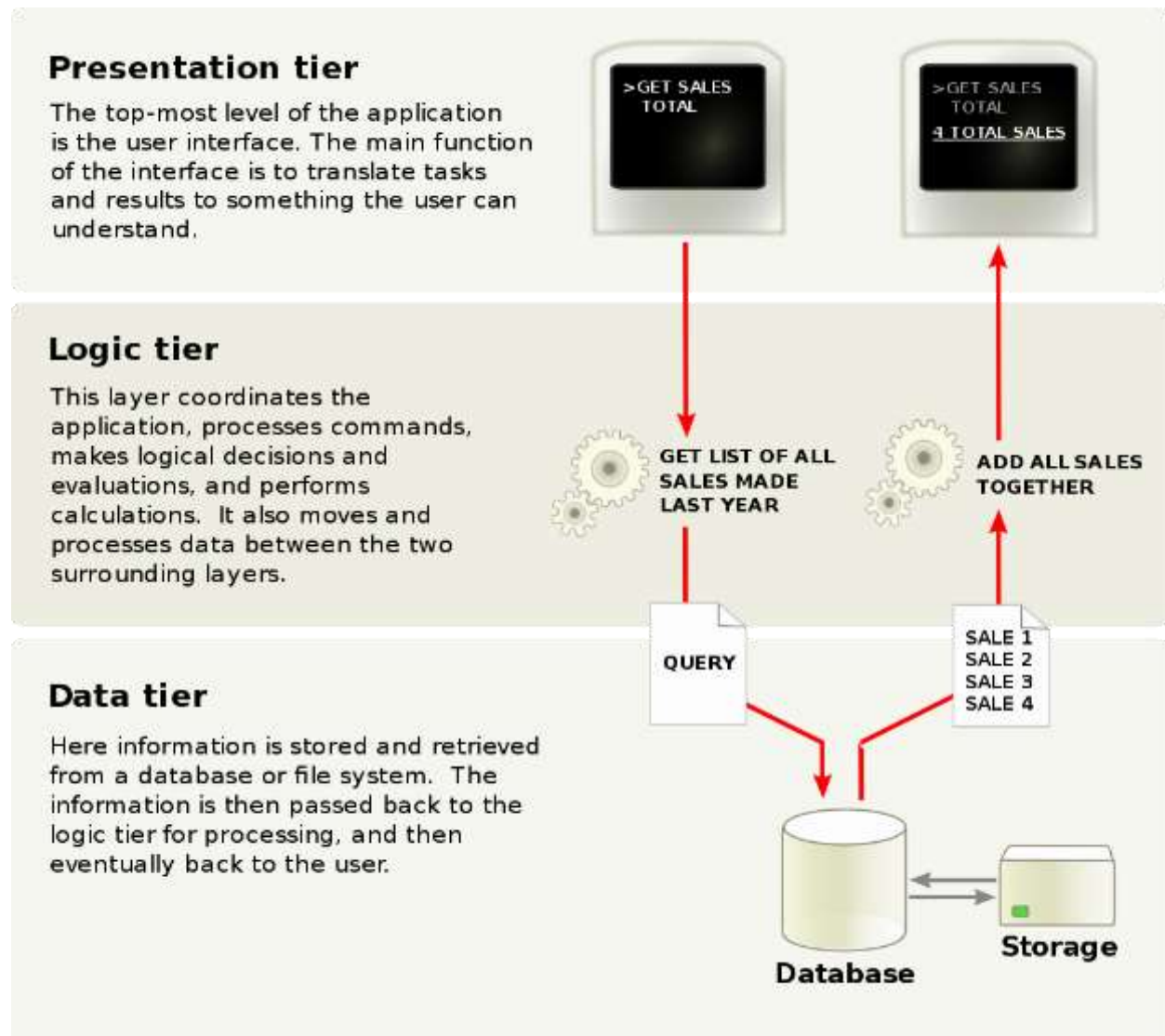
Bogus requires that the ProductModel class have an **empty constructor**, shown here.



## Data Repository

Next, we are going to refactor the application to reflect a better design pattern. We are going to take the sample data that is defined in the Products Controller and move it to a data service layer.

Consider the following diagram of a **3-tier application**. Applications that scale to large user numbers well are typically separated into layers according to specific tasks. Splitting the work allows multiple servers to handle the workload.



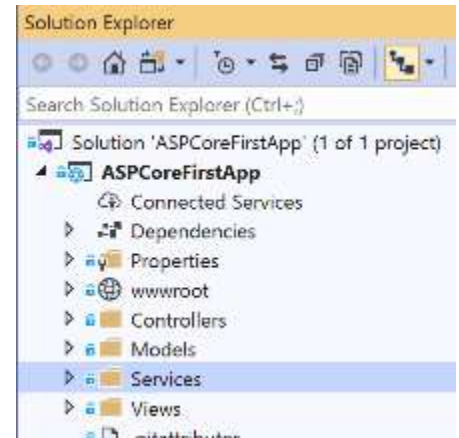
In our application:

- the View components (HTML + Razor) correspond to the **Presentation Layer**.
- the controller corresponds to the **Logic Tier**.
- the Data Tier is **currently absent** from our application.

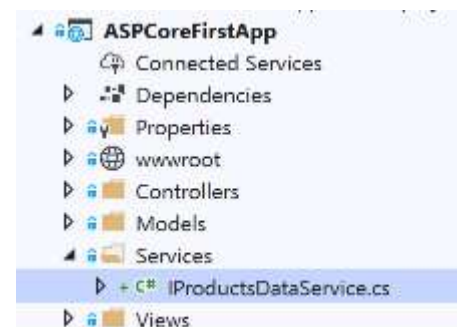
We currently have stored the **list of products** in the controller. We will now create the **Data Tier** layer and provide a pathway to making the data storage a distinct module of the application that can be easily changed according to our storage needs. We will eventually replace the mock data source we currently have with a **SQL database**.

### Create an Interface to Define the Data Service

1. Create a new folder called **Services** in the project.



2. Create a new class called **IProductsDataService** in the Services folder.



3. Add the following code to the new class.

This interface will define three operations that our application will eventually be able to perform. These are the basic **CRUD** (Create Read Update Delete) operations that are common in data storage applications.

A screenshot of the Visual Studio code editor showing the content of the 'IProductsDataService.cs' file. The code is as follows:

```
1 using System.Collections.Generic;
2
3
4 namespace ASPCoreFirstApp.Services
5 {
6     public interface IProductsDataService
7     {
8         List<ProductModel> AllProducts();
9         List<ProductModel> SearchProducts(string searchTerm);
10        ProductModel GetProductById(int id);
11        int Insert(ProductModel product);
12        bool Delete(ProductModel product);
13        int Update(ProductModel product);
14    }
15 }
```

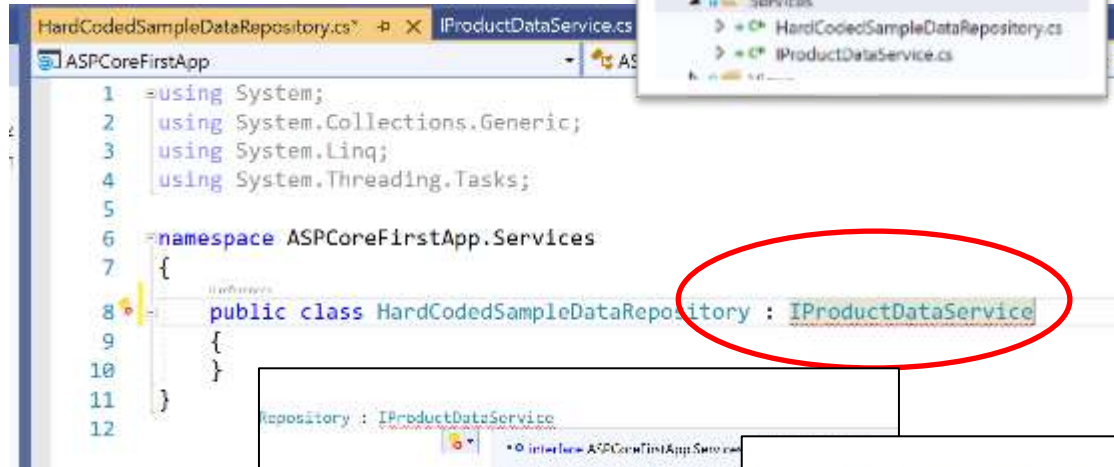
The line 'public interface IProductsDataService' is circled in red.



4. Create another class in the Services folder. Name it **HardCodedSampleDataRepository.cs**

5. Add the implementation of the new interface we just defined.

Notice that the red-line error on the new code says we need to implement the methods defined in the interface.



