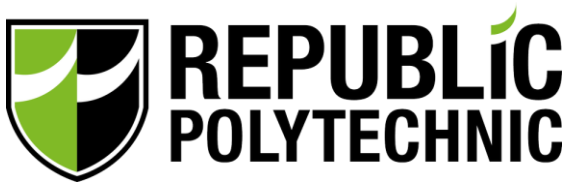
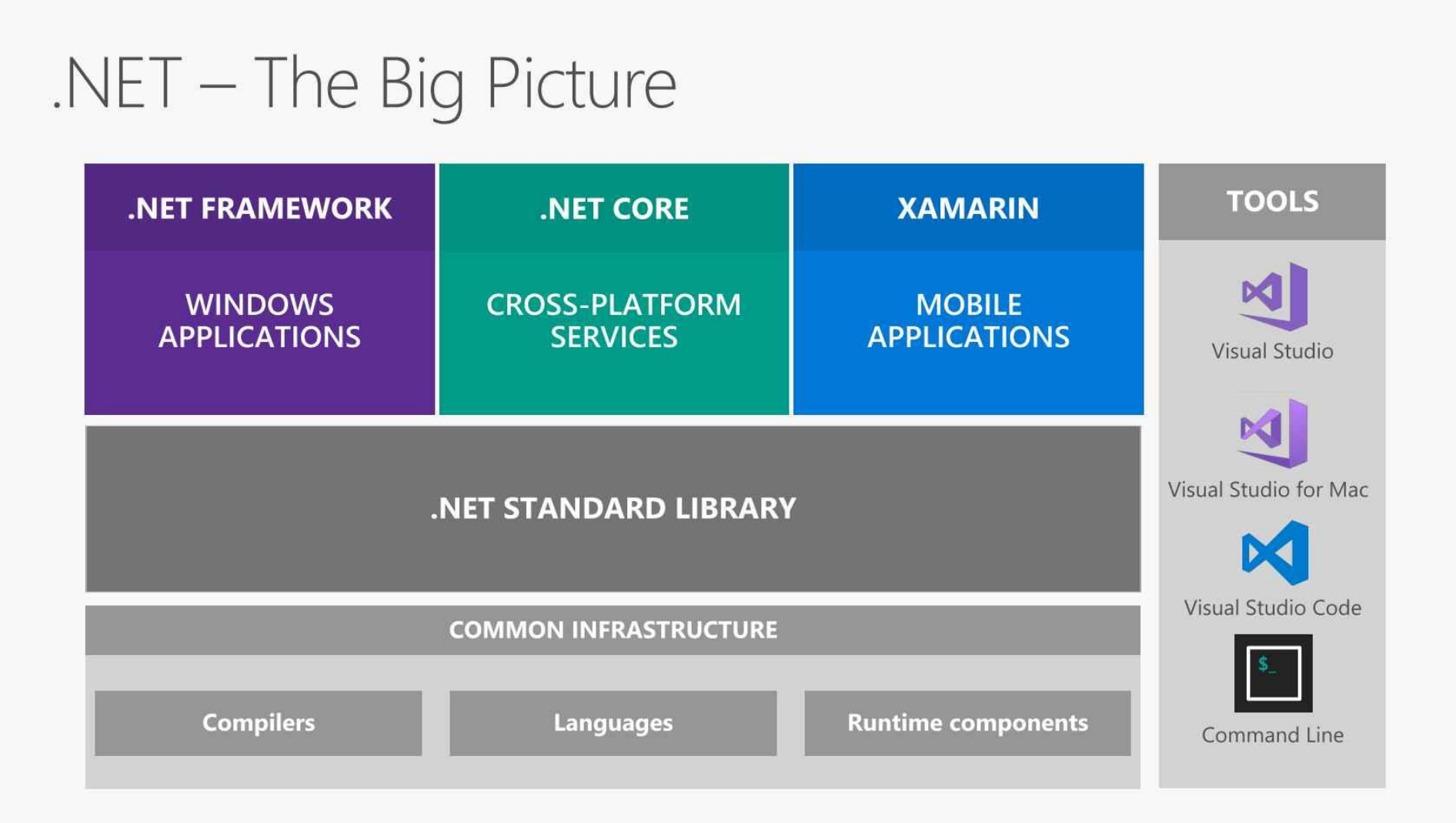


Lesson 1

Introduction to .NET, HTML and CSS

Microsoft .NET 7

.NET ecosystem



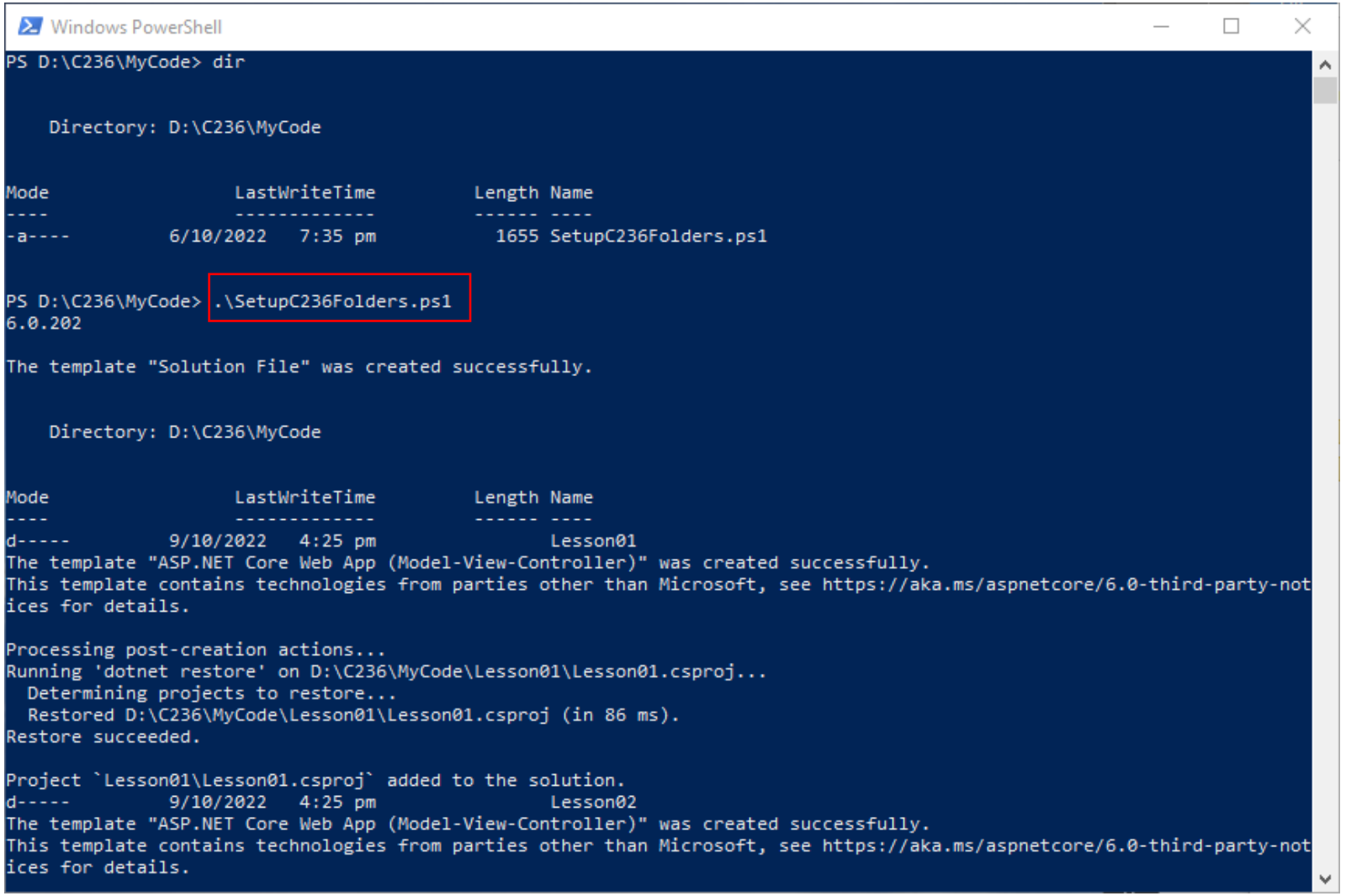
Create solution and projects

Creating the solution & project structure

A script in the student package will create a **solution file** and **13 projects** for you automatically.

- Use Windows explorer to copy the **SetupC236Folders.ps1** script from your LEO2.0 download location and paste it into your **C236\MyCode** folder.
- With the **C236\MyCode** folder still selected in Windows Explorer, type **CTRL-L**. Your cursor will be placed inside the address bar of Windows Explorer.
- Type **PowerShell** and click enter (Don't think just type).
- PowerShell will start inside the newly created folder.
- Run the script by typing **.\SetupC236Folders.ps1** from the PowerShell prompt.

Creating the solution & project structure



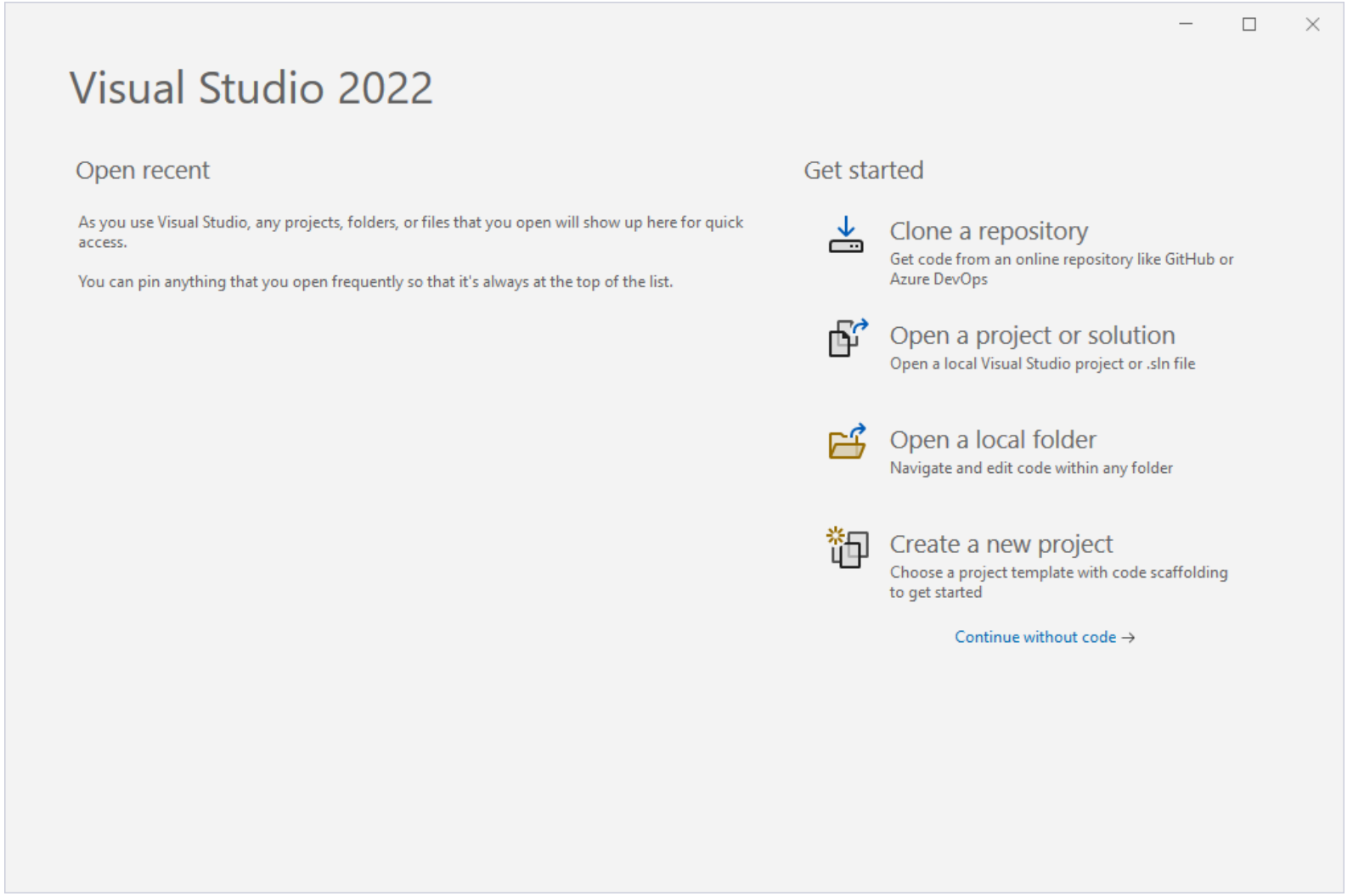
Apprentice Activity

Complete worksheet activities 1 – 8

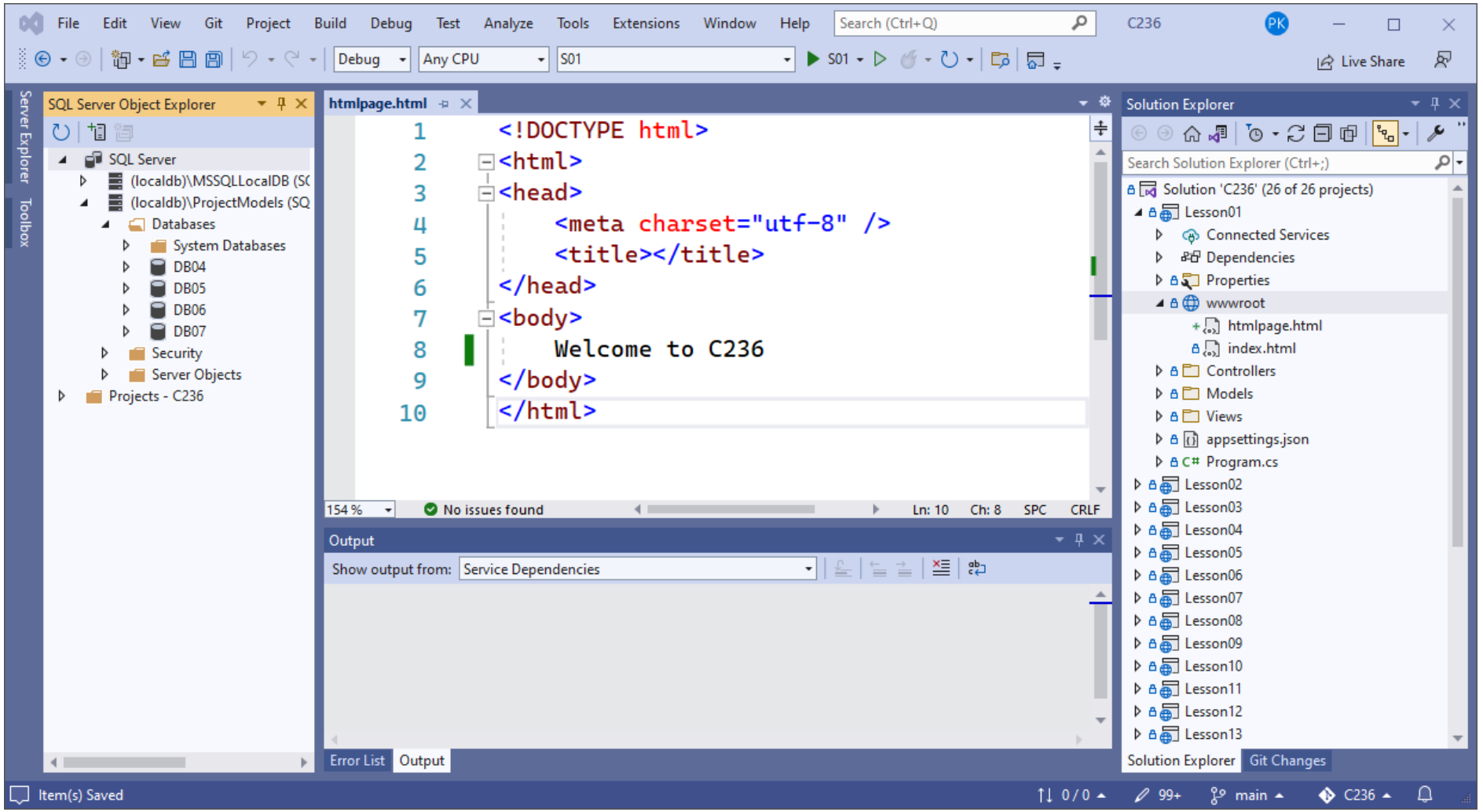
(10 mins)

Visual Studio 2022

VS2022 IDE


Visual Studio 2022
Open recent
As you use Visual Studio, any projects, folders, or files that you open will show up here for quick access.
You can pin anything that you open frequently so that it's always at the top of the list.
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

VS2022 IDE


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
C236
Debug
Any CPU
S01
Live Share
SQL Server Object Explorer
SQL Server
(localdb)\MSSQLLocalDB (SC
(localdb)\ProjectModels (SQ
Databases
System Databases
DB04
DB05
DB06
DB07
Security
Server Objects
Projects - C236
htmlpage.html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title></title>
</head>
<body>
Welcome to C236
</body>
</html>
154 %
No issues found
Ln: 10 Ch: 8 SPC CRLF
Output
Show output from: Service Dependencies
Error List
Output
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'C236' (26 of 26 projects)
Lesson01
Connected Services
Dependencies
Properties
wwwroot
htmlpage.html
index.html
Controllers
Models
Views
appsettings.json
Program.cs
Lesson02
Lesson03
Lesson04
Lesson05
Lesson06
Lesson07
Lesson08
Lesson09
Lesson10
Lesson11
Lesson12
Lesson13
Solution Explorer
Git Changes
Item(s) Saved
0 / 0
99+
main
C236

Lecturer demo: VS 2022

Topics to demonstrate:

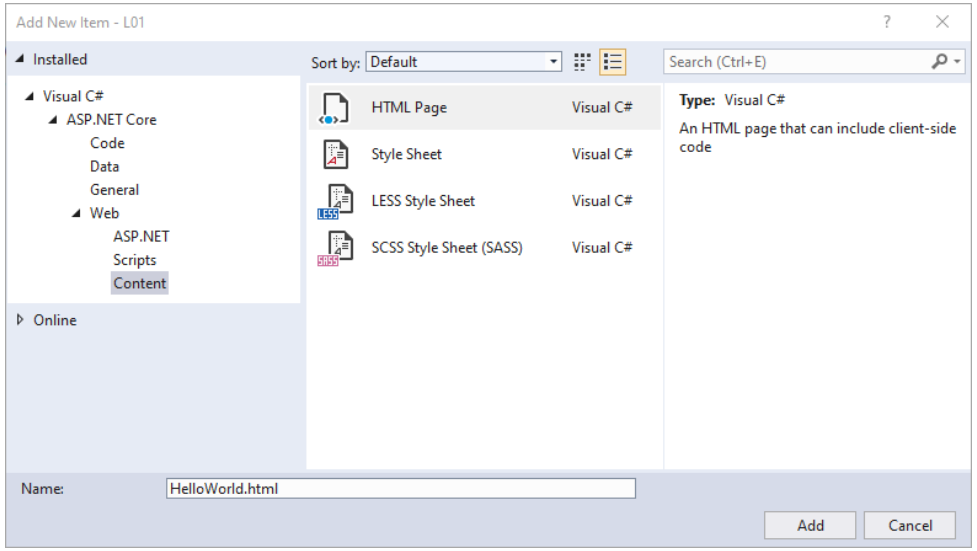
Solution and project structure, moving windows, resetting layout, editing, auto-format, split screen, collapse all, show all files, set start-up project. Also shortcuts for save (Ctrl S), save all (Ctrl Shft S) format (Ctrl K D) and show task list (Ctrl \ T)

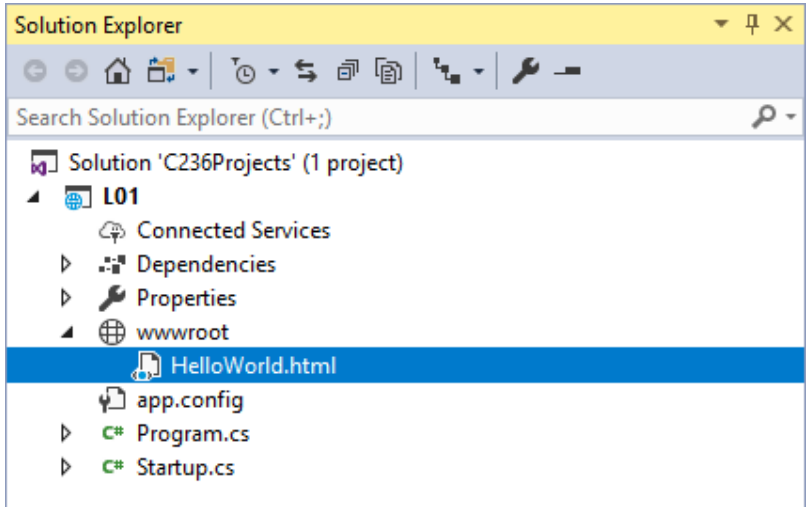
Create a new HTML page

Right-click on wwwroot

- Click Add > New Items…
- Select HTML Page and enter HelloWorld.html and click [Add]

The HelloWorld.html page will appear in the Solution Explorer





Editing the HTML page

Edit the code as follows

HelloWorld.html*

```
1   <!DOCTYPE html>
2   <html>
3   <head>
4       <meta charset="utf-8" />
5       <title>Hello World</title>
6   </head>
7   <body>
8       This is my first HTML Page.
9   </body>
10  </html>
```

Right-click **HelloWorld.html** in Solution Explorer

- Right click HelloWorld.html > **View in Browser**

Apprentice Activity

Perform Task 9 – 13

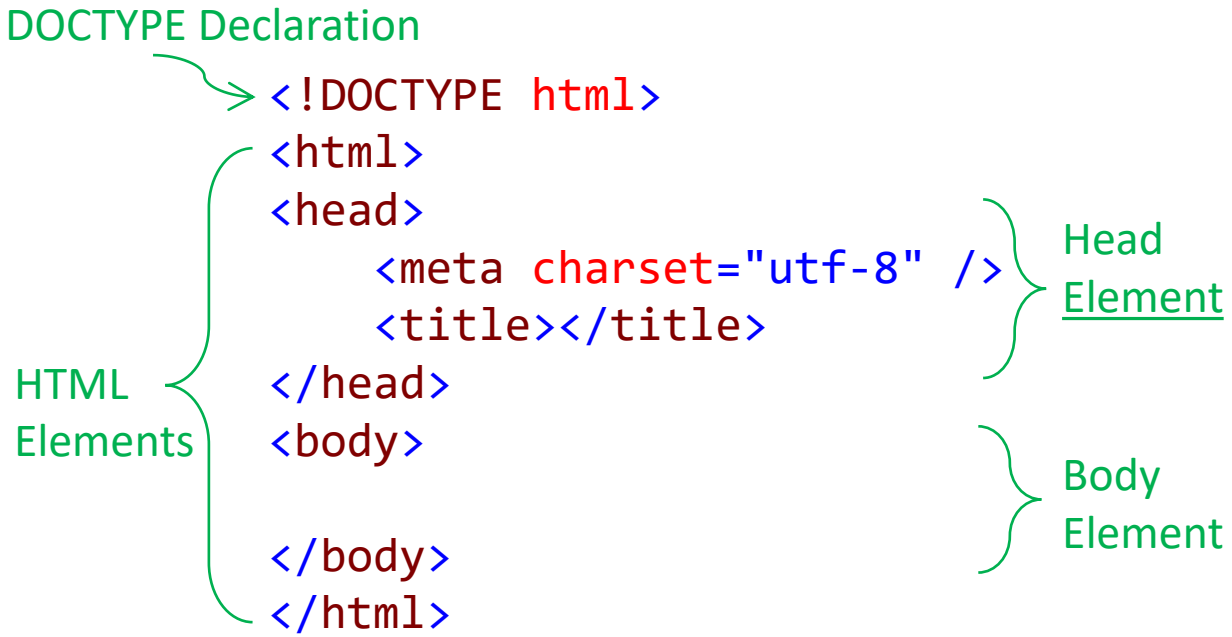
(5-10 minutes)

HTML

What is HTML?

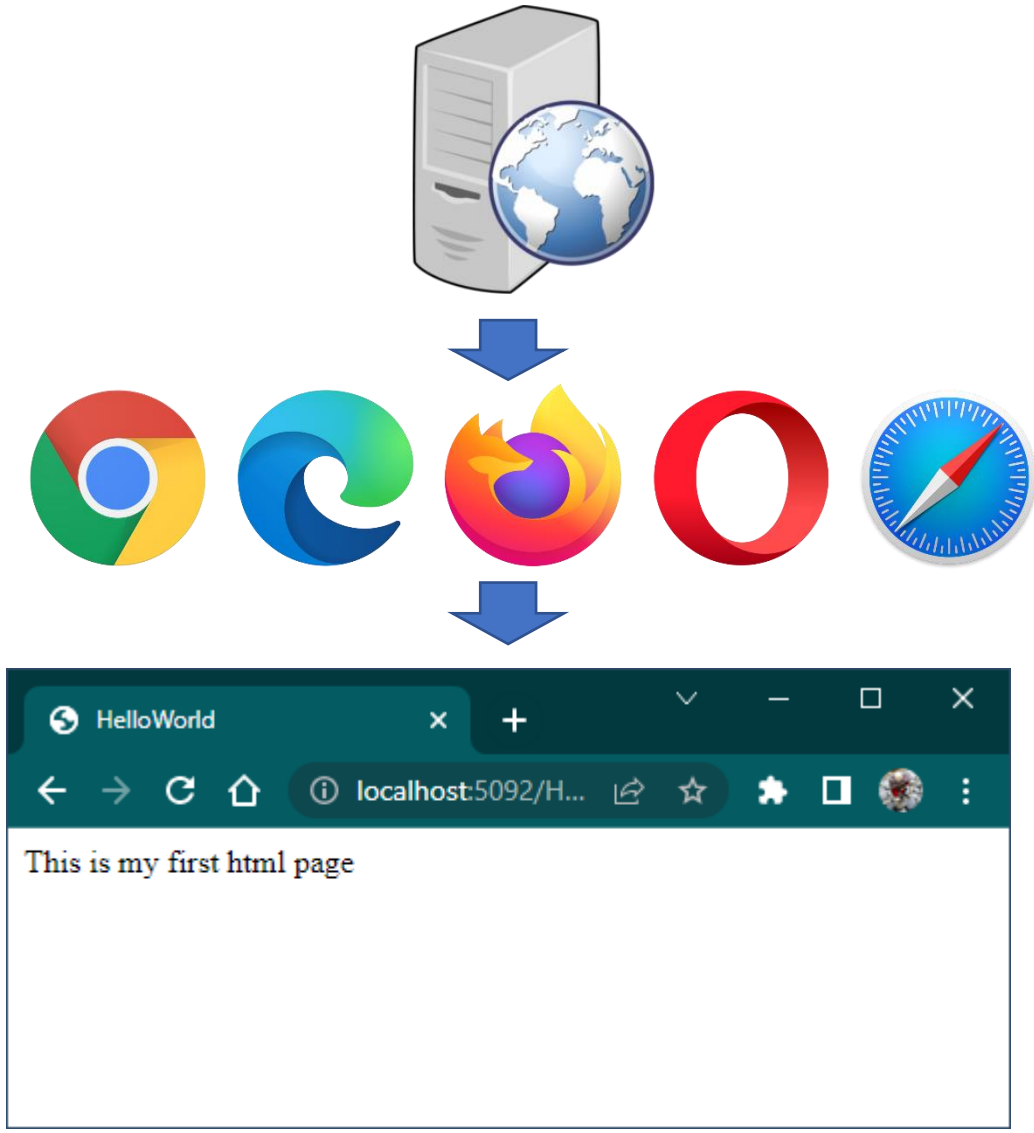
Hypertext Markup Language (HTML)

- Web pages are written in HTML
- Browsers display HTML as web pages
- HTML defines the structure and the contents of a Web page



HTML and web pages

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>HelloWorld</title>
</head>
<body>
    This is my first html page
</body>
</html>
```



HTML elements

HTML elements are structures that describe parts of an HTML document.

- HTML elements are written with a **start tag**, with an **end tag**, with the **content** in between
```
<tagname>content</tagname>
```

- The HTML **element** is everything from the start tag to the end tag:
```
<title>Hello World</title>
```

- Some HTML elements do not have content and therefore do not need an end tag.
```
<br />
```

- HTML elements can be nested in other elements.
```
<head>
<meta charset="utf-8" />
<title>Hello World</title>
</head>
```

Test your understanding

- Please start the C236 Lesson 01 SCORM package in LEO2.0 in a **new browser window**. **Pin** the window to make it easier to take the second quiz.
- Take the first quiz - **Mini Quiz 01 - HTML Elements**.
- Do not attempt the remaining quizzes until instructed to do so as the material is not yet covered.

Notes:

- Quizzes are formative in nature.
- Quiz answers do not contribute to your continuous assessment grade (CAG).
- You may attempt the SCROM package more than once.

Quiz 01 answers §

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Hello World</title>
</head>
<body>
    This is my first html page
</body>
</html>
```

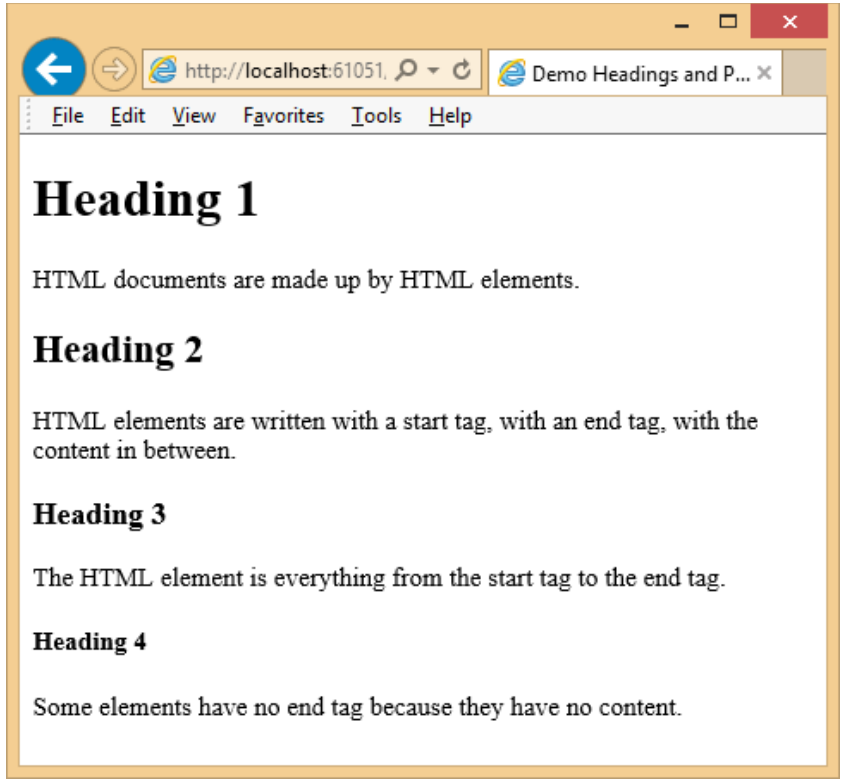
How many elements are there? **5**

How many nested elements are there? **4**

What are the elements? **html head title body meta**

Headings and paragraphs

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Demo Headings and Paragraphs</title>
</head>
<body>
    <h1>Heading 1</h1>
    <p>HTML documents are made up by HTML elements.</p>

    <h2>Heading 2</h2>
    <p>
        HTML elements are written with a start tag,
        with an end tag, with the content
        in between.
    </p>

    <h3>Heading 3</h3>
    <p>
        The HTML element is everything from the
        start tag to the end tag.
    </p>

    <h4>Heading 4</h4>
    <p>
        Some elements have no end tag because
        they have no content.
    </p>
</body>
</html>
```

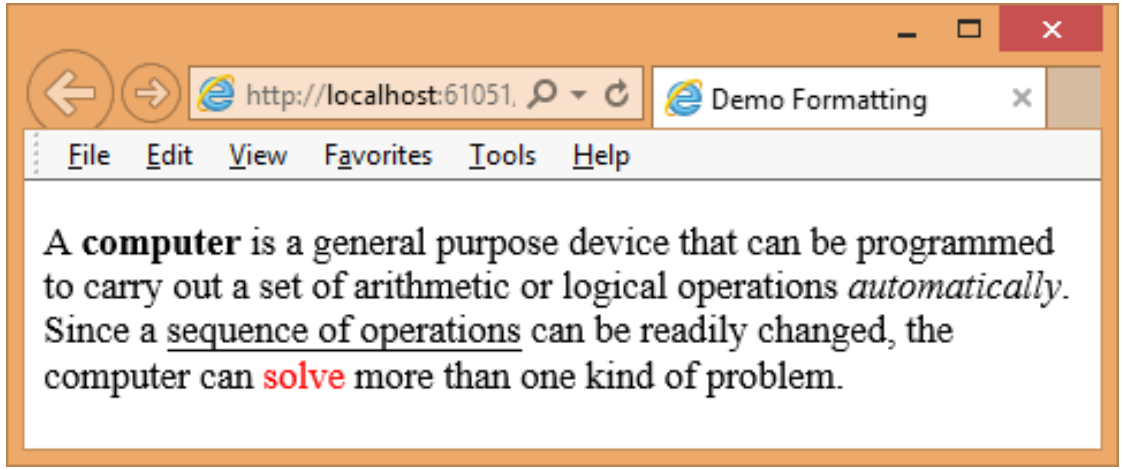


Apprentice Activity

Complete worksheet exercise A

Formatting

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Demo Formatting</title>
</head>
<body>
    <p>
        A <b>computer</b> is a general purpose device that can be programmed to
        carry out a set of arithmetic or logical operations <i>automatically</i>.
        Since a <u>sequence of operations</u> can be readily changed, the computer
        can <span style="color:red;">solve</span> more than one kind of problem.
    </p>
</body>
</html>
```

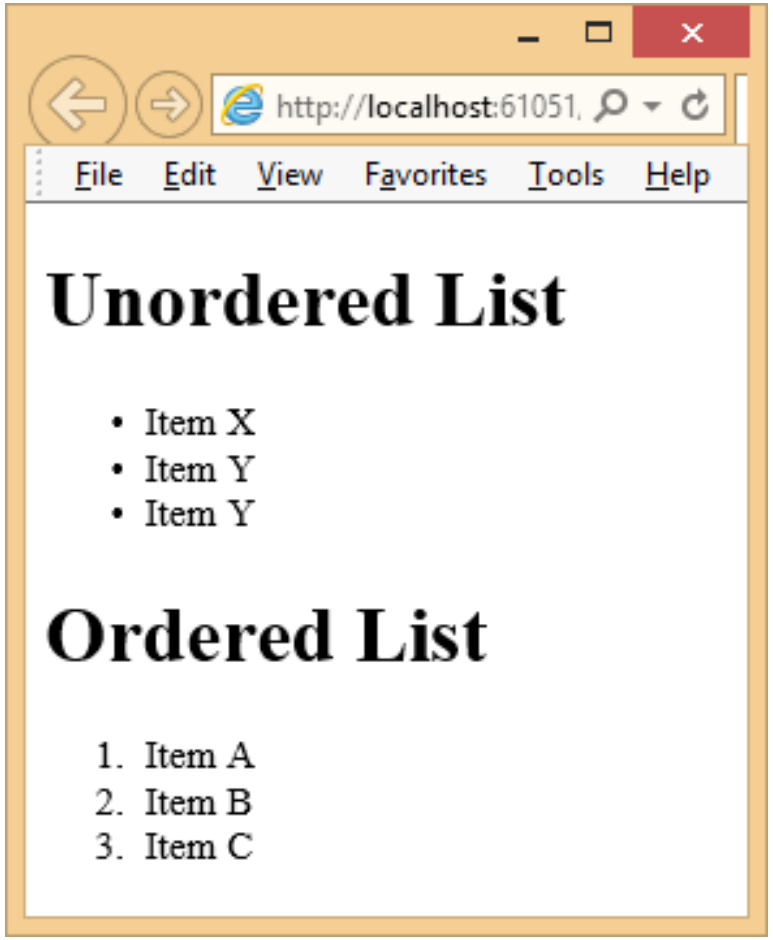


Apprentice Activity

Complete worksheet exercise B

Lists

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Lists</title>
</head>
<body>
    <h1>Unordered List</h1>
    <ul>
        <li>Item X</li>
        <li>Item Y</li>
        <li>Item X</li>
    </ul>
    <ol>
        <li>Item A</li>
        <li>Item B</li>
        <li>Item C</li>
    </ol>

</body>
</html>
```

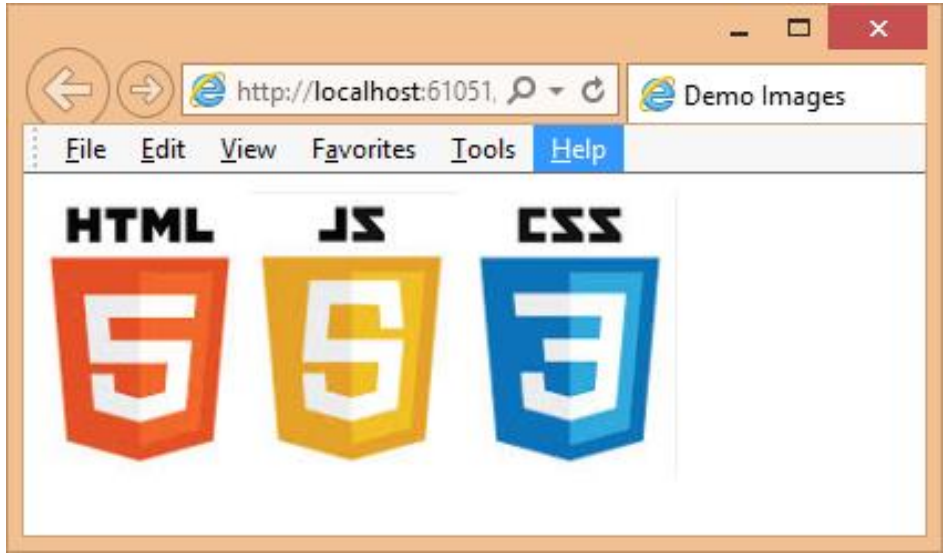


Apprentice Activity

Complete worksheet exercise C

Images

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Demo Images</title>
</head>
<body>
    <img src="/images/shield1.png" width="50" height="60"/>
    <img src="/images/shield2.png" />
    <img src="/images/shield3.png" />
</body>
</html>
```



Apprentice Activity

Complete worksheet exercise D

HTML attributes

HTML elements can have attributes

- Attributes provide additional information about an element
- Attributes are always specified in the start tag
- Attributes come in key/value pairs like: **key="value"**

Example 1

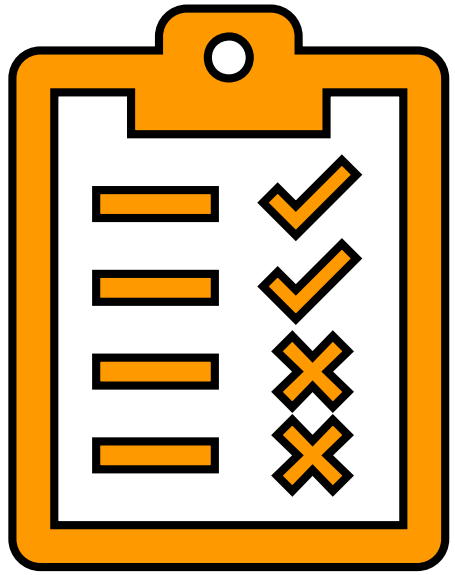
- `<img src="/images/shield1.png" width="50" height="60"/>`
- The filename of the source (**src**), and the size of the image (**width** and **height**) are all provided as **attributes**

Example 2

- `<p title="About RP">The Republic Polytechnic was set up on 1 August 2002</p>`
- The **<p>** element has a **title** attribute and the value is "About RP"

Test your understanding

- Return to your pinned C236 Lesson 01 SCORM package window and take the second quiz - **Mini Quiz 02 - HTML Attributes**.
- Do not attempt the remaining quizzes.