6. Choose **Implement interface** from the "Show potential fixes" menu.

7. All of the methods defined in the interface should now appear in the new service.



8. Cut all of the Fake data generation code from the **ProductsController** file.

```
13 public class ProductsController : Controller
14 {
15     public IActionResult Index()
16     {
17
18         List<ProductModel> productList = new List<ProductModel>();
19
20         productList.Add(new ProductModel(1, "Mouse Pad", 5.99m, "A square piece of plastic to make mousing
21             easier"));
22         productList.Add(new ProductModel(2, "Web Cam", 45.50m, "Enables you to attend more Zoom meetings"));
23         productList.Add(new ProductModel(3, "4 TB USB Hard Drive", 130.00m, "Back up all of your work. You
24             won't regret it.));
25         productList.Add(new ProductModel(4, "Wireless Mouse", 15.99m, "Notebook mice really don't work very
26             well. Do they?));
27         for (int i = 0; i < 100; i++)
28         {
29             productList.Add(new Faker<ProductModel>()
30                 .RuleFor(p => p.Id, i + 5)
31                 .RuleFor(p => p.Name, f => f.Commerce.ProductName())
32                 .RuleFor(p => p.Price, f => f.Random.Decimal(100))
33                 .RuleFor(p => p.Description, f => f.Rant.Review())
34             );
35         }
36
37         return View(productList);
38     }
39 }
```



9. In the **HardCodedSampleDataRepository** file, create a constructor and paste the code inside. Make modifications as shown here.

```
8 namespace ASPCoreFirstApp.Services
9 {
10     public class HardCodedSampleDataRepository : IProductDataService
11     {
12         // use static to ensure the data set does not change
13         static List<ProductModel> productList;
14
15         // create the list in the constructor of the service.
16         public HardCodedSampleDataRepository()
17         {
18             productList = new List<ProductModel>();
19             productList.Add(new ProductModel(1, "Mouse Pad", 5.99m, "A square piece of plastic to make mousing
20                 easier"));
21             productList.Add(new ProductModel(2, "Web Cam", 45.50m, "Enables you to attend more Zoom meetings"));
22             productList.Add(new ProductModel(3, "4 TB USB Hard Drive", 130.00m, "Back up all of your work. You
23                 won't regret it.");
24             productList.Add(new ProductModel(4, "Wireless Mouse", 15.99m, "Notebook mice really don't work very
25                 well. Do they?"));
26             for (int i = 0; i < 100; i++)
27             {
28                 productList.Add(new Faker<ProductModel>()
29                     .RuleFor(p => p.Id, i + 5)
30                     .RuleFor(p => p.Name, f => f.Commerce.ProductName())
31                     .RuleFor(p => p.Price, f => f.Random.Decimal(100))
32                     .RuleFor(p => p.Description, f => f.Rant.Review())
33                 );
34             }
35         }
36
37         public List<ProductModel> AllProducts()
38         {
39             return productList;
40         }
41     }
42 }
```

productList is a class-member variable, accessible throughout all of the methods below.

The data is randomly generated inside the constructor.

The first method only needs to return the entire productList.

10. In the ProductsController, make the following changes.

```
1 namespace ASPCoreFirstApp.Controllers
2 {
3     public class ProductsController : Controller
4     {
5         public IActionResult Index()
6         {
7             HardCodedSampleDataRepository repository = new HardCodedSampleDataRepository();
8             return View(repository.AllProducts());
9         }
10    }
11 }
```

Instantiate the data repository. This repository will be replaced in a future version of the application using a technique called Dependency Injection.

Get all of the data records via the AllProduct method.

11. Run the program. There should be no visual or behavior changes to the application. We have simply refactored the application to implement a better code structure.

### **Coding Challenge: Create another model**


To reinforce the process of managing models, controller, and views, create another model class in addition to the Products Model. This will go in the Models folder. Create a controller and View to display the new item in a table.

1. Create a class for another object which represents a product (**Movie, Car, Airline Ticket**, etc.) with at least the following properties: String, Integer, and Date.
2. Create another **Controller** and **View** based on the example created in this exercise.
3. Generate some **sample data** for the new item.
4. Display the resulting list in a **table**.
5. Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.

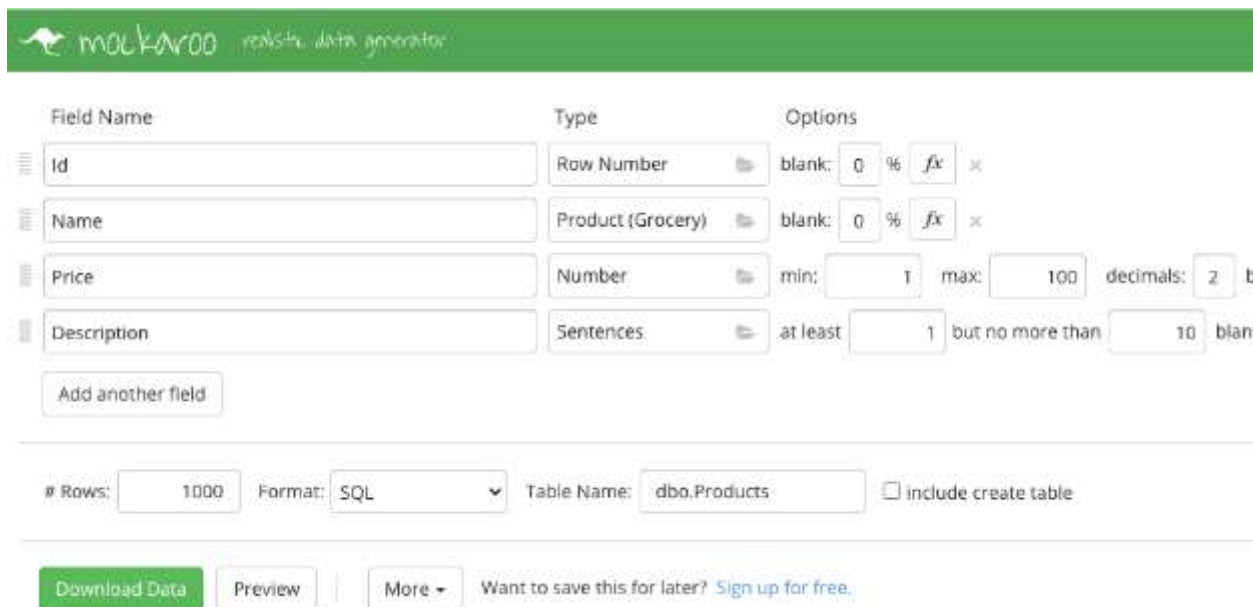
## Database table for products

In this section, we will create a database source for product information replacing the process of generating the data within the controller from the Bogus library.

1. In the test database (where the users table already exists), create a new table called **dbo.Products** using the values shown here.
2. Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.
3. Fill the table with some sample data. Use the site Mockaroo to generate 1000 products automatically. Import the SQL file into the database. Here is an example of one way you could generate sample data. Take a screenshot of the database at this stage. Paste it into a Word document and caption the image with a brief explanation of what you just demonstrated.



Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
Name	nvarchar(50)	<input checked="" type="checkbox"/>
Price	decimal(18,2)	<input checked="" type="checkbox"/>
Description	nvarchar(500)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>



mockaroo realistic data generator

Field Name	Type	Options
Id	Row Number	blank: 0 % fx
Name	Product (Grocery)	blank: 0 % fx
Price	Number	min: 1 max: 100 decimals: 2 b
Description	Sentences	at least 1 but no more than 10 blank

[Add another field](#)

---

# Rows: 1000 Format: SQL Table Name: dbo.Products ☐ include create table

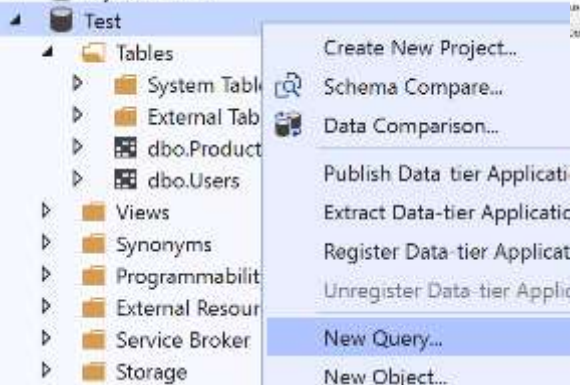
---

[Download Data](#) [Preview](#) [More](#) Want to save this for later? [Sign up for free.](#)

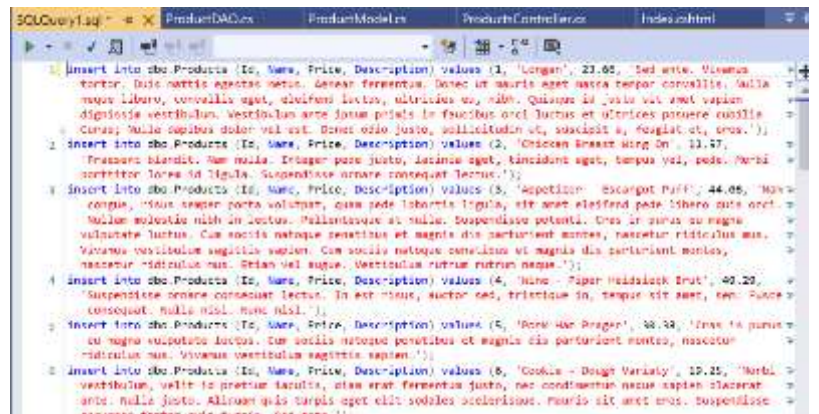
- Download the data and open it in a text editor.



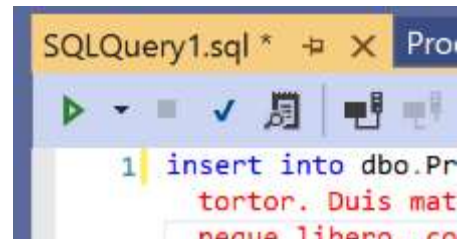
- In the SQL Server window, right-click on the test database and start a **New Query...**



- Paste** the SQL statements from Mockaroo into the query window.



- Execute** the query by clicking the **green triangle** at the top of the query window.

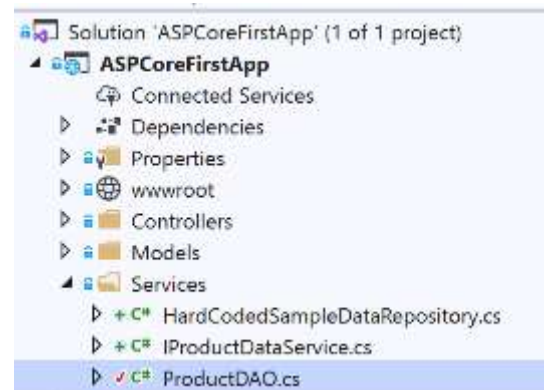


You should see the database filled with values.

8. In the **Services** folder in the project, add a **ProductsDAO** class.

9. Implement the **IProductDataServices** interface.

```
1 using ASPCoreFirstApp.Models;
2 using System;
3 using System.Collections.Generic;
4 using System.Data.SqlClient;
5
6 namespace ASPCoreFirstApp.Services
7 {
8     // references
9     public class ProductDAO : IProductDataService
10    {
11    }
12 }
13
```



10. Implement all the methods.



11. Add code to the method

**AllProducts** that connects to the database and returns all records from the Products table.



```

9 public class ProductDAO : IProductDataService
10 {
11     // copy this string from the test database properties values.
12     string connectionString = @"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Test;Integrated Security=True;Connect
13         Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False";
14
15     public List<ProductModel> AllProducts()
16     {
17         // assume nothing is found
18         List<ProductModel> foundProducts = new List<ProductModel>();
19
20         // uses prepared statements for security. @username @password are defined below
21         string sqlStatement1 = "SELECT * FROM dbo.Products";
22
23         using (SqlConnection connection = new SqlConnection(connectionString))
24         {
25             SqlCommand command = new SqlCommand(sqlStatement, connection);
26
27             try
28             {
29                 connection.Open();
30                 SqlDataReader reader = command.ExecuteReader();
31
32                 while (reader.Read())
33                 {
34                     foundProducts.Add(new ProductModel((int)reader[0], (string)reader[1], (decimal)reader[2], (string)
35                         reader[3]));
36                 }
37             }
38             catch (Exception ex)
39             {
40                 Console.WriteLine(ex.Message);
41             }
42         }
43         return foundProducts;
44     }
45 }

```

12. Add another method to perform a search by name.



```

62 public List<ProductModel> SearchProducts(string searchTerm)
63 {
64     // assume nothing is found
65     List<ProductModel> foundProducts = new List<ProductModel>();
66
67     // uses prepared statements for security. @username @password are defined below
68     string sqlStatement = "SELECT * FROM dbo.Products WHERE Name LIKE @Name";
69
70     using (SqlConnection connection = new SqlConnection(connectionString))
71     {
72         SqlCommand command = new SqlCommand(sqlStatement, connection);
73
74         // define the values of the two placeholders in the sqlStatement string
75         command.Parameters.AddWithValue("@Name", '%' + searchTerm + '%');
76
77
78         try
79         {
80             connection.Open();
81             SqlDataReader reader = command.ExecuteReader();
82
83             while (reader.Read())
84             {
85                 foundProducts.Add(new ProductModel((int)reader[0], (string)reader[1], (decimal)reader[2], (string)
86                     reader[3]));
87             }
88         }
89         catch (Exception ex)
90         {
91             Console.WriteLine(ex.Message);
92         }
93     }
94     return foundProducts;
95 }

```

Microsoft has examples for all types of SQL queries at their official documentation  
<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/retrieving-and-modifying-data>

Other methods, such as **delete**, **selectOne**, and **update** are future enhancements for the application and are not required at this time.

13. In the **ProductsController**, use the **ProductsDAO** to **fetch the data** from the database instead of generating new fake data in the controller.

```

10
11 namespace ASPCoreFirstApp.Controllers
12 {
13     public class ProductsController : Controller
14     {
15         ProductDAO repository = new ProductDAO();
16         public ProductsController()
17         {
18             repository = new ProductDAO();
19         }
20
21         public IActionResult Index()
22         {
23             return View(repository.AllProducts());
24         }
25 }

```

14. The app should display the contents of the database instead of the Bogus data.
15. Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.

Id number	Product Name	Cost to customer	What you get...
1	Longan	\$25.66	Sed ante eiusmodi lobis. Duis nulla egeta ornare, delecta feconibus. Donec ut mauris eget, nulla lacus conat. Nunc massa libero, conat eget, vestibul lacus, utiles eu, with. Quisque id justo et amet sapien dignidit. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nulla dapibus dolor vel est. Donec odio justo, sollicitudin ut, suscipit a. Voluta et, eros.
2	Chicken Breast Wing (11	11.07	Prosser blandi. Nam nulla integer pede justo, lectus eget, vivit eget, tempus sed, pede. Morbi porttitor, lobis id ligula. Suspendisse ornare conat conat.
3	Apple Pie - Enchanted But	\$45.06	Nam conat, eros tempus porta voluget, quam pede lobis ligula, sit amet oblectat pede libero, quis orci. Nullam molestie nibh in lectus. Pellentesque ornare. Suspendisse porttitor. Cras in purus eu magna vulputate lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridicula mus. Vivamus vulputate sem sagittis sapien. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridicula mus. Etiam sed augue, vestibulum rutrum rutrum tempus.
4	Wine - Piper Heidsieck Brut	\$49.29	Suspendisse ornare conat conat lectus, in est risus, auctor sed, tristique in, tempus ut amet, enim. Fusce conat conat. Nulla nisi. Nunc nisi.
5	Rock Ram Page	\$30.48	Cras in purus eu magna vulputate lacus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridicula mus. Vivamus vulputate sem sagittis sapien.

16. Add **another method** to the **ProductsController** to respond to search results.

```

13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
249
```

Let's add a form that will allow the user to perform a search.

19. Add a new method in the **ProductsController** called **SearchForm**.

```
21 public IActionResult Index()
22 {
23     return View(repository.AllProducts());
24 }
25
26 public IActionResult SearchResults(string searchTerm)
27 {
28     List<ProductModel> productList = repository.SearchProducts(searchTerm);
29     return View("Index", productList);
30 }
31
32 public IActionResult SearchForm()
33 {
34     return View();
35 }
36
```

20. Right-click inside the SearchForm method and choose Add View...

21. Choose Razor View.

22. Choose name = SearchForm, Template = Create, Model class = ProductModel.

Add Razor View

View name: SearchForm

Template: Create

Model class: ProductModel (ASPCoreFirstApp.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

23. Modify the resulting HTML code for a data entry form.
24. Run the application and use the URL:

```

1 <h4>Search Form</h4>
2 <hr />
3 <div class="row">
4   <div class="col-md-4">
5     <form asp-action="SearchResults">
6
7       <div class="form-group">
8         <label class="control-label">Search for a Product</label>
9         <input name="SearchTerm" class="form-control" />
10      </div>
11
12      <div class="form-group">
13        <input type="submit" value="Search" class="btn btn-primary" />
14      </div>
15    </form>
16  </div>
17 </div>
18 </div>
19
20 <div>
21   <a asp-action="Index">Back to List</a>
22 </div>
23
24

```

<https://localhost:44396/products/searchForm>

25. Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.
26. The new form should produce search results according to the keyword you supply.

27. Take a screenshot of the app at this stage. Paste it into a Word document and caption the image with a brief explanation of what you just demonstrated.

ASPCoreFirstApp Home Privacy

Create New

Id number	Product Name	Cost to customer	What you get...
4	Wine - Piper Heidsieck Brut	\$49.29	Suspendisse ornare consequat. lectus. In est risus, auctor s Fusce consequat. Nulla nisl. Nunc nisl.
22	Wine - Magnotta - Bell Paese White	\$4.08	Filam faucibus cursus urna. Ut tellus. Nulla ut erat id maur
33	Wine - Barolo Fontanafredda	\$28.79	Integer ac neque.
34	Wine - Guy Sage Touraine	\$97.19	Proin risus. Praesent lectus. Vestibulum quam sapien, vari Vestibulum ante ipsum primis in faucibus orci luctus et ul faucibus accumsan odio.

## Add Links in the Navbar

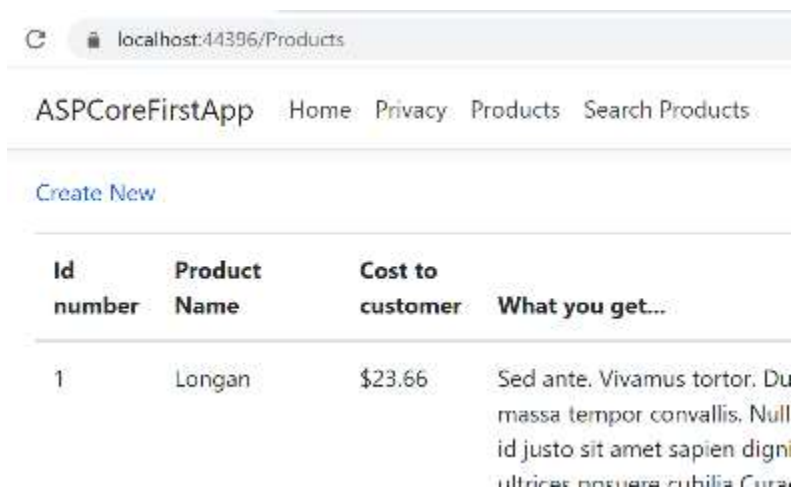
To make this app much more usable, add some links to the Navbar to allow the user to open the Products page as well as the SearchForm page.

1. Open **\_Layout.cshtml** in Views > Shared folder.
2. Add two more links in the navbar section as shown here. Copy and paste a previous link and modify the details.

```
10 <body>
11 <header>
12 <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
13 <div class="container">
14 <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">ASPCoreFirstApp</a>
15 <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
16 <span class="navbar-toggler-icon"></span>
17 </button>
18 <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
19 <ul class="navbar-nav flex-grow-1">
20 <li class="nav-item">
21 <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
22 </li>
23 <li class="nav-item">
24 <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
25 </li>
26 <li class="nav-item">
27 <a class="nav-link text-dark" asp-area="" asp-controller="Products" asp-action="Index">Products</a>
28 </li>
29 <li class="nav-item">
30 <a class="nav-link text-dark" asp-area="" asp-controller="Products" asp-action="SearchForm">Search Products</a>
31 </li>
32 </ul>
33 </div>
34 </div>
35 </div>
36 </div>
37 </div>
38 </div>
```

You should be able to navigate to the two new pages.