Notes:

- Quizzes are formative in nature.
- Quiz answers do not contribute to your continuous assessment grade (CAG).
- You may attempt the SCROM package more than once.

Quiz 02 answers §

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Demo Images</title>
</head>
<body>
    <img src="/images/shield1.png" width="50" height="60"/>
    <img src="/images/shield2.png" />
    <img src="/images/shield3.png" />
</body>
</html>
```

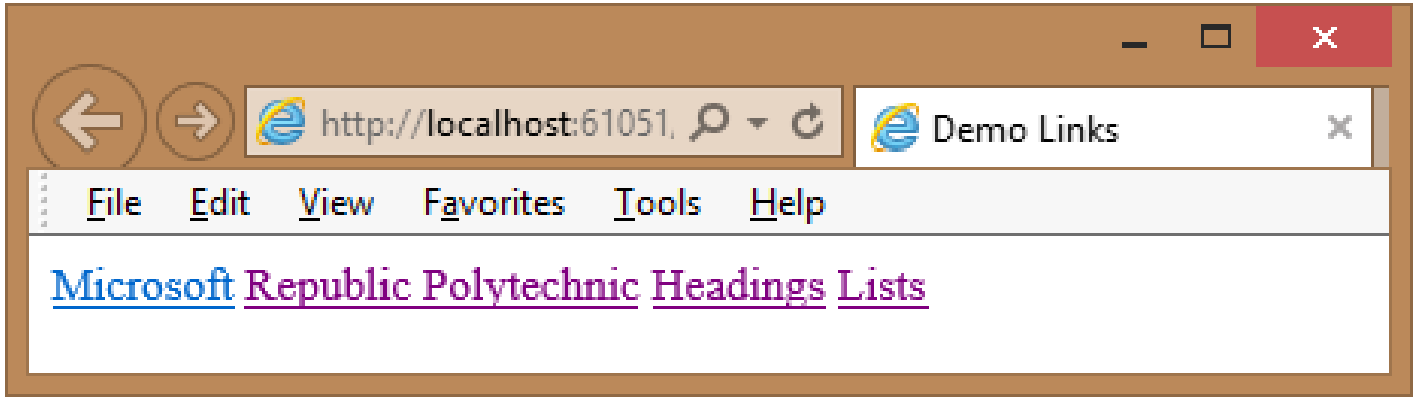
How many attributes are there? **4**

Which two elements have attributes? **meta img**

What are the attribute names? **charset src width height**

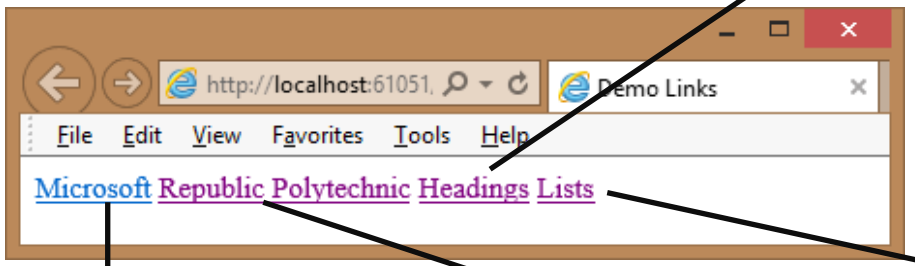
Links

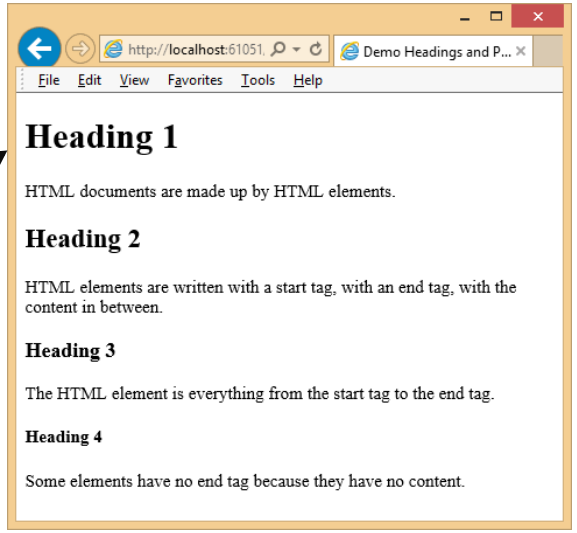
```
<html>
<head>
    <meta charset="utf-8" />
    <title>exerciseE</title>
</head>
<body>
    <a href="http://www.microsoft.com">Microsoft</a>
    <a href="http://www.rp.edu.sg">Republic Polytechnic</a>
    <a href="exerciseA.html">Headings</a>
    <a href="exerciseC.html">Lists</a>
</body>
</html>
```

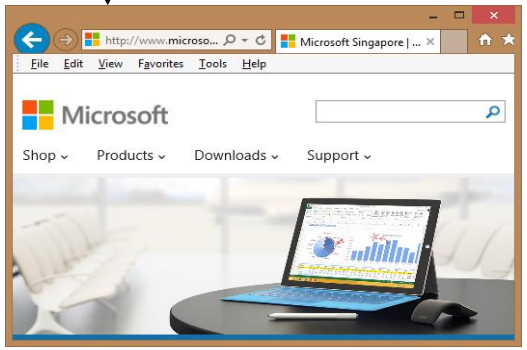


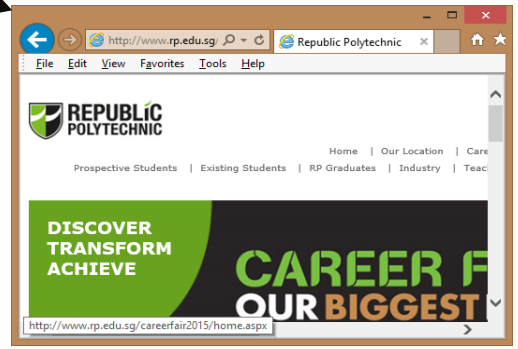
Links

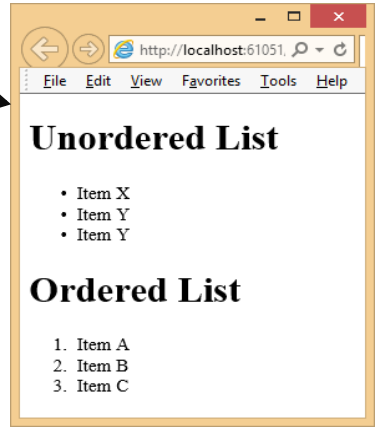
```
<html>
<head>
    <meta charset="utf-8" />
    <title>exerciseE</title>
</head>
<body>
    <a href="http://www.microsoft.com">Microsoft</a>
    <a href="http://www.rp.edu.sg">Republic Polytechnic</a>
    <a href="exerciseA.html">Headings</a>
    <a href="exerciseC.html">Lists</a>
</body>
</html>
```











Apprentice Activity

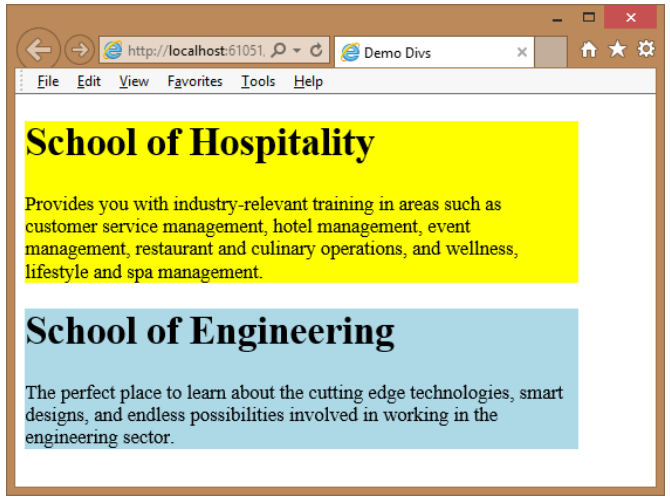
Complete worksheet exercise E

Element: div

The element is a block element that can be used as a container for other HTML elements. **In-line CSS** is used to add a background colour to the divs.

It can be used to group <h1> and <p> into a block.

```
<div style="width: 450px; background-color: yellow;">
    <h1>School of Hospitality</h1>
    <p>
        Provides you with industry-relevant training in areas
        such as customer service management, hotel management,
        event management, restaurant and culinary operations,
        and wellness, lifestyle and spa management.
    </p>
</div>

<div style="width: 450px; background-color: lightblue;">
    <h1>School of Engineering</h1>
    <p>
        The perfect place to learn about the cutting
        edge technologies, smart designs, and
        endless possibilities involved in working
        in the engineering sector.
    </p>
</div>
```



CSS box model

Margin

- Space outside/around element

Border

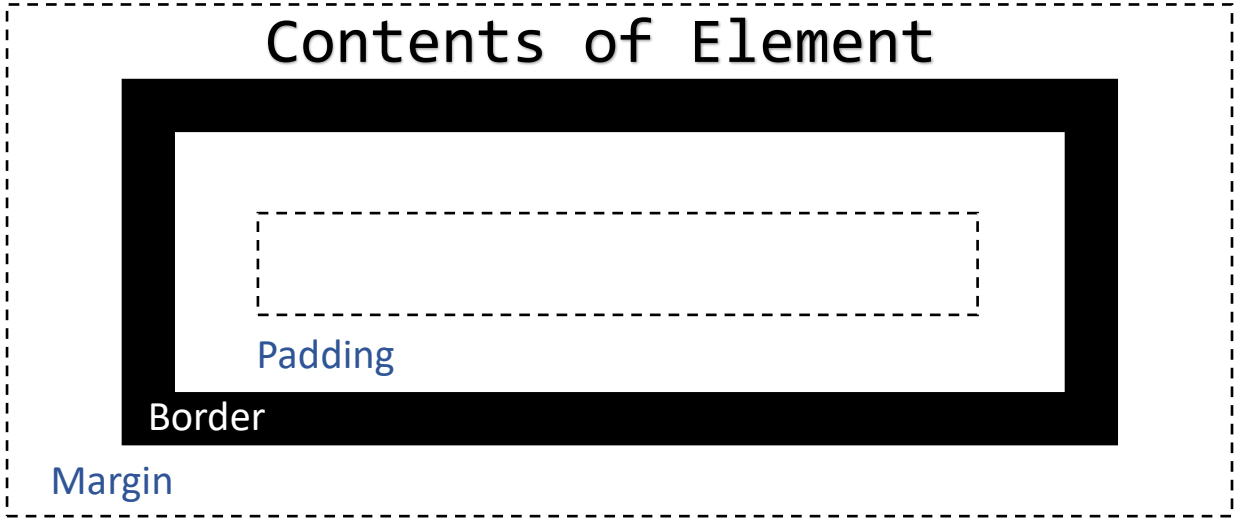
- Visible border around element

Padding

- Space within an element, but outside the content

Internal CSS

- Use **internal CSS** to apply a style to multiple elements. In the example given all **div** elements will have a margin of 12px.




```
<head>
    <title>exerciseF</title>
    <style>
        div {
            margin:12px;
        }
    </style>
</head>
```

CSS Units

Absolute Sizes

- px
  - Pixels (1px = 1/96 of an inch)
- pt
  - Points (1pt = 1/72 of an inch)
- cm, mm, in
  - Centimeters, Millimeters and Inches respectively

Relative Sizes

- em
  - Based on the font size of the current element
- rem
  - Based on the font size of the html element (16px by default)

For more units, see

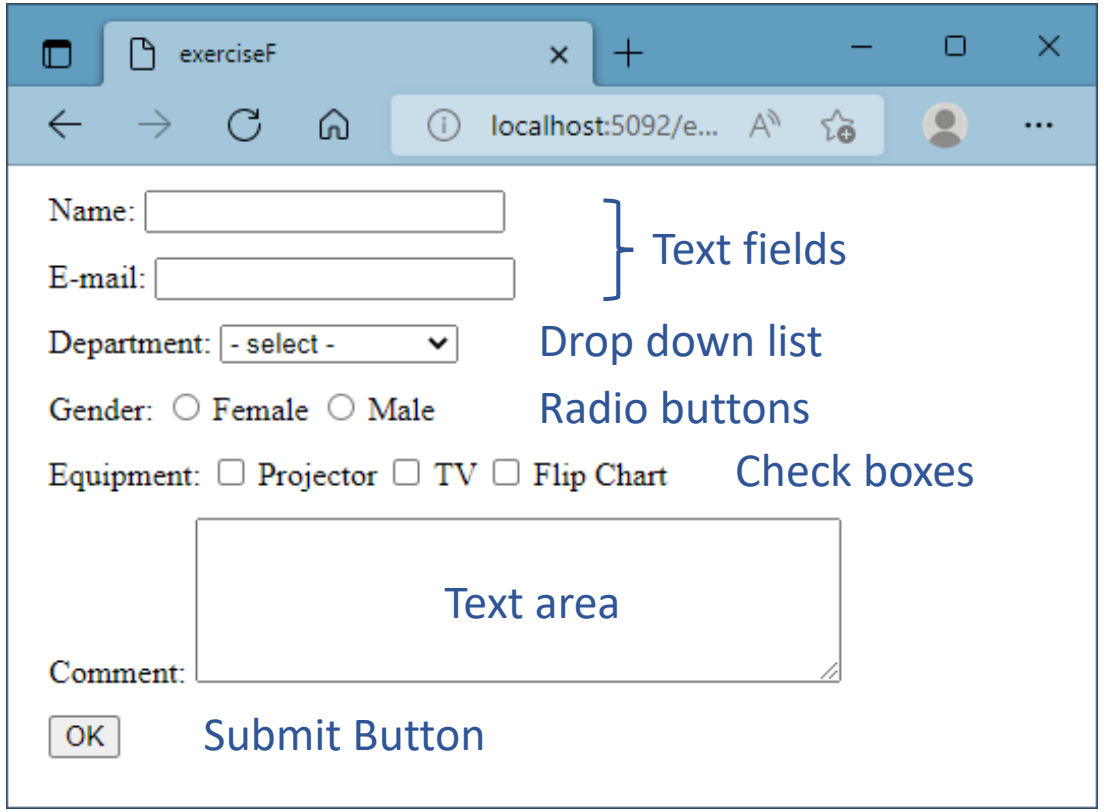
- CSS Units (w3schools.com)

Element: form

The **<form>** element is used to collect user input.

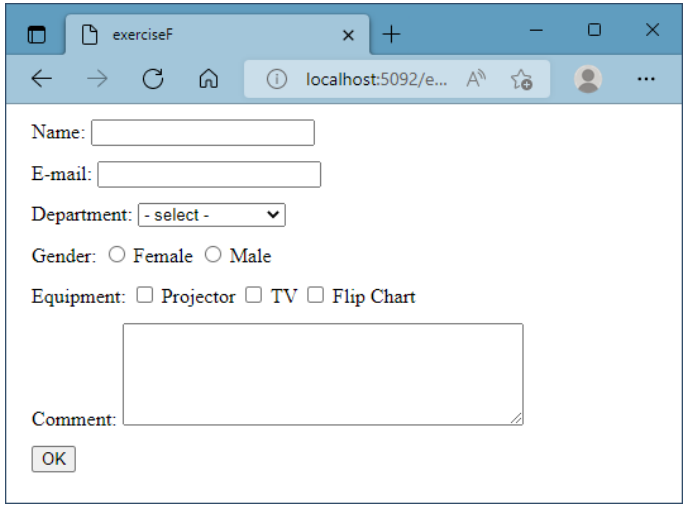
Within the **<form>** element, there are different types of input elements

- Text Fields
- Text Areas
- Lists
- Drop Down Lists
- Check Boxes
- Radio Buttons
- Submit Buttons



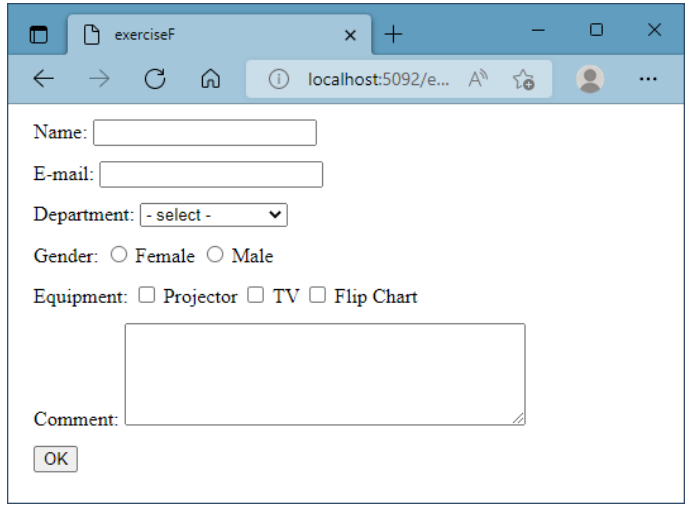
Element: form

```
<form id="feedback">
    <div>
        <label for="name">Name:</label>
        <input type="text" id="name" name="Name">
    </div>
    <div>
        <label for="email">E-mail:</label>
        <input type="text" id="email" name="Email">
    </div>
    <div>
        <label for="department">Department:</label>
        <select id="department" name="Department">
            <option>- select -</option>
        </select>
    </div>
    <div>
        <label>Gender:</label>
        <input type="radio" id="gender1" name="Gender"> Female
        <input type="radio" id="gender2" name="Gender"> Male
    </div>
```



Element: form

```
    <div>
        <label>Equipment:</label>
        <input type="checkbox" id="Equip1" name="Proj"> Projector
        <input type="checkbox" id="Equip2" name="Tv"  > TV
        <input type="checkbox" id="Equip3" name="Flip"> Flip Chart
    </div>
    <div>
        <label for="comment">Comment:</label>
        <textarea id="comment" name="Comment" rows="5" cols="40"></textarea>
    </div>
    <div>
        <input type="submit" id="btn" value="OK" />
    </div>
</form>
```



Apprentice Activity

Complete worksheet exercise F

CSS

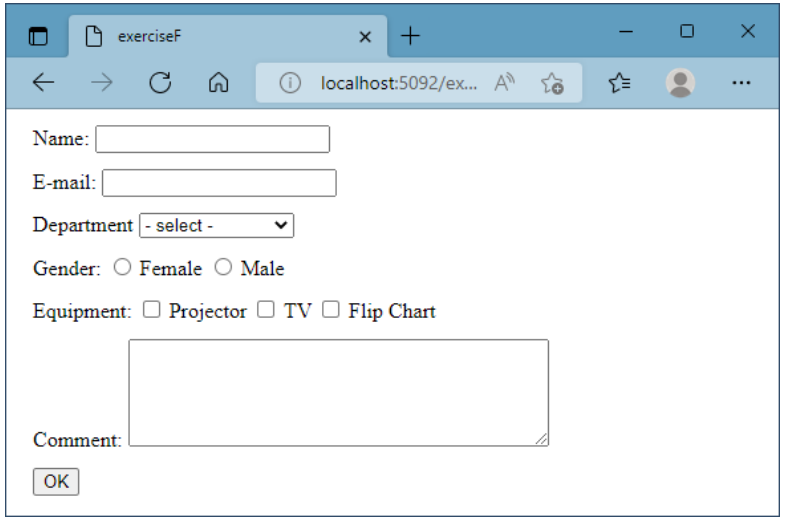
Identifying HTML elements for styling with CSS

The id attribute specifies a unique id for an HTML element

- This value must be unique within the HTML document

The id attribute is used

- by CSS
  - to point to a style in a style sheet
- by JavaScript
  - to manipulate the element of the specific id



Examples

- `<div id='menu'>`
- `<input type='text' id='sno'>`

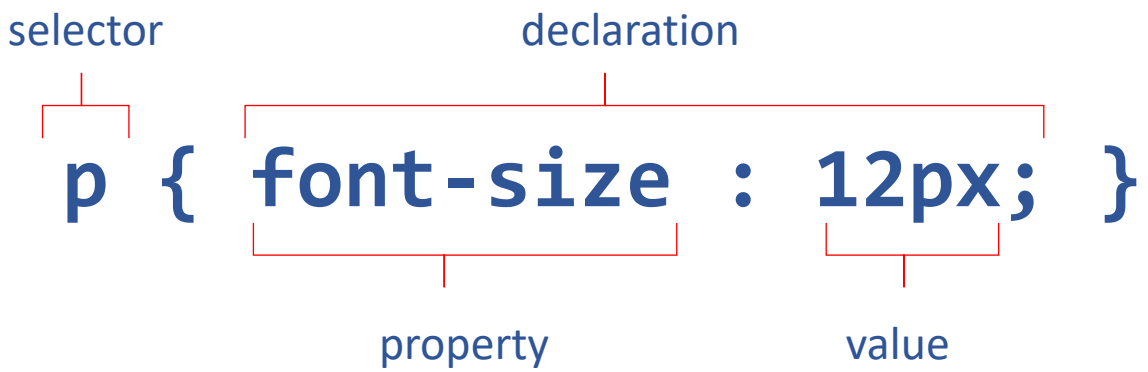
CSS

Cascading Style Sheets (CSS)

- Defines how HTML elements are to be formatted, layout and displayed

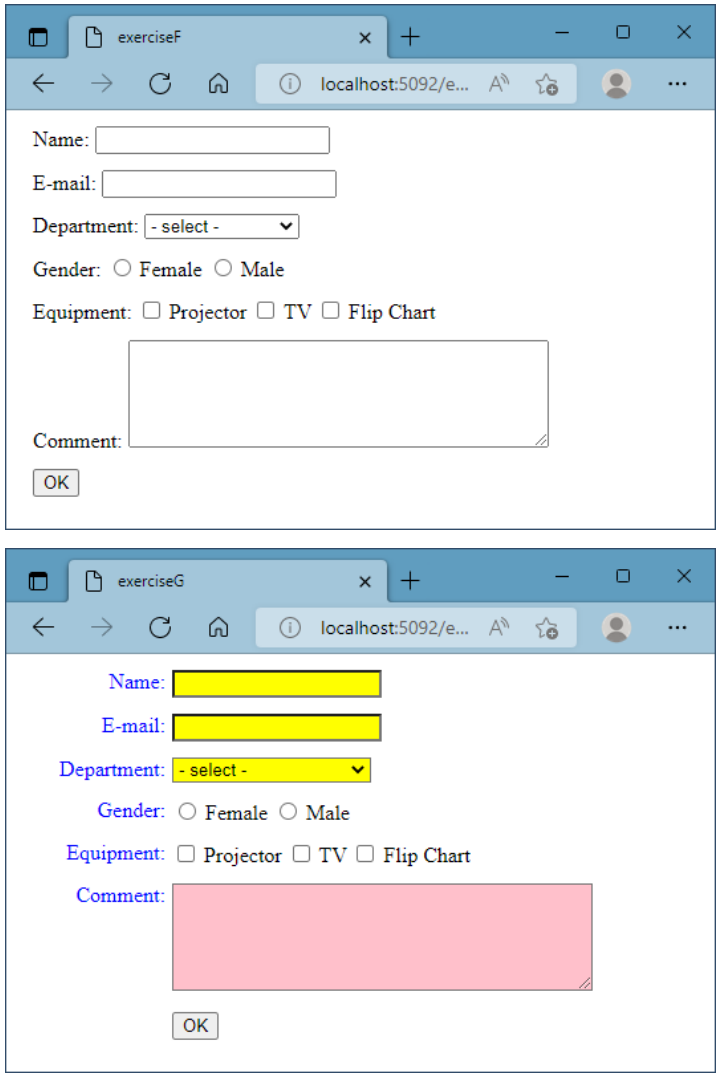
CSS Syntax

- A CSS Rule comprises
  - Selector – selects the HTML element to style
  - Declaration Block – specifies the properties to apply



Styling HTML using both HTML elements and ids (#)

```
<head>
    <title>exerciseG</title>
    <style>
        div {
            margin: 12px;
        }
        #feedback label {
            color: blue;
            float: left;
            width: 100px;
            text-align: right;
            margin-right: 5px;
        }
        #name, #email, #department {
            width: 150px;
            background-color: yellow;
        }
        #comment {
            background-color: pink;
        }
        #btn{
            margin-left: 105px;
        }
    </style>
</head>
```



Apprentice Activity

Complete worksheet exercise G

Using CSS

Applying CSS to HTML Elements

- **External CSS:** Link to **external file**

```
<link href="main.css" rel="stylesheet" />
```

- **Internal CSS:** Add **<style> section** to page

```
<style>
  /* standard CSS syntax */
</style>
```

- **Inline CSS:** Use the **style attribute**

```
<div style='property1:value1; property2:value2'>
```

CSS - Basic element selection

Using Element

- Applies to element
- Simply use the name of the **element** in the CSS

```
h2 {
    font-family: verdana, sans-serif;
    font-weight: bold;
    color: blue;
}
```

```
p {
    text-align:
    color: blue;
}
```

CSS - Basic id selection

Using ID

- Applies to element with ID
  - IDs must be unique
- ID must be prefixed by **"#"** in the CSS

```
#header {
    font-family: verdana, sans-serif;
    font-weight: bold;
    color: blue;
}
```

```
#submit {
    margin-left: 105px;
}
```

CSS - Basic class selection

Using Class

- Classes are available to all elements
  - Specified by the **class attribute** in an element
- Applies to all elements with a matching class
  - `<p class="bright">`
  - `<div class="bright">`
  - `<h1 class="bright">`
- Class must be prefixed by "." in the CSS

```
.bright {
    color : yellow;
}
```

Basic element selection

Combining Classes and Element: The **cascading** in CSS

```
.bright {
    color : yellow;
}
```

Apply to all elements with the "bright" **class attribute**.

```
div.bright {
    color : lightyellow;
}
```

Apply to all **div** elements with the "bright" **class attribute**.

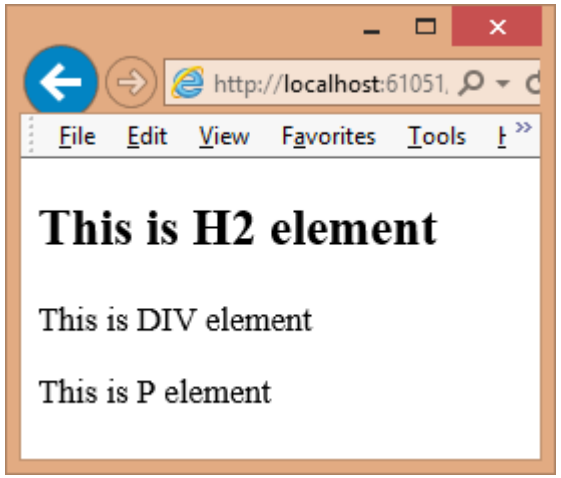
```
p.bright {
    color : orange;
}
```

Apply to all **p** elements with the "bright" **class attribute**.

Test your understanding

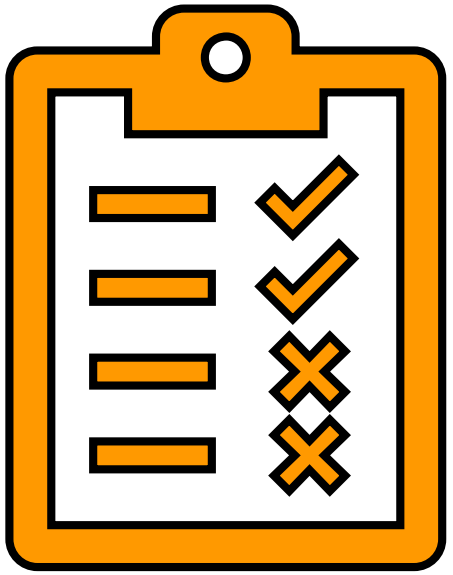
```
<style>
    .bright {
        color: orange;
    }

    div.bright {
        color: magenta;
    }

    p.bright {
        color: green;
    }
</style>
```

```
<h2 class="bright">
    This is H2 element
</h2>

<div class="bright">
    This is DIV element
</div>

<p class="bright">
    This is P element
</p>
```



Test your understanding

- Return to your pinned C236 Lesson 01 SCORM package window and take the second quiz - **Mini Quiz 03 – CSS Properties**.
- Take the third quiz - **Mini Quiz 03 - CSS Properties**.



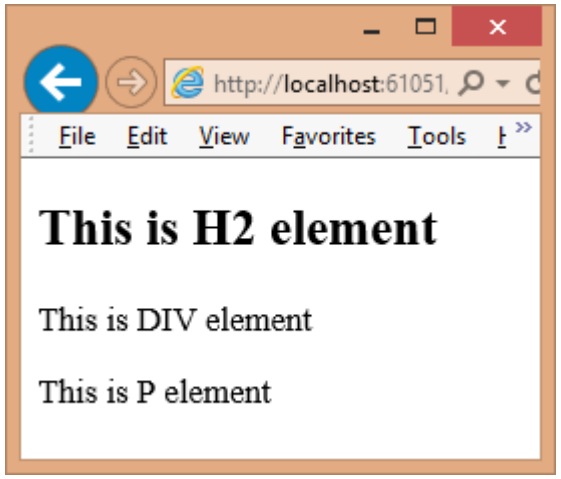
Notes:

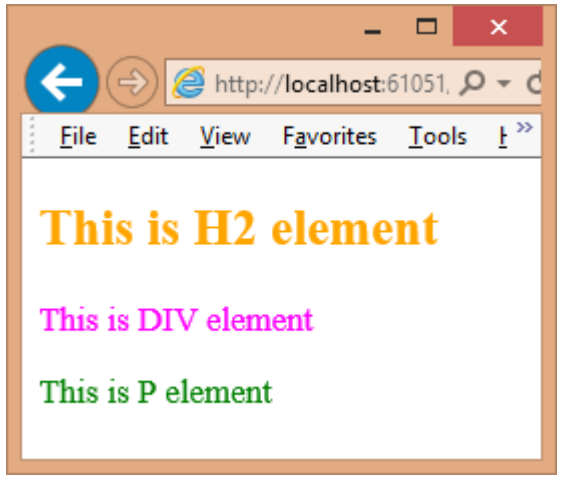
- Quizzes are formative in nature.
- Quiz answers do not contribute to your continuous assessment grade (CAG).
- You may attempt the SCROM package more than once.

Test your understanding §

```
<style>
    .bright {
        color: orange;
    }

    div.bright {
        color: magenta;
    }

    p.bright {
        color: green;
    }
</style>
```

```
<h2 class="bright">
    This is H2 element
</h2>

<div class="bright">
    This is DIV element
</div>

<p class="bright">
    This is P element
</p>
```





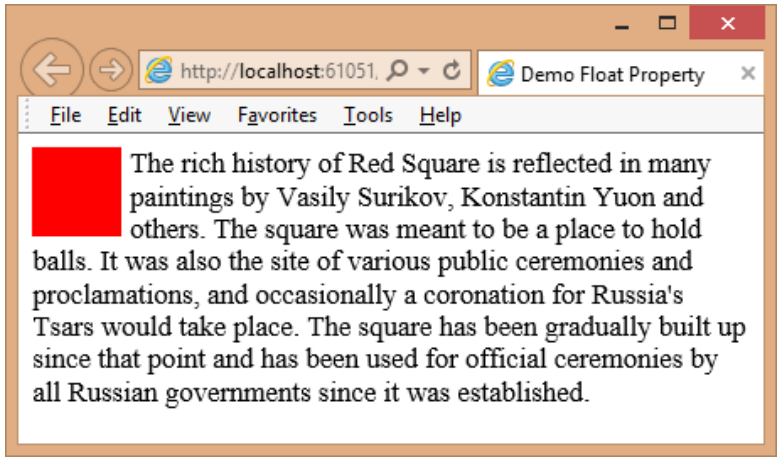
float

This property

- horizontally anchors an element
- enables other elements to flow around
- enables block elements to be positioned on the same row

```
<div id="redsquare"></div>
<div>
    The rich history of Red Square is reflected in many paintings by Vasily Surikov,
    Konstantin Yuon and others. The square was meant to be a place to hold balls.
    It was also the site of various public ceremonies and proclamations,
    and occasionally a coronation for Russia's Tsars would take place.
    The square has been gradually built up since that point and
    has been used for official ceremonies by all Russian governments
    since it was established.
</div>
```

```
#redsquare {
    float : left;
    width : 50px;
    height : 50px;
    margin-right : 5px;
    background-color : red;
}
```

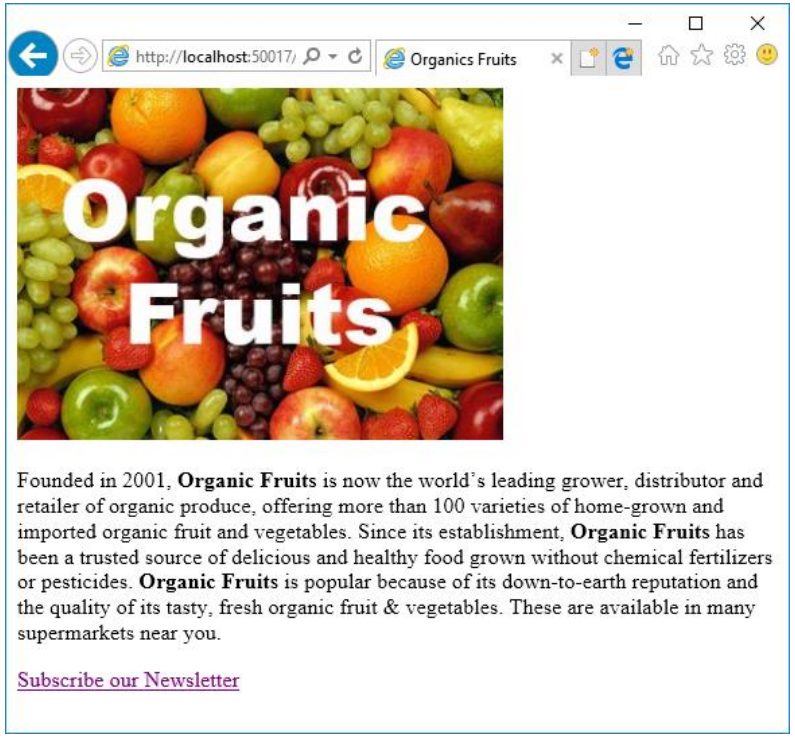


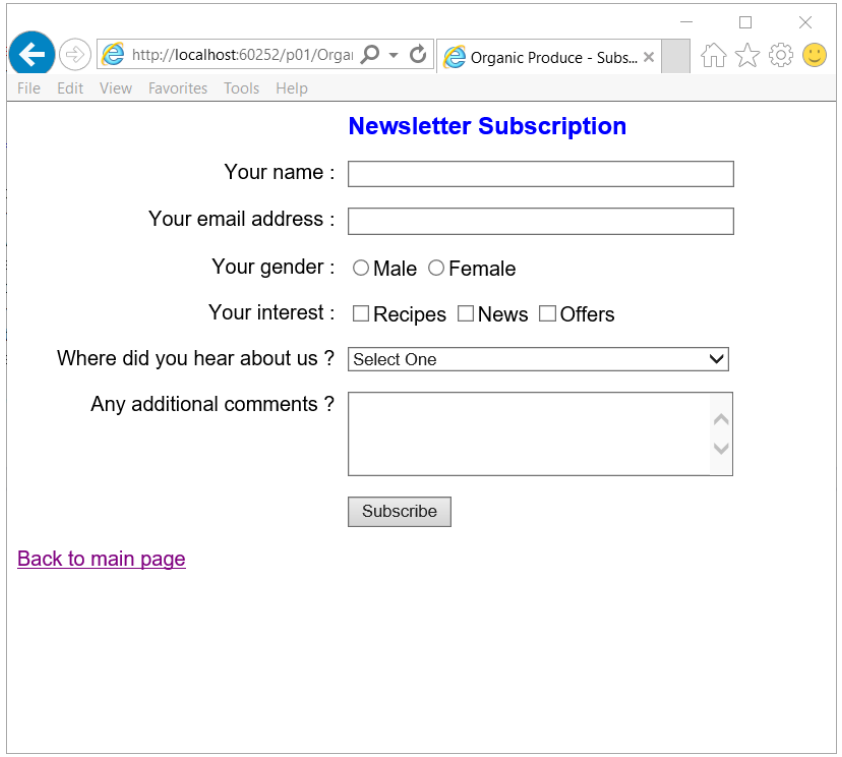
§

Problem solution

Organic Fruits §

Web pages §


http://localhost:50017/
Organics Fruits
Organic Fruits
Founded in 2001, **Organic Fruits** is now the world's leading grower, distributor and retailer of organic produce, offering more than 100 varieties of home-grown and imported organic fruit and vegetables. Since its establishment, **Organic Fruits** has been a trusted source of delicious and healthy food grown without chemical fertilizers or pesticides. **Organic Fruits** is popular because of its down-to-earth reputation and the quality of its tasty, fresh organic fruit & vegetables. These are available in many supermarkets near you.
Subscribe our Newsletter


http://localhost:60252/p01/Orga
Organic Produce - Subs...
File Edit View Favorites Tools Help
Newsletter Subscription
Your name :
Your email address :
Your gender : Male Female
Your interest : Recipes News Offers
Where did you hear about us ? Select One
Any additional comments ?
Subscribe
Back to main page

main.html §

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Organics Fruits</title>
</head>
<body>
    <img src="/images/company.jpg" />
    <p>
        Founded in 2001, <b>Organic Fruits</b> is now the world's leading grower,
        distributor and retailer of organic produce, offering more than
        100 varieties of home-grown and imported organic fruit and vegetables.
        Since its establishment, <b>Organic Fruits</b> has been a trusted source of
        delicious and healthy food grown without chemical fertilizers or pesticides.
        <b>Organic Fruits</b> is popular because of its down-to-earth reputation
        and the quality of its tasty, fresh organic fruit & vegetables.
        These are available in many supermarkets near you.
    </p>
    <a href="subscribe.html">Subscribe our Newsletter</a>
</body>
</html>
```

subscribe.html §

```
<body>
   <form id="subForm">
      <h2>Newsletter Subscription</h2>
      <div class="form-group">
         <label for="name" class="control-label">Your name : </label>
         <input type="text" id="name" name="Name" />
      </div>
      <div class="form-group">
         <label for="email" class="control-label">Your email address : </label>
         <input type="text" id="email" name="Email" />
      </div>
      <div class="form-group">
         <label class="control-label">Your gender : </label>
         <input type="radio" name="Gender" id="male"   value="male" />Male
         <input type="radio" name="Gender" id="female" value="female" />Female
      </div>
      <div class="form-group">
         <label class="control-label">Your interest :</label>
         <input type="checkbox" id="recipes" value="Recipes" />Recipes
         <input type="checkbox" id="news"    value="News" />News
         <input type="checkbox" id="offers"  value="Offers" />Offers
      </div>
```

subscribe.html §

```
      <div class="form-group">
         <label for="refer" class="control-label">Where did you hear about us ? </label>
         <select name="Refer" id="refer">
            <option value="0">Select One</option>
            <option value="1">Friend</option>
            <option value="2">Magazine</option>
            <option value="3">Newspaper</option>
            <option value="4">Radio</option>
         </select>
      </div>
      <div class="form-group">
         <label for="comments" class="control-label">Any additional comments ? </label>
         <textarea name="Comments" id="comments" cols="35" rows="4"></textarea>
      </div>
      <div class="form-group">
         <input type="submit" name="Submit" id="subscribe" value="Subscribe" />
      </div>
   </form>
   <a href="main.html">Back to main page</a>
</body>
```