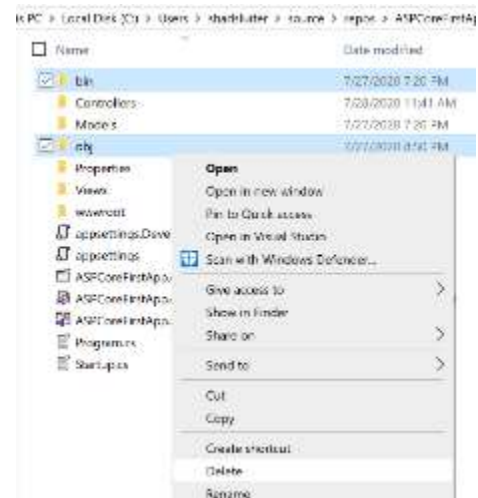
3. Take a screenshot of the app at this stage. Paste it into a Word document and caption the image with a brief explanation of what you just demonstrated.



Id number	Product Name	Cost to customer	What you get...
1	Longan	\$23.66	Sed ante. Vivamus tortor. Du massa tempor convallis. Null id justo sit amet sapien digni ultrices nequea. Eubilia Cura

## Deliverables:

1. This activity has multiple parts. Complete all parts before submitting.
2. Submit a Microsoft Word document with screenshots of the application being run at each stage of development. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
3. In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson.
4. Submit a ZIP file of the project file. In order to save space, you can delete the bin and the obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.





## Part 2 Working with Bootstrap

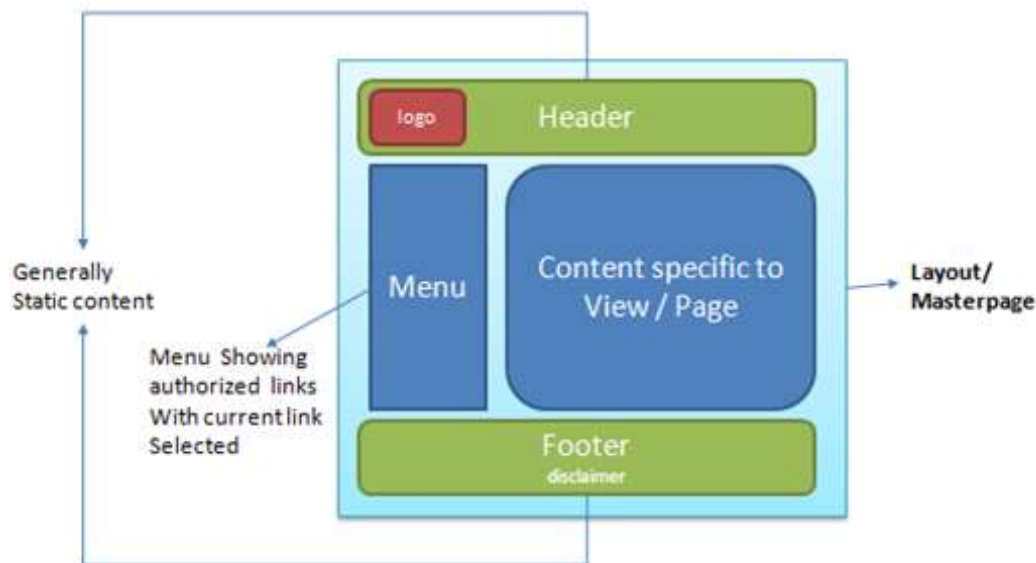
### Goal:

In this activity, you will:

- Use Razor Layouts.
- Format the site using Bootstrap CSS.
- Change the default Bootstrap CSS theme.
- Use Flexbox as an alternative to a table.

### About Layouts

Layouts are designed to reduce code duplication. We create a piece of the page, such as the header or footer, only once and reference it in a layout file. Consider the part of a website as shown here.



To use layouts in this example, we would design each region as a separate piece of html code and then combine them using one master layout file, pictured here as the light blue box surrounding all of the other pieces. The pieces are reassembled dynamically when the server generates a new webpage for a browser request.

1. Open the **\_Layout.cshtml** file in the View > Shared folder.

Notice the sections where the pieces of a layout are assembled. The @ symbol is used when in the dynamically-inserted sections.

```

31         <li class="nav-item">
32             <a class="nav-link text-dark" asp-area="" asp-controller="Products" asp-action="SearchForm">Search Products</a>
33         </li>
34     </ul>
35 </div>
36 </div>
37 </div>
38 </div>
39 </div>
40 </div>
41 </div>
42 <div class="container">
43     <main role="main" class="pb-3">
44         @RenderBody()
45     </main>
46 </div>
47 <div class="border-top footer text-muted">
48     <div class="container">
49         <small>© 2020 - ASPCoreFirstApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
50     </div>
51 </div>
52 </div>
53 </div>
54 </div>
55 </div>
56 </div>
57 </div>
58 </div>

```

@RenderBody() is a Razor function that inserts a View

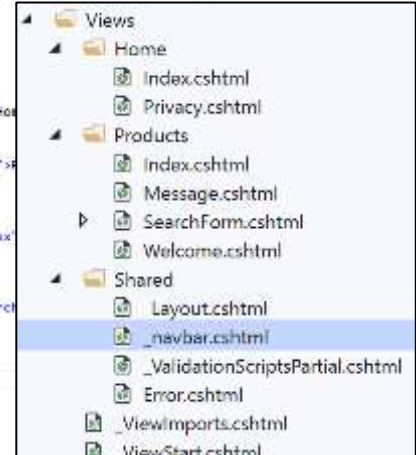
Let's create a partial view for the navbar in order to make the \_layout page smaller.

2. **Cut out the entire section** from <nav> to </nav> and put it into a separate file. Save the file as **\_navbar.cshtml** in the shared folder. Here is the contents of the new file taken from layout.

```

1 <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
2     <div class="container">
3         <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">ASPCoreFirstApp</a>
4         <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
5             aria-expanded="false" aria-label="Toggle navigation">
6             <span class="navbar-toggler-icon"></span>
7         </button>
8         <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
9             <ul class="navbar-nav flex-grow-1">
10                 <li class="nav-item">
11                     <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
12                 </li>
13                 <li class="nav-item">
14                     <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
15                 </li>
16                 <li class="nav-item">
17                     <a class="nav-link text-dark" asp-area="" asp-controller="Products" asp-action="Index">Products</a>
18                 </li>
19                 <li class="nav-item">
20                     <a class="nav-link text-dark" asp-area="" asp-controller="Products" asp-action="SearchForm">Search Products</a>
21                 </li>
22             </ul>
23         </div>
24     </div>
25 </nav>

```



3. In the `_Layout.cshtml` file, replace the missing content with `@Html.Partial("_navbar.cshtml");` as shown here.

```
9 </head>
10 <body>
11 <header>
12     @Html.Partial("_navbar.cshtml");
13 </header>
14 <div class="container">
15     <main role="main" class="pb-3">
16         @RenderBody()
17     </main>
18 </div>
19
20 <footer class="border-top footer text-muted">
21     <div class="container">
22         &copy; 2020 - ASPCoreFirstApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
23     </div>
24 </footer>
25 <script src="~/lib/jquery/dist/jquery.min.js"></script>
26 <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
27 <script src="~/js/site.js" asp-append-version="true"></script>
28 @RenderSection("Scripts", required: false)
29 </body>
30 </html>
31
```

4. Run the application to ensure it still shows the **nav bar** correctly. There should be **no visible change** to the application.

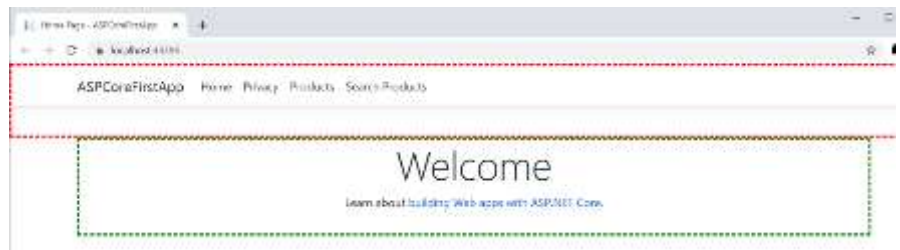
## Visual Confirmation of Page Sections

Next, we will add some **CSS formatting** to visually **emphasize** the separate sections on the layout page.

1. Open the `site.css` in the folder `wwwroot > css`. Notice that line 27 of the layout file (pictured above) has a `<script>` reference to this file. The `site.css` file is the preferred location to customize the site.
2. To visually demonstrate the use of the template, we are going to add a **large border** around the `<main>` and `<header>` sections of the page. I am using an ugly, but noticeable, **3px** wide border around each section.

```
_navbar.cshtml  site.css  ProductDAO.cs
1 /* Please see documentation at https://docs.
2 for details on configuring this project to b
3
4 header {
5     border: 3px dashed red;
6 }
7
8 main {
9     border: 3px dashed green;
10 }
11
12 a.navbar-brand {
13     white-space: normal;
14     text-align: center;
15     word-break: break-all;
16 }
17
18 /* Provide sufficient contrast against white
```

3. Run the application and verify that the sections are displayed. You may have to press **shift while reloading the page** on the browser to force the new CSS changes to appear.

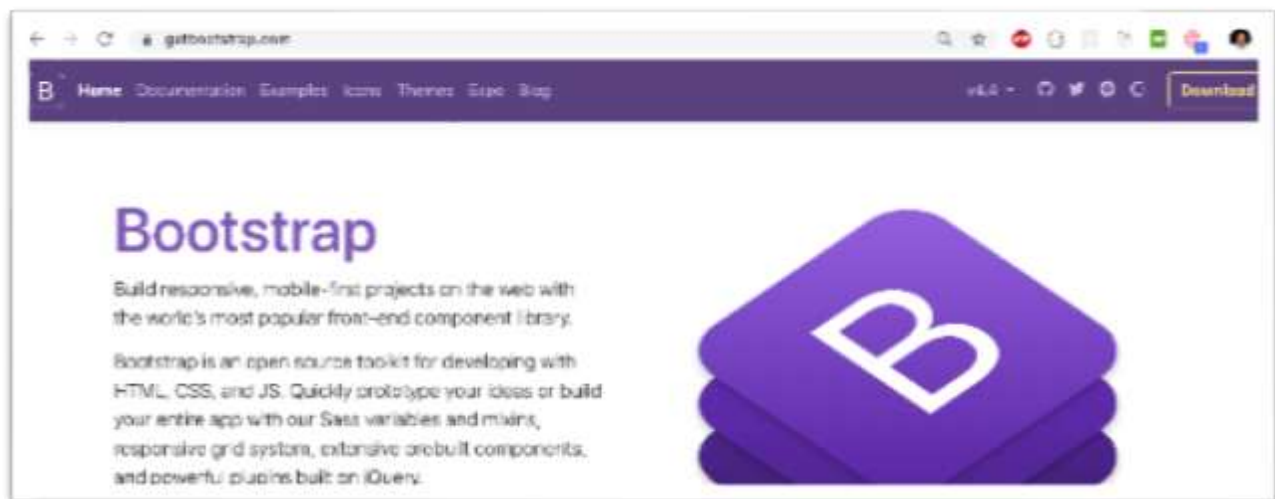


4. Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.

## Introduction to Bootstrap CSS

Bootstrap is a popular CSS library that provides you with many pre-defined styles. You should use Bootstrap or some similar library when you are trying to save time in creating a functional website. Bootstrap 4 is the de facto CSS framework for creating a quick design.

The official website for Bootstrap is shown below. The documentation and examples on the official website are very clearly written.



A good tutorial for using Bootstrap is shown here at w3schools.



You will notice that links to Bootstrap CSS and Bootstrap JavaScript are included in the `_Layout.cshtml` file. That means that every page in the application is styled using Bootstrap.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>@ViewData["Title"] - ASPCoreFirstApp</title>
7   <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
8   <link rel="stylesheet" href="~/css/site.css" />
9 </head>
10 <body>
11   <header>
12     @Html.Partial("_navbar.cshtml");
13   </header>
14   <div class="container">
15     <main role="main" class="pb-3">
16       @RenderBody()
17     </main>
18   </div>
19
20   <footer class="border-top footer text-muted">
21     <div class="container">
22       &copy; 2020 - ASPCoreFirstApp - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
23     </div>
24   </footer>
25   <script src="~/lib/jquery/dist/jquery.min.js"></script>
26   <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
27   <script src="~/js/site.js" asp-append-version="true"></script>
28   @RenderSection("Scripts", required: false)
29 </body>
30 </html>
31
```

Bootstrap class names are used in nearly every Razor page in the application. For example, `SearchForm.cshtml`, created in a previous tutorial, has Bootstrap CSS class names in nearly every line of the form. Knowing Bootstrap class names is a key skill to being able to style an ASP.NET application.

```
SearchForm.cshtml  X  navbar.cshtml  site.css  ProductModel.cs  Index.cs
1 <h4>Search From</h4>
2 <hr />
3 <div class="row">
4   <div class="col-md-4">
5     <form asp-action="SearchResults">
6
7       <div class="form-group">
8         <label class="control-label">Search for a Product</label>
9         <input name="searchTerm" class="form-control" />
10      </div>
11
12      <div class="form-group">
13        <input type="submit" value="Search" class="btn btn-primary" />
14      </div>
15    </form>
16  </div>
17 </div>
18
19 <div>
20   <a asp-action="Index">Back to List</a>
21 </div>
22
```

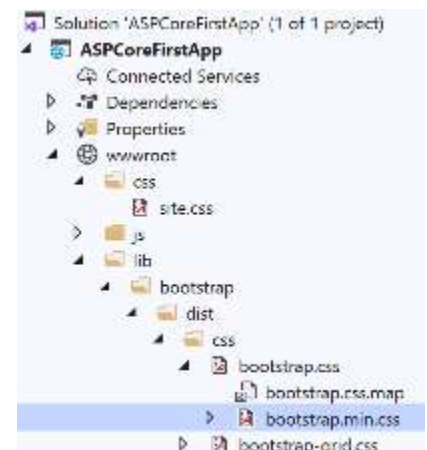
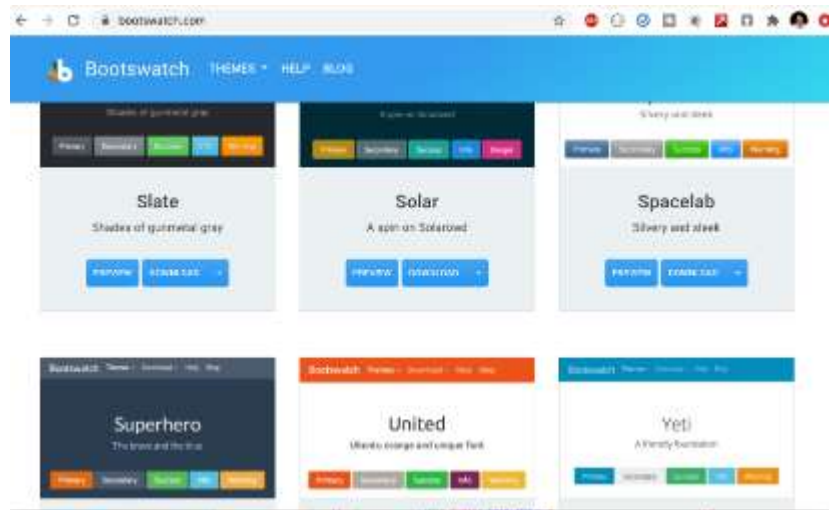
Bootstrap class names.



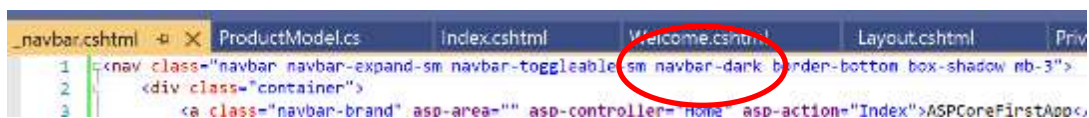
## Bootswatch Themes

It is fairly easy to change the theme in an ASP.NET project. Sites like **Bootswatch** provide customized versions of the bootstrap.min.css files.

1. Download a theme from **Bootswatch**. You should receive a file called **bootstrap.min.css**
2. Place the file in the **wwwroot > lib > bootstrap > dist > css** folder. You will have to **replace** the existing file.
3. Run the application and refresh the page. You may have to hold the **shift key** while **refreshing** to see the new CSS changes.



4. The CSS class name in the navbar is changed from **navbar-light** to **navbar-dark** to blend better with the chosen theme, *Superhero*.



5. Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.



## Alternatives to Bootstrap

Bootstrap may be the most popular of the CSS frameworks, but there are dozens of CSS frameworks available online. However, the pages autogenerated by Visual Studio for an ASP.NET project contain references to Bootstrap CSS class styles. You will have to change all these class names manually in order to use an alternative CSS framework. Here are three alternatives to Bootstrap to investigate.

### 1. Foundation by Zurb

Second only to Bootstrap in popularity. Foundation has most of the features of Bootstrap.

### 2. MaterializeCSS

Created and designed by Google, Material Design is a design language that combines the classic principles of successful design along with innovation and technology. Google's goal is to develop a system of design that allows for a unified user experience across all their products on any platform.

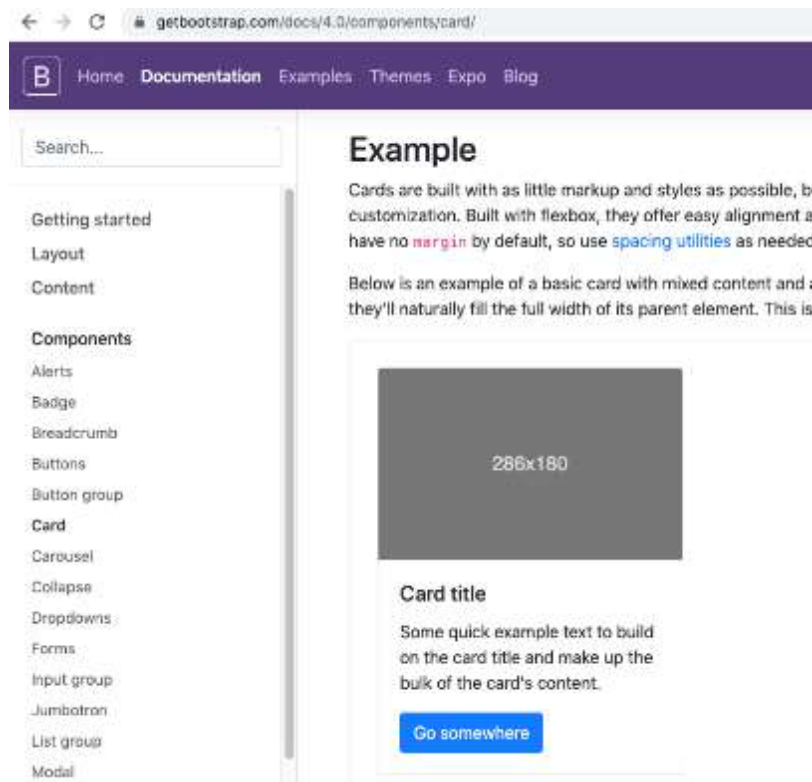
### 3. PureCSS

Known for being extremely small file sizes to reduce webpage load time. Minimal options, but perhaps more than enough to get the basics done.

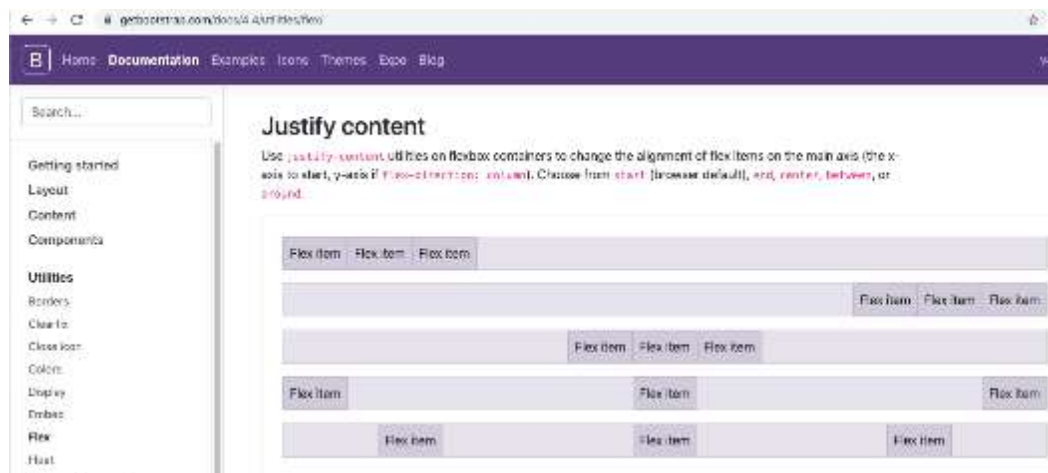
## Change a Table to a Flexbox container of cards

In this section, we will replace the table display with a flex box container full of Card objects that looks like the image below. Everything you need to complete the challenge is found in these pages:

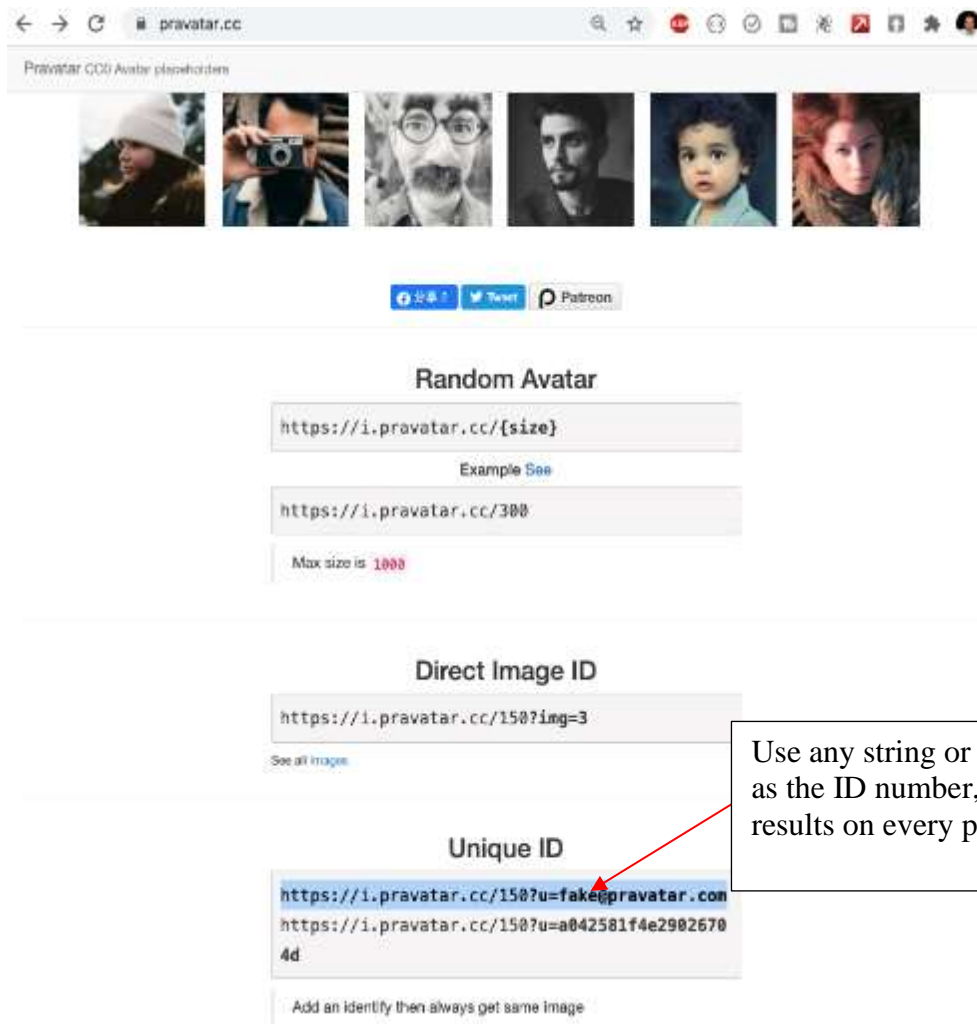
Bootstrap cards are described on their site as shown here.



Also shown is how Bootstrap uses Flex box CSS styling to arrange items within a container.



**Pravatar** is a free service that will help you generate random avatar photos for mock data. Our tutorial app uses products, not avatars, but the effect will still be noticeable.



Use any string or number here, such as the ID number, to get consistent results on every page refresh.

1. Modify the **index.cshtml** view page in the products folder to use:

- a flexbox container

- a set of Bootstrap cards instead of a table
- Pravatar images