Internal CSS for subscribe.html §

Using <style> within <head>

```
<style>
    html {
        font-family: sans-serif;
    }

    .form-group {
        padding: 5px;
    }

    .control-label {
        float: left;
        width: 250px;
        text-align: right;
        margin-right: 10px;
    }

    #name, #email, #refer {
        width: 300px;
    }

    #subscribe {
        margin-left: 260px;
    }

    h2 {
        margin-left: 260px;
        color: blue;
        font-size: 1.2em;
        font-weight: bold;
    }
</style>
```

External CSS for subscribe.html §

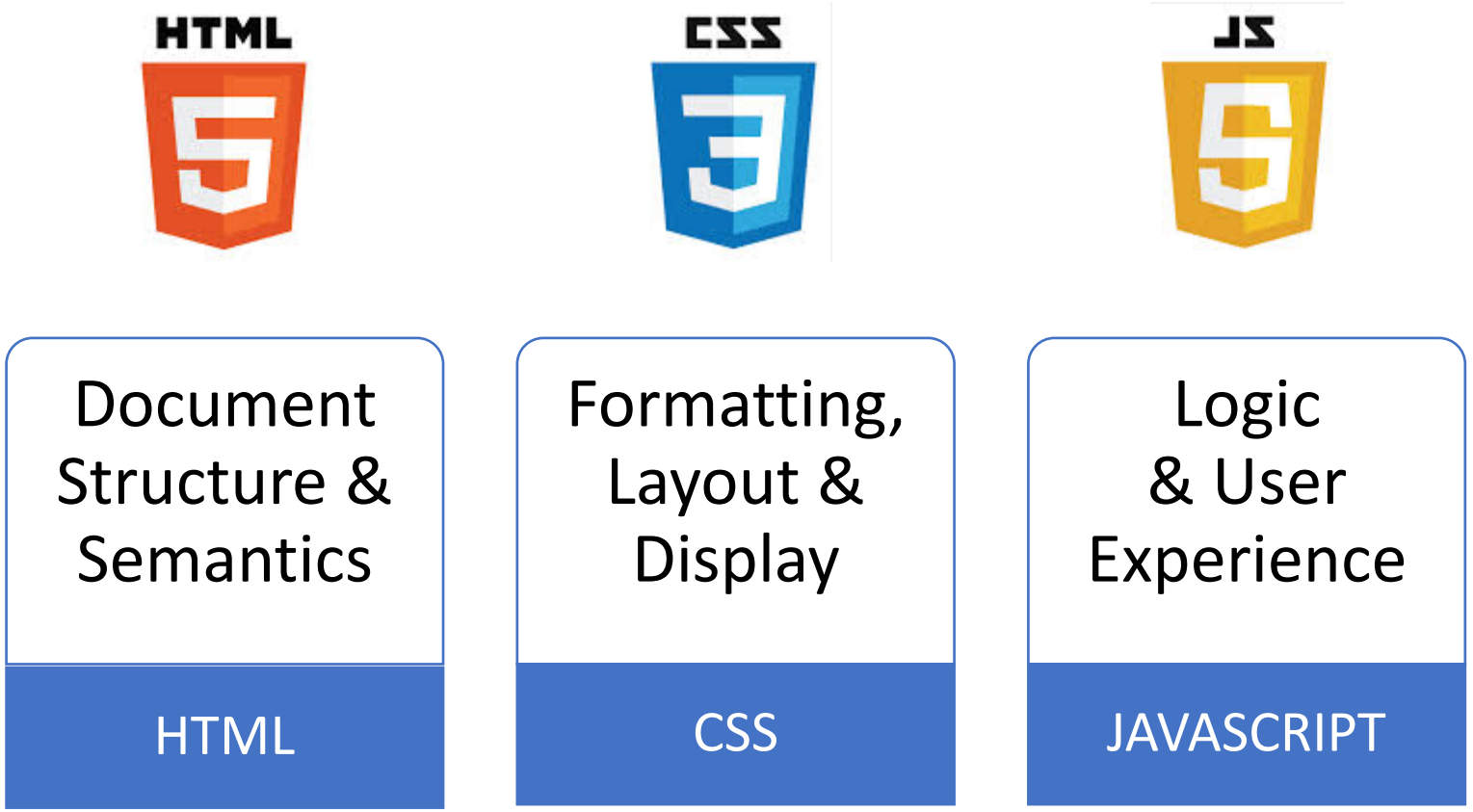
Using <link> within <head>

```
<head>
    <title>Organic Fruits - Subscription</title>
    <link href="/styles/organic.css" rel="stylesheet" />
</head>
```

organic.css

```
1   html {
2       font-family: sans-serif;
3   }
4
5   .form-group {
6       padding: 5px;
7   }
8
9   .control-label {
10      float: left;
11      width: 250px;
12      text-align: right;
13      margin-right: 10px;
14  }
15
16  #name, #email, #refer {
17      width: 300px;
```

Separation of concerns



Summary

What you have learned

- Build web pages using HTML and CSS
- State the usage of common HTML elements
- Apply CSS rules to format and layout HTML elements
- Separate the concerns between content and presentation in web pages

Reminder: Read the Start document

- You need to complete a **Linked-in learning course by Week 04**.
- Details are in the Start document.
- Course contributes to CAG.

Additional resources

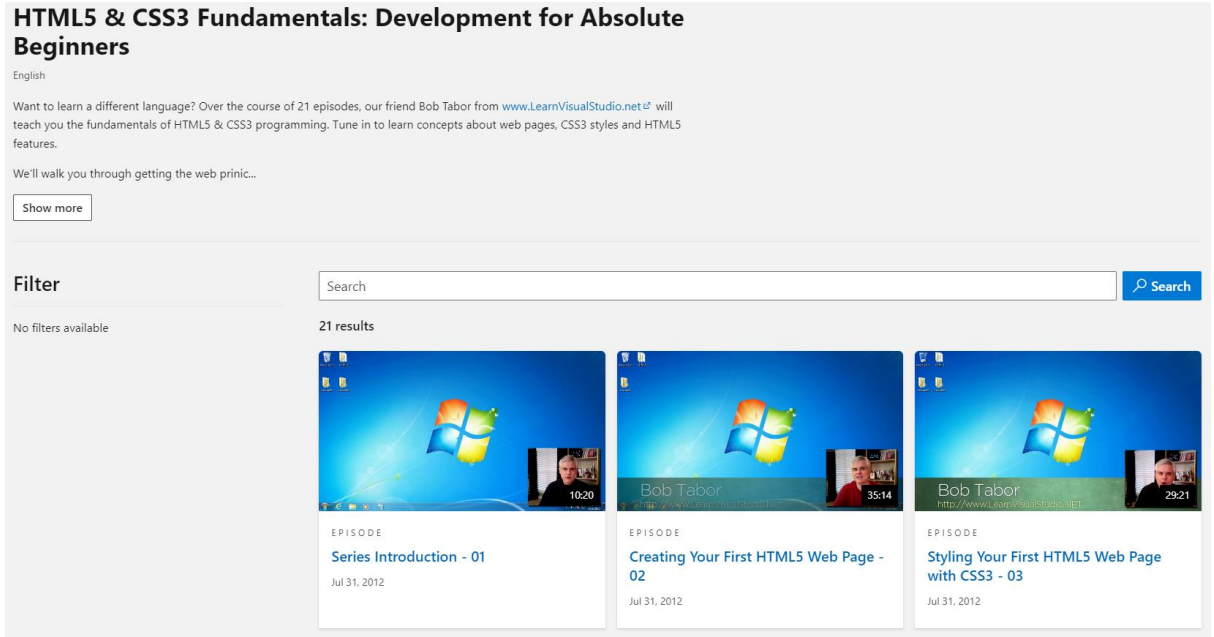
W3schools.com

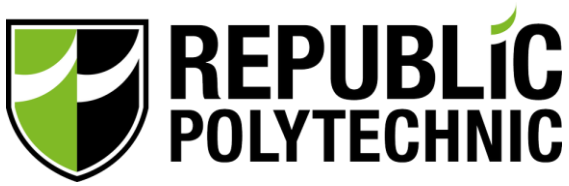
- HTML Tutorial (w3schools.com)
- CSS Tutorial (w3schools.com)

MSDN Channel 9

- HTML5 & CSS3 Fundamentals: Development for Absolute Beginners

Note: This series is recommended and is given as part of the **e-learning resources** for this module. It is well organised and covers many of the core HTML concepts in simple, bite-size pieces.





Lesson 2

Introduction to C# and MVC

C#

Overview of C#

Object-Oriented

- Very similar to Java.
- Everything belongs to a class
- All data types derived from `System.Object`

Complete C# Program

```
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

Program Structure

Namespaces

- Contain types and other namespaces

Type Declarations

- Classes, structs, interfaces, delegates

Members

- Fields, methods, properties, events, constructors

Data Types

Integer Types

- `byte, short, int, long`

Floating Types

- `float, double`

Exact Numeric Type

- `decimal`

Character Types

- `char, string`

Boolean Type

- `bool`

Arrays

Built on .NET class System.Array

Declared with type and dimension

- `int [ ] array1D`
- `int [ , ] array2D`

Created using new with bounds

- `array1D = new int[20]`
- `array2D = new int[10, 5]`

Using initialisers

- `int[] myarray = { 1, 2, 3, 4 };`

Statements & Comments

Case-sensitive

Statement delimiter is semi-colon `;`

- Curly Brackets to enclose multiple statements into a block

```
{statement;
            statement;
            }
```

Single line comment `// one line comment`

Block comment `/* comments */`

`if … else` `// same as Java`

`for ( ; ; )` `// same as Java`

Operators

Arithmetic

- + addition
- - subtraction
- * multiplication
- / division
- % modulus
- ++ increment
- -- decrement

Conditional

- || or
- && and
- ! not

Comparison

- != not equal
- == equal
- >= greater or equal
- <= lesser or equal
- > greater
- < lesser

Characters

A char literal is specified inside **single** quotes

- `char c = 'A';  // simple character`

Escape sequences express special characters

- `char newLine = '\n';`
- `char backslash = '\\';`

Strings

A string literal is specified inside **double** quotes

The escape sequences valid for char also work inside strings

```
string abc = "Apple";
string xyz = "aPPLE";
string msg = "Hello\t Blah Blah\n"; // Tab, NewLine

// No Difference in C#
if (abc == xyz)      // Does NOT compare object references
                     // Still compares string values
if (abc.Equals(xyz)) // Same as Java, compares string values

// Case Sensitive
bool notEqual = abc.Equals(xyz);

// Ignore Case
bool yesEqual = abc.Equals(xyz,
                StringComparison.InvariantCultureIgnoreCase);
```

Strings (cont.)

More Examples

```
string s1 = "abc";
string s2 = "abc";
string s3 = "ABC";
Object s4 = s2.ToUpper();

Console.WriteLine(s1 == s2);            // True
Console.WriteLine(s1 == s3.ToLower()); // True
Console.WriteLine(s3 == s4);            // False

Console.WriteLine(s1.Equals(s2));           // True
Console.WriteLine(s1.Equals(s3.ToLower())); // True
Console.WriteLine(s3.Equals(s4));           // True

s1 = null; s2 = null;
Console.WriteLine(s1 == s2);      // True
Console.WriteLine(s1.Equals(s2)); // Null Pointer Exception
```

Strings (cont.)

Verbatim strings are strings that do not support escape sequences and can span multiple lines

- Must be prefixed by **@**

```
string msg1 = @"Escape \n makes no difference";
string msg2 = @"Line 1
                Line 2"; // Can span multiple lines
string msg3 = "Line 1" +
              "Line 2";  // Normal string must use +
```

Strings (cont.)

Interpolated strings are template strings that contain variable definitions.

- Must be prefixed by **$**
- Variables are delimited by **{ }**

```
int a = 12;
int b = 13;
String c = "25";
String answer =
   $"{a} + {b} = {c}";  // answer is "12 + 13 = 25"
```

.Trim Method

Syntax

- Calls from a string
- Takes no argument
- Returns a string with trailing and leading spaces removed

```
string x = "  Nissan";
string y = "  Toyota Camry  ";
string z = "Hyundai   ";
string a = x.Trim();  // a is "Nissan"
string b = y.Trim();  // b is "Toyota Camry"
string c = z.Trim();  // c is "Hyundai"
```

`.Substring` Method

Syntax

- Calls from a string
- Takes two arguments startIndex and length
- Returns a sub string

```
string x = "Happiness";
string y = x.Substring(0, 3); // y is "Hap"
string z = x.Substring(4, 2); // z is "in"
```

Model View Controller

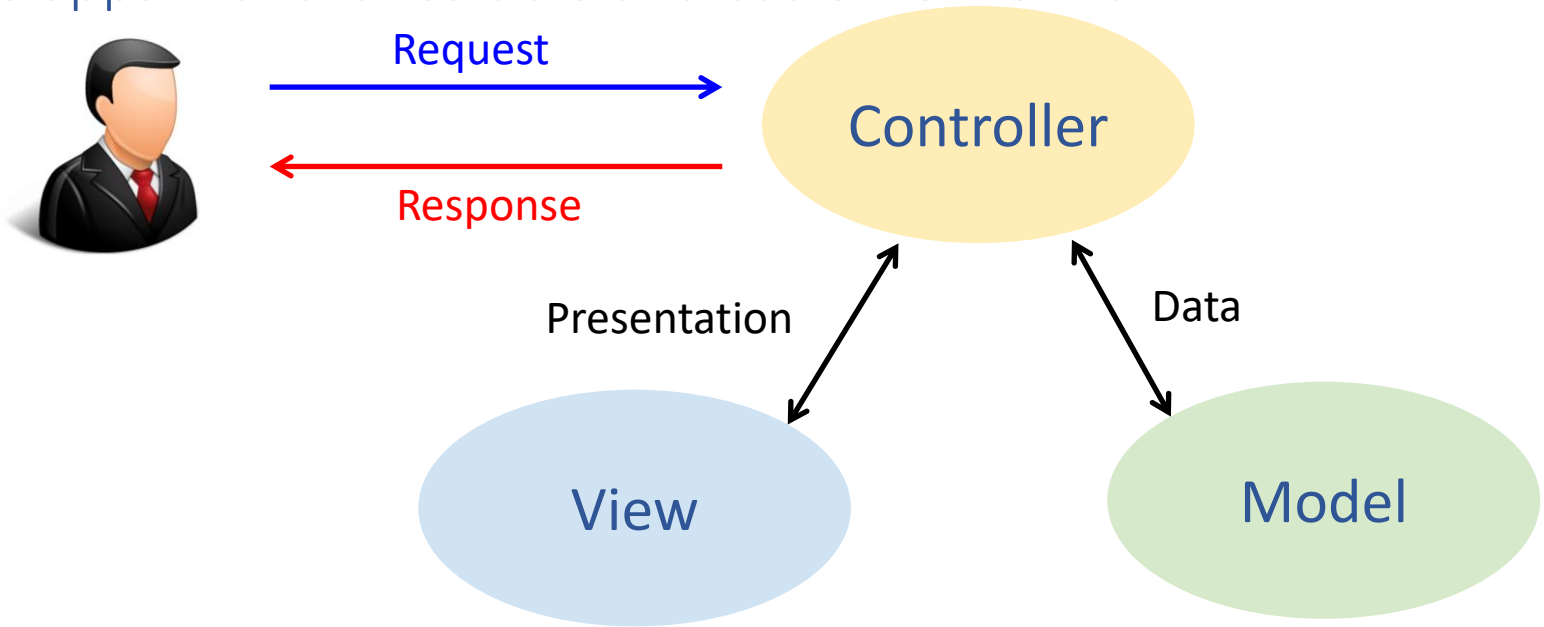
Model-View-Controller (MVC)

It is a Software Architectural Pattern

Separates an app into three main components

- Model
- View
- Controller

Helps to create apps that are testable and easier to maintain



Model-View-Controller (MVC)

Models

- Classes that represent the data of the app
- Enforce validation and business rules

Views

- Components that handle the UX/UI of the app

Controllers

- Classes that handle browser requests
- Retrieve model data
- Specify view template
- Return response to the browser

Motivation for MVC

Create apps that separate the different aspects of the app

- Input Logic
- Business Logic
- UI Logic

Maintains a loose coupling between these elements.

MVC specifies where each kind of logic should be located in the app

- UI logic in the view
- Input logic in the controller
- Business logic in the model

Helps to manage complexity when building app

URL Routing

MVC URL Routing Patterns

Static html, jpeg, css files

- Every URL must match with a specific file in wwwroot

```
http://localhost/start.html
http://localhost/customer/home.html
http://localhost/images/company.jpg
http://localhost/css/main.css
```

MVC

- Use routing to eliminate the need of mapping each URL with a physical file
- Routing enables a URL to map to **controller** classes, **action** methods and **id** parameters.

```
http://localhost/home/index
http://localhost/home/customer/123
http://localhost/store/index
http://localhost/store/product/p001
```

Configuration for Routing

Routing Logic is defined in the `Configure` method of `Startup.cs`

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

- 1st Segment is Controller
- 2nd Segment is Action
- 3rd Segment is id (optional, `?` means zero or one)
- Segments are separated by slashes (`/`)

Default Controller is Home (as specified by `"=Home"`)
Default Action is Index (as specified by `"=Index"`)
Controller and Action are case-insensitive in the URLs: `"Home", "hoMe", "HOME", "home"` are all the same.

Understanding Routing

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

`http://locahost/Pokedex/Number/50`

- Controller is **Pokedex**
- Action is **Number**
- Id is **50**

`http://locahost/Pokedex/`

- Controller is **Pokedex**
- Action is **Index**
- Id is **null**

`http://localhost/`

- Controller is **Home**
- Action is **Index**
- Id is **null**

More Routing Examples

URL	Controller	Action	Id
http://localhost/home	HomeController	Index	null
http://localhost/Home/index/123	HomeController	Index	123
http://localhost/demo/about	DemoController	About	null
http://localhost/Contact/demo	ContactController	Demo	null
http://localhost/student	StudentController	Index	null
http://localhost/student/edit/123	StudentController	Edit	123
http://localhost/id	IdController	Index	null
http://localhost/index	IndexController	Index	null
http://localhost/home/home/home	HomeController	Home	"home"

Test your understanding

- Please start the C236 Lesson 02 SCORM package called MiniQuiz 1

Notes:

- Quizzes are formative in nature.
- Quiz answers do not contribute to your continuous assessment grade (CAG).
- You may attempt the SCROM package more than once.

MiniQuiz 1 Answers §

`http://localhost/bbb/aaa/ccc`

Controller = bbb

Action = aaa

Id = ccc

`http://localhost/pokedex/number/25`

Controller = pokedex

Action = number

Id = 25

`http://localhost/subject/`

Controller = subject

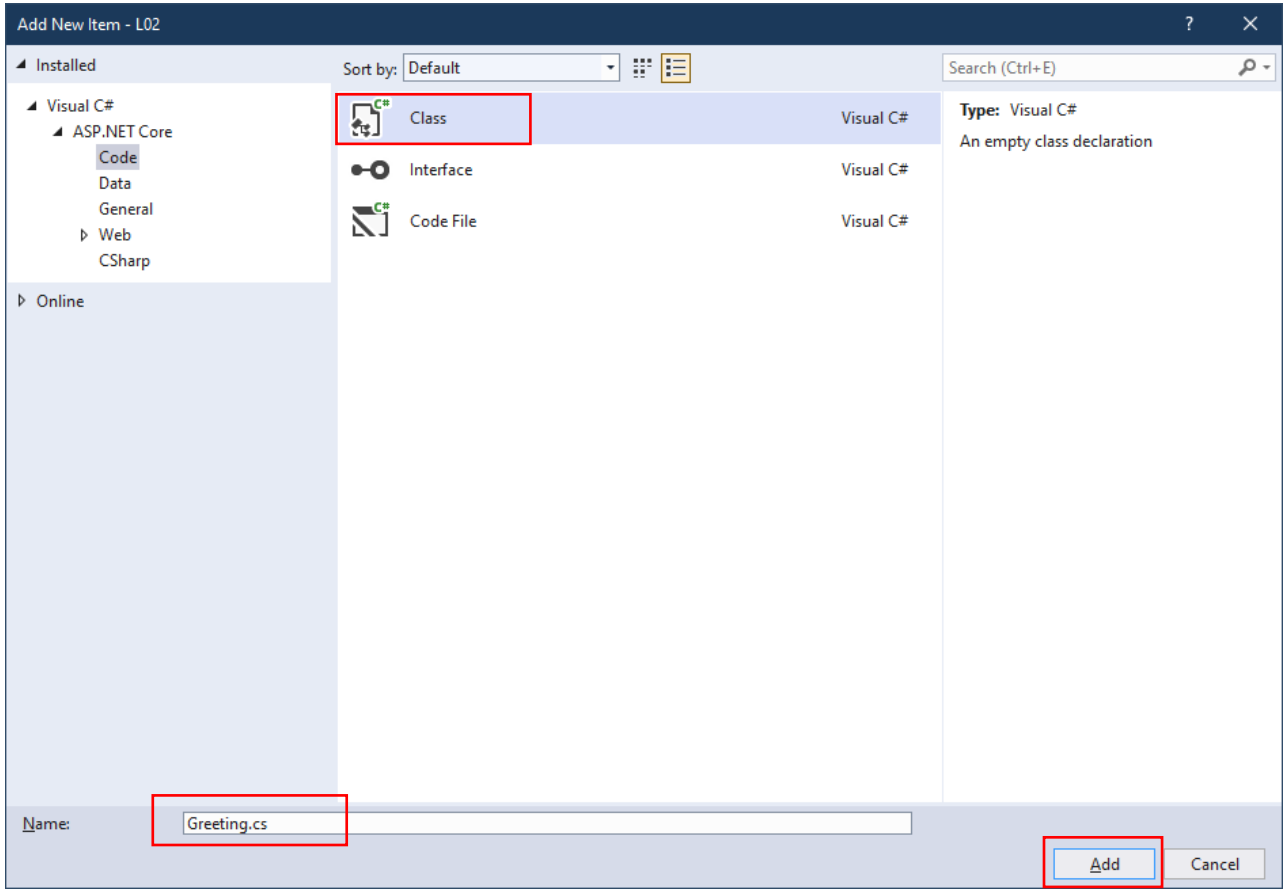
Action = index

Id = null

Creating a Model

Models, Add, New Items

Right-click **Models > Add > Class**

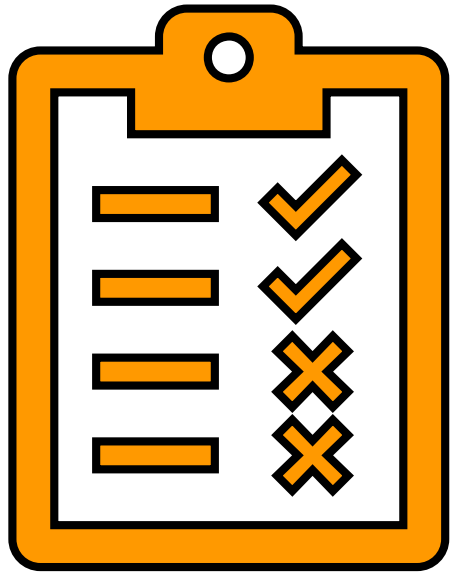


Greeting.cs

```
namespace L02.Models
{
    public class Greeting
    {
        public string To { get; set; }
        public string From { get; set; }
        public string Title { get; set; }
        public string Message { get; set; }

        public Greeting(string to,
                        string from,
                        string title,
                        string message)
        {
            To = to;
            From = from;
            Title = title;
            Message = message;
        }
    }
}
```

Test your understanding

- Please start the C236 Lesson 02 SCORM package called MiniQuiz 2



Notes:

- Quizzes are formative in nature.
- Quiz answers do not contribute to your continuous assessment grade (CAG).
- You may attempt the SCROM package more than once.

MiniQuiz 2 Answers §

What is the name of the Class?

- Greeting

How many properties are there?

- Four (4)

What is the data type of these properties?

- string

What is the access of these properties (Read-Write, Read-Only, Write-Only)?

- Read-Write

How many arguments does the constructor take?

- Four(4)

```
namespace L02.Models
{
    public class Greeting
    {
        public string To { get; set; }
        public string From { get; set; }
        public string Title { get; set; }
        public string Message { get; set; }

        public Greeting(string to,
                        string from,
                        string title,
                        string message)
        {
            To = to;
            From = from;
            Title = title;
            Message = message;
        }
    }
}
```

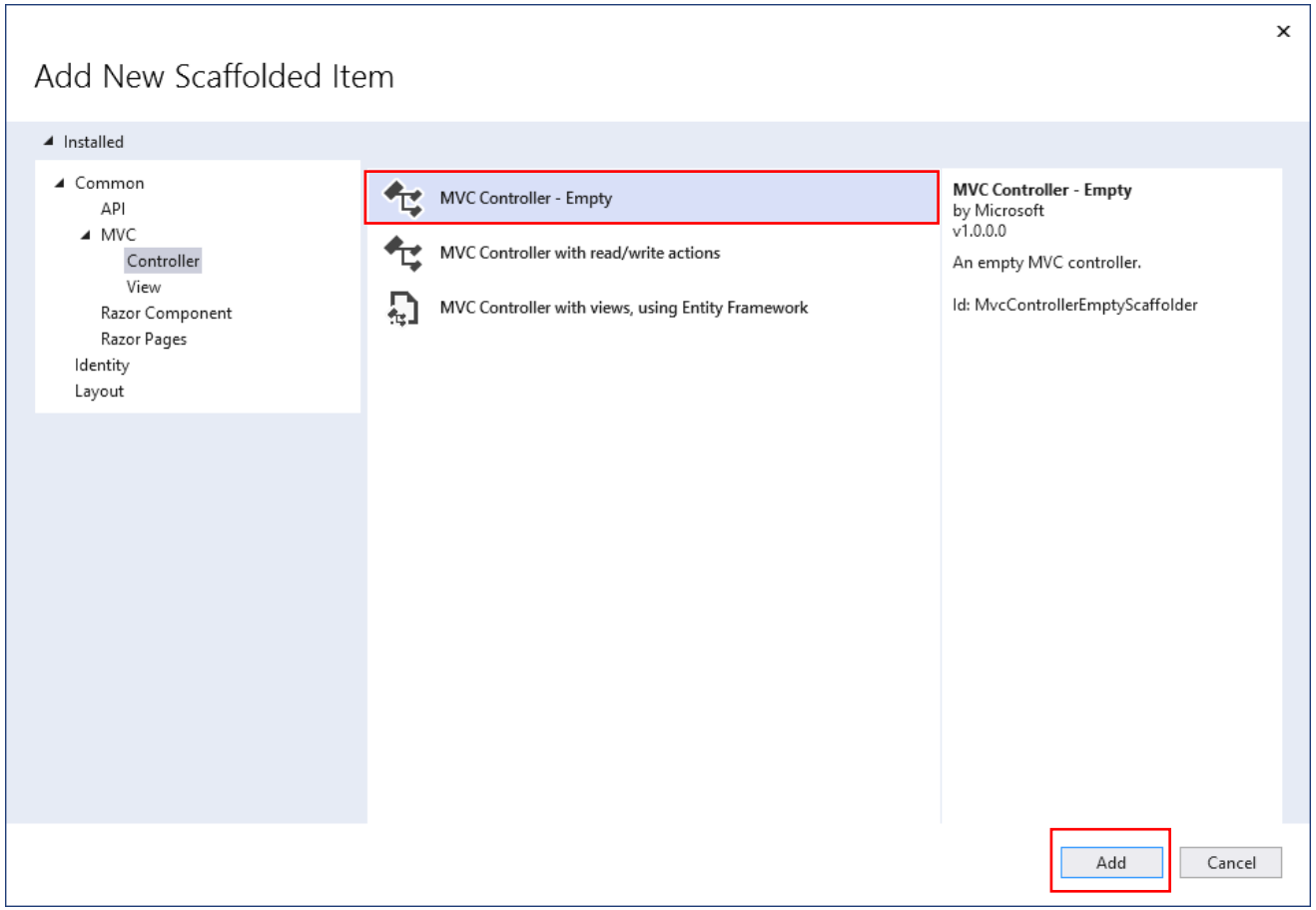
Apprentice Activity

Activity 1: Create the Greeting.cs Model

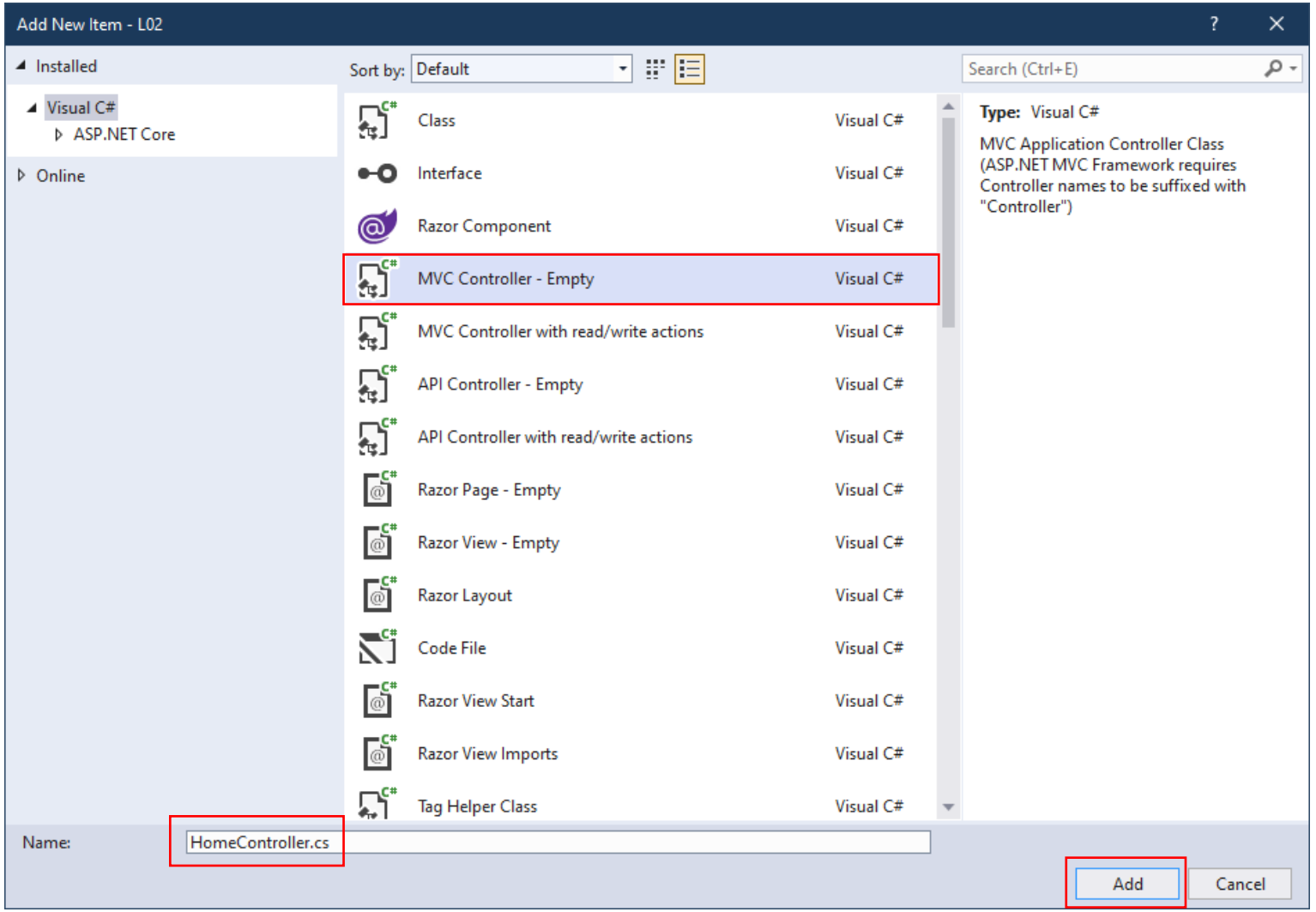
Creating a Controller

Add a new controller

Right-click Controllers folder > Add > Controller



Add a new controller



HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using L02.Models;

namespace L02.Controllers
{
    public class HomeController : Controller
    {

        public IActionResult Welcome()
        {
            // Create the model to be used in the View
            Greeting greet =
                new Greeting("SOI Students",
                             "C236 Module Chair",
                             "Welcome to Web App Dev in .NET",
                             @"This is a very useful module for your FYP.
                               Study hard and All the BEST!");

            // Put the model in ViewData
            ViewData["Hello"] = greet;

            // Use the default Welcome.cshtml view in Views/Home folder
            return View();
        }


    }
}
```

Controllers

Important Points

- All controllers derive (inherit) from the base class `Microsoft.AspNetCore.Mvc.Controller`

- Class name must end with "`Controller`".
  - Home**`Controller`**
  - Demo**`Controller`**
  - Student**`Controller`**

- Public instance methods in the class represent **actions**
  - Return a `View` object of type `IActionResult`

- Use `ViewData` to pass data to View

Using ViewData

ViewData

ViewData is a dictionary

- Contains key-value pairs
  - Each key must be string
  - Each value can be any object
- Used to transfer data from action to view, not vice-versa
- It is valid only during the **current request**.

```
// In the Action, put data in ViewData
ViewData["name"] = "Robert Sim";
ViewData["age"] = 34;
ViewData["lastLogin"] = DateTime.Now;

// In the View, get data from ViewData
string n      = (string) ViewData["name"];
int a         = (int) ViewData["age"];
DateTime when = (DateTime) ViewData["lastLogin"];
```

Transferring Data

In the Action

```
public IActionResult Welcome()
{
    // Create the model to be used in the View
    Greeting greet =
        new Greeting("SOI Students",
                     "C236 Module Chair",
                     "Welcome to Web App Dev in .NET",
                     @"This is a very useful module for your FYP.
                       Study hard and All the BEST!");

    // Put the model in ViewData
    ViewData["Hello"] = greet;

    // Use the default Welcome.cshtml view in Views/Home folder
    return View();
}
```

In the View

```
@using L02.Models
@{
    Greeting gtg = (Greeting)ViewData["Hello"];
}

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>C236 Lesson 2</title></head>
<body>
    <h2>Welcome</h2>
    <h3>To :</h3>
    <p>@gtg.To</p>
```

Apprentice Activity

Activity 2: Create the Home.cs controller

Creating a View

Views in ASP.NET MVC Core

Some Important Points

- A view encapsulates an application's **presentation details.**
- Views are **HTML templates** with embedded code that generate content to send to the client.
- Views use **Razor** syntax which enables C# code to interact with HTML code easily.
- A **controller** passes data defined in a **model** to a **view.**
- A **view** uses data defined in **model** to render web page.
- A **controller** returns rendered web pages to the client (browser).