```

1 @model IEnumerable<ASPCoreFirstApp.Models.ProductModel>
2
3 <div class="container d-flex flex-wrap">
4     @foreach (var item in Model)
5     {
6         <div class="card" style="width: 18rem; margin: 5px;">
7             
8             <div class="card-body">
9                 <h5 class="card-title">@item.Name</h5>
10
11                 <p class="card-text"> @Html.DisplayFor(modelItem => item.Price)</p>
12                 <a href="#" class="btn btn-primary">Go somewhere</a>
13             </div>
14         </div>
15     }
16 </div>
17

```

Flexbox container div

A set of bootstrap cards instead of a table

Pravatar images

2. Modify the layout to use another placeholder service. This time from **Loremflickr**, which uses images from Flickr instead of avatar pics.

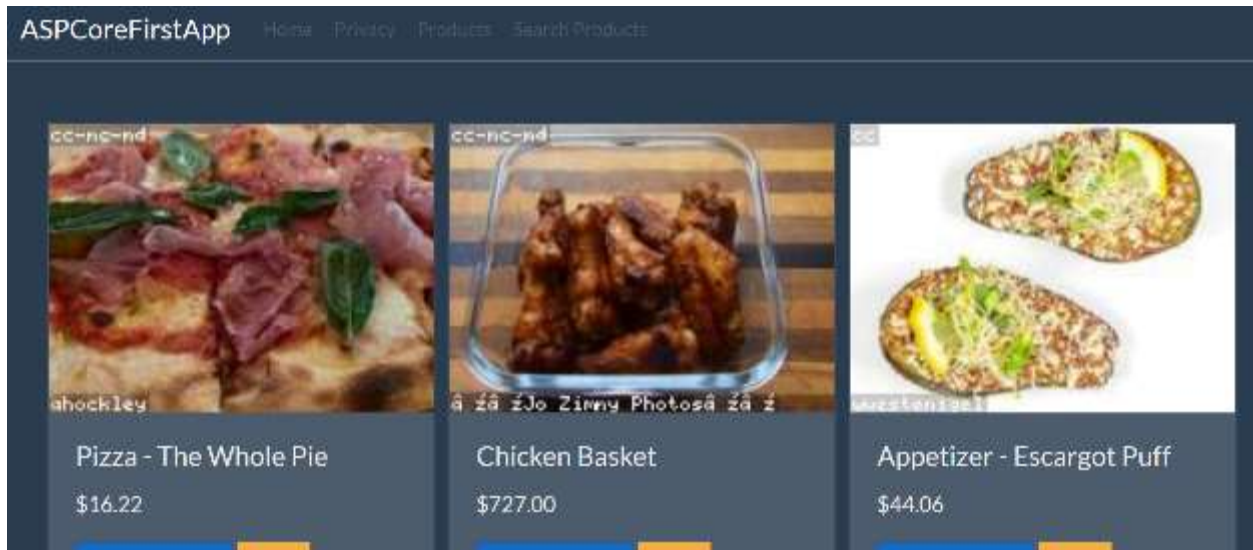
```