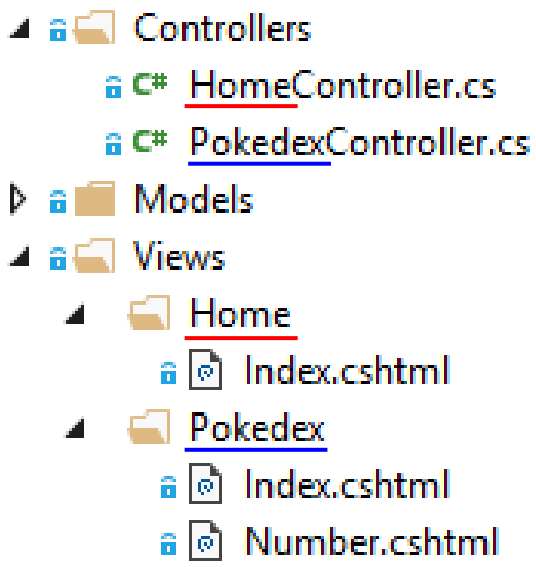
Naming Convention

- The view **folder** needs to have the same name as the **controller**
- The view **name** is the same as the **action** in the **controller**

View folders

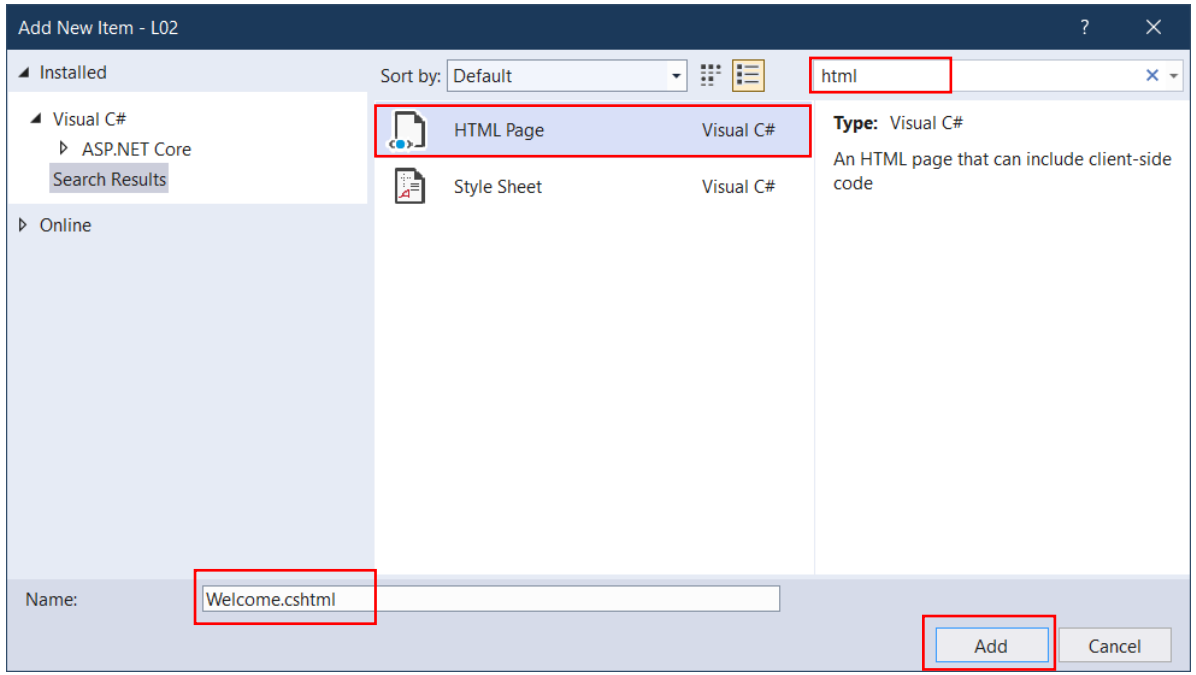
The **view folder** must have the same name as the **controller.**

Create the folders before creating the views



Add a view

1. Right-click the view folder matching the controller
2. Select **Add > New Item** (Do **not** select Add > View …)
3. Filter for **HTML** in the Search box.
4. **Replace the entire Name**: with a new name and the extension **.cshtml** (Do **not** use the .html extension)



Home/Welcome.cshtml

```
@using L02.Models
@{
    Greeting gtg = (Greeting)ViewData["Hello"];
}
```

Razor Directive

Razor Code Block

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>C236 Lesson 2</title></head>
<body>
    <h2>Welcome</h2>
    <h3>To :</h3>
    <p>@gtg.To</p>
```

Home/Welcome.cshtml

```
<body>
    <h2>Welcome</h2>

    <h3>To :</h3>
    <p>@gtg.To</p>

    <h3>From :</h3>
    <p>@gtg.From</p>

    <h3>Title :</h3>
    <p>@gtg.Title</p>

    <h3>Message :</h3>
    <p>@gtg.Message</p>

</body>
</html>
```

Razor Expressions

Files containing Razor generally have a .cshtml file extension.

Razor

- Razor is a markup syntax for embedding .NET based code into webpages.
- The Razor syntax consists of Razor markup, C#, and HTML
- Files containing Razor generally have a .cshtml file extension.
- Razor Markup
    - `@`
    - `@( expression )`
    - `@{ statements }`
    - `@* comments *@`

- Reference
  https://docs.asp.net/en/latest/mvc/views/razor.html

Apprentice Activity

Activity 3: Creating Home/Index View

Handling Form Inputs

Form Element

Form element should have two attributes.

- **method** (post)
- **action** (controller/action)

```
<!-- inside .cshtml file -->
<form method="post"
      action="/Form/Action_Post">
   <label for="id1">Input Field 1:</label>
   <input type="text" name="name1" id="id1" />
   <br/>
   <input type="submit" value="OK"/>
</form>
```

The form requires a **Submit** button so the form inputs can be posted to the controller/action specified in the form's **action** attribute.

The input element should have two attributes **id** (used in the view by CSS or JavaScript) and **name** (used in the controller to get the input).

id and **name** could be the same.

id must be unique but **name** need not be.

Controller Action

```
<!-- inside .cshtml file -->
<form method="post"
      action="/Form/Action_Post">
   <label for="id1">Input Field 1:</label>
   <input type="text" name="name1" id="id1" />
   <br/>
   <input type="submit" value="OK"/>
</form>
```

The action specifies the **Controller** (Form) and the **Action** (`Action_Post`) to process the post request.

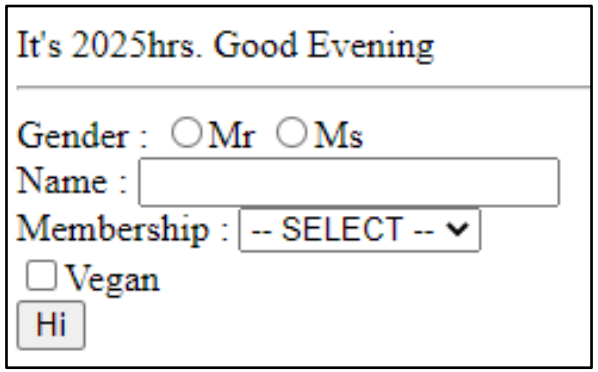
Use `HttpContext.Request.Form` to read input values from form. The **name1** attribute (NOT **id**) of the input element is used here to get the value.

```
public class FormController : Controller
{
   public IActionResult Action_Post()
   {
      string n = HttpContext.Request.Form["name1"];

      ViewData["Person"] = n;
      return View();
   }
…
```

SayHello5

```
<form method="post" action="~/Demo/SayHello5_Post">
    @ViewData["Greeting"]
    <hr />
    <label for="male">Gender : </label>
    <input type="radio" id="male" name="Gender" value="Mr" />Mr
    <input type="radio" id="female" name="Gender" value="Ms" />Ms
    <br />
    <label for="Name">Name : </label>
    <input type="text" name="Name" id="Name" />
    <br />
    <label for="mem">Membership :</label>
    <select name="membership" id="mem">
        <option value="">-- SELECT --</option>
        <option value="Silver">Silver</option>
        <option value="Gold">Gold</option>
        <option value="Platinum">Platinum</option>
    </select>
    <br />
    <label for="vegan">
        <input type="checkbox"
               name="Vegan" id="vegan" value="Vegan" />Vegan
    </label>
    <br />
    <input type="submit" id="ok" value="Hi" />

    @if (ViewData["Message"] != null)
    {
        <hr />
        @ViewData["Message"]
    }
</form>
```

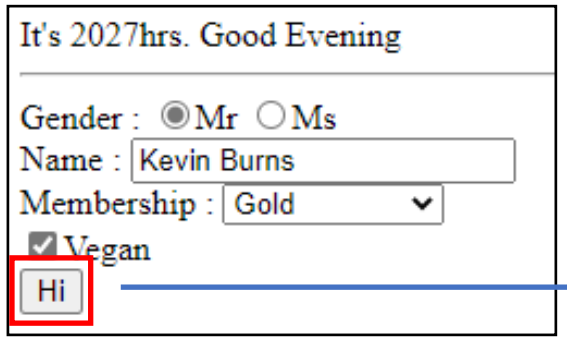
Rendered view



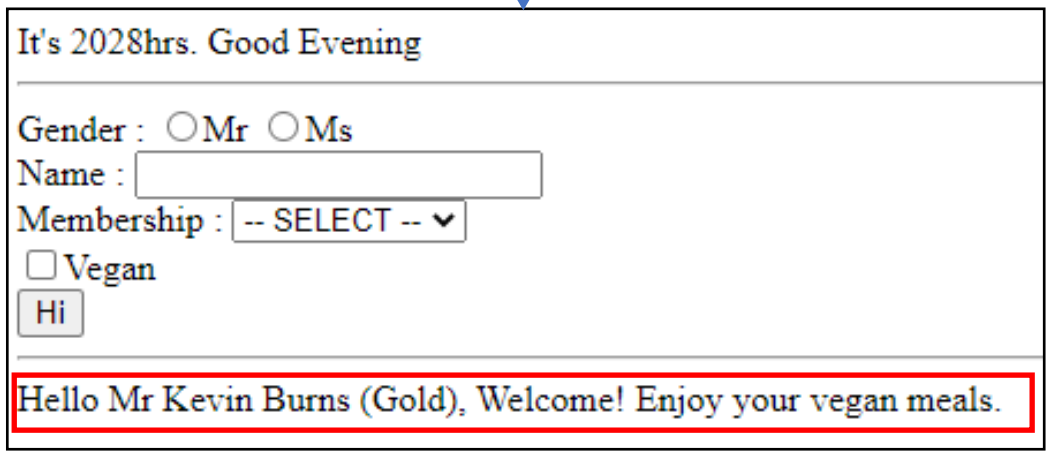
Controller code

```
public IActionResult SayHello5()
{
    DoGreeting();
    return View(); // Views/Demo/SayHello5.cshtml
}
```

SayHello5




```
public IActionResult SayHello5_Post()
{
    DoGreeting();
    string name    = HttpContext.Request.Form["Name"];
    string salute  = HttpContext.Request.Form["Gender"].ToString();
    string memship = HttpContext.Request.Form["Membership"];
    string vegan   = HttpContext.Request.Form["Vegan"].ToString();

    ViewData["Message"] = $"Hello {salute} {name} ({memship}), Welcome!";
    if (String.Equals(vegan, "Vegan"))
       ViewData["Message"] += " Enjoy your vegan meals.";

    return View("SayHello5");
}
```

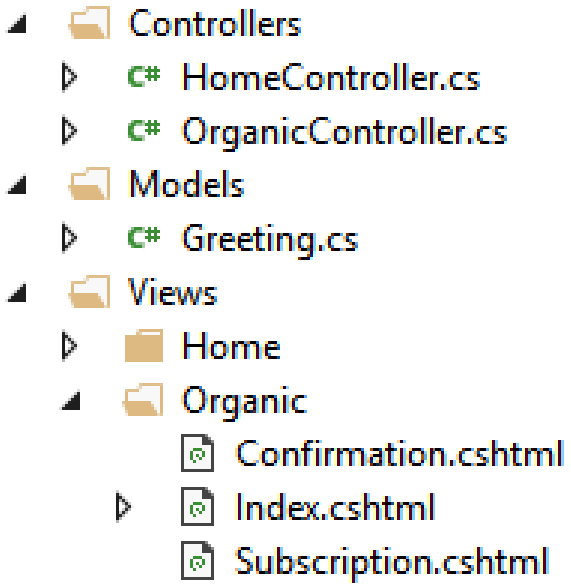


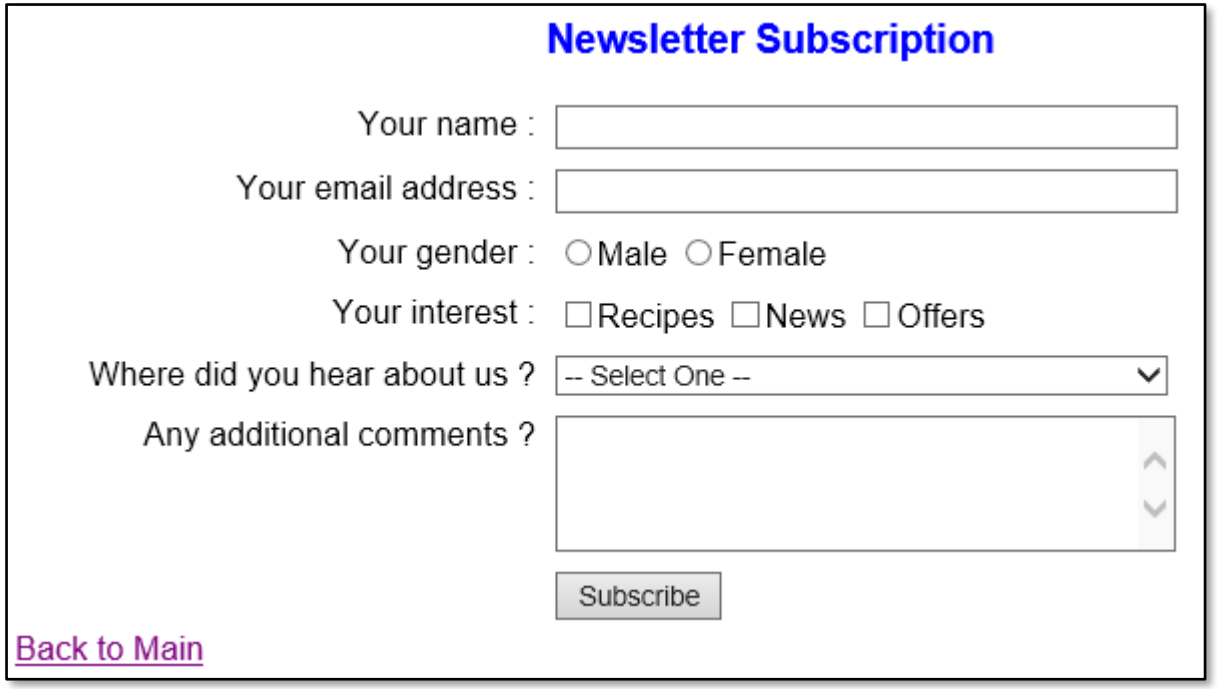
Apprentice Activity

Activity 4: Complete the "SayHello" series.

Organic Fruits

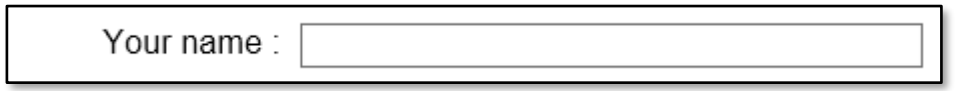
View: Subscription.cshtml





Subscription.cshtml

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Organic Fruits - Subscription</title>
   <link href="/styles/organic.css" rel="stylesheet" />
</head>
<body>
   <form method="post" action="/Organic/Confirmation">
      <h2>Newsletter Subscription</h2>
      <div>
         <label for="name" class="control-label">Your name : </label>
         <input type="text" id="name" name="Name" />
      </div>

      <div>
         <label for="email" class="control-label">Your email address : </label>
         <input type="text" id="email" name="Email" />
      </div>
```

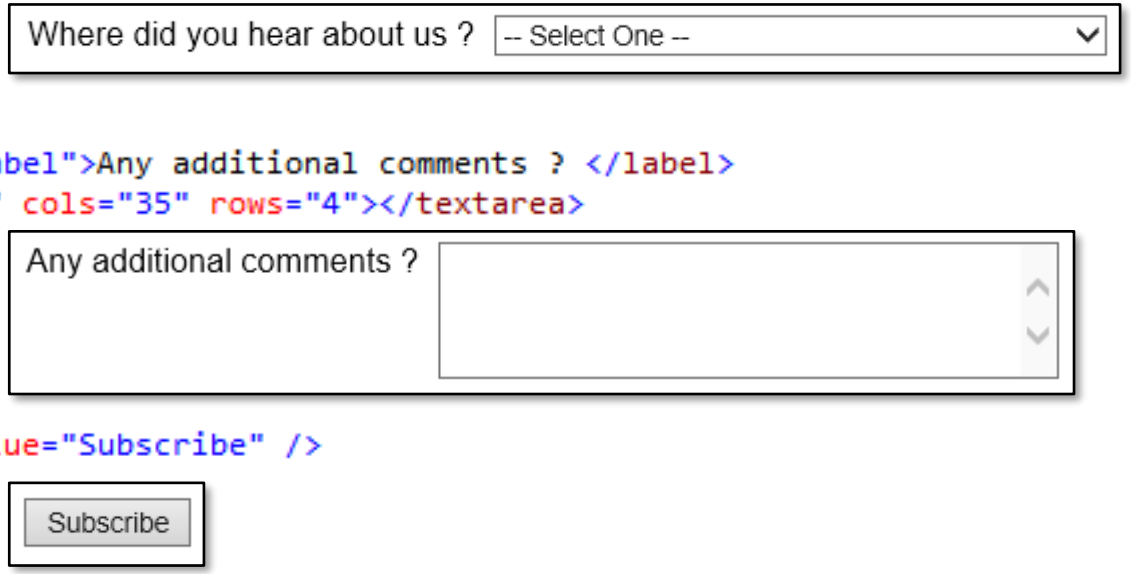


Subscription.cshtml

```
<div>
   <label class="control-label">Your gender : </label>
   <label for="male">
      <input type="radio" id="male" name="Gender" value="Male" />Male
   </label>
   <label for="female">
      <input type="radio" id="female" name="Gender" value="Female" />Female
   </label>
</div>

<div>
   <label class="control-label">Your interest :</label>
   <label for="recipe">
      <input type="checkbox" id="recipe" name="Recipe" value="Recipe" />Recipes
   </label>
   <label for="news">
      <input type="checkbox" id="news" name="News" value="News" />News
   </label>
   <label for="offers">
      <input type="checkbox" id="offers" name="Offers" value="Offers" />Offers
   </label>
</div>
```

Subscription.cshtml

```
        <div>
            <label for="refer" class="control-label">Where did you hear about us ? </label>
            <select id="refer" name="Refer">
                <option value="">-- Select One --</option>
                <option value="Friend">Friend</option>
                <option value="Magazine">Magazine</option>
                <option value="Newspaper">Newspaper</option>
                <option value="Radio">Radio</option>
            </select>
        </div>

        <div>
            <label for="comments" class="control-label">Any additional comments ? </label>
            <textarea id="comments" name="Comments" cols="35" rows="4"></textarea>
        </div>



        <div>
            <input type="submit" id="subscribe" value="Subscribe" />
        </div>
    </form>
    <a href="/Organic/Index">Back to Main</a>
</body>
</html>
```



OrganicController

Index and Subscription Actions

```
public class OrganicController : Controller
{
    public IActionResult Index()
    {
        return View(); // Views/Organic/Index.cshtml
    }

    public IActionResult Subscription()
    {
        return View(); // Views/Organic/Subscription.cshtml
    }
```

OrganicController

Confirmation Action

```
public IActionResult Confirmation()
{
    // Use IFormCollection for shorter coding
    IFormCollection form = HttpContext.Request.Form;

    // Remove leading/trailing spaces for TextFields
    string name     = form["Name"].ToString().Trim();
    string email    = form["Email"].ToString().Trim();
    string refer    = form["Refer"].ToString().Trim();
    string comments = form["Comments"].ToString().Trim();

    // ToString() for RadioButtons - change null to empty
    string gender = form["Gender"].ToString();

    // ToString() for CheckBoxes - change null to empty
    string recipe = form["Recipe"].ToString();
    string news   = form["News"].ToString();
    string offers = form["Offers"].ToString();
```

OrganicController

Confirmation Action (cont.)

```
    // Get CheckBoxes for Interest
    string interest = "";
    if (recipe.Equals("Recipe"))
        interest += "Recipe, ";
    if (news.Equals("News"))
        interest += "News, ";
    if (offers.Equals("Offers"))
        interest += "Offers, ";
    if (!interest.Equals(""))
        interest = interest.Substring(0, interest.Length - 2);

    // Display View
    ViewData["name"] = name;
    ViewData["email"] = email;
    ViewData["gender"] = gender;
    ViewData["interest"] = interest ;
    ViewData["refer"] = refer;
    ViewData["comments"] = comments;

    return View(); // Views/Organic/Confirmation.cshtml
}
```

Confirmation.cshtml

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Organics Fruits</title>
</head>
<body>
   <h1>Thank You</h1>

   <h4>Name : </h4>
   @ViewData["name"]<br />

   <h4>Email : </h4>
   @ViewData["email"]<br />

   <h4>Gender : </h4>
   @ViewData["gender"]<br />

   <h4>Interest :</h4>
   @ViewData["interest"]<br />

   <h4>Referral : </h4>
   @ViewData["refer"]<br />

   <h4>Comments : </h4>
   @ViewData["comments"]<br />

   <a href="~/Organic/Index">Back to Main</a>

</body>
</html>
```

§

Solving the Problem

Business Logic §

```
private int CalcCreditPoint(string activity, int days)
{
    int credits = 0;
    if (activity.Equals("Story Telling") ||
        activity.Equals("Art and Craft"))
    {
        credits = days * 10;
    }
    else if (activity.Equals("Traffic Control"))
    {
        credits = days * 5;
    }
    else if (activity.Equals("Music Appreciation"))
    {
        credits = days * 15;
    }
    return credits;
}
```

Validation Logic §

```
private bool CheckIfEmpty(params string[] list)
{
    foreach (string o in list)
    {
        if (o == null || o.Trim().Equals(""))
        {
            return true;
        }
    }
    return false;
}
```

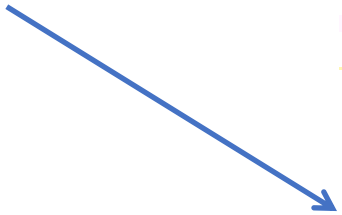
The keyword **params** allows a method to take a variable number of arguments

```
// Validation - Mandatory Fields must be entered
if (CheckIfEmpty(title, name, mobile, postal, activity))
{
    ViewData["Message"] = "Please enter all fields";
    return View("Volunteer");
}
```

Parent Controller §

Volunteer() action

- To display an empty form for input
- Views/Parent/Volunteer.cshtml



```
namespace L02.Controllers
{
    public class ParentController : Controller
    {

        public IActionResult Volunteer()
        {
            return View();
        }
```

Submit() action

- To read form entries
- To perform validation
- To calculate credit points
- To display "Thank you" confirmation
- Views/Parent/Submit.cshtml

Submit() Action §

```
public IActionResult Submit()
{
    // Use IFormCollection for shorter coding
    IFormCollection form = HttpContext.Request.Form;

    // Reading Input from TextFields
    string name   = form["Name"].ToString().Trim();
    string mobile = form["Mobile"].ToString().Trim();
    string postal = form["Postal"].ToString().Trim();

    // Reading Input from Dropdown List
    string activity = form["Activity"].ToString();

    // Reading Input from RadioButtons
    string title = form["Title"].ToString();

    // Reading Input from CheckBoxes
    string mon = form["Mon"].ToString();
    string wed = form["Wed"].ToString();
    string fri = form["Fri"].ToString();

    // Validation - Mandatory Fields must be entered
    if (CheckIfEmpty(title, name, mobile, postal, activity))
    {
        ViewData["Message"] = "Please enter all fields";
        return View("Volunteer");
    }
```

Submit() Action §

```
// Determine Number of Days Checked
int days = 0;
string daysSelected = "";

if (mon.Equals("Mon"))
{
    days++;
    daysSelected += "Monday, ";
}
if (wed.Equals("Wed"))
{
    days++;
    daysSelected += "Wednesday, ";
}
if (fri.Equals("Fri"))
{
    days++;
    daysSelected += "Friday, ";
}

// Validation - At least ONE day must be checked
if (days == 0)
{
    ViewData["Message"] = "Please check Days";
    return View("Volunteer");
}
```

Submit() Action §

```
    // Remove the last comma, E.g.
    // "Monday, Wednesday, Friday, "
    //                          ==> "Monday, Wednesday, Friday"
    // "Monday, Friday, "       ==> "Monday, Friday"
    // "Wednesday, Friday, " ==> "Wednesday, Friday"
    // "Wednesday, "            ==> "Wednesday"
    daysSelected = daysSelected.Substring(0, daysSelected.Length - 2);

    // Display Acknowledge View
    ViewData["FullName"] = title + " " + name;
    ViewData["Activity"] = activity;
    ViewData["Days"]     = daysSelected;
    ViewData["Credits"]  = CalcCreditPoint(activity, days);
    return View(); // Submit.cshtml
}
```

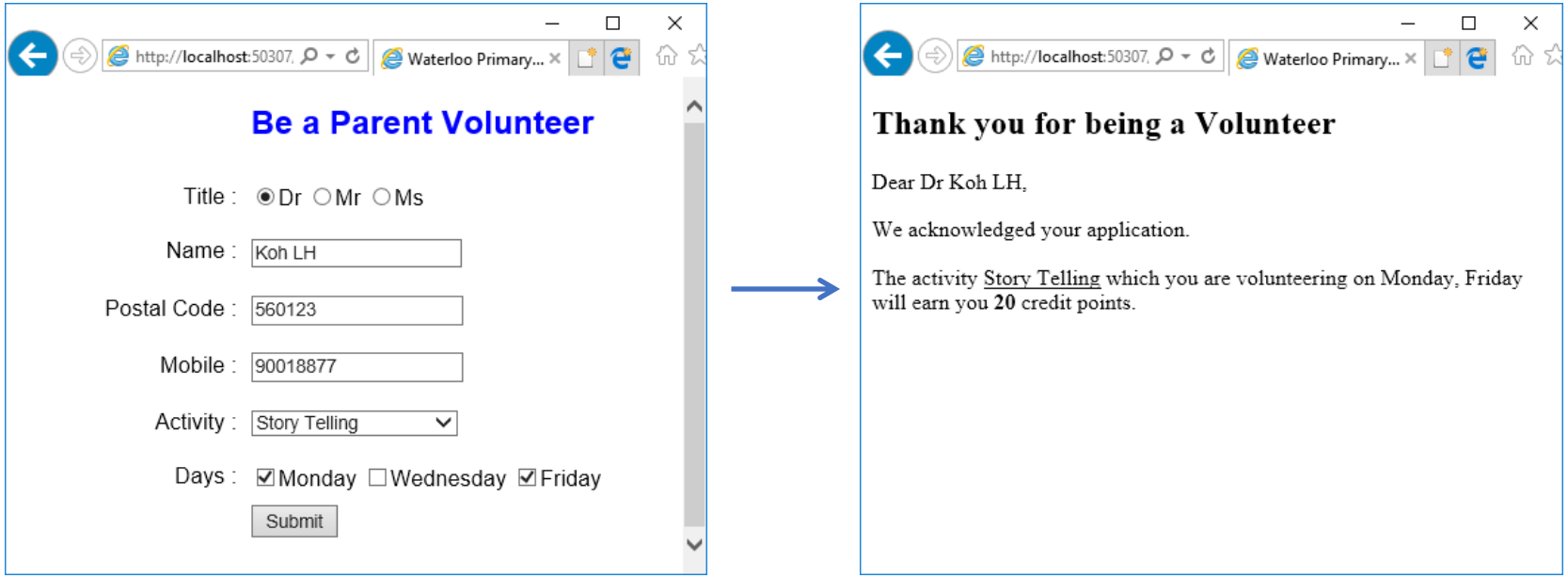
Views §

Volunteer.cshtml

- Empty form to allow parents to input

Submit.cshtml

- Thank You form to acknowledge parents' submissions



Volunteer.cshtml §

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Waterloo Primary School</title>
    <link href="/styles/waterloo.css" rel="stylesheet" />
</head>
<body>

    <form action="/Parent/Submit" method="post">

        <h2>Be a Parent Volunteer</h2>

        <div>
            <label class="control-label">Title :</label>
            <label for="dr">
                <input type="radio" name="Title" id="dr" value="Dr" />Dr
            </label>
            <label for="mr">
                <input type="radio" name="Title" id="mr" value="Mr" />Mr
            </label>
            <label for="ms">
                <input type="radio" name="Title" id="ms" value="Ms" />Ms
            </label>
        </div>
```

External stylesheet

Form declaration

Radio group

Volunteer.cshtml §

```
<div>
   <label class="control-label" for="name">Name :</label>
   <input type="text" name="Name" id="name" class="form-control" />
</div>

<div>
   <label class="control-label" for="postal">Postal Code :</label>
   <input type="text" name="Postal" id="postal" class="form-control" />
</div>

<div>
   <label class="control-label" for="mobile">Mobile :</label>
   <input type="text" name="Mobile" id="mobile" class="form-control" />
</div>

<div>
    <label class="control-label" for="activity">Activity :</label>
   <select name="Activity" id="activity" class="form-control">
      <option value="">-- SELECT --</option>
      <option value="Story Telling">Story Telling</option>
      <option value="Traffic Control">Traffic Control</option>
      <option value="Music Appreciation">Music Appreciation</option>
      <option value="Art and Craft">Art and Craft</option>
   </select>
</div>
```

Text field

Text field

Text field

Drop List

Volunteer.cshtml §

```
<div>
    <label class="control-label" for="StartDate">Days :</label>
    <label for="mon">
        <input type="checkbox" name="Mon" id="mon" value="Mon" />Monday
    </label>
    <label for="wed">
        <input type="checkbox" name="Wed" id="wed" value="Wed" />Wednesday
    </label>
    <label for="fri">
        <input type="checkbox" name="Fri" id="fri" value="Fri" />Friday
    </label>
</div>
```

Checkbox

```
<input type="submit" id="btn" value="Submit" />

<div class="message">
    @ViewData["Message"]
</div>
```

Submit button

```
</form>

</body>
</html>
```

Submit.cshtml §

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Waterloo Primary School</title>
</head>
<body>

    <h2>Thank you for being a Volunteer</h2>
    <p>
        Dear @ViewData["Fullname"],
    </p>
    <p>
        We acknowledged your application.
    </p>
    <p>
        The activity <u>@ViewData["Activity"]</u>
        which you are volunteering on @ViewData["Days"]
        will earn you <b>@ViewData["Credits"]</b> credit points.
    </p>

</body>
</html>
```

What You Have Learned

Overview of Programming in C#

Introduction to the MVC Framework

- Creating a Model
- Creating a View (Razor)
- Creating a Controller

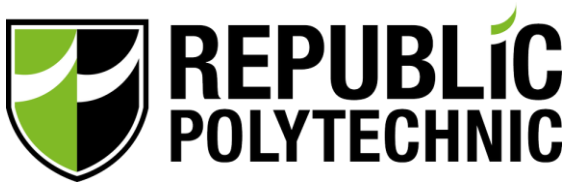
URL routing in MVC

Handling form inputs using MVC

Additional Resources

LinkedIn Learning

- Learning C#
- C# and .NET Essential Training



Lesson 3

Responsive Forms & Validation

C#

`String.Format` method

Syntax

```
String.Format(
  "Format String with {0} and {1}",
  item1, item2, item3, …… );
```

Takes a **format string** and several **arguments**

The **format string** consists of **format items** delimited by { }

- `{0}` for the first item
- `{1}` for the second item

Returns a formatted string

`String.Format` method

Examples

```
string aaa = String.Format("Apple={0}, Orange={1}", 12, 34);
// aaa is "Apple=12, Orange=34"

string bbb = String.Format("{1}{0}{2}{1}", "E", "T", "S")
// bbb is "TEST"

string ccc = String.Format("{0,7}", 35);
// width of the item is specified after the comma"
// ccc is "     35"

string cash = String.Format("{0:C}", 200);
// type of the item is specified after the colon"
// currency value in dollars is "$200.00"
```

DateTime

A structure that represents an instant in time

To create a date with a specific year, month, day, hour, minute, and second

```
DateTime date1 = new DateTime(2023, 10, 20, 9, 15, 1);
// 9:15:01am 20 October 2023

DateTime currentMoment = DateTime.Now;
```

Properties

- Day, Month, Year, Hour, Minute, Second

DateTime Struct (System) | Microsoft Docs

Working with `DateTime`

```
DateTime dobMary = new DateTime(1973, 2, 28);
DateTime dobJohn = new DateTime(1974, 7, 31);
DateTime dobMike = new DateTime(1974, 7, 31);
DateTime dobPaul = new DateTime(1975, 12, 10);

bool younger = dobMike > dobMary;   // true
bool older   = dobJohn < dobPaul;   // true
bool same    = dobJohn == dobMike;  // true

DateTime aDay = DateTime.Now;
// Add Years and Days
DateTime bDay = aDay.AddYears(2);
DateTime cDay = aDay.AddDays(12);

// Add Hours, Minutes, Seconds, Milliseconds, and Ticks
DateTime eDay = aDay.AddHours(4.25);
DateTime gDay = aDay.AddMinutes(15);
DateTime hDay = aDay.AddSeconds(45);
DateTime iDay = aDay.AddMilliseconds(200);
```

Formatting `DateTime`

```
DateTime dt = new DateTime(2008, 3, 9, 16, 5, 7, 123);

String.Format("{0:y yy yyy yyyy}", dt);  // "8 08 008 2008"   year
String.Format("{0:M MM MMM MMMM}", dt);  // "3 03 Mar March"  month
String.Format("{0:d dd ddd dddd}", dt);  // "9 09 Sun Sunday" day
String.Format("{0:h hh H HH}",     dt);  // "4 04 16 16"      hour 12/24
String.Format("{0:m mm}",          dt);  // "5 05"            minute
String.Format("{0:s ss}",          dt);  // "7 07"            second
```

```
DateTime dt = new DateTime(2023, 4, 9, 8, 30, 17)
```

Year, month, day, hour, minute, second.

Test Yourself –

- What format string for **dt** will output "8:30 9-Apr-23"?

```
String.Format("{0:HH:mm d-MMM-yy}", dt);
```

String Format for DateTime [C#] (csharp-examples.net)

`.Split` method

Syntax

- Calls from a string
- Takes one char argument for separator
  - Use ' ' instead of " "
- Returns an array of strings

Examples

```
string text = "red, green, blue";
string [] colors = text.Split(',');
// colors[0] is "red"
// colors[1] is " green"
// colors[2] is " blue"
```

`.ToUpper` and `.ToLower` methods

Syntax

- Calls from a string
- Takes no argument
- Returns a string with characters made uppercase or lowercase respectively

Examples

```
string x = "  Nissan";
string y = "  Toyota Camry  ";
string z = "Hyundai   ";
string a = x.ToLower();          // a is "  nissan"
string b = y.Trim().ToUpper();   // b is "TOYOTA CAMRY"
string c = z.ToUpper();          // c is "HYUNDAI   "
```

More about strings

`.Substring()` Method

- Takes 2 parameters
  - Start Position
  - Length
- Returns the specified sub-string

`.Length` Property

- Returns the length of the string

```
DateTime dt = new DateTime(2014, 4, 9, 8, 30, 17);

string str1 = "XYZABC";
string str2 = "1123611";
int num = str1.Length;                    // num  is     6
string str3 = str1.Substring(5, 1);       // str3 is     "C"
string str4 = str2.Substring(2, 3);       // str4 is     "236"
string msg = String.Format("I LOVE {0}", str3+str4);

                                          // I LOVE C236
```

Int32.TryParse & Double.TryParse

Purpose: To convert strings into numbers

Syntax:

- Takes two parameters
  - **string** containing a number to convert
  - out **int** or out **double**
- Returns a **bool**
  - true if string represents a number that can be converted

Examples:

```
bool outcome; int num1; double num2;
outcome = Int32.TryParse("-99", out num1);
// outcome is true, num1 is -99

outcome = Double.TryParse("12.5", out num2);
// outcome is true, num2 is 12.5
```

Int32.TryParse Method (System) | Microsoft Docs

Double.TryParse Method (System) | Microsoft Docs

`Int32.Parse` & `Double.Parse`

Purpose: To convert strings into numbers

Syntax:

- Takes one parameter
  - `String` containing a number to convert
- Returns a `Int32` or `Double` respectively

Examples:

```
int apple = Int32.Parse("-99");
// apple is -99
double orange = Double.Parse("88.1");
// orange is 88.1
int bomb = Int32.Parse("KaBoom");
// exception will be thrown
```

When given an invalid string,

- TryParse will return false and **not throw** exception
- Parse will **throw** exception

Extension methods

Methods that are added to an object without modifying its class or super classes.

Extension Methods in C#

- Must be defined as **`static`** methods in **`static`** classes
- The first argument of the method specifies the object to extend
  - The class name must be preceded by the keyword **`"this"`**

```
public static string Stretch(this String str)
    {
        string newstr = "";
        for (int i = 0; i < str.Length; i++)
        {
            newstr += str.Substring(i, 1) + " ";
        }
        return newstr.ToUpper();
    }
```

Extension methods

```
// Without Extension Methods
string abc = "APPLE";
string xyz = abc.Stretch(); // Compilation Error
```

```
// Extension Methods defined in Static Class
public static class StringExtMethods
{
    public static string Stretch(this String str)
    {
        string newstr = "";
        for (int i = 0; i < str.Length; i++)
        {
            newstr += str.Substring(i, 1) + " ";
        }
        return newstr.ToUpper();
    }
}
```

```
string abc = "APPLE";
string xyz = abc.Stretch(); // xyz is "A P P L E "
```

Generating random numbers

`Random` Class

- `Next()`
  - Returns a non-negative random integer
- `Next(int max)`
  - Returns a non-negative random integer less than max
- `Next(int min, int max)`
  - Returns a non-negative random integer within min (inclusive) and max (exclusive)

```
Random rnd = new Random();
shape.Side1 = rnd.Next(1,5);
shape.Side2 = rnd.Next(1,5);
shape.Side3 = rnd.Next(1,5);
```

ValidUtl.cs

This is the first "helper" class created by the module team. It enables you to use functions developed in your own code.

CheckIfEmpty()

```
public static bool CheckIfEmpty(params string[] list)
{
   foreach (string o in list)
   {
      if (o == null || o.Trim().Equals(""))
      {
         return true;
      }
   }
   return false;
}
```

The keyword `params` allows a variable *(multiple)* number of parameters.