1 @model IEnumerable<ASPCoreFirstApp.Models.ProductModel>
2
3 <div class="container d-flex flex-wrap">
4     @foreach (var item in Model)
5     {
6         <div class="card" style="width: 18rem; margin: 5px;">
7             @{
8                 // get just the first word of the item.Name for searching purposes
9                 var s = item.Name;
10                // ternary operator - look it up
11                var firstWord = s.IndexOf(" ") > -1 ? s.Substring(0, s.IndexOf(" ")) : s;
12            }
13
14            @*look at loremflickr.com for details on how this works*@
15
16            
17            <div class="card-body">
18                <h5 class="card-title">@item.Name</h5>
19
20                <p class="card-text"> @Html.DisplayFor(modelItem => item.Price)</p>
21                <a href="#" class="btn btn-primary">Go somewhere</a>
22            </div>
23        </div>
24    }
25 </div>
26

```

Lorem flicker image

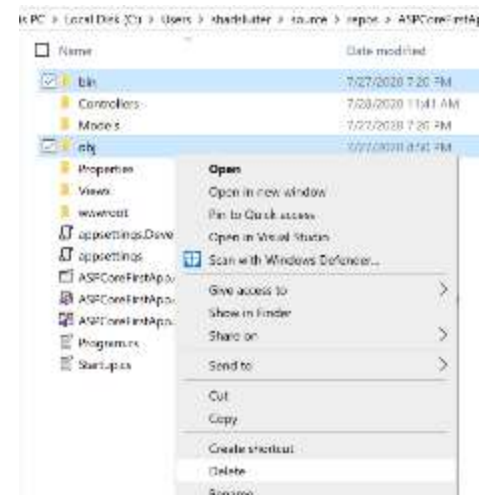
3. Run the application. You should see a grid of cards with custom pictures.



- Take a screenshot of the app at this stage. Paste it into a Microsoft Word document and caption the image with a brief explanation of what you just demonstrated.

### Deliverables:

- This activity has multiple parts. Complete all parts before submitting.
- Submit a Word document with screenshots of the application being run at each stage of development. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
- In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson.
- Submit a ZIP file of the project file. In order to save space, you can delete the bin and the obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.





## Part 3 CRUD SQL operations

### Goals:

In this activity, we will return to the Products app and complete the remaining features in the app – **Create, Get One Item, Update, and Delete**.

You will need the code from a previous activity, the **Products Application**.

### Show One Item

The first feature that we will complete is the **show one product** on its own **details** page.

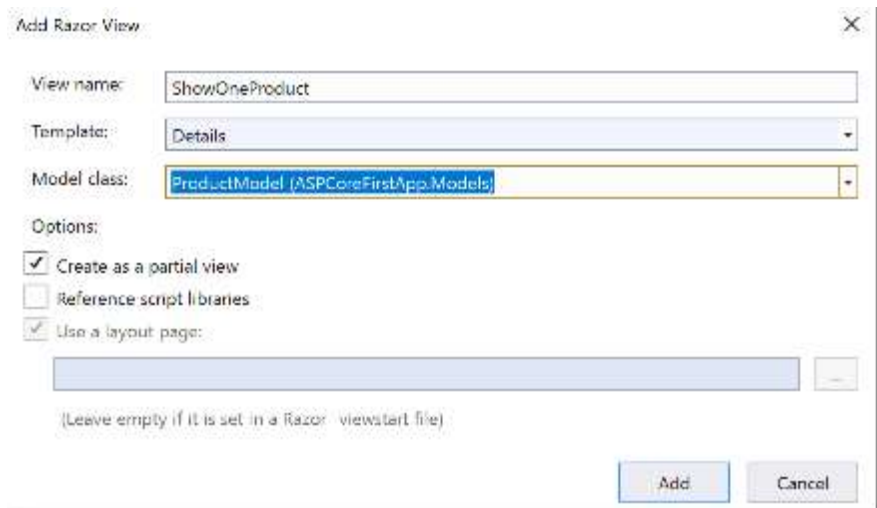
1. In the **ProductDAO** file, implement the **GetProductById** method as shown here.

```
115 public ProductModel GetProductById(int id)
116 {
117     ProductModel foundProduct = null;
118
119     // Uses prepared statements for security. @username @password are defined below
120     string sqlStatement = "SELECT * FROM dbo.Products WHERE Id = @id";
121
122     using (SqlConnection connection = new SqlConnection(connectionString))
123     {
124         SqlCommand command = new SqlCommand(sqlStatement, connection);
125
126         // define the values of the two placeholders in the sqlStatement string
127         command.Parameters.AddWithValue("@id", id);
128
129
130         try
131         {
132             connection.Open();
133             SqlDataReader reader = command.ExecuteReader();
134
135             while (reader.Read())
136             {
137                 foundProduct = new ProductModel((int)reader[0], (string)reader[1], (decimal)reader[2], (string)reader
138                 [3]);
139             }
140         }
141         catch (Exception ex)
142         {
143             Console.WriteLine(ex.Message);
144         }
145     }
146     return foundProduct;
147 }
```

2. In the **ProductsController**, add the method **ShowOneProduct** as shown.

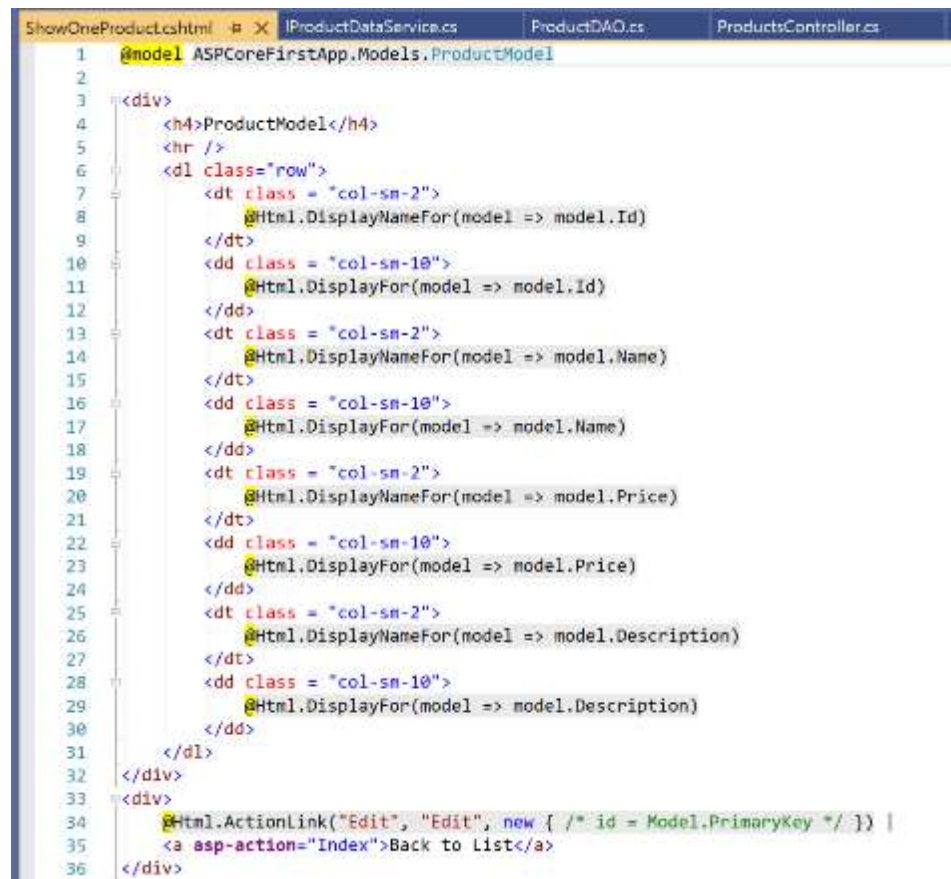
```
36
37 public IActionResult ShowOneProduct(int Id)
38 {
39     return View(repository.GetProductById(Id));
40 }
41
```

3. **Right-click** inside the ShowOneProduct method and choose **Add > View**. Create a new Razor View with the Details template and the ProductModel.



The 'Add Razor View' dialog box is shown. It has a title bar with a close button. The 'View name' field contains 'ShowOneProduct'. The 'Template' dropdown is set to 'Details'. The 'Model class' dropdown is set to 'ProductModel (ASPCoreFirstApp.Models)'. Under the 'Options' section, there are three checkboxes: 'Create as a partial view' (checked), 'Reference script libraries' (unchecked), and 'Use a layout page' (checked). There is a text box for a layout page name, which is empty. Below the text box is the instruction '(Leave empty if it is set in a Razor viewstart file)'. At the bottom right are 'Add' and 'Cancel' buttons.

You should see the following code in the new file that displays the default code for details of a single product record.



```
1 @model ASPCoreFirstApp.Models.ProductModel
2
3 <div>
4     <h4>ProductModel</h4>
5     <hr />
6     <dl class="row">
7         <dt class="col-sm-2">
8             @Html.DisplayNameFor(model => model.Id)
9         </dt>
10        <dd class="col-sm-10">
11            @Html.DisplayFor(model => model.Id)
12        </dd>
13        <dt class="col-sm-2">
14            @Html.DisplayNameFor(model => model.Name)
15        </dt>
16        <dd class="col-sm-10">
17            @Html.DisplayFor(model => model.Name)
18        </dd>
19        <dt class="col-sm-2">
20            @Html.DisplayNameFor(model => model.Price)
21        </dt>
22        <dd class="col-sm-10">
23            @Html.DisplayFor(model => model.Price)
24        </dd>
25        <dt class="col-sm-2">
26            @Html.DisplayNameFor(model => model.Description)
27        </dt>
28        <dd class="col-sm-10">
29            @Html.DisplayFor(model => model.Description)
30        </dd>
31    </dl>
32 </div>
33 <div>
34     @Html.ActionLink("Edit", "Edit", new { /* id = Model.PrimaryKey */ }) |
35     <a asp-action="Index">Back to list</a>
36 </div>
```

4. In the **Index.cshtml** file, modify the `<a>` tag inside the product card. This will direct the browser to display the new view we just created.

```
1  @model IEnumerable<ASPCoreFirstApp.Models.ProductModel>
2
3
4  <div class="container d-flex flex-wrap">
5      @foreach (var item in Model)
6      {
7          <div id="prod-card-@item.Id" class="card" style="width: 18rem; margin: 5px;">
8              @
9              // get just the first word of the item.Name for searching purposes
10             var s = item.Name;
11             // ternary operator - look it up
12             var firstWord = s.IndexOf(" ") > -1 ? s.Substring(0, s.IndexOf(" ")) : s;
13
14
15             @*look at loremflickr.com for details on how this works*@
16
17             
18             <div class="card-body">
19                 <h5 class="card-title">@item.Name</h5>
20
21                 <p class="card-text"> @Html.DisplayFor(modelItem => item.Price)</p>
22                 <a href="/products/ShowOneProduct/@item.Id" class="btn btn-primary">Show Details</a>
23             </div>
24         </div>
25     }
26 </div>
27
28
```

## Edit an Item

Next, we will create the ability for the user to update any of the items in the product catalog. We need to do the following:

1. Create the appropriate method in the ProductsDAO.
2. Create two event handlers in the Controller, (1) show the edit form (2) process the update.
3. Create a View to show an edit for the product.
4. Create a button on the products page to allow navigation to the edit form.

## New Method in Products DAO

1. In the **ProductsDAO**, implement the **Update** method as shown here.