```
if (ValidUtl.CheckIfEmpty(name, email, refer, comments, gender))
    {
        ViewData["Message"] = "Please enter or select all fields";
        return View("Subscription");
    }
```

`ValidUtl.cs` extension methods

```
String.IsInteger()
String.IsNumeric()
String.IsDate(string format)
String.ToDate(string format)
```

BootStrap

A CSS Library for creating Responsive Web Applications

http://getbootstrap.com/

For 2022, C236 will use Bootstrap 5.2

Introduction

Bootstrap is …

- A very popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first web sites.

We use bootstrap because …

- It can automatically adjust web applications to Desktops, Tablets and Smart Phones.

Bootstrap is open source and …

- Contains beautiful and functional built-in components which are easy to customize.

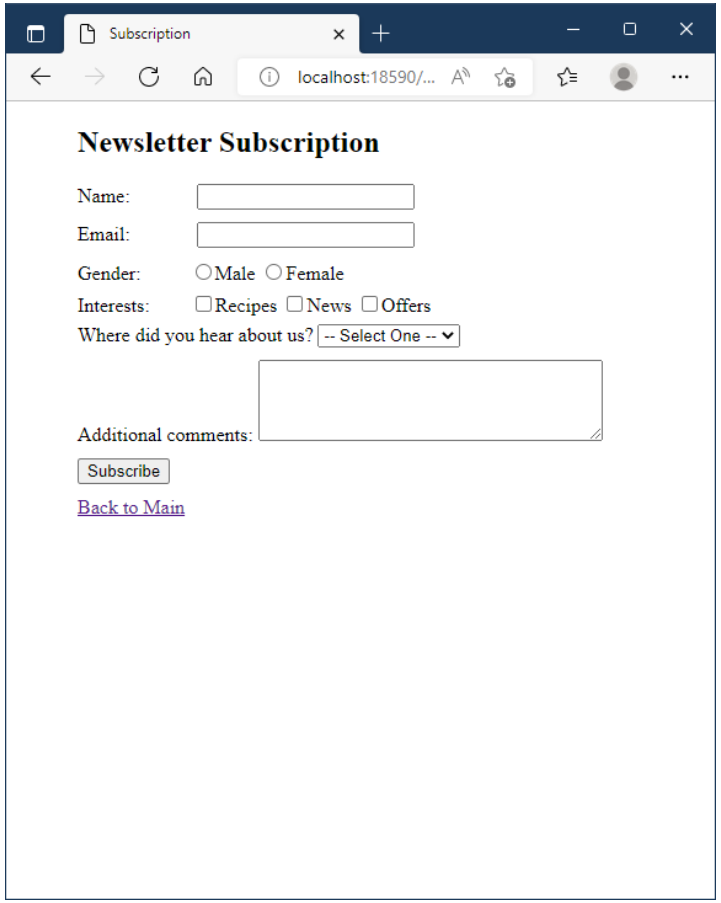
Demonstration

Bootstrap Grid System

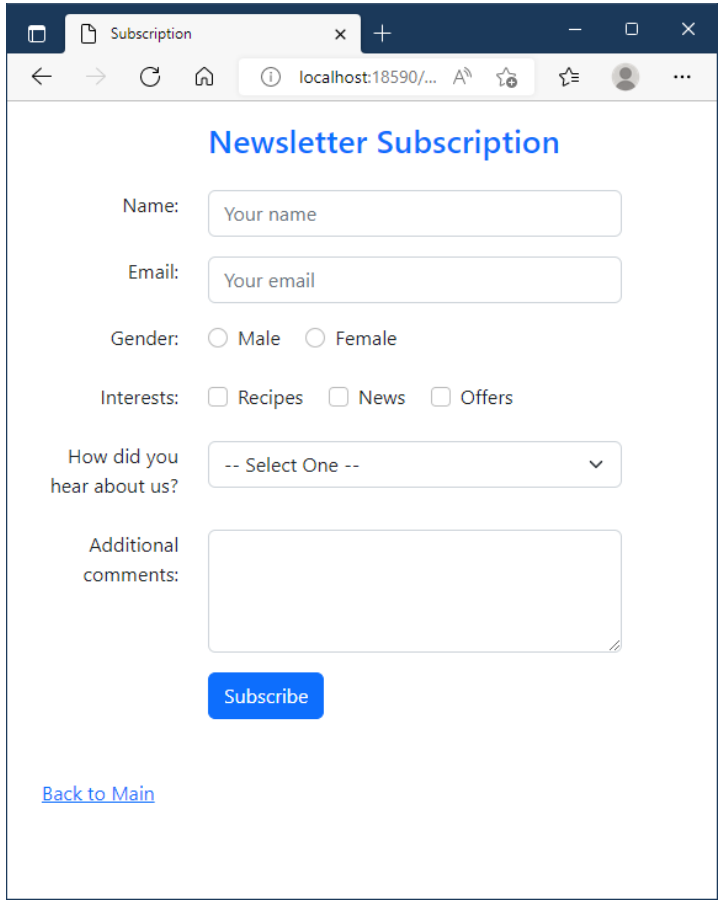
Bootstrap Forms

Creating Responsive Forms using Bootstrap 5

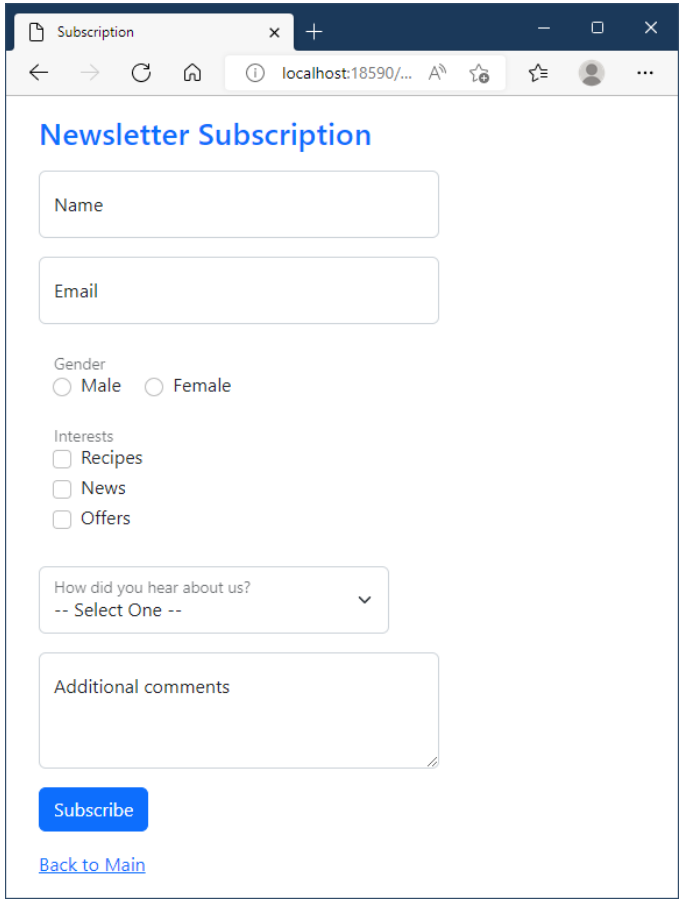
Applying Bootstrap classes to forms



No bootstrap
HTML & CSS only



Bootstrap 5 with labels
on left hand side



Bootstrap 5 with floating
labels

Bootstrap forms

Once bootstrap classes are correctly applied, the form becomes **responsive**. **Responsive** means that the form fields will automatically adjust their size, depending on the view port (screen resolution).

This module will examine **labels on the left hand side** first as there are some unique classes. (**row**, **offset**). This method also aligns better to the original (no bootstrap) form.

However, this module will concentrate on forms that use **floating labels** because:

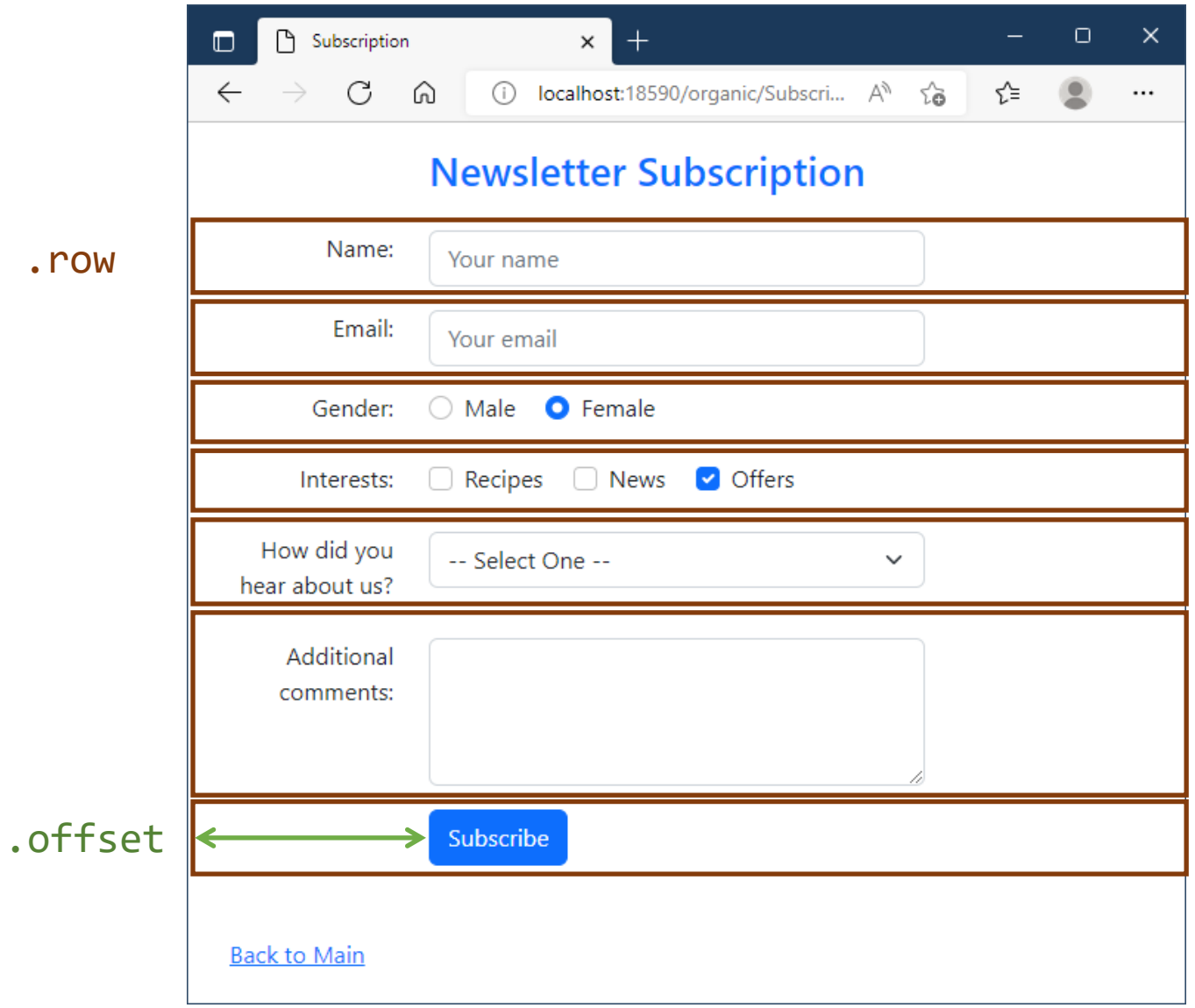
1. Fields can occupy more horizontal space without UI differences.
2. The UI is more consistent across various screen sizes.
3. The code is slightly simplified (fewer tags).

Labels Left Bootstrap Form

Full code is available to examine. Refer to Organic sample.

Organic Subscription Labels Left (Responsive)

.row and .offset classes



Rows and offsetting columns

Rows are a fundamental part of the **grid system** and are considered essential for responsive applications.

If you need more than one element in a given row, **you must use the .row** class. For example, in Left Labels, **both the label and the text box** are in the same row.

https://getbootstrap.com/docs/5.0/layout/columns/#offsetting-columns

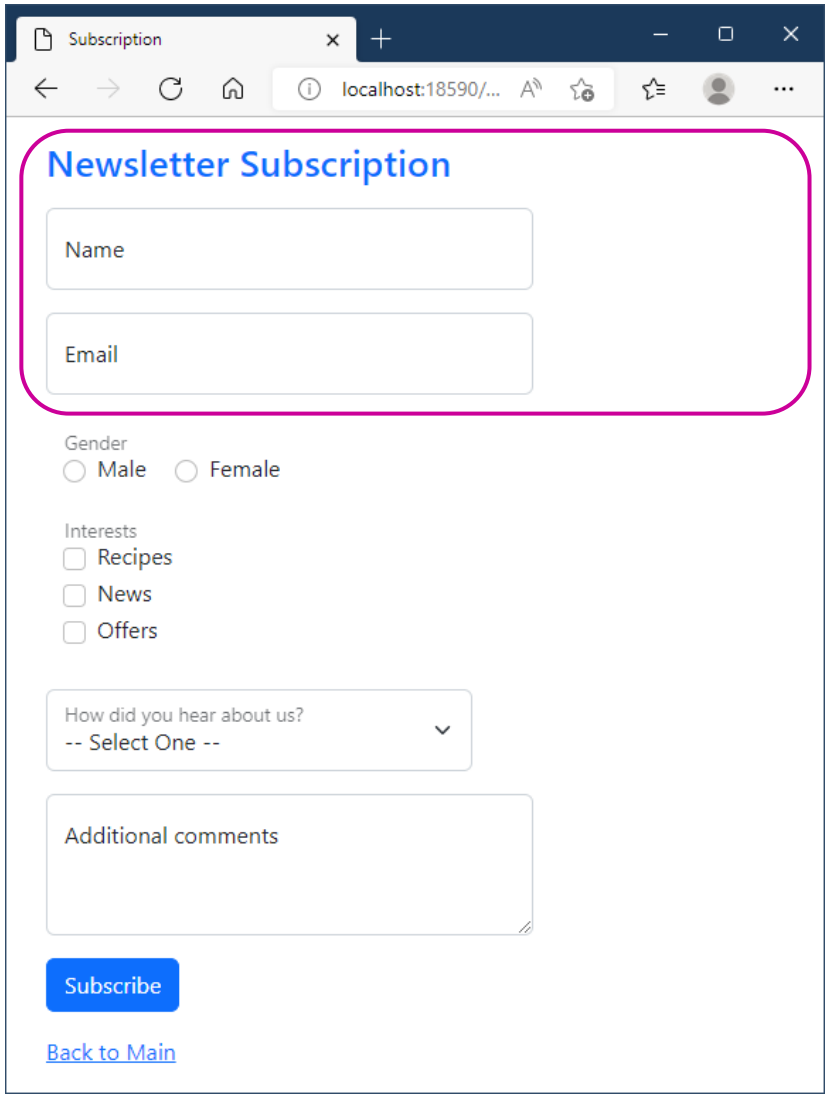
Offset: You can offset grid columns in two ways: using responsive **.offset-grid** classes and our **margin utilities**. Grid classes are sized to match columns. In Left Labels, we used offset to **align the button and the alert message** with the other form controls.

Floating Labels Bootstrap Form

Full code is available to examine. Refer to Organic sample.

Organic Subscription Floating Labels (Responsive)

Applying bootstrap classes to a form




```
<div class="container m-3">
  <form method="post" action="~/Organic/Confirmation">

    <div class="mb-3 text-primary">
        <h2>Newsletter Subscription</h2>
    </div>

   <div class="mb-3 col-8 form-floating">
        <input type="text" class="form-control" id="name" name="Name"
        placeholder="Your name" />
        <label for="name">Name</label>
    </div>

    <div class="mb-3 col-8 form-floating">
        <input type="email" class="form-control" id="email" name="Email"
        placeholder="Your email" />
        <label for="email">Email address</label>
    </div>
```

New Bootstrap 5 classes used:

container, m-3, mb-3, text-primary*, col-8, form-floating, form-control

* *text-primary simply derives the text color from bootstrap buttons (later)*

Grid system

https://getbootstrap.com/docs/5.0/layout/grid/

Breakpoints: Breakpoints are the building blocks of responsive design. Bootstrap's grid system uses a series of containers, rows, and columns to layout and align content.

Breakpoint	Class infix	Dimensions
X-Small	*None*	<576px
Small	`sm`	≥576px
Medium	`md`	≥768px
Large	`lg`	≥992px
Extra large	`xl`	≥1200px
Extra extra large	`xxl`	≥1400px

Spacing

https://getbootstrap.com/docs/5.0/utilities/spacing/

- The classes are named using the format {property}{sides}-{size} (Simpified version for C236)
- Property is m for margin and p for padding
- Sides are either t (margin or padding) top, b bottom, l left, r right, x both left and right, y both top and bottom or blank for all sides
- Size is either 0 (eliminate margin or padding) or 1 to 5 or auto.
- `<div class="mb-3">` A div with a bottom margin of 3
  - mb-3 (margin bottom of size 3)
  - m-3 (all margins of size 3)
  - py-2 (padding top and bottom of 2)

Grid system (cont.)

Containers: Containers center and horizontally pad your content. Use **.container** for a responsive pixel width, **.container-fluid** for width: 100% across all viewports and devices.

Example:

- `<div class="container">` A **div** which is a container.
- `<div class="container m-3">` A **div** which is a container that has a margin of 3 in all directions.

Grid system (cont.)

Columns: There are 12 template columns available per row, allowing you to create different combinations of elements that span any number of columns. Column classes indicate the number of template columns to span.

Example:

- `<div class="mb-3">` A div with a **bottom margin** of 3.
- `<div class="mb-3 col-8">` A div with a **bottom margin** of 3 that spans 8 out of a maximum of 12 equal columns of the container in which it resides.
- `<div class="mb-3 col-8 form-floating">`A div with a **bottom margin** of 3 that spans 8 out of a maximum of 12 equal columns of the container in which it resides. This div also has a **floating** label (later).

Form controls

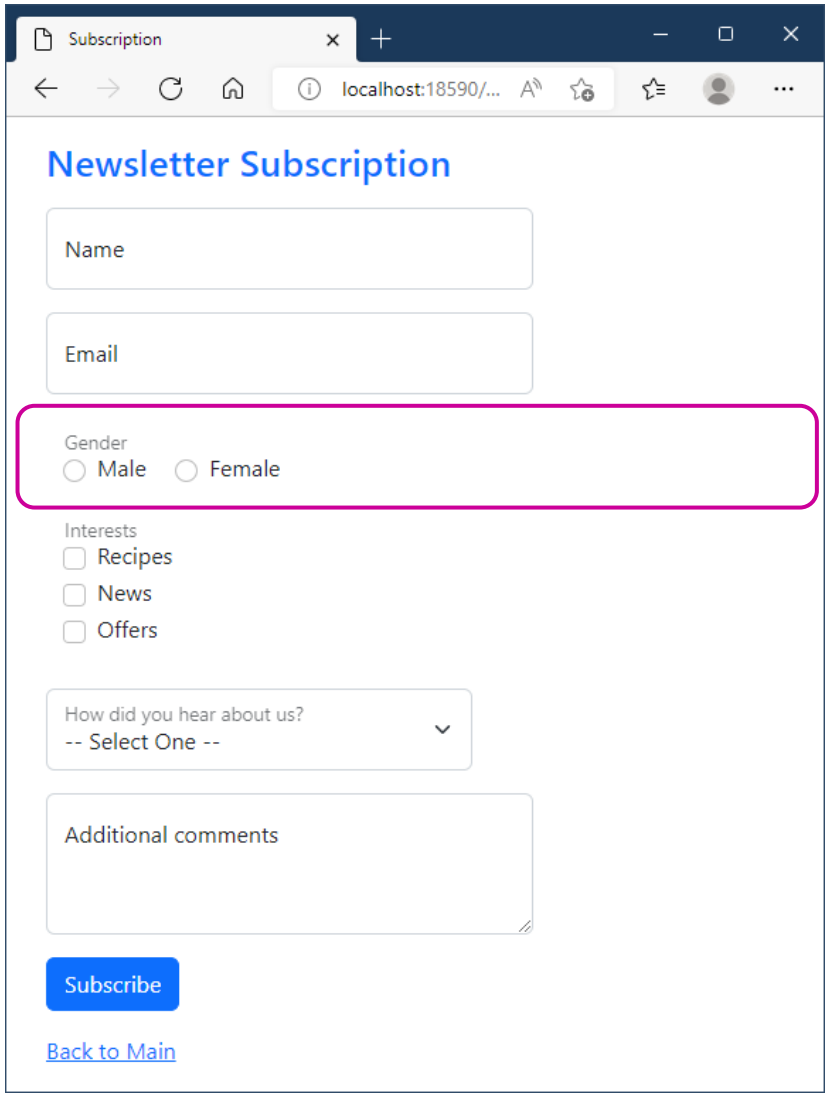
https://getbootstrap.com/docs/5.0/forms/form-control/

Form Controls: Give textual form controls like **<input>**s and **<textarea>**s an upgrade with custom styles, sizing, focus states, and more.

Example:

- `<input type="email" class="form-control"/>` An input that is decorated by a Bootstrap 5 form-control.
- `<input type="email" class="form-control" id="email" name="Email" placeholder="Your email" />` An input that is decorated by a Bootstrap 5 form-control. Note the placeholder is required for the animation effect on the title when the user starts to type.

Applying bootstrap classes to a form

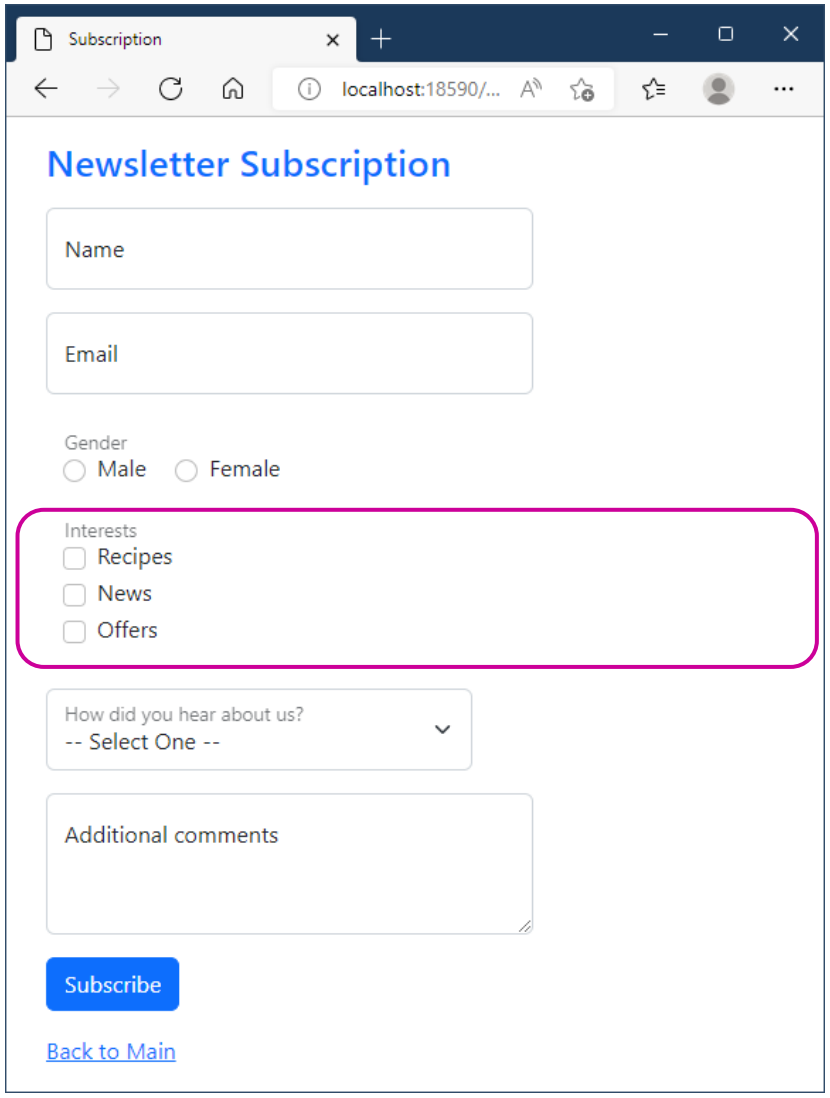



```
@*The following div has no mb-3 class to reduce the distance between
    two controls that have no borders. No need for col-x class either.*@
<div class="form-floating">
    <div class="border-0 form-control h-auto">
        <div class="form-check form-check-inline">
            <input type="radio" class="form-check-input"
                   id="male" name="Gender" value="Male" />
            <label class="form-check-label" for="male">Male</label>
        </div>
        <div class="form-check form-check-inline">
            <input type="radio" class="form-check-input"
                   id="female" name="Gender" value="Female" />
            <label class="form-check-label" for="female">Female</label>
        </div>
    </div>
    <label>Gender</label>
</div>
```

Changes:

- New divs are added. Class h-auto is for automatic height.
- For horizontal presentation of radio groups add the class form-check-inline. Remove that class for vertical presentation.
- border-0 removes the border from the group.

Applying bootstrap classes to a form




```
<div class="mb-3 form-floating">
    <div class="border-0 form-control h-auto">
        <div class="form-check">
            @*Remove the form-check-inline class for vertical alignment/>*@
            <input type="checkbox" class="form-check-input"
                   id="recipe" name="Recipe" value="Recipe" />
            <label class="form-check-label" for="recipe">Recipes</label>
        </div>
        <div class="form-check">
            <input type="checkbox" class="form-check-input"
                   id="news" name="News" value="News" />
            <label class="form-check-label" for="news">News</label>
        </div>
        <div class="form-check">
            <input type="checkbox" class="form-check-input"
                   id="offers" name="Offers" value="Offers" />
            <label class="form-check-label" for="offers">Offers</label>
        </div>
    </div>
    <label>Interests</label>
</div>
```

Changes:

- New **div**s are added.
- For horizontal presentation of radio groups add the class **form-check-inline**. Remove that class for vertical presentation.
- **border-0** removes the border from the group.

Checks and radios

https://getbootstrap.com/docs/5.0/forms/checks-radios/

Checks and radios: Browser default checkboxes and radios are replaced with the help of **.form-check-input**, a series of classes for both input types that improves the layout and behavior of their HTML elements, that provide greater customization and cross browser consistency. Note the only difference is `type="checkbox"` or `type="radio"` (Radio groups require the same `name`)

Example:

- `<input type="radio" class="form-check-input" id="male" name="Gender" value="Male" />` A **radio** control that is decorated by a Bootstrap 5.
- `<input type="checkbox" class="form-check-input" id="recipe" name="Recipe" value="Recipe" />` A **checkbox** control that is decorated by a Bootstrap 5.

Labels

https://getbootstrap.com/docs/5.0/forms/floating-labels/

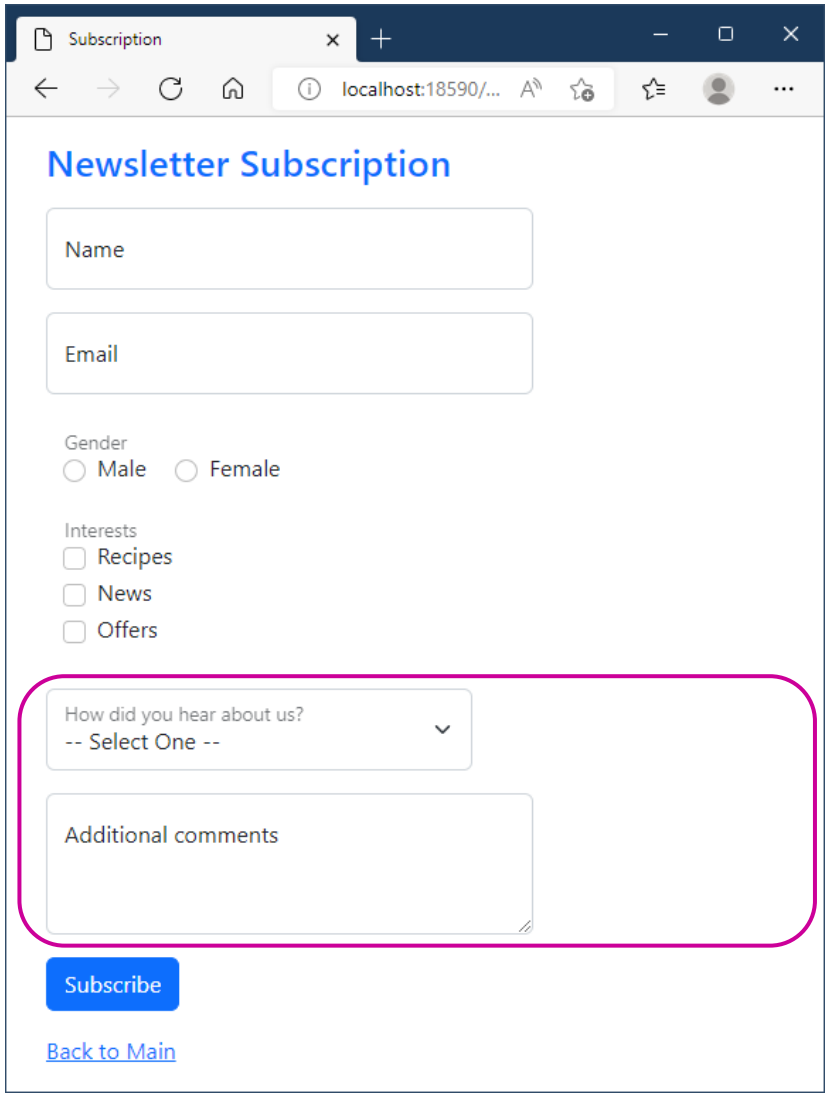
Bootstrap 5 has launched a new floating label element to the Forms component. It works on the **<input>**, **<select>** and **<textarea>** elements.

Wrap a pair of **<input class="form-control">** and **<label>** elements in **.form-floating** to enable floating labels with Bootstrap's textual form fields. A **placeholder is required** on each **<input>**. Also note that the **<input> must come first.**

Example:

- `<label for="male" class="form-check-label">Male</label>` A **label** control that is decorated by a Bootstrap 5.

Applying bootstrap classes to a form




```
<div class="mb-3 col-7 form-floating">
    <select class="form-select" id="refer" name="Refer">
        <option value="">-- Select One --</option>
        <option value="Friend">Friend</option>
        <option value="Magazine">Magazine</option>
        <option value="Newspaper">Newspaper</option>
        <option value="Radio">Radio</option>
    </select>
    <label for="refer">How did you hear about us?</label>
</div>

<div class="mb-3 col-8 form-floating">
    <textarea class="form-control" id="comments" name="Comments"
              style="height: 100px" placeholder="Add comments here">
    </textarea>
    <label for="comments">Additional comments</label>
</div>
```

New Bootstrap 5 classes used:

form-select

Select

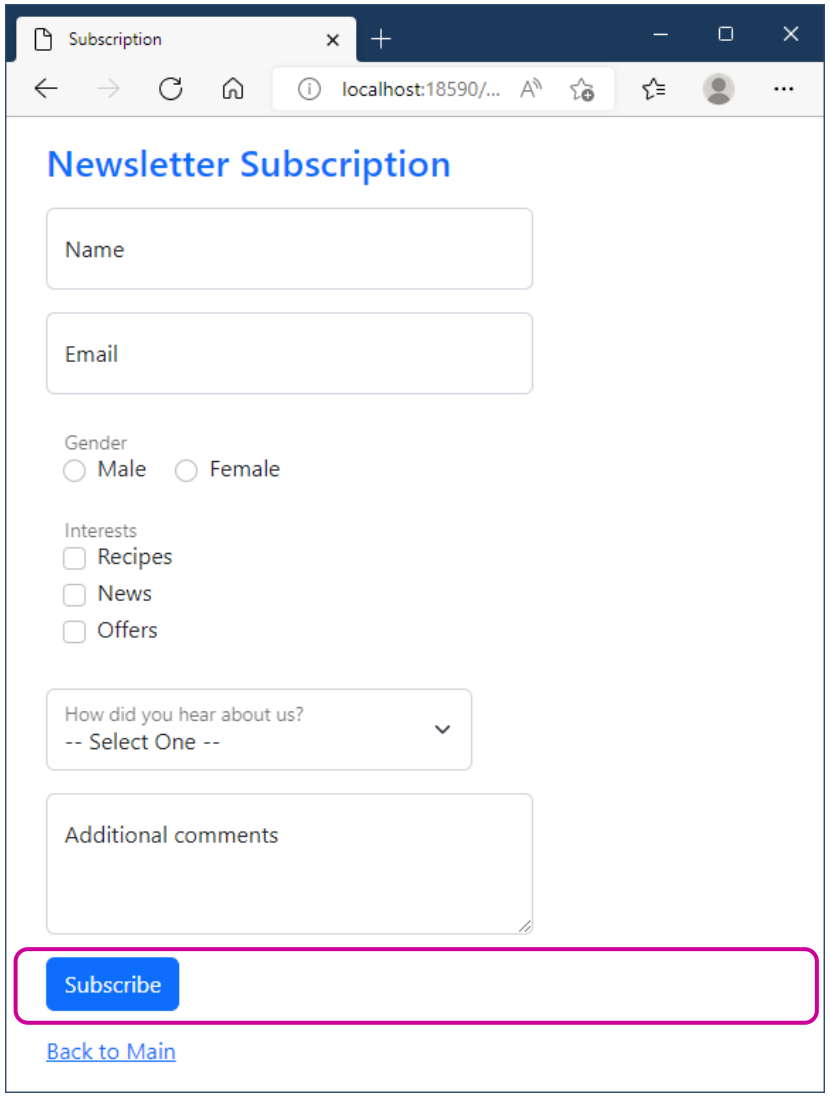
https://getbootstrap.com/docs/5.0/forms/select/

Select: Custom <select> menus need only a custom class, .form-select to trigger the custom styles. Custom styles are limited to the <select>'s initial appearance and cannot modify the <option>s due to browser limitations.

Example:

- `<select class="form-select" id="refer" name="Refer">` A **select** control that is decorated by a Bootstrap 5.
- `<option value="">-- Select One --</option>` A **select option** control.

Applying bootstrap classes to a form




```
<input type="submit" class="mb-3 btn btn-primary"
value="Subscribe" />
```

Bootstrap classes used:

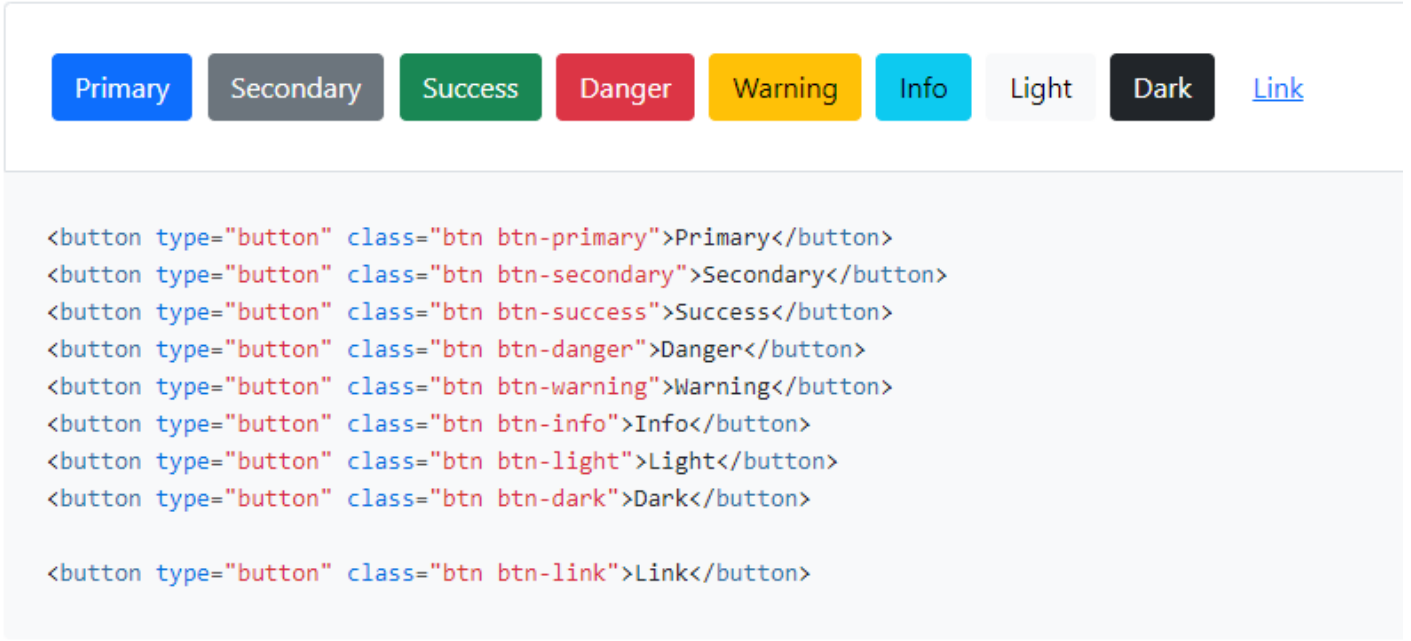
btn btn-primary

Buttons

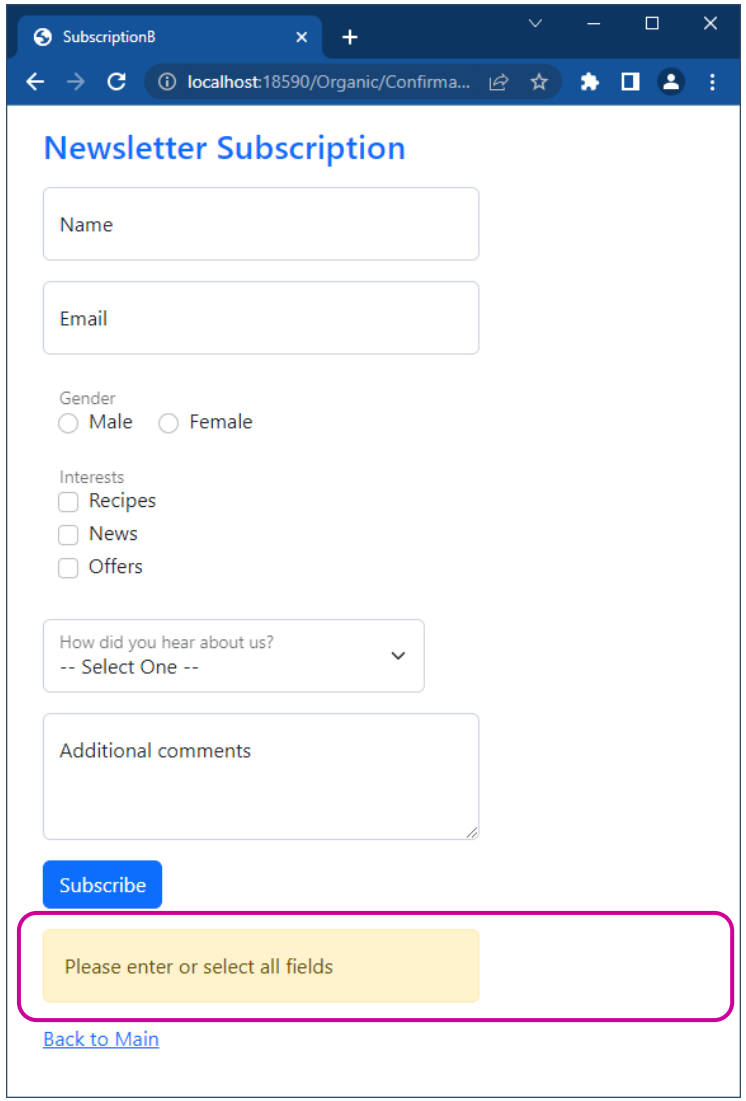
https://getbootstrap.com/docs/5.0/components/buttons/

Buttons: **Use Bootstrap's custom button styles for actions in forms, dialogs, and more with support for multiple sizes, states, and more.** Example:

- `<input type="submit" class="mb-3 btn btn-primary" value="Subscribe" />` A **submit button** that is decorated by a Bootstrap 5.



Applying bootstrap classes to a form




```
@if (ViewData["Message"] != null)
    {
        <div class="col-8">
            <div class="alert alert-warning" role="alert">
                @ViewData["Message"]
            </div>
        </div>
    }
```

New Bootstrap 5 classes used:

alert alert-warning

Alerts

https://getbootstrap.com/docs/5.0/components/alerts/

Alert: Provide contextual feedback messages for typical user actions with the handful of available and flexible alert messages.

Example:

- `<div class="alert alert-warning" role="alert">` A **div** control that is decorated by a Bootstrap 5 warning alert.
- `<div class="alert alert-warning" role="alert">`

  `@ViewData["Message"]`

  `</div>` A **div** control illustrating the complete Bootstrap 5 warning alert.

Alerts

A simple primary alert—check it out!

A simple secondary alert—check it out!

A simple success alert—check it out!

A simple danger alert—check it out!

A simple warning alert—check it out!

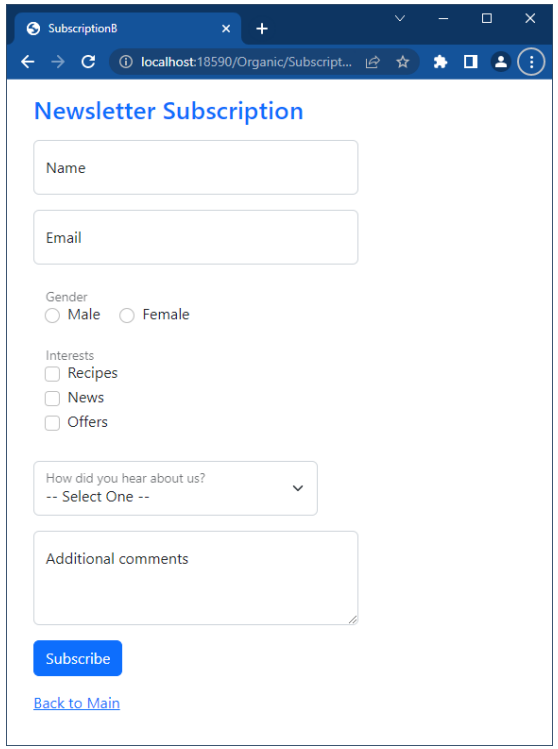
A simple info alert—check it out!

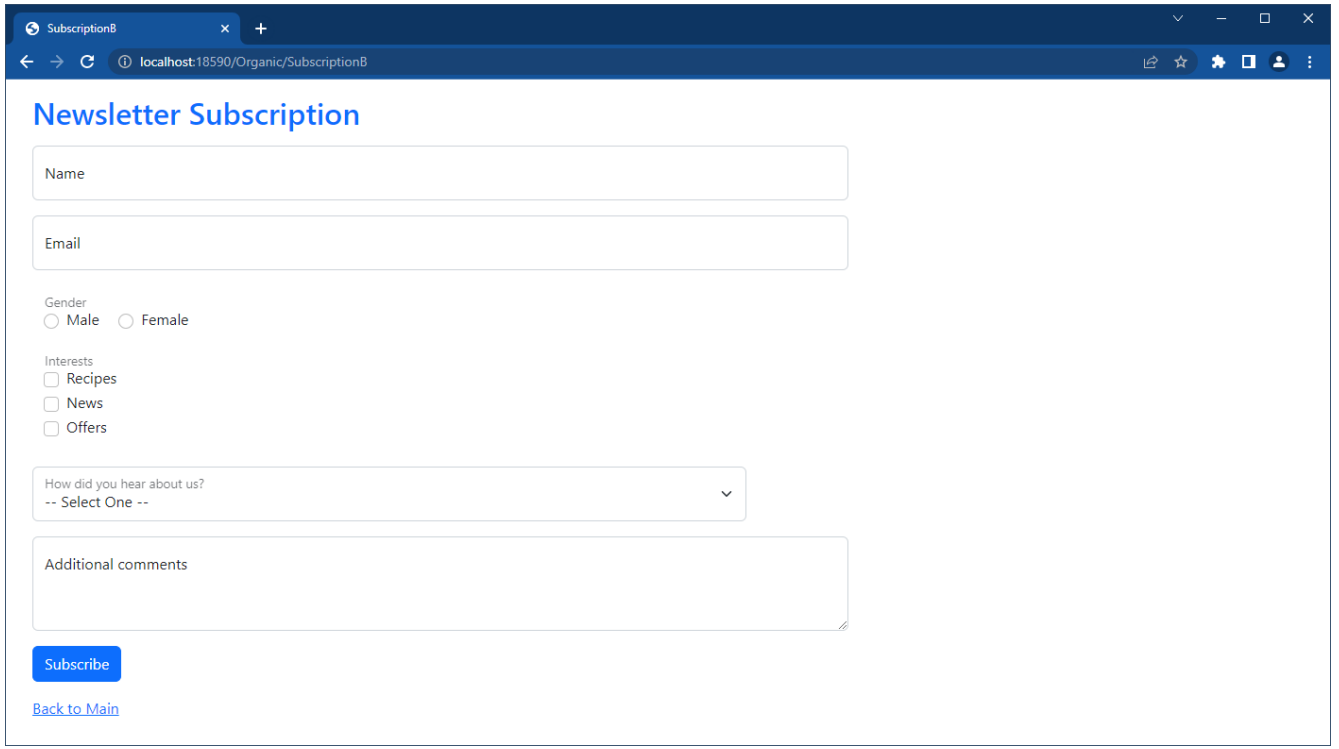
A simple light alert—check it out!

A simple dark alert—check it out!

Bootstrap 5 field resizing

For grids that are the same from the smallest of devices to the largest, use the .col and .col-* classes (e.g., .col-8). There is no need to specify specific breakpoint classes (e.g., col-**lg**-2).





Hypertext Transfer Protocol (HTTP)

HTTP Essential Training (linkedin.com)

COURSE

HTTP Essential Training

By: Morten Rand-Hendriksen

22,529 viewers · Released Apr 4, 2018

What is HTTP?

The **foundation** of data communication for the World Wide Web

- Enables communications between clients and servers.

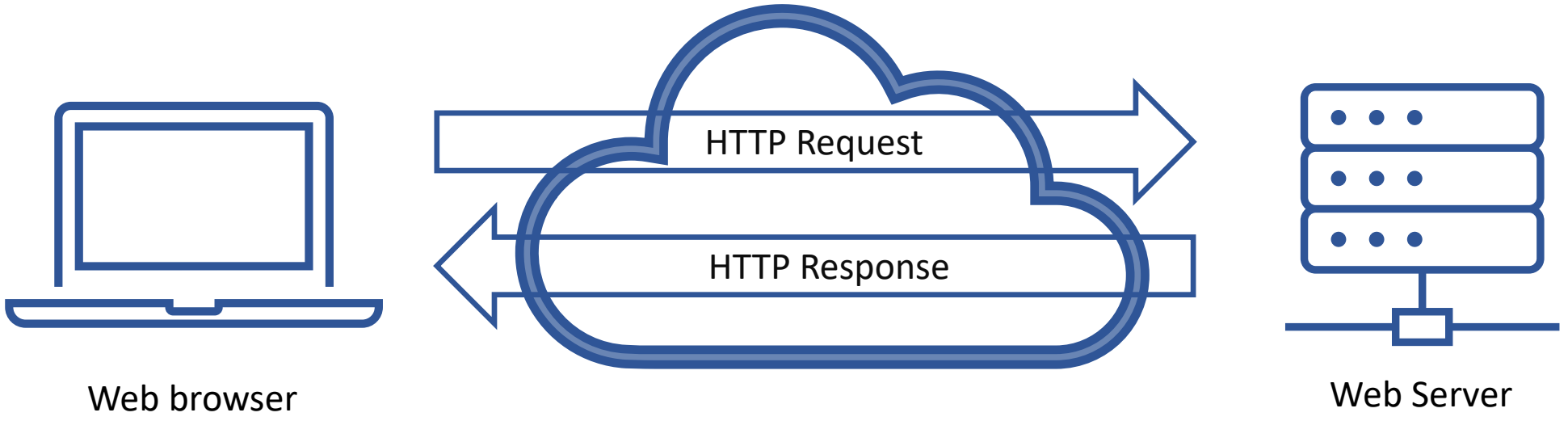
The **protocol** to exchange or transfer hypertext / hypermedia

- **Hypertext** is structured text which contains links to other texts.
- **HyperMedia** is a term used for **hypertext** which is not constrained to be text: it can include graphics, video, sound and other rich media.

HTTP works as a **request-response** protocol between a client and a server.

- A web browser may be the client
- An application on a computer that hosts a web site may be the server

Request and response protocols



HTTP request methods

GET method

- Requests data from a specified resource

POST method

- Submits data to be processed to a specified resource

HEAD method

- Same as GET but returns only HTTP headers and no document body

PUT method

- Uploads a representation of the specified URI

DELETE method

- Deletes the specified resource

OPTIONS method

- Returns the HTTP methods that the server supports

CONNECT method

- Converts the request connection to a transparent TCP/IP tunnel

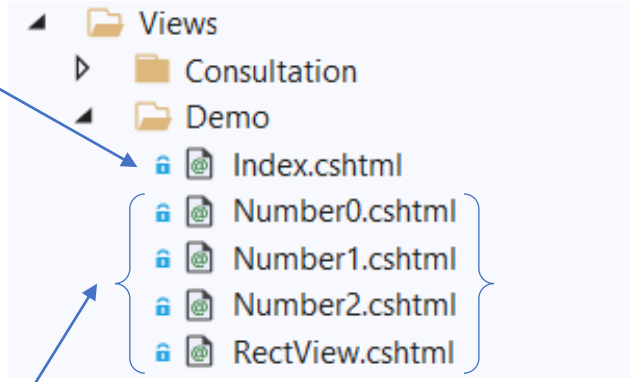
More MVC

Views

Default View

- Same name as the Action

```
public class DemoController : Controller
  {
     public IActionResult Index()
     {
        return View(); // Which cshtml?
     }
```



Named Views

- As specified in the argument

```
public class DemoController : Controller
  {
     public IActionResult Number(int id)
     {
        return View("Number" + id);
     }
```

Models

Two ways to pass information to Views

- Using `ViewData`
  - Putting objects into `ViewData["key"]` from the Controller
  - Getting objects from `@ViewData ["key"]` in the View

- Using `@model` directive
  - Passing the object as an argument to the `View()` method in the Controller
  - Specifying the class of the object using `@model` directive at the top of the View
  - Using `@Model` to reference to the object in the View

Using @model directive

Controllers/DemoController.cs

```
public class DemoController : Controller
{
    public IActionResult DemoRectangle()
    {
        Rectangle square = new Rectangle();
        square.Side1 = 3;
        square.Side2 = 3;
        return View("RectView", square);
    }
}
```

Models/Rectangle.cs

```
namespace L03.Models
{
    public class Rectangle
    {
        public int Side1 { get; set; }
        public int Side2 { get; set; }
    }
}
```

Views/Demo/RectView.cshtml

```
@model L03.Models.Rectangle

<html>
<head>
    <title>Rectangle</title>
</head>
<body>
    Length is @Model.Side1<br />
    Width is  @Model.Side2<br />
</body>
</html>
```

More Razor

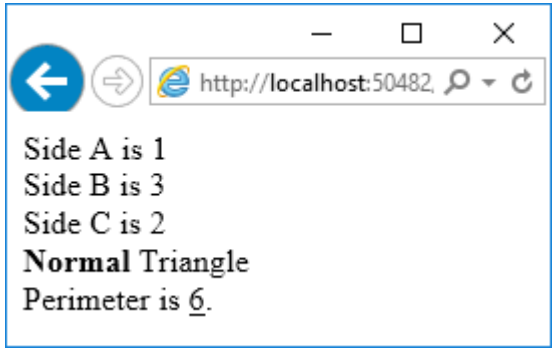
Markup syntax for embedding server-based code into webpages

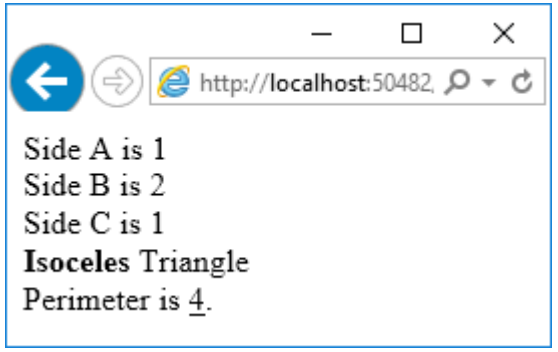
Conditionals

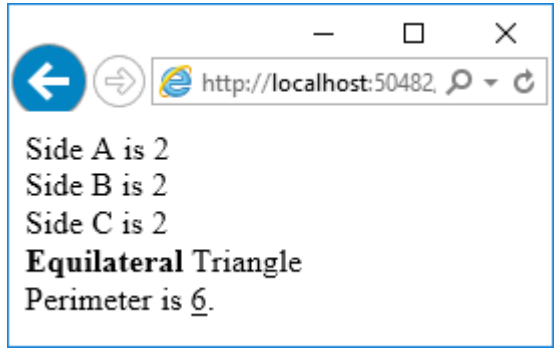
```
@if (value % 2 == 0)
{
    <p>The value is even</p>
}
```

```
@if (value % 2 == 0)
{
    <p>The value is even</p>
}
else if (value >= 1337)
{
    <p>The value is large.</p>
}
else
{
    <p>The value is not large and is odd.</p>
}
```

ShowTriangle web app








```
public IActionResult ShowTriangle()
     {
        Triangle shape = new Triangle();

        Random rnd = new Random();
        shape.Side1 = rnd.Next(1, 5);
        shape.Side2 = rnd.Next(1, 5);
        shape.Side3 = rnd.Next(1, 5);

        return View("TriView", shape);
     }
```

```
namespace L03.Models
{
   public class Triangle
   {
      public int Side1 { get; set; }
      public int Side2 { get; set; }
      public int Side3 { get; set; }
   }
}
```

Named view, passing model

Views\Demo\TriView.cshtml

```
@model L03.Models.Triangle

<html>
<head>
   <title>Triangle</title>
</head>
<body>
   Side A is @Model.Side1<br />
   Side B is @Model.Side2<br />
   Side C is @Model.Side3<br />
```

Razor Directive

Razor Expressions

Views\Demo\TriView.cshtml

```
@if (Model.Side1 == Model.Side2 &&
     Model.Side2 == Model.Side3)
{
   <b>Equilateral</b>
}
else if (Model.Side1 == Model.Side2 ||
         Model.Side2 == Model.Side3 ||
         Model.Side1 == Model.Side3)
{
   <b>Isoceles</b>
}
else
{
   <b>Normal</b>
}
Triangle<br />

Perimeter is <u>@(Model.Side1 + Model.Side2 + Model.Side3)</u>.
</body>
</html>
```

Razor Conditionals

Razor Explicit Expression

`View()` methods

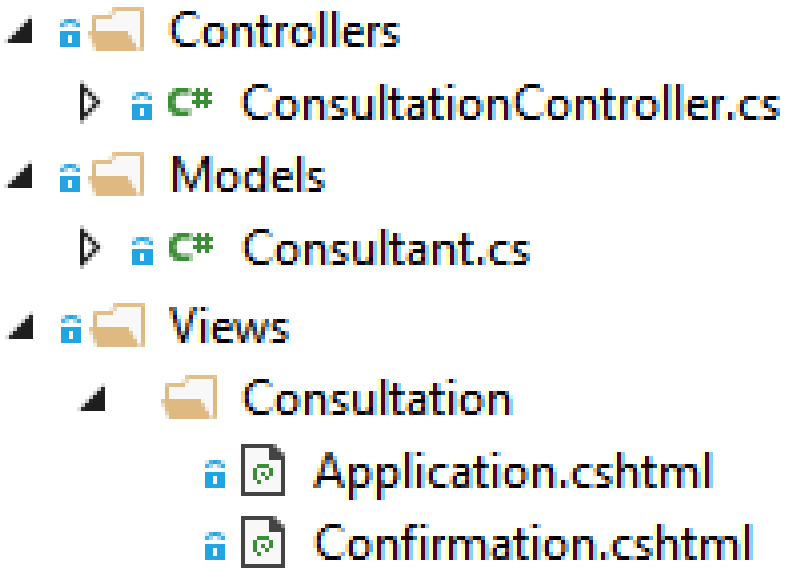
Four Ways to use the View() method in a controller

- Without Arguments – `View()`
  - Call the default view, that is the view with the same name as the action
- With a string – `View("vName")`
  - Call the view that is specified by "vName"
- With a model – `View(object)`
  - Call the default view passing the object over
- With a string and a model – `View("vName", object)`
  - Call the view that is specified by "vName" passing the object over

§

Solving the Problem

Project Structure §



Model – Consultant.cs §

```
namespace Lesson03.Models;

public class Consultant
{
    public string StaffName { get; set; } = null!;
    public string Email { get; set; } = null!;
    public string AcadPos { get; set; } = null!;
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }
}
```

Note: C# introduced **null safety**. For this reason StaffName, Email and AcadPos are declared with

property_name = null!;

This is known as the **null forgiving operator.** There will be more on **null safety** in subsequent weeks. But, for the moment, know that the **null forgiving operator** tells the compiler that something that could be null, is **safe to be accessed**. More detail in the **stack** overflow link below.

https://stackoverflow.com/questions/54724304/what-does-null-statement-mean

Business Logic §

```
private static int CalcWeekendDays(DateTime start, DateTime end)
{
    int wk = 0;
    int days = (end - start).Days + 1;
    for (int i = 0; i < days; i++)
    {
        DateTime day = start.AddDays(i);
        if (day.DayOfWeek == DayOfWeek.Saturday ||
            day.DayOfWeek == DayOfWeek.Sunday)
            wk++;
    }
    return wk;
}
```

Business Logic §

```
private const int WKDAY_PROF = 120;
private const int WKDAY_LECT = 100;
private const int WKDAY_ADJT = 90;

private const int WKEND_EXTRA = 80;

private static int CalcRenumeration(Consultant cst)
{
    int days = (cst.EndDate - cst.StartDate).Days + 1;
    int sat_sun = CalcWeekendDays(cst.StartDate, cst.EndDate);
    int amount = sat_sun * WKEND_EXTRA;

    if (cst.AcadPos.Equals("Professor"))
    {
        amount += WKDAY_PROF * days;
    }
    else if (cst.AcadPos.Equals("Lecturer"))
    {
        amount += WKDAY_LECT * days;
    }
    else if (cst.AcadPos.Equals("Adjunct"))
    {
        amount += WKDAY_ADJT * days;
    }
    return amount;
}
```

Consultation Controller §

```
public class ConsultationController : Controller
{
    public IActionResult Apply()
    {
        return View("Application");
    }

    public IActionResult Confirm()
    {
        IFormCollection form = HttpContext.Request.Form;
        string staffName = form["StaffName"].ToString().Trim();
        string email     = form["Email"].ToString().Trim();
        string acadPos   = form["AcadPosition"].ToString();
        string startDate = form["StartDate"].ToString().Trim();
        string endDate   = form["EndDate"].ToString().Trim();
        string agree     = form["Agree"].ToString();
```

Validation Logic §

```
if (ValidUtl.CheckIfEmpty(staffName, email, acadPos, startDate, endDate, agree))
{
    ViewData["Message"] = "Please enter or select all fields";
    return View("Application");
}

if (!startDate.IsDate("yyyy-MM-dd") ||
    !endDate.IsDate("yyyy-MM-dd"))
{
    ViewData["Message"] = "Invalid Date Format";
    return View("Application");
}

if (startDate.ToDate("yyyy-MM-dd") > endDate.ToDate("yyyy-MM-dd"))
{
    ViewData["Message"] = "End date should be later than Start date";
    return View("Application");
}
```

Passing Info to View §

```
// Create Consultant object
Consultant cst = new();
cst.StartDate = startDate.ToDate("yyyy-MM-dd");
cst.EndDate = endDate.ToDate("yyyy-MM-dd");
cst.StaffName = staffName;
cst.Email = email;
cst.AcadPos = acadPos;

// Calculate Renumeration
int payment = CalcRenumeration(cst);

// Pass renumeration to View using ViewData
ViewData["Amount"] = payment;
// Pass Consultant object to View as Model
return View("Confirmation", cst);
```

Confirmation.cshtml §

```
@model Lesson03.Models.Consultant

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <title>Confirmation</title>
</head>
<body>
    <div class="container m-3">
        <div class="row">
            <div class="col-4">
                <h2>Consultation</h2>
            </div>
        </div>

        <div class="row">
            <label class="control-label col-3">Contact:</label>
            <div class="col-5">
                @Model.StaffName (@Model.Email)
            </div>
        </div>
```

Confirmation.cshtml (cont.)

```
        <div class="row">
            <label class="control-label col-3">Position:</label>
            <div class="col-5">
                @Model.AcadPos
            </div>
        </div>

        <div class="row">
            <label class="control-label col-3">Period:</label>
            <div class="col-6">
                @String.Format("{0:dd/MM/yyyy}", Model.StartDate) to
                @String.Format("{0:dd/MM/yyyy}", Model.EndDate)
            </div>
        </div>

        <div class="row">
            <label class="control-label col-3">Renumeration:</label>
            <div class="col-5">
                @String.Format("{0:C}", ViewData["Amount"])
            </div>
        </div>
    </div>
</body>
</html>
```

Application.cshtml §

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <title>ApplicationBS</title>
</head>
<body>
    <div class="container m-3">
        <form method="post" action="/Consultation/Confirm">

            <div class="mb-3 text-primary">
                <h2>Be a Consultant</h2>
            </div>

            <div class="mb-3 col-9 form-floating">
                <input type="text" class="form-control" id="staffname" name="StaffName"
                       placeholder="Staff name" />
                <label for="name">Staff Name</label>
            </div>

            <div class="mb-3 col-9 form-floating ">
                <input type="email" class="form-control" id="email" name="Email"
                       placeholder="Your email" />
                <label for="email">Email</label>
            </div>
```

Application.cshtml §

```
<div class="mb-3 col-5 form-floating">
    <select class="form-select" id="acadposition" name="AcadPosition">
        <option value="">-- Select One --</option>
        <option value="Professor">Professor</option>
        <option value="Lecturer">Lecturer</option>
        <option value="Adjunct">Adjunct</option>
    </select>
    <label for="email">Position</label>
</div>

<div class="mb-3 col-5 form-floating ">
    <input type="date" class="form-control" id="startdate" name="StartDate"
           placeholder="DD/MM/YYYY" />
    <label for="email">Start Date</label>
</div>

<div class="mb-3 col-5 form-floating ">
    <input type="date" class="form-control" id="enddate" name="EndDate"
           placeholder="DD/MM/YYYY" />
    <label for="email">End Date</label>
</div>
```

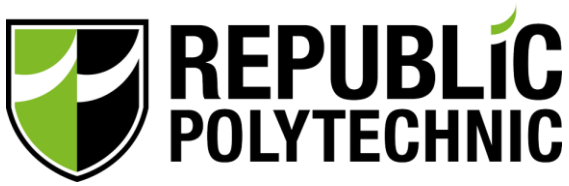
Application.cshtml §

```
            <div class="mb-3 col-9 form-floating">
                <div class="form-control">
                    <input type="checkbox" class="form-check-input"
                           id="terms" name="Agree" value="Agree" />
                    <label class="form-check-label" for="terms">
                        I agree to the terms and conditions
                    </label>
                </div>
                <label for="terms">Acceptance</label>
            </div>

            <input type="submit" class="mb-3 btn btn-primary" value="Submit" />

            <div class="row mb-3">
                <div class="col-sm-4 offset-sm-2">
                    @if (ViewData["Message"] != null)
                    {
                        <div class="alert alert-warning">
                            @ViewData["Message"]
                        </div>
                    }
                </div>
            </div>
        </form>
    </div>
```

Summary

What you have learned

- C# DateTime structs, String API's, Extension Methods
- ValidUtl.cs
- Bootstrap Grid concepts
- Using Bootstrap classes to create responsive UI's
- Razor's Conditional Structures



Lesson 4

Database and Presenting Tabular Data

Microsoft SQL Server

Create a database using Visual Studio

Database creation

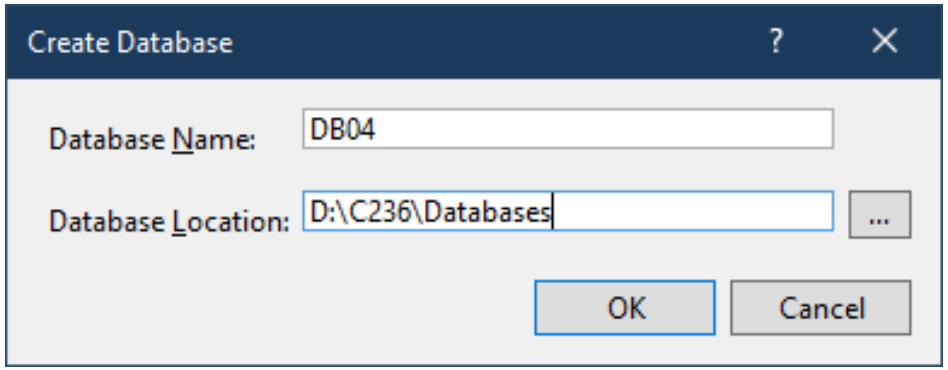
Open **SQL Server Object Explorer** from **View > SQL Server Object Explorer** if the window is not open

Expand **(localdb)\ProjectModels**

- Do **not** expand (localdb)\MSSQLLocalDB

Right-click **Databases > Add New Database**

- The Create Database window will appear



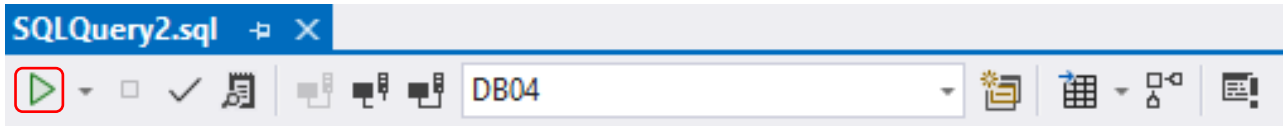
Create new tables and new data

In the **SQL Server Object Explorer**,

- Right-click on **DB04 > New Query**
- A new window called **SQLServer1.sql** will open

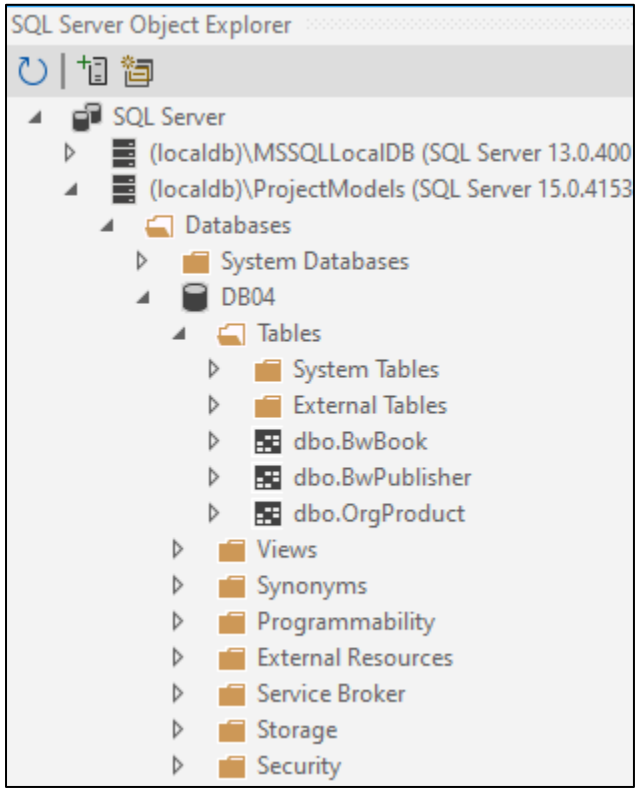
In the SQLServer1.sql window

- Enter `CREATE TABLE` statements to create tables
- Enter `INSERT` statements to add records to a table
- Enter `SELECT` statements to view records of a table
- To execute any statement, you must click the execute (green triangle) button on the top left.



Database management

In the **SQL Server Object Explorer**, expand the database **DB04** to view database components.



Apprentice activity

Setup DB

Worksheet Setup 1 -> 5.

C#

More advanced features of C#

var

Local variables can be declared as **`var`** instead of an explicit type (e.g., **`int, double, char, string`**)

The compiler will infer the type of the variable from the initialization.

The inferred type may be

- Built-in types (**`int, bool, DateTime, List<T>`** etc.)
- User-defined types (**`Customer, Pokedex`** etc.)

```
var i = 5;                      // i is int
var s = "Hello";                // s is string
var a = new[] { 0, 1, 2 };      // a is int[]
var list = new List<int>();     // list is List<int>
```

Formatted output

Number

```
double a = 5.5;
// output is "05.50"
String output = $"{a:00.00}";
```

Currency

```
double a = 5.5;
// output is "$5.50"
String output = $"{a:c}";
```

DateTime

```
DateTime a = DateTime.Today;
// output is "2016-11-02"
String output = $"{a:yyyy-MM-dd}";
```

Formatting types in .NET | Microsoft Docs

List<T>

```
using System.Collections.Generic;
```

- Similiar to Java ArrayList
- Creating the List
- Adding elements into List
- Accessing the List
- Iterating the List
- Insert into List
- Remove elements from List

Lists

Creating Lists

```
List<int> intList1 = new List<int>();
List<int> intList2 = new List<int>(){ 11, 21, 13, 4};

List<string> strList = new List<string>();

List<Pokedex> PokeList = new List<Pokedex>() {
  new Pokedex(1,"Bulbasaur","Grass","Poison",126,126,90),
  new Pokedex(2,"Ivysaur","Grass","Poison",156,158,120)
};
```

Adding Elements

```
intList1.Add(10);
intList1.Add(20);

strList.Add("Hello");

Pokedex p = new
        Pokedex(11,"Metapod","Bug","",56,86,100);

PokeList.Add(p);
```

Lists

Accessing Lists

```
List<string> myList =
   new List<string>() {"a","b","c","d","e"};

int len = myList.Count;      // 5
string item1 = myList[0];    // "a"
string item2 = myList[2];    // "c"

String abba = String.Format("{0}{1}{1}{0}",
                            myList[0], myList[1]);
```

Iterating through a List

```
string msg = "";
foreach (string x in myList)
{
   msg = msg + x;
}
// msg is "abcde"
```

Null safety

What it is and why it's important.

Null safety

If you're a .NET developer, chances are you've encountered the **System.NullReferenceException**.

This occurs at run time when a null is **dereferenced**—that is, when a variable is **evaluated at runtime**, but the variable refers to null.

```
// Declare variable and assign it as null.
FooBar fooBar = null;

// Dereference variable by calling ToString.
// This will throw a NullReferenceException.
_ = fooBar.ToString();

// The FooBar type definition.
record FooBar(int Id, string Name);
```

```
// Declare variable and assign it as null.
FooBar fooBar = null;

// Check for null
if (fooBar is not null)
{
    _ = fooBar.ToString();
}

// The FooBar type definition for example.
record FooBar(int Id, string Name);
```

The _ is called a discard character in C#. You are not interested in the result.

Null safety (cont.)

https://learn.microsoft.com/en-us/training/modules/csharp-null-safety/2-understand-nullability

C# 8.0 introduced nullable reference types, where you can express your **intent** that a reference type **might be null** or **is always non-null**.

```
#nullable enable
string first = string.Empty;    << Never null. It assigned, in this case the empty string.
string second;                  << Should never be null, even though it's initially null.
string? third;                  << Might be null
```

Nullable context

In the **Lesson[99].csproj** file for all projects, you will find the following properties set.

```
<Nullable>enable</Nullable>
<ImplicitUsings>enable</ImplicitUsings>
```

disable	The compiler behaves similarly to C# 7.3 and earlier
enable	The compiler enables all null reference analysis and all language features

Other possibilities include **warnings** and **annotations.**

! Null forgiving operator

! Null-forgiving operator. This is one way to tell the compiler that you know what you're doing, but it comes with the caveat that you *actually know what you're doing!*

```
namespace Lesson04.Models;

public class Person
{

    // Name is mandatory
    public string Name { get; set; } = "";             // Empty string ("" ≠ null)
    public string Name4 { get; set; } = String.Empty; // Empty string
    public string Name3 { get; set; } = null!;         // Set to null (dammit)

    //Name is optional
    public string? Name2 { get; set; }                 // Optional

    // DateTime, int and float CANNOT be null
    public DateTime Birthdate { get; set; }            // DateTime cannot be set to null
    public int Height { get; set; }                    // int cannot be set to null
    public float Weight { get; set; }                  // float cannot be set to null

}
```

?? Null-coalescing operator

Coalescing = *def:* combine (elements).

?? Null-coalescing operator. Returns the left operand if the operand is not null, otherwise it returns the right operand.

```
static void Sample()
{
    Person winner = new();
    winner.Name = DBFind("Kevan") ?? "No winner";
}
```

```
static string GetName()
{
    string name = GuiRead() ?? "No Name";
    return name;
}
```

If **DBFind("Kevan")** is **not null** then **winner.Name** = value returned from call to **DBFind("Kevan")**
If **DBFind("Kevan")** is **null** then **winner.Name = "No Winner"**

?. Null-conditional operator.

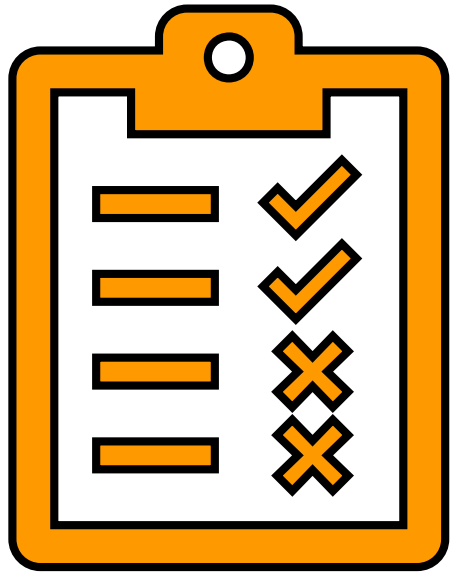
- ?. Returns null if the left operand is null, else it is treated just like a normal . operator.

```
static string GetAward()
{
    Person winner = new();
    var position = winner?.Placing; // int
    var award = winner?.AmtWon;     // float
    return position + award.ToString();
}
```

If **winner is null** then **position** = **null;**
If **winner is not null** then **position** = the value of **winner.Placing**

Test your understanding – Null Safety

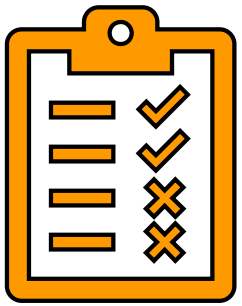
- Please start the C236 Lesson 04 SCORM package called MiniQuiz 1



Notes:

- Quizzes are formative in nature.
- Quiz answers do not contribute to your continuous assessment grade (CAG).
- You may attempt the SCROM package more than once.

Mini Quiz 1 Answers – Q1 & Q2 §



Q1

With reference to c#8 nullable reference types, match the following code on the left to the correct description on the right.

1. string v1 = "";	1. String cannot be null
2. string v3 = null!;	2. Programmer intends the string cannot be null
3. string? v2 = null;	3. String can be null

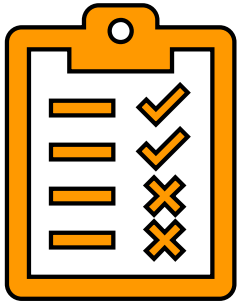
Q2

```
namespace Lesson04.Models;

public class Person
{
    public string Name { get; set; } = null!;
    public int Position { get; set; }
    public float AmtWon { get; set; }
}

public class Nullability
{
    public string? DBFind(string search)
    {
        if (search == "Kean")
            return "Kevan";
        return null;
    }

    public void Sample()
    {
        Person winner = new();
        winner.Name = DBFind("Kevan") ?? "No winner";
        Console.WriteLine(winner.Name);
    }
}
```

As a result of running the code on the left, what will be printed to the console?

- ✔ (●) No Winner
- ( ) Kean
- ( ) Nothing (blank, empty, nothing printed)
- ( ) Kevan

Mini Quiz 1 Answers – Q3 & Q4 §



Q3

```
namespace Lesson04.Models;

public class Person
{
    public string Name { get; set; } = null!;
    public int Position { get; set; }
    public float AmtWon { get; set; }
}

public class Nullability
{
    public string? DBFind(string search)
    {
        if (search == "Kean")
            return "Kevan";
        return null;
    }

    public void Sample()
    {
        Person winner = new();
        winner.Name = DBFind("Kevan") ?? "No winner";
        Console.WriteLine(winner.Name);
    }
}
```

Why is there a '?' behind the word 'string' in the definition of 'DBFind'?