```

154 public int Update(ProductModel product)
155 {
156     // Returns -1 if the update fails.
157     int newIdNumber = -1;
158     using (SqlConnection connection = new SqlConnection(connectionString))
159     {
160         string query = "UPDATE dbo.Products SET Name = @Name, Price = @Price, Description = @Description WHERE Id = @Id";
161
162         SqlCommand myCommand = new SqlCommand(query, connection);
163         myCommand.Parameters.AddWithValue("@Id", product.Id);
164         myCommand.Parameters.AddWithValue("@Name", product.Name);
165         myCommand.Parameters.AddWithValue("@Price", product.Price);
166         myCommand.Parameters.AddWithValue("@Description", product.Description);
167
168         try
169         {
170             connection.Open();
171
172             newIdNumber = Convert.ToInt32(myCommand.ExecuteScalar());
173         }
174         catch (Exception ex)
175         {
176             Console.WriteLine(ex.Message);
177         }
178         return newIdNumber;
179     }
180 }
181
182
183
184 }

```

2. In the **Products** Controller, add the following two methods: **ShowEditForm** and **ProcessEdit**

```

41
42 public IActionResult ShowEditForm(int Id)
43 {
44     return View(repository.GetProductById(Id));
45 }
46
47 public IActionResult ProcessEdit(ProductModel product)
48 {
49     repository.Update(product);
50     return View("Index", repository.AllProducts());
51 }
52

```

3. Right-click in the **ShowEditForm** method and choose **Add > View**. Choose the **Edit** template with the **ProductModel** for the Model class.

Add Razor View

View name: ShowEditForm

Template: Edit

Model class: ProductModel (ASPCoreFirstApp.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

4. Change the form action to **ProcessEdit**

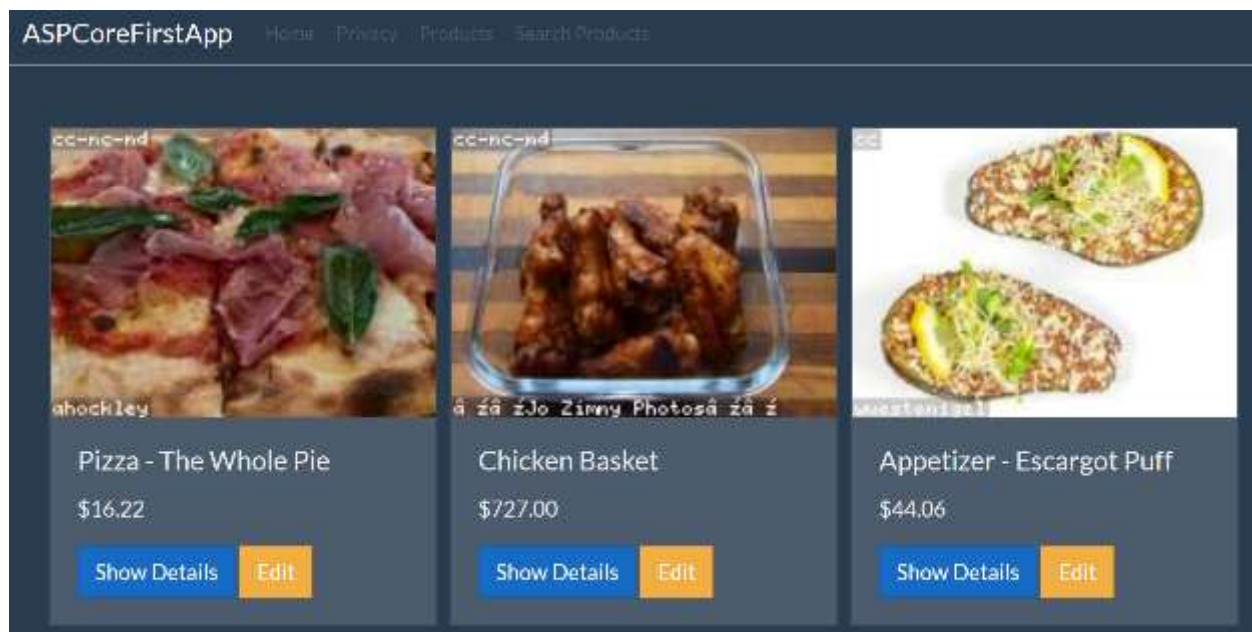
```
1 @model ASPCoreFirstApp.Models.ProductModel
2
3 <h4>ProductModel</h4>
4 <hr />
5 <div class="row">
6     <div class="col-md-4">
7         <form asp-action="ProcessEdit">
8             <div asp-validation-summary="ModelOnly" class="text-danger"></div>
9             <div class="form-group">
10                 <label asp-for="Id" class="control-label"></label>
11                 <input asp-for="Id" class="form-control" />
12                 <span asp-validation-for="Id" class="text-danger"></span>
13             </div>
14             <div class="form-group">
15                 <label asp-for="Name" class="control-label"></label>
16                 <input asp-for="Name" class="form-control" />
17                 <span asp-validation-for="Name" class="text-danger"></span>
18             </div>
19             <div class="form-group">
20                 <label asp-for="Price" class="control-label"></label>
21                 <input asp-for="Price" class="form-control" />
22                 <span asp-validation-for="Price" class="text-danger"></span>
23             </div>
24             <div class="form-group">
25                 <label asp-for="Description" class="control-label"></label>
26                 <input asp-for="Description" class="form-control" />
27                 <span asp-validation-for="Description" class="text-danger"></span>
28             </div>
29             <div class="form-group">
30                 <input type="submit" value="Save" class="btn btn-primary" />
31             </div>
32         </form>
33     </div>
34 </div>
35
36 <div>
37     <a asp-action="Index">Back to List</a>
38 </div>
39
```



5. Add another **button** to the **index.cshtml** file to allow for **edits**.

```
1 @model IEnumerable<ASPCoreFirstApp.Models.ProductModel>
2
3
4 <div class="container d-flex flex-wrap">
5     @foreach (var item in Model)
6     {
7         <div id="prod-card-@item.Id" class="card" style="width: 18rem; margin: 5px;">
8             @if
9             {
10                 // get just the first word of the item.Name for searching purposes
11                 var s = item.Name;
12                 // ternary operator - look it up
13                 var firstWord = s.IndexOf(" ") > -1 ? s.Substring(0, s.IndexOf(" ")) : s;
14             }
15
16             @*look at loremflickr.com for details on how this works*@
17
18             
19             <div class="card-body">
20                 <h5 class="card-title">@item.Name</h5>
21
22                 <p class="card-text"> @Html.DisplayFor(modelItem => item.Price)</p>
23                 <a href="/products/ShowOneProduct/@item.Id" class="btn btn-primary">Show Details</a>
24                 <a href="/products/ShowEditForm/@item.Id" class="btn btn-warning">Edit</a>
25             </div>
26         </div>
27     }
28 </div>
```

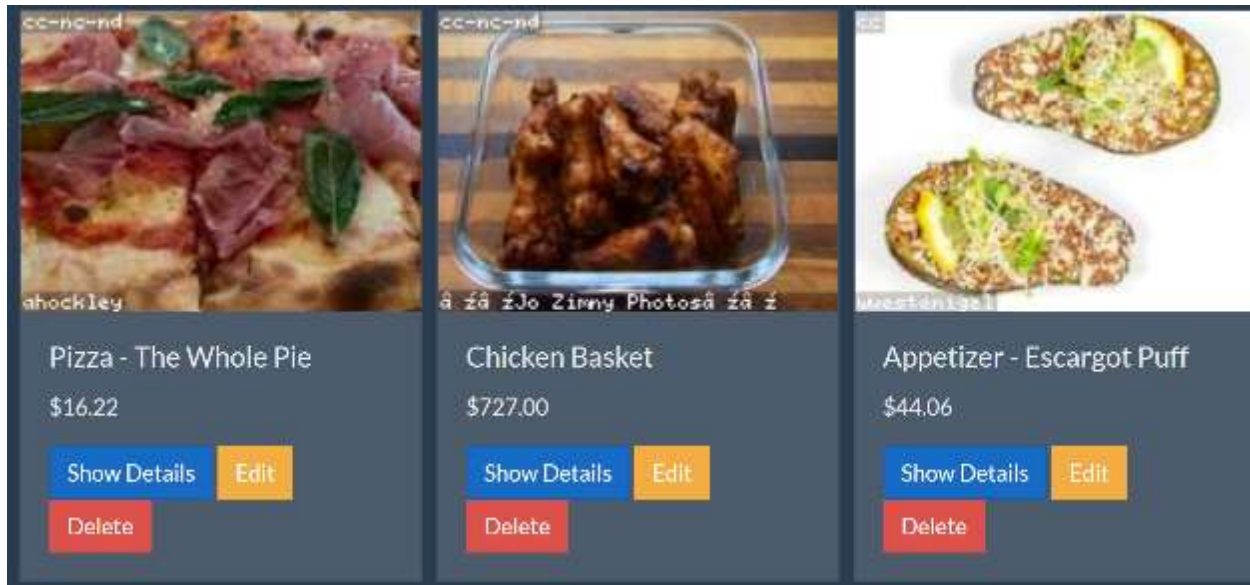
You should be able to run the program, show the **details** of one item and **edit** an item.



## Programming Challenge

The last database operation to perform is the **delete** command. Add the following changes to the app:

1. Implement the **Delete** method in **ProductsDAO**.
2. Add a **Delete** method to the controller.
3. Add a **Delete** button to the Index view.



For reference materials on SQL statements, see Microsoft's documentation at <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/retrieving-and-modifying-data>

## Deliverables:

1. This activity has multiple parts. Complete all parts before submitting.
2. Submit a Microsoft Word document with screenshots of the application being run at each stage of development. Show each screen of the output and put a caption under each picture explaining what is being demonstrated.
3. In the same document, in one paragraph, write a summary of the key concepts that were demonstrated in this lesson.
4. Submit a ZIP file of the project file. In order to save space, you can delete the bin and the obj folders of the project. These folders contain the compiled version of the application and are automatically regenerated each time the build or run commands are executed.