- ○ Kevan may not be found
- ✔ ◉ The return string may be null
- ○ Kean may not be found
- ○ The search string may be null

Q4

Which of the following illustrates the null-conditional operator?

- ○ var age = p.Age?;
- ✔ ◉ var age = p?.Age;
- ○ var age = p??Age;
- ○ var age? = p.Age;

Apprentice activity

Setup Application

Worksheet Setup 6 -> 19.

HTML tables

Please watch Chapter 10

HTML tables

Purpose

- Provides a straightforward way to mark up structured tabular data
- Display that data in a form that is easy for users to read and digest

Elements

- Table
  `<table></table>`
- Rows
  `<tr></tr>`
- Data
  `<td></td>`

HTML table with CSS

```
<table>
    <tr>
        <td>NP</td>
        <td>Ngee Ann Polytechnic</td>
    </tr>
    <tr>
        <td>NYP</td>
        <td>Nanyang Polytechnic</td>
    </tr>
    <tr>
        <td>RP</td>
        <td>Republic Polytechnic</td>
    </tr>
    <tr>
        <td>SP</td>
        <td>Singapore Polytechnic</td>
    </tr>
    <tr>
        <td>TP</td>
        <td>Temasek Polytechnic
    </td>
</table>
```

NP Ngee Ann Polytechnic
NYP Nanyang Polytechnic
RP Republic Polytechnic
SP Singapore Polytechnic
TP Temasek Polytechnic

```
<style>
   table {
      width: 100%;
      border: 1px solid;
      border-collapse: collapse;
   }
   th, td {
      border: 1px solid;
   }
</style>
```

NP	Ngee Ann Polytechnic
NYP	Nanyang Polytechnic
RP	Republic Polytechnic
SP	Singapore Polytechnic
TP	Temasek Polytechnic

Table header

- Element

  `<th></th>`

```
<thead>
    <tr>
        <th>Code</th>
        <th>Name</th>
    </tr>
</thead>
<tbody>
   <tr>
        <td>NP</td>
        <td>Ngee Ann Polytechnic</td>
   </tr>
   …
```

Code	Name
NP	Ngee Ann Polytechnic
NYP	Nanyang Polytechnic
RP	Republic Polytechnic
SP	Singapore Polytechnic
TP	Temasek Polytechnic

Country codes

Use HTML to represent the following table

ISO	COUNTRY	CODE
HK	Hong Kong	852
ID	Indonesia	62
MY	Malaysia	60
PH	Philippines	63
SG	Singapore	65
TD	Thailand	62

Bootstrap tables

Bootstrap table CSS classes

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

`.table`

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

`.table-striped`

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

`.table-hover`

Bootstrap table CSS classes

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

`.table`

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

`.table-bordered`

[Bootstrap Tables (w3schools.com)]

Apprentice activity

Create HTML Tables

Worksheet Setup 20 -> 23.

Solution (HTML) §

```
<body>
    <div>
        <table>
            <tr>
                <th>ISO</th>
                <th>COUNTRY</th>
                <th>CODE</th>
            </tr>
            <tr>
                <td>HK</td>
                <td>Hong Kong</td>
                <td>852</td>
            </tr>
            <tr>
                <td>ID</td>
                <td>Indonesia</td>
                <td>62</td>
            </tr>

         …
```

Solution (HTML with bootstrap) §

```
<body>
    <div class="container">
        <div class="col-8">
            <h3>Country Codes</h3>
            <table class="table table-bordered table-striped table-hover">
                <tr>
                    <th>ISO</th>
                    <th>COUNTRY</th>
                    <th>CODE</th>
                </tr>
                <tr>
                    <td>HK</td>
                    <td>Hong Kong</td>
                    <td>852</td>
                </tr>
                <tr>
                    <td>ID</td>
                    <td>Indonesia</td>
                    <td>62</td>
                </tr>

                …
```

Razor control structures

`while, for, foreach`

Looping

new

```
@{
    var people =
        new List<Person> {
            new Person("John", 33),
            new Person("James", 41)
        };
}
```

while

```
@{ int i = 0; }
@while (i < people.Count)
{
    var p = people[i];
    <div>Name: @p.Name</div>
    <div>Age:  @p.Age</div>
    i++;
}
```

Looping

for

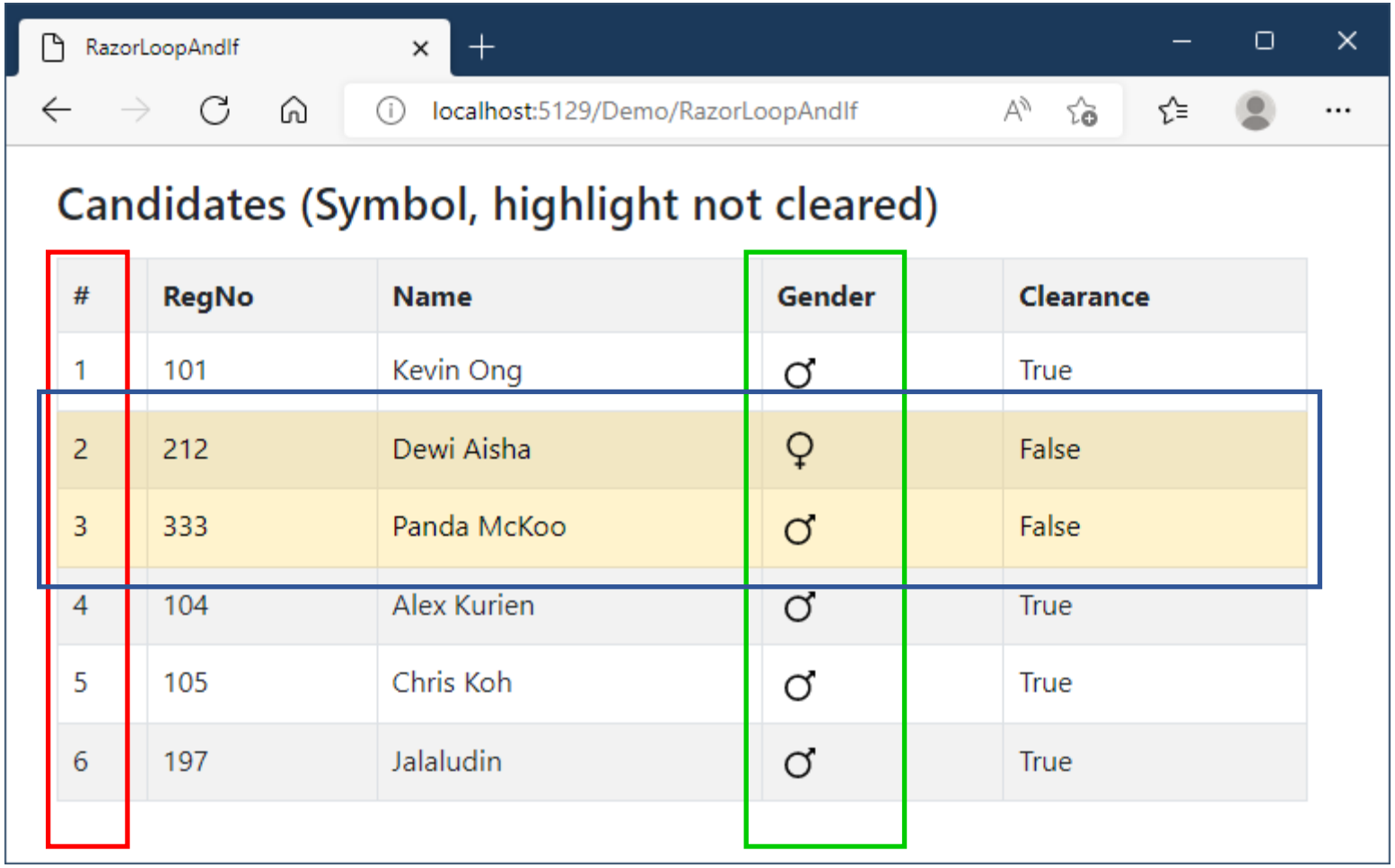
```
@for (int i = 0; i < people.Count; i++)
{
    var p = people[i];
    <div>Name: @p.Name</div>
    <div>Age:  @p.Age</div>
}
```

foreach

```
@foreach (var p in people)
{
    <div>Name: @p.Name</div>
    <div>Age:  @p.Age</div>
}
```

Razor's control structures

Highlighted Rows



Serial Numbers

Images for Gender

Razor code

```
@using Lesson04.Models
@model List<Candidate>

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <title>RazorLoopAndIf</title>
</head>
<body>
    <div class="container m-3">
        <h3 class="mb-3">Candidates (Symbol, highlight not cleared)</h3>
        <table class="table table-bordered table-hover table-striped">
            <tr>
                <th>#</th>
                <th>RegNo</th>
                <th>Name</th>
                <th>Gender</th>
                <th>Clearance</th>
            </tr>
```

Razor code

```
@{
    var sno = 1;
}

@foreach (var c in Model)
{
    var highlight = "";
    if (!c.Clearance)
    {
        highlight = "table-warning";
    }
    var image = c.Gender.Equals("M") ? "male.svg" : "female.svg";

    <tr class="@highlight">
        <td>@(sno++)</td>
        <td>@c.RegNo</td>
        <td>@c.Name</td>
        <td><img src="~/images/@image" width="25" /></td>
        <td>@c.Clearance</td>
    </tr>
}
        </table>
    </div>
</body>
</html>
```

Apprentice activity

Experiment with Razor

Worksheet Setup 24 -> 28.

SQL quick recap

CREATE TABLE statement

Recap from C207 Database Systems

```
CREATE TABLE <table name> (
   <col 1> <col 1 type> (<precision>) <constraint>,
   <col 2> <col 2 type> (<precision>) <constraint>,
   …

   PRIMARY KEY <primary key>

   CONSTRAINT <foreign key name>
       FOREIGN KEY <column name>
       REFERENCES <table>(<col>)
)
```

CREATE TABLE BwPublisher

```
CREATE TABLE BwPublisher (
  PubID      INT           NOT NULL IDENTITY,
  PubName    VARCHAR(250) NOT NULL,
  PubAddr    VARCHAR(250) NOT NULL,
  PRIMARY KEY (PubID)
);
```

IDENTITY

- Value of this column is auto-incremented from the previous value
- Equivalent to MYSQL's AUTO_INCREMENT

NOT NULL

- Column cannot take null value.

PRIMARY KEY

- Values of column(s) must be unique

CREATE TABLE BwBook

```
CREATE TABLE BwBook (
  Isbn                VARCHAR(20) PRIMARY KEY,
  Title               VARCHAR(250) NOT NULL,
  Lang                VARCHAR(45)  NOT NULL,
  Price               FLOAT        NOT NULL,
  Qty                 INT          NOT NULL,
  BkType              VARCHAR(45)  NULL,
  PubID               INT          NOT NULL,
  CONSTRAINT fk FOREIGN KEY (PubID)
     REFERENCES BwPublisher(PubID)
);
```

FOREIGN KEY

- To maintain referential integrity
- `publisher_id` in Book table must exist as a Primary Key in Publisher table

SELECT statement

Syntax

```
SELECT <column list>
FROM   <table list>
WHERE  <condition>
       [[AND|OR]<condition>]
```

Examples

```
SELECT isbn, title
FROM   bwbook
WHERE  qty > 10
```

```
SELECT *
FROM bwpublisher
WHERE PubName LIKE 'M%'
```

SELECT statement

More Examples

```
SELECT *
FROM   bwbook b
INNER JOIN bwpublisher p
ON     b.PubID = p.PubID
WHERE  (Lang = 'English' OR Lang = 'Mandarin')
AND    Price > 70
```

```
SELECT Country
FROM   OrgProduct
GROUP BY Country
HAVING COUNT(OrgDesc) > 1
```

Database connection string

In the Solution Explorer, open the file **appsettings.Development.json**

```
{...
    },
    "ConnectionStrings": {
        "DefaultConnection": "DataSource=(localdb)\\ProjectModels;
                    Initial Catalog=DB04;
                    Integrated Security=True",
        "ProductionConnection":"Server=tcp:sqlsoirp.database.windows.net,1433;
                    Database=dbsoirp;UserID=<userid>;
                    Password=<pw>;
                    Encrypt=True;
                    TrustServerCertificate=False;
                    Connection Timeout=45;"
    }
}
```

Note: You **cannot split** lines for connection strings. It is done here simply for ease of illustration.

Accessing databases

Microsoft's APIs for Data Access

`DBUtl` class

Helper Class for Database Access

`string DB_CONNECTION;`

- Determines which database to connect

`string DB_Message`

- Stores any error message

`GetTable(string sql)` method

- Method to get records from database using SQL

`ExecSQL(string sql)` method

- Method to **`insert`**, **`update`**, **`delete`** records from database using SQL

DB_CONNECTION

Inside `DBUtl.cs` helper class

```
private static readonly string DB_CONNECTION = config.GetValue<String>("DefaultConnection") ?? "";
```

Inside `appsettings.Development.json`

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=(localdb)\\
          ProjectModels;
          Initial Catalog=DB04;
          Integrated Security=True",
    "ProductionConnection":
          …
  }
}
```

`GetTable` method

```
public static DataTable GetTable(string sql, params object?[] list)
    {
        ...// Escape quotes on all params.
        DB_Message = "";
        DB_SQL = string.Format(sql, list);
        DataTable dt = new();
        using SqlConnection dbConn = new(DB_CONNECTION);
        using SqlDataAdapter dAdptr = new(DB_SQL, dbConn);

        try
        {
            dAdptr.Fill(dt);
            return dt;
        }
        catch (System.Exception ex)
        {
            DB_Message = ex.Message;
            return dt;
        }
    }
```

`SqlConnection`

Represents an open connection to a SQL Server database

SqlConnection Class (System.Data.SqlClient) | Microsoft Docs

`SqlDataAdapter`

Represents a set of data commands and a database connection that are used to fill the DataSet and update a SQL Server database.

SqlDataAdapter Class (System.Data.SqlClient) | Microsoft Docs

DataTable

A **DataTable** represents an in-memory cache of data.

- Used to store results of **SELECT** queries from database.
- Part of the **System.Data** namespace

```
using System.Data;
```

- Consists of a collection of other objects

DataTable Class (System.Data) | Microsoft Docs

DataTable construction

A DataTable consists of

- DataRowCollection (Rows)
  - DataRow (Each Row)
  - Count
- DataColumnCollection (Columns)
  - DataColumn (Each Column)
  - Count

```
string sql = "SELECT * FROM table";
DataTable dt = DBUtl.GetTable(sql);
DataRowCollection rows    = dt.Rows;
DataColumnCollection cols = dt.Columns;
int rr = rows.Count;    // number of rows
int cc = cols.Count;    // number of columns
```

DataTable structure

DataTable dt

cols

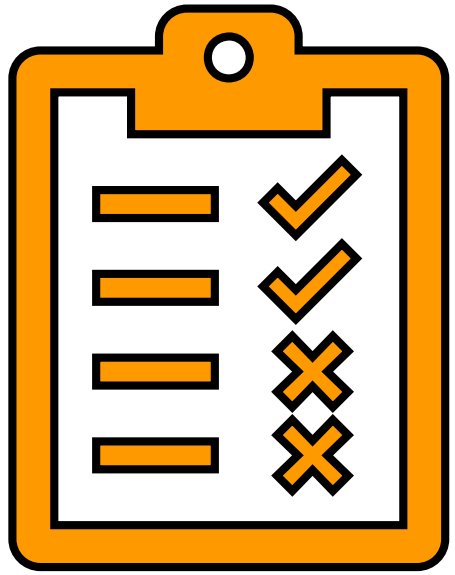
	0	1	2	……	Count-1
	"col1"	"col2"	"col3"		
0					
1		"pear"			
2		"apple"			
…					
Count-1					

rows

```
string a = dt.Rows[2][1].ToString();    // "apple"
string b = dt.Columns[2].ToString();    // "col3"
```

Test your understanding - DataTables

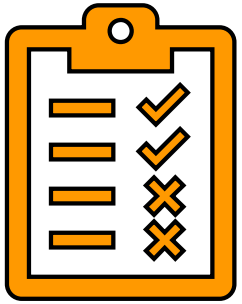
- Please start the C236 Lesson 04 SCORM package called MiniQuiz 2



Notes:

- Quizzes are formative in nature.
- Quiz answers do not contribute to your continuous assessment grade (CAG).
- You may attempt the SCROM package more than once.

Mini Quiz 2 - Answers §



Q1

Drag and drop the words to their places:

```
DataTable dt = DBUtl.GetTable(sql);
```

The total number of columns is [dt.columns.Count]

The total number of rows is [dt.rows.Count]

Name of the third column is [dt.Columns[2]]

Name of the last column is [dt.Columns] [[dt.Columns.Count-1]]

Q2

```
DataTable dt = DBUtl.GetTable(sql);
```

The value of the "salary" column in the second record is dt.Rows[[1]]["salary"]

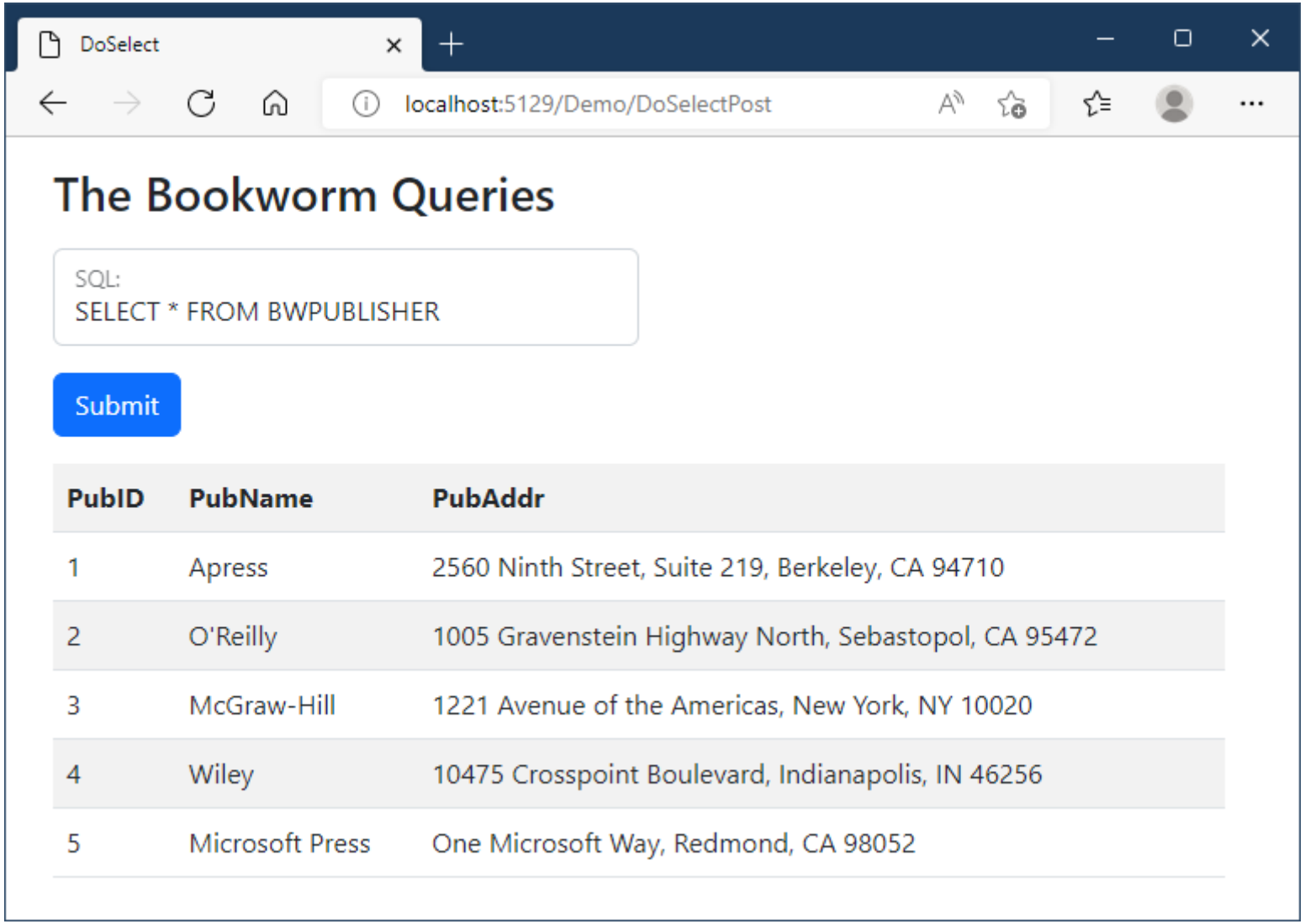
Q3

Drag and drop the words to their places:

The value of the last column in the last record is dt.Rows[[dt.Rows.Count-1]][[dt.Columns.Count-1]].

Data access in web-apps

Demo/DoSelect



Controller

```
public IActionResult DoSelect()
{
    return View();
}

public IActionResult DoSelectPost()
{
    string sql = HttpContext.Request.Form["sql"].ToString().ToUpper().Trim();

    DataTable dt = DBUtl.GetTable(sql);
    if (dt.Rows.Count == 0)
    {
        ViewData["Message"] = DBUtl.DB_Message;
    }
    ViewData["ExecSQL"] = DBUtl.DB_SQL;
    return View("DoSelect", dt);
}
```

View

```
@using System.Data
@model DataTable

<!DOCTYPE html>
<html>
<head>
    <title>DoSelect</title>
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
</head>

<body>
    <div class="container m-3">
        <form method="post" action="~/Demo/DoSelectPost">

            <div class="mb-3">
                <h2>The Bookworm Queries</h2>
            </div>

            <div class="mb-3 col-6 form-floating">
                @*Example of null-conditional perator (?.) below in attribute 'value'*@
                <input type="text" class="form-control" id="sql" name="Sql"
                       placeholder="Your query" value="@ViewData["ExecSQL"]?.ToString()" />
                <label for="name">SQL:</label>
            </div>
```

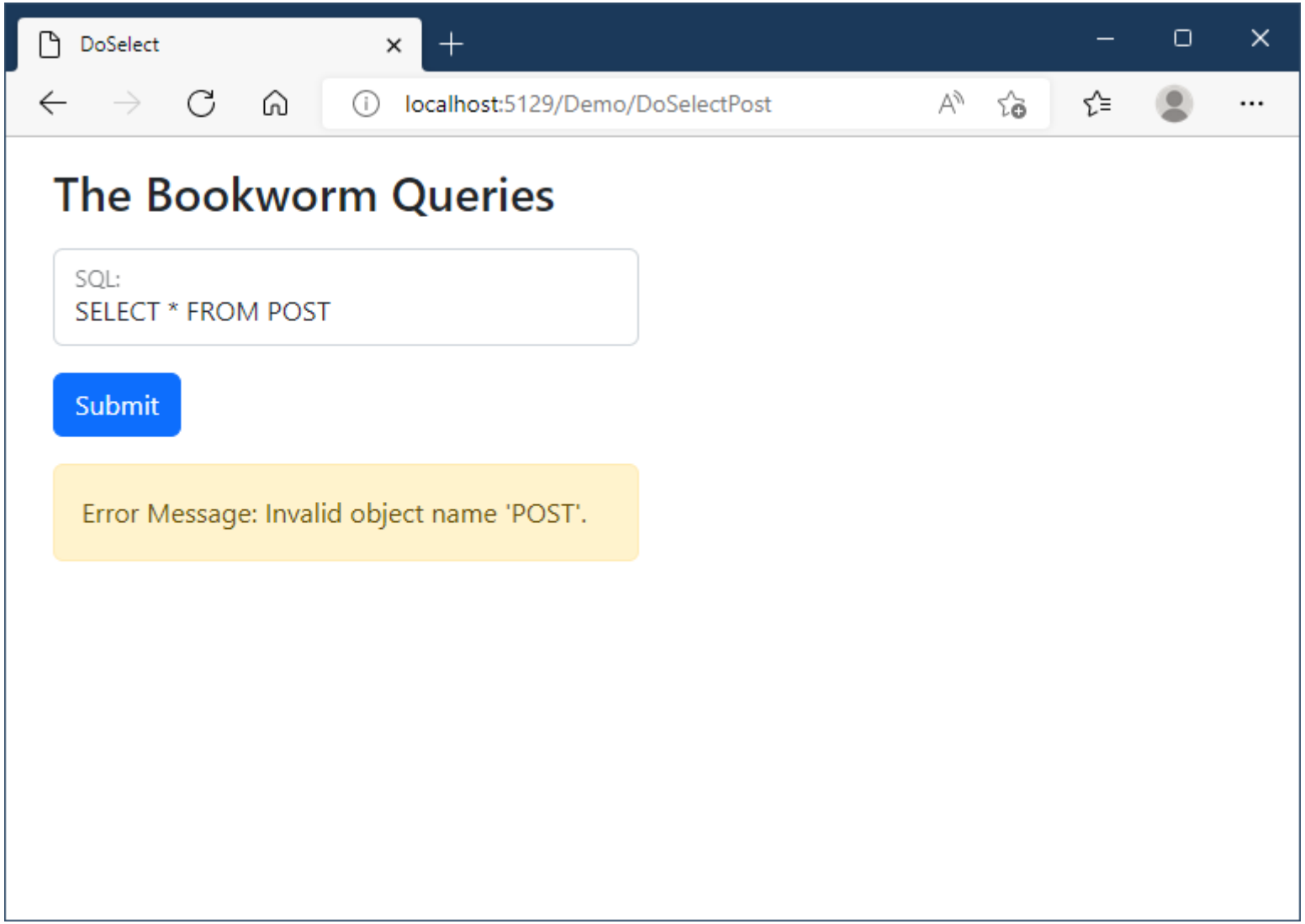
View

```
<input type="submit" class="mb-3 btn btn-primary" value="Submit" />

@if (Model != null && Model.Rows.Count != 0)
{
    <table class='table table-striped table-hover'>
        <tr>
            @for (int col = 0; col < Model.Columns.Count; col++)
            {
                <th>@Model.Columns[col]</th>
            }
        </tr>

        @for (int row = 0; row < Model.Rows.Count; row++)
        {
            <tr>
                @for (int col = 0; col < Model.Columns.Count; col++)
                {
                    <td>@Model.Rows[row][col]</td>
                }
            </tr>
        }
    </table>
}
```

View

```
    @{
        var message = ViewData["Message"] as string;
        if (!string.IsNullOrEmpty(message))
        {
            <div class="mb-3">
                <div class="col-6 alert alert-warning">
                    Error Message: @message
                </div>
            </div>
        }
    }
    </form>
    </div>
</body>
</html>
```

View cont'd


DoSelect
localhost:5129/Demo/DoSelectPost
The Bookworm Queries
SQL:
SELECT * FROM POST
Submit
Error Message: Invalid object name 'POST'.

Passing a `DataTable` to a view

There are two ways to pass table data from a controller to a view

- Using **`ViewData`** (GetTable1)
- Using **`Model`** (GetTable2) **<<< preferred method**

```
public IActionResult GetTable1()
{
    ViewData["dbtable"] = DBUtl.GetTable("SELECT * FROM BwPublisher");
    return View();
}
```

```
public IActionResult GetTable2()
{
    DataTable dt = DBUtl.GetTable("SELECT * FROM BwPublisher");
    return View(dt);
}
```

Using `ViewData` (View: GetTable1)

```
@using System.Data
@{
    DataTable dt = (ViewData["dbtable"] as DataTable)!;
}
<html>
<head>
    <title>GetTable1</title>
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
</head>

<body>
    <div class="container m-3">
        <h3 class=mb-3>Get Table (ViewData)</h3>
        <table class='table table-bordered table-hover table-striped'>
            <tr>
                @for (int col = 0; col < dt.Columns.Count; col++)
                {
                    <th>@dt.Columns[col]</th>
                }
            </tr>

            @for (int row = 0; row < dt.Rows.Count; row++)
            {
                <tr>
                    @for (int col = 0; col < dt.Columns.Count; col++)
                    {
                        <td>@dt.Rows[row][col]</td>
                    }
                </tr>
            }
        </table>
    </div>
```

Using a **model** - preferred (View: GetTable1)

```
@using System.Data
@model DataTable

<html>
<head>
    <title>GetTable2</title>
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
</head>

<body>
    <div class="container m-3">
        <h3 class=mb-3>Get Table (Model)</h3>
        <table class='table table-bordered table-hover table-striped'>
            <tr>
                @for (int col = 0; col < Model.Columns.Count; col++)
                {
                    <th>@Model.Columns[col]</th>
                }
            </tr>

            @for (int row = 0; row < Model.Rows.Count; row++)
            {
                <tr>
                    @for (int col = 0; col < Model.Columns.Count; col++)
                    {
                        <td>@Model.Rows[row][col]</td>
                    }
                </tr>
            }
        </table>
    </div>
```

§

Solving the problem

Thought process §

For each of Rachel's queries,

- Determine the **SELECT** statement that is required

For each **SELECT** statement

- Use `DBUtl.GetTable` to query the Database
- Use the action `DoSelect()` in `DemoController` and the `DoSelect view` to display the results using a HTML table

SELECT statements §

```
const string SELECT1 = // QUESTION ONE
   @"SELECT Isbn, Title, Lang
       FROM BwBook
      WHERE Lang != 'English'";

const string SELECT2 = // QUESTION TWO
   @"SELECT Isbn, Title, Qty
       FROM BwBook
      WHERE Qty = 0";

const string SELECT3 = // QUESTION THREE
   @"SELECT P.PubName AS Publisher, COUNT(*) AS Titles
       FROM BwPublisher P
      INNER JOIN BwBook B
         ON B.PubID = P.PubID
      GROUP BY P.PubName";

const string SELECT4 = // QUESTION FOUR
   @"SELECT Title
       FROM BwBook
      GROUP BY title
     HAVING COUNT(Lang) > 1";
```

§

Rachel's queries

Answers to Rachel's queries

Question 1 §

What are the non-English titles?

The Bookworm Queries

How many non-English titles are there?

ISBN	Title	Lang
978-0-07-124739-3	Management Control of Multinational Enterprises in China	Mandarin
978-0-59651-816-5	Security Monitoring - Proven Methods for Incident Detection on Enterprise Networks	Mandarin
978-0-631-16754-2	Children and Emotion: The Development of Psychological Understanding	Spanish
978-1-4302-1954-6	Beginning Java EE 6 Platform with GlassFish 3: From Novice to Professional	Mandarin

Question 2 §

Which titles are out of stock?

The Bookworm Queries

Which titles are out of stock?

ISBN	Title	Qty
978-0-470-05367-6	High Performance Switches and Routers	0
978-1-4051-7058-1	Explaining the Breakdown of Ethnic Relations	0

Question 3 §

How many titles are there for each publisher?

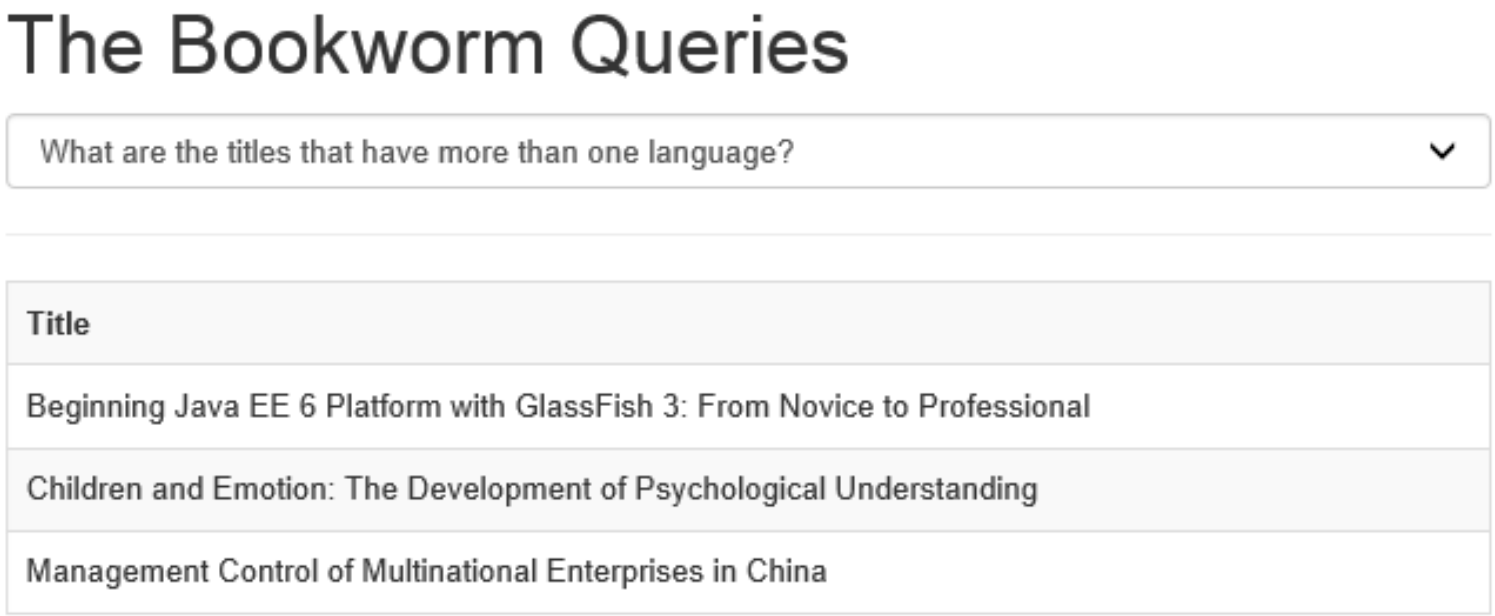
The Bookworm Queries

How many titles are there for each publisher?

Publisher	Titles
Apress	6
McGraw-Hill	5
Microsoft Press	2
O'Reilly	3
Wiley	12

Question 4 §

What are the titles that have more than one language?



Query.cshtml §

```
@using System.Data
@model DataTable
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
   <title>Query</title>
</head>
<body>
   <div class="container m-3">
      <form method="post" action="~/BookWorm/Submit">
          <div class="mb-3">
              <h3>Bookworm Queries</h3>
          </div>
      <div class="mb-3">
          <div class="col-8">
              <select name="Question" id="question" class="form-control"
                    onchange="this.form.submit();">
                  <option selected="selected" value="">-- Select a Question --</option>
                  <option value="1">What are the non-English titles?</option>
                  <option value="2">Which titles are out of stock?</option>
                  <option value="3">How many titles are there for each publisher?</option>
                  <option value="4">What are the titles that have more than one language?</option>
               </select>
            </div>
         </div>
```

Query.cshtml (cont'd)

```
@if (Model != null)
{
    <table class='table table-bordered table-hover table-striped'>
        <tr>
            @for (int col = 0; col < Model.Columns.Count; col++)
            {
                <th>@Model.Columns[col]</th>
            }
        </tr>

        @for (int row = 0; row < Model.Rows.Count; row++)
        {
            <tr>
                @for (int col = 0; col < Model.Columns.Count; col++)
                {
                    <td>@Model.Rows[row][col]</td>
                }
            </tr>
        }
    </table>
}
```

Query.cshtml (cont'd)

```
            @{
                var message = ViewData["Message"] as string;
                var sql = ViewData["ExecSQL"] as string;
                if (!string.IsNullOrEmpty(message))
                {
                    <div class="mb-3">
                        <div class="col-8 alert alert-warning">
                            <b>Error Message:</b> @message<br />
                            <b>SQL Executed: </b> @sql<br />
                        </div>
                    </div>
                }
            }

        </form>
    </div>
</body>
</html>
```

BookWormController.cs §

```
public IActionResult Submit(IFormCollection form)
{
    string question = form["Question"].ToString();

    string sql = "";
    if (question.Equals("1"))
    {
        sql = SELECT1;
    }
    else if (question.Equals("2"))
    {
        sql = SELECT2;
    }
    else if (question.Equals("3"))
    {
        sql = SELECT3;
    }
    else if (question.Equals("4"))
    {
        sql = SELECT4;
    }

    DataTable dt = DBUtl.GetTable(sql);
    if (dt.Rows.Count == 0)
    {
        ViewData["Message"] = DBUtl.DB_Message;
        ViewData["ExecSQL"] = DBUtl.DB_SQL;
    }
    return View("Query", dt);
}
```

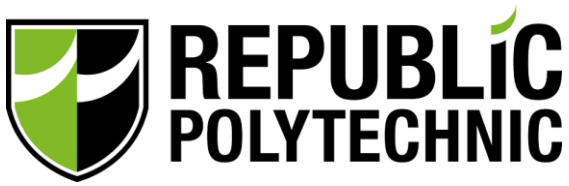
DataTables – one more thing §

```
string msg = "";
string id, name, addr;
DataTable dt = DBUtl.GetTable("SELECT * FROM BwPublisher");
if (dt.Rows.Count == 0)
{
   msg = "No records found";
}
else
{
   id   = dt.Rows[0]["PubId"].ToString();
   name = dt.Rows[0]["PubName"].ToString();
   addr = dt.Rows[0]["PubAddr"].ToString();

   // OR
   id   = dt.Rows[0][0].ToString();
   name = dt.Rows[0][1].ToString();
   addr = dt.Rows[0][2].ToString();
}
```

Summary

What you have learned

- Use **HTML** table related elements to present data
- Create a database in Visual Studio
- Setup a database connection string in **appsettings.json**
- Use the **DBUtl.GetTable** helper method
- Identify various components of a **DataTable**
- Display information from a database in a web page



Lesson 5

Modifying Data Part 1

Bootstrap

`navbar` and `alert`

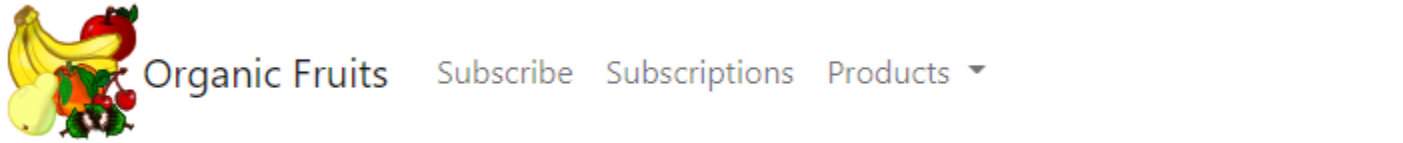
Navigation bar

What is a Navigation Bar?

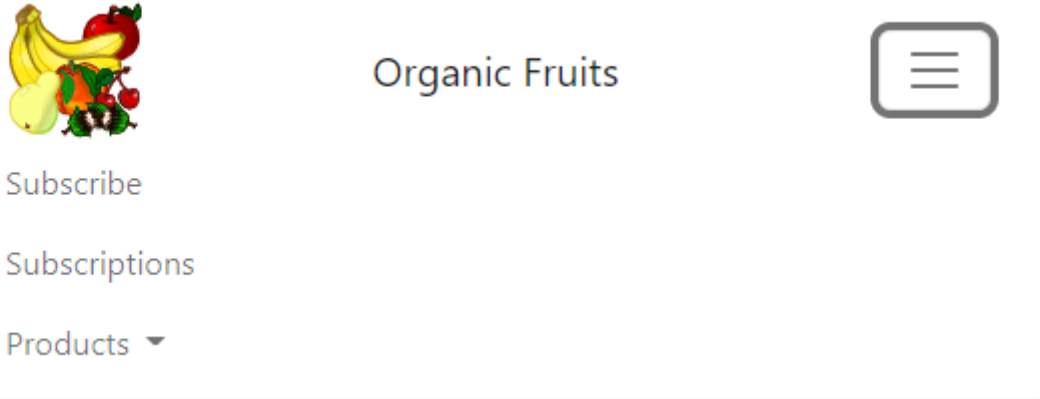
- Part of the Web Application that allows its users to easily access information.
- Basically, a list of links to different pages in the Web Application.
- Easily implemented using `<nav>`, `<ul>` and `<li>` elements

Kinds of Navigation Bar

Medium Viewport



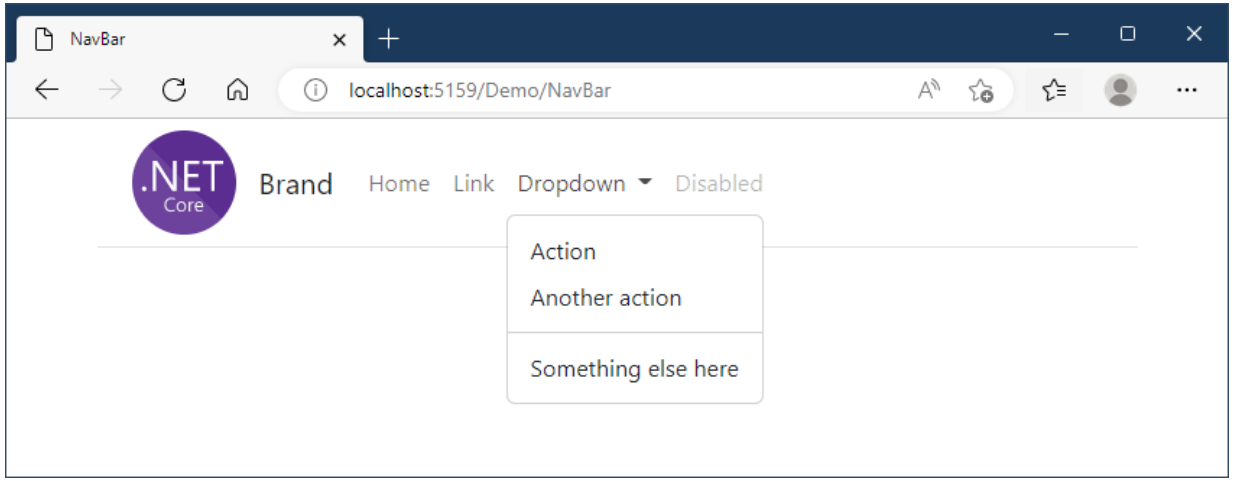
Small Viewport



Demonstration

Bootstrap `navbar` and `alert`

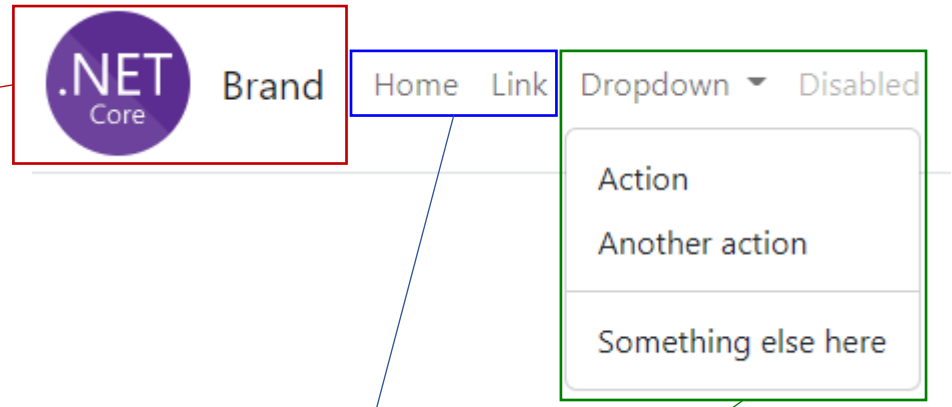
Bootstrap **navbar**




```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <title>NavBar</title>
</head>
```

Bootstrap **navbar** (cont.)

```
<nav class="navbar navbar-expand-md">
<div class="container-fluid">
    <img src="~/images/netcore.svg" height="72" class="me-3" />
    <a class="navbar-brand" href="#">Brand</a>
    @*Button below (hamburger menu) appears when viewport too small.*@
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
            data-bs-target="#navbarSupportedContent">
            <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav">
            <li class="nav-item"><a class="nav-link" href="#">Home</a></li>
            <li class="nav-item"><a class="nav-link" href="#">Link</a></li>
            <li class="nav-item dropdown">
                <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
                   data-bs-toggle="dropdown">Dropdown</a>
                <ul class="dropdown-menu">
                    <li><a class="dropdown-item" href="#">Action</a></li>
                    <li><a class="dropdown-item" href="#">Another action</a></li>
                    <li><hr class="dropdown-divider"></li>
                    <li><a class="dropdown-item" href="#">Something else here</a></li>
                </ul>
            </li>
            <li class="nav-item"><a class="nav-link disabled" href="#" tabindex="-1">Disabled</a></li>
        </ul>
    </div>
</div>
</nav>
```



Alerts

Purpose

- Alerts provide contextual feedback messages for typical user actions, with a handful of available and flexible alert messages.

EXAMPLE

Well done! You successfully read this important alert message.

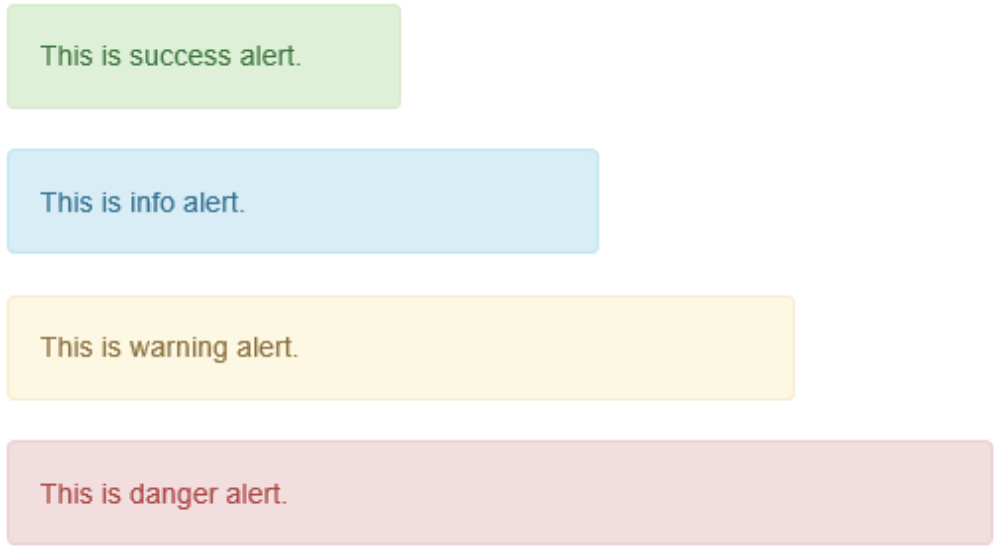
Heads up! This alert needs your attention, but it's not super important.

Warning! Better check yourself, you're not looking too good.

Oh snap! Change a few things up and try submitting again.

Using an `alert`

```
<div class="row">
    <div class="alert alert-success col-3">
    This is success alert.</div>
</div>
<div class="row">
    <div class="alert alert-info col-4">
    This is info alert.</div>
</div>
<div class="row">
    <div class="alert alert-warning col-5">
    This is warning alert.
    </div>
</div>
<div class="row">
    <div class="alert alert-danger col-6">
    This is danger alert.
    </div>
</div>
```



Views

Introducing Partial Views and
`@await Html.PartialAsync()` method

Partial Views

Partial views are special type of views that are rendered inside other views.

Common parts, such as NavBars can be reused using Partial Views.

Partial views are rendered in a view using the method

```
@await Html.PartialAsync()
```

Organic Fruits has a partial view called **`OrgNavBar.cshtml`**,
which is used by **all other views** in Organic Fruits.

OrgNavBar.cshtml

```
<div class="container border-bottom">
    <nav class="navbar navbar-expand-md">
        <div class="container-fluid">
            <img src="~/images/organic.svg" height="72">
            <a class="navbar-brand" href="~/Organic">Organic Fruits</a>
            @*Button below (hamburger menu) appears when viewport too small.*@
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
                    data-bs-target="#navbarSupportedContent">
                <span class="navbar-toggler-icon"></span>
            </button>

            <div class="collapse navbar-collapse" id="navbarSupportedContent">
                <ul class="navbar-nav">
                    <li class="nav-item"><a href="~/Organic/Subscription" class="nav-link">Subscribe</a></li>
                    <li class="nav-item"><a href="~/Organic/SubscriptionList" class="nav-link">Subscriptions</a>

                    <li class="nav-item dropdown">
                        <a class="nav-link dropdown-toggle" role="button"
                           href="#" id="navbarDropdown" data-bs-toggle="dropdown">
                            Products
                        </a>
                        <ul class="dropdown-menu">
                            <li><a class="dropdown-item" href="~/Organic/Products">View All</a></li>
                            <li><div class="dropdown-divider"></div></li>
                            <li><a class="dropdown-item" href="#">Add Product (Lesson 06)</a></li>
                            <li><a class="dropdown-item" href="#">Remove Product (Lesson 06)</a></li>
                        </ul>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</div>
```

Organic.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <title>Organic</title>
</head>

<body>
    @await Html.PartialAsync("OrgNavBar")
    <div class="container mt-3">
        <div class="bg-light rounded-3">
            <img src="/images/company.jpg" class="rounded-3 m-3" />
            <p class="m-3">
                Founded in 2001, <b>Organic Fruits</b> is now the world's leading grower,
                distributor and retailer of organic produce, offering more than
                100 varieties of home-grown and imported organic fruit and vegetables.
                Since its establishment, <b>Organic Fruits</b> has been a trusted source of
                delicious and healthy food grown without chemical fertilizers or pesticides.
                <b>Organic Fruits</b> is popular because of its down-to-earth reputation
                and the quality of its tasty, fresh organic fruit & vegetables.
                These are available in many supermarkets near you.
            </p>
             
        </div>
    </div>
</body>
</html>
```

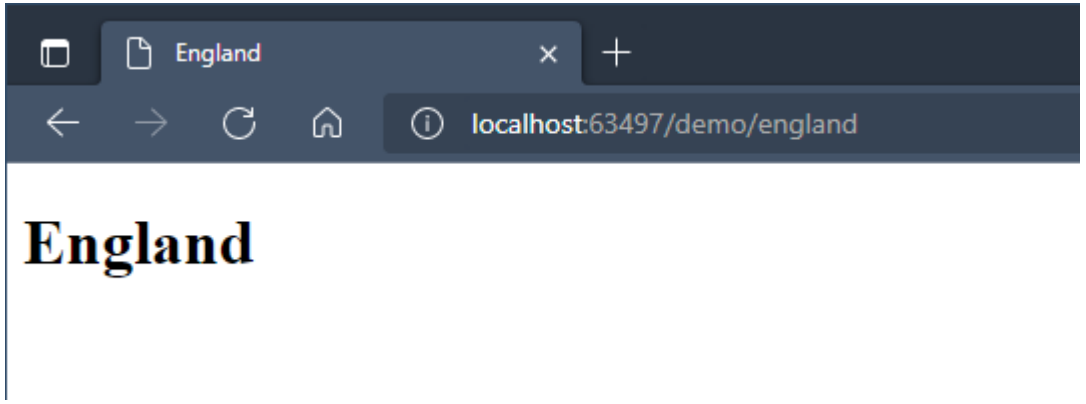
Using the Partial View defined in Views/Organic/OrgNavBar.cshtml

Passing Data in MVC

`RedirectToAction()` and `TempData`

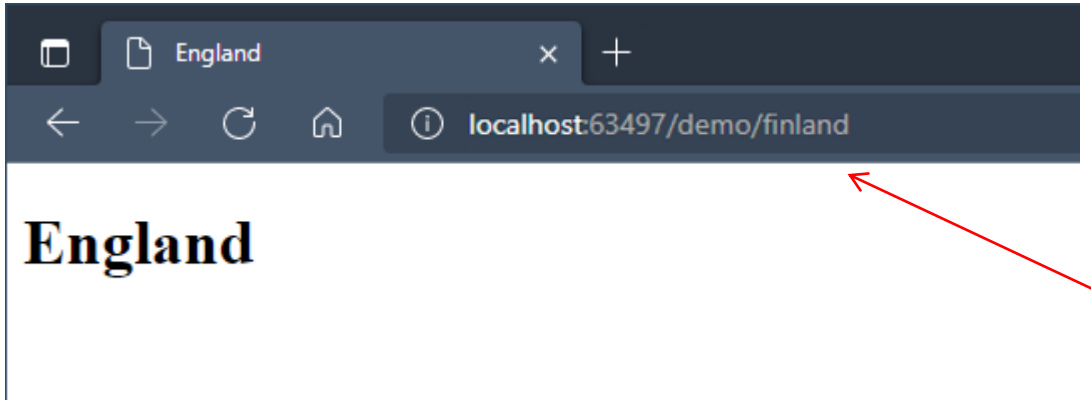
Demo/England

```
public IActionResult England()
     {
        return View();
     }
```



Demo/Finland

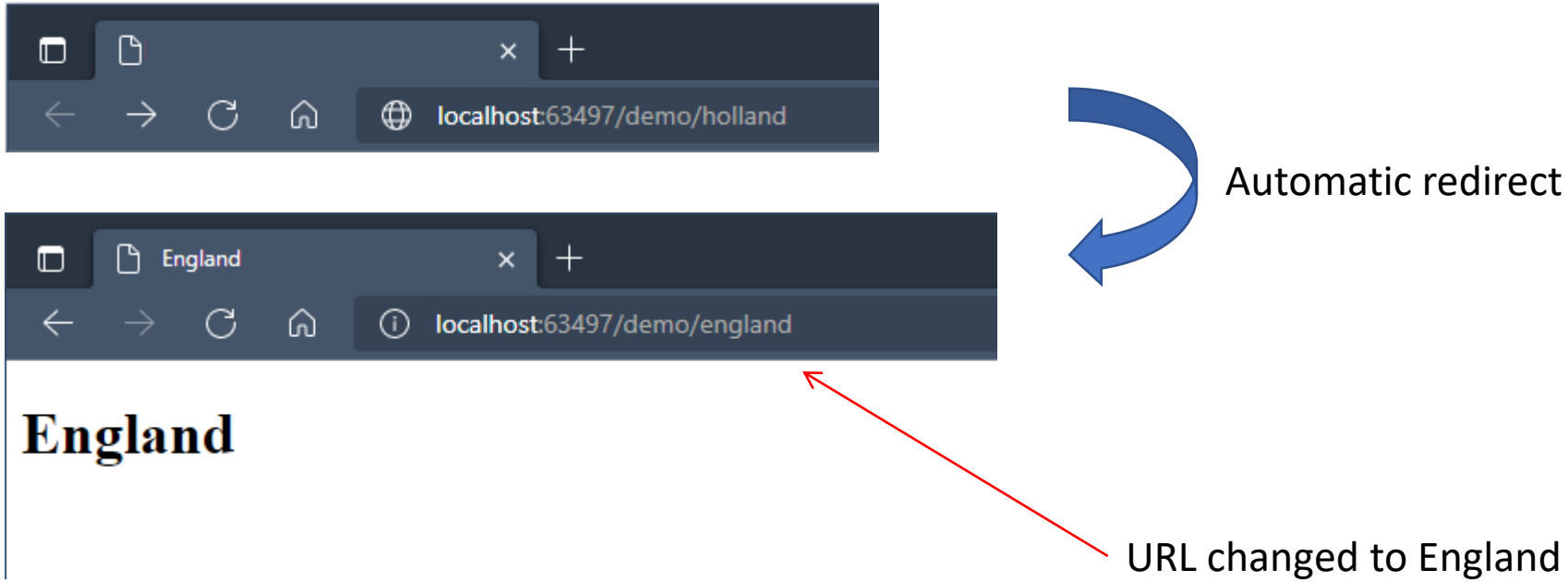
```
public IActionResult Finland()
     {
        return View("England");
     }
```



URL unchanged, although the View displayed is for England

Demo/Holland

```
public IActionResult Holland()
     {
        return RedirectToAction("England");
     }
```



RedirectToAction

```
return RedirectToAction("actionname");
```

- Returns an HTTP 302 response to the browser.
- Causes the browser to make a GET request to the specified action.
- The Browser's URL will change because of the redirection.

```
return View("viewname")
```

- Does not cause the browser to redirect.
- The browser's URL remains **unchanged**.

TempData

`TempData` stores data that persists only from **one request to the next.**

- It is used to pass data from **one action** to **another action** when `RedirectToAction()` method is called
- The usage is similar to `ViewData`, except `TempData` can only store primitive types.
- Demonstration

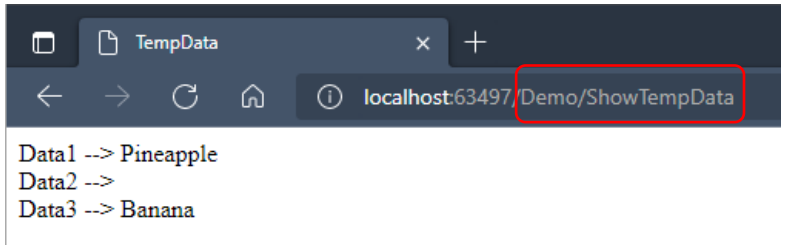
```
<body>
    <div class="alert alert-info">
        Data1 --> @TempData["Data1"]
    </div>

    <div class="alert alert-info">
        Data2 --> @ViewData["Data2"]
    </div>

    <div class="alert alert-info">
        Data3 --> @ViewData["Data3"]
    </div>
</body>
```

TempData – Same result

1.

```
public class DemoController : Controller
  {
     public IActionResult UseTempData()
     {
        TempData["Data1"] = "Pineapple";
        ViewData["Data2"] = "Orange";
        return RedirectToAction("ShowTempData", "Demo");
     }
```

Controller specified

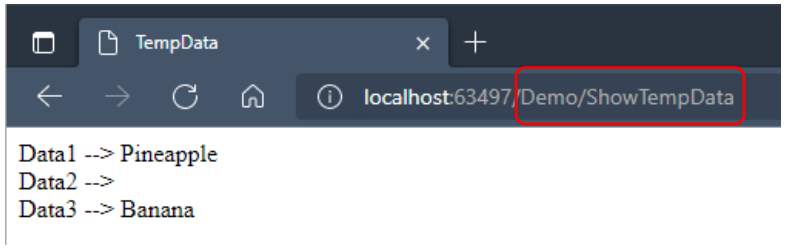


2.

```
public class DemoController : Controller
  {
     public IActionResult UseTempData()
     {
        TempData["Data1"] = "Pineapple";
        ViewData["Data2"] = "Orange";
        return RedirectToAction("ShowTempData");
     }
```

Controller not specified

```
public IActionResult ShowTempData()
     {
        ViewData["Data3"] = "Banana";
        return View();
     }
```



SQL recap

From C207 Database Systems

INSERT statement

```
INSERT INTO Contact
  (contact_id, last_name, first_name,
   email, phone, school_id)
VALUES
  (1, 'Goh', 'Ivan',
  'igoh@bendemeersec.edu.sg', '64527676', 4);
```

- The columns are listed in the brackets after the table name.
- The values to be inserted are given in brackets after the keyword **VALUES**.
- The Values given must correspond to the order in which the columns are listed.
- What is the name of the Table? `contact`
- How many columns are there in the Table? `6`
- Which columns are of type `int`? `contact_id, school_id`

DELETE statement

```
DELETE
FROM Contact
WHERE contact_id = 7;
```

```
DELETE
FROM Contact;
```

- The table from which the records are to be deleted is specified after **FROM**
- The **WHERE** condition determines which records are to be deleted.
- If the primary key is used as a condition in the **WHERE** clause, at most ONE record will be deleted. <<< Normal case.
- If the **WHERE** clause is not specified, then all records will be deleted. <<< Generally, this is not the intended action.

Updating a database

Microsoft's APIs for data access

`DBUtl` class

Helper Class for Database Access

`string DB_CONNECTION;`

- Determines which database to connect

`string DB_Message`

- Stores any error message

`GetTable(string sql)` method <<< Last week

- Method to get records from database using SQL

`ExecSQL(string sql)` method <<< This week

- Method to **`insert`**, **`update`**, **`delete`** records from database using SQL

ExecSQL method (*simplified*)

```
public static int ExecSQL(string sql)
   {
      int rowsAffected = 0;
      using (SqlConnection dbConn = new SqlConnection(DB_CONNECTION))
      using (SqlCommand dbCmd = dbConn.CreateCommand())
      {
         dbConn.Open();
         dbCmd.CommandText = sql;
         rowsAffected = dbCmd.ExecuteNonQuery();
      }
      return rowsAffected;
   }
```

SqlConnection

- Same as in **GetTable()**

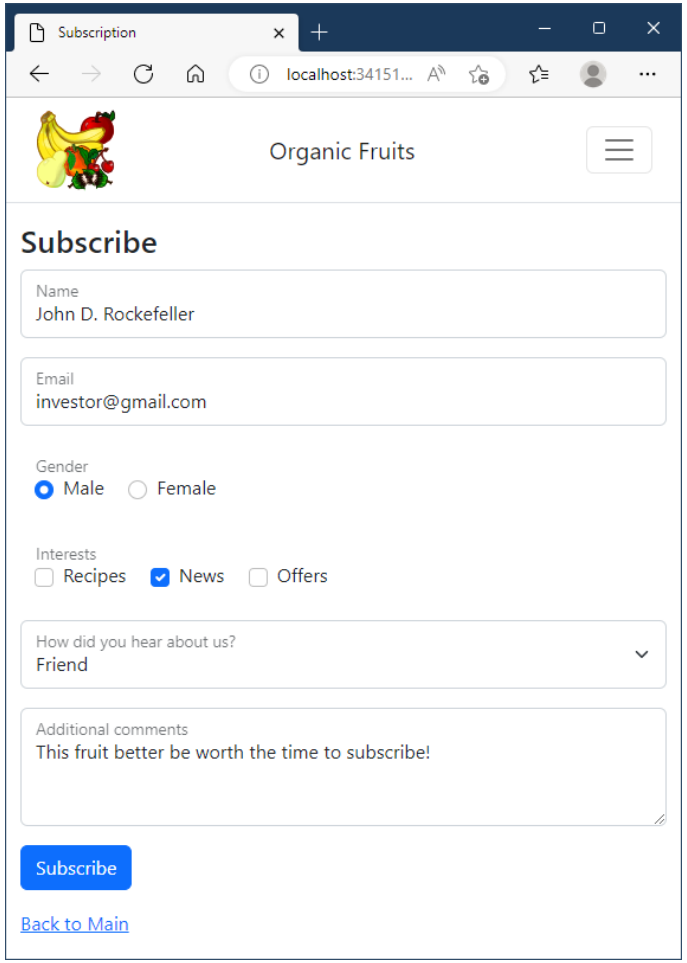
SqlCommand

- Represents a SQL statement to execute against a SQL Server database.

SqlCommand Class (System.Data.SqlClient) | Microsoft Docs

ExecSQL for INSERT

Organic Fruits' Newsletter Subscription




```
CREATE TABLE OrgSubscription (
   Sno          INT PRIMARY KEY IDENTITY,
   UserName     VARCHAR(50) NOT NULL,
   SubEmail     VARCHAR(50) NOT NULL,
   Gender       VARCHAR(1)  NOT NULL,
   Recipes      BIT,
   News         BIT,
   Offers       BIT,
   Referral     VARCHAR(20),
   Comments     VARCHAR(1000)
```

Organic/Confirmation

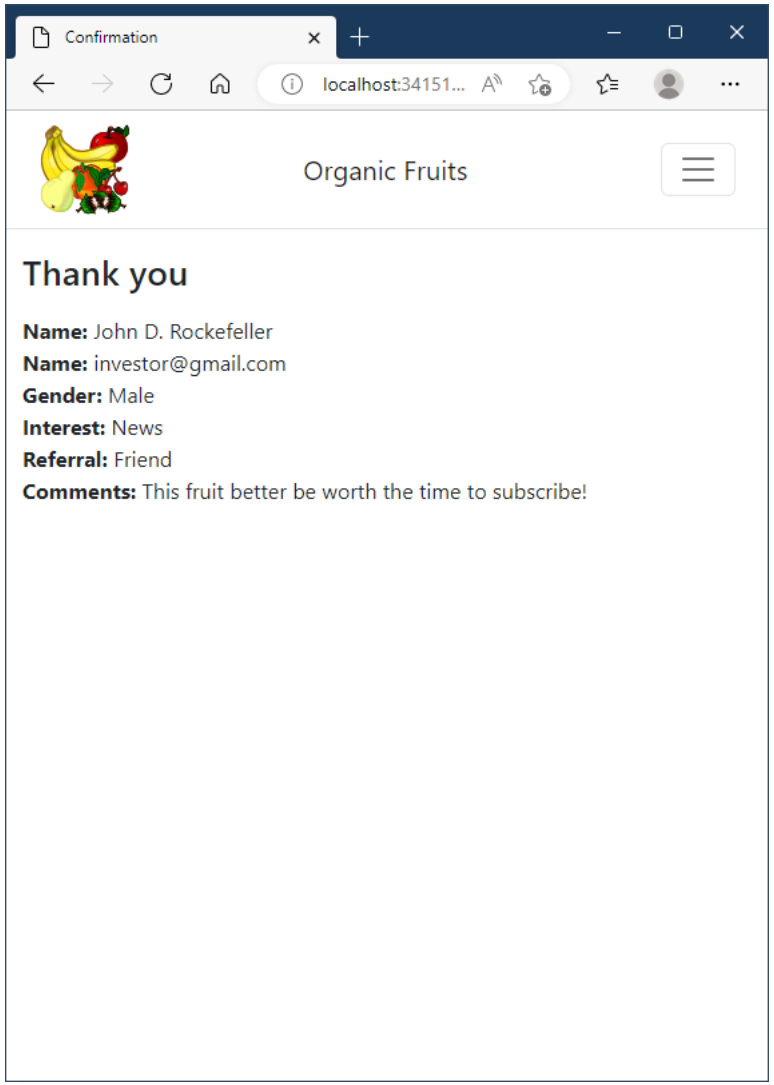
```
// Add Record to Database
string sql = @"INSERT INTO
               OrgSubscription(UserName, SubEmail, Gender,
               Recipes, News, Offers, Referral, Comments)
               VALUES('{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}')";

string insert = String.Format(sql, name, email,
                              gender.Substring(0, 1),  // F or M
                              recipe.Equals("Recipe"), // True or False
                              news.Equals("News"),
                              offers.Equals("Offers"),
                              refer, comments);

int rowsAffected = DBUtl.ExecSQL(insert);
```

```
CREATE TABLE OrgSubscription (
   Sno          INT PRIMARY KEY IDENTITY,
   UserName     VARCHAR(50) NOT NULL,
   SubEmail     VARCHAR(50) NOT NULL,
   Gender       VARCHAR(1)  NOT NULL,
   Recipes      BIT,
   News         BIT,
   Offers       BIT,
   Referral     VARCHAR(20),
   Comments     VARCHAR(1000)
```

<< Why is Sno is not included in the SQL?

Organic/Confirmation (cont.)

```
int rowsAffected = DBUtl.ExecSQL(insert);

// Check Insert is Successful
if (rowsAffected == 1)
{
    Subscription s = new()
        {
            UserName = name,
            SubEmail = email,
            Gender   = gender,
            Interest = interest,
            Referral = refer,
            Comments = comments
        };
    return View("Confirmation", s);
}
else
{
    ViewData["Message"] = DBUtl.DB_Message;
    return View("Subscription");
}
```

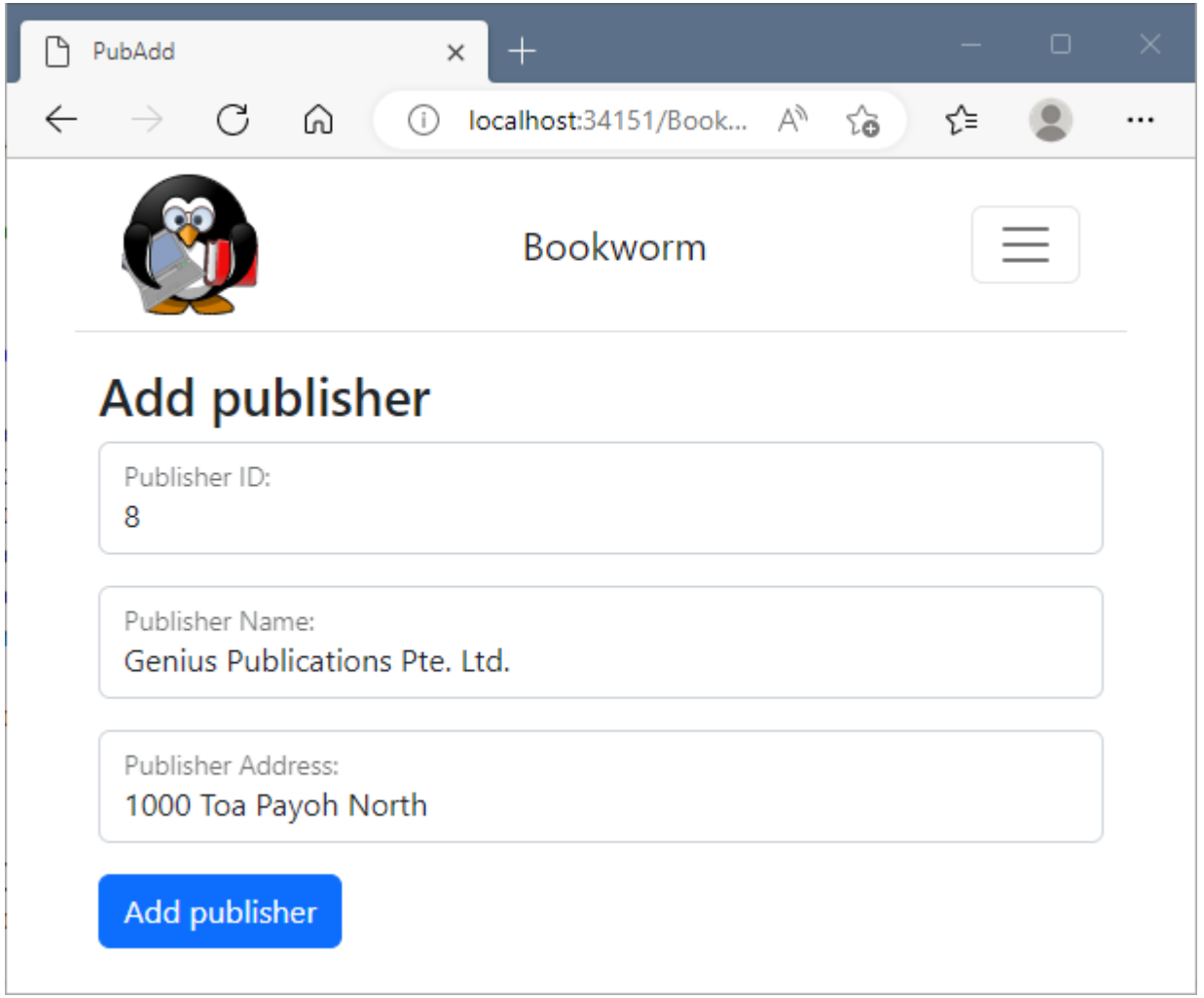


Adding a publisher

BookWormController.cs

- PubAdd Action
- PubAddPost Action

PubAdd.cshtml



PubAdd.cshtml

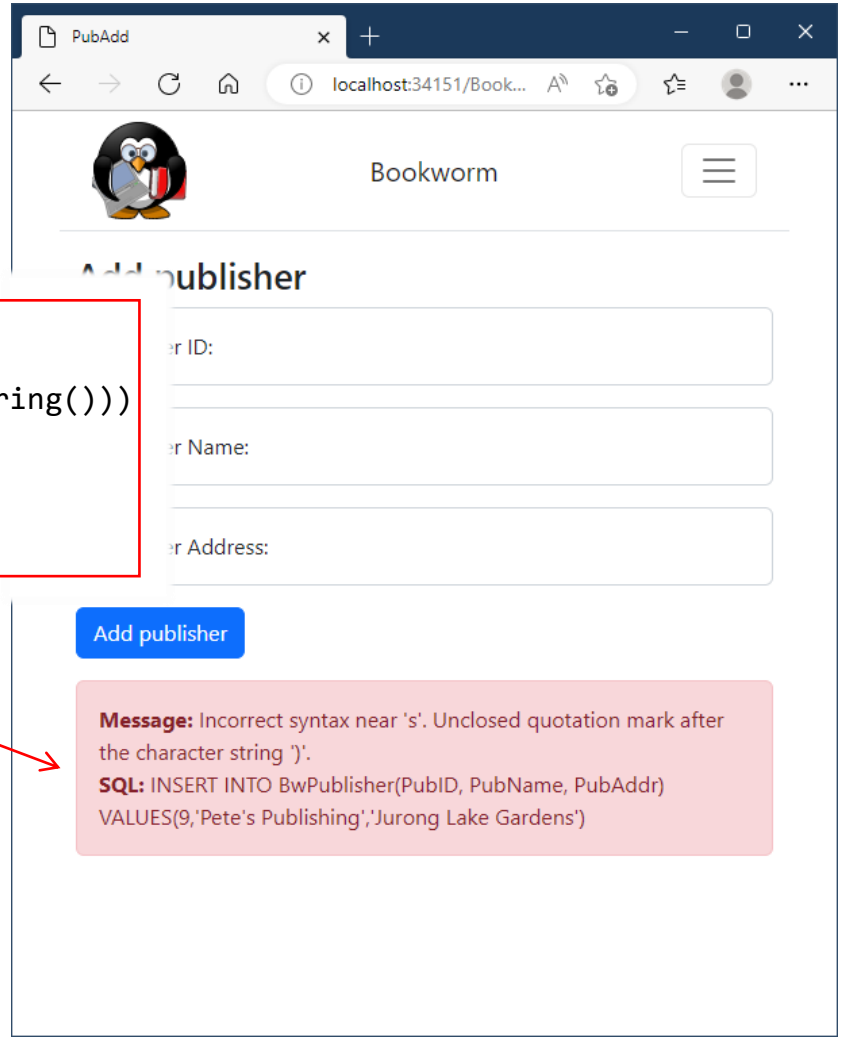
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <title>PubAdd</title>
</head>
<body>
    @await Html.PartialAsync("BwNavBar")
    <div class="container">

        <div class="mt-3">
            <h2>Add publisher</h2>
        </div>

        <form method="post" action="~/BookWorm/PubAddPost">

            <div class="col-md-5 mb-3 form-floating">
                <input type="text" class="form-control" id="pubid" name="PubId"
                       placeholder="pubid" />
                <label for="isbn">Publisher ID:</label>
            </div>
```

PubAdd.cshtml (cont.)

```
 <div class="col-md-5 mb-3 form-floating">
    <input type="text" class="form-control" id="pubname" name="PubName"
           placeholder="pubname" />
    <label for="isbn">Publisher Name:</label>
</div>

<div class="col-md-5 mb-3 form-floating">
    <input type="text" class="form-control" id="pubaddr" name="PubAddr"
           placeholder="pubaddr" />
    <label for="isbn">Publisher Address:</label>
</div>

<div class="form-group mb-3">
    <div class="col-5">
        <input type="submit" value="Add publisher" class="btn btn-primary" />
    </div>
</div>
```

PubAdd.cshtml (cont.)

```
        @if (ViewData["Message"] != null)
        {
            <div class="form-group row">
                <div class="col-md-6">
                    <div class="alert alert-@ViewData["MsgType"]">
                        <b>Message: </b>@ViewData["Message"]<br />
                        @if (!String.IsNullOrEmpty(ViewData["ExecSQL"]?.ToString()))
                        {
                            <b>SQL: </b>@ViewData["ExecSQL"]?.ToString()
                        }
                    </div>
                </div>
            </div>
            }
        </form>
    </div>
</body>
</html>
```



Bookworm/PubAddPost

```
public IActionResult PubAddPost()
    {
        IFormCollection form = HttpContext.Request.Form;
        string pubid = form["Pubid"].ToString().Trim();
        string pubname = form["Pubname"].ToString().Trim();
        string pubaddr = form["Pubaddr"].ToString().Trim();

        if (ValidUtl.CheckIfEmpty(pubid, pubname, pubaddr))
        {
            ViewData["Message"] = "Please enter all fields.";
            ViewData["MsgType"] = "warning";
            return View("PubAdd");
        }

        if (!pubid.IsInteger())
        {
            ViewData["Message"] = "Pub ID must be an integer.";
            ViewData["MsgType"] = "warning";
            return View("PubAdd");
        }

        string sql =
           @"INSERT INTO BwPublisher(PubID, PubName, PubAddr)
              VALUES({0},'{1}','{2}')";
```

Bookworm/PubAddPost (cont.)

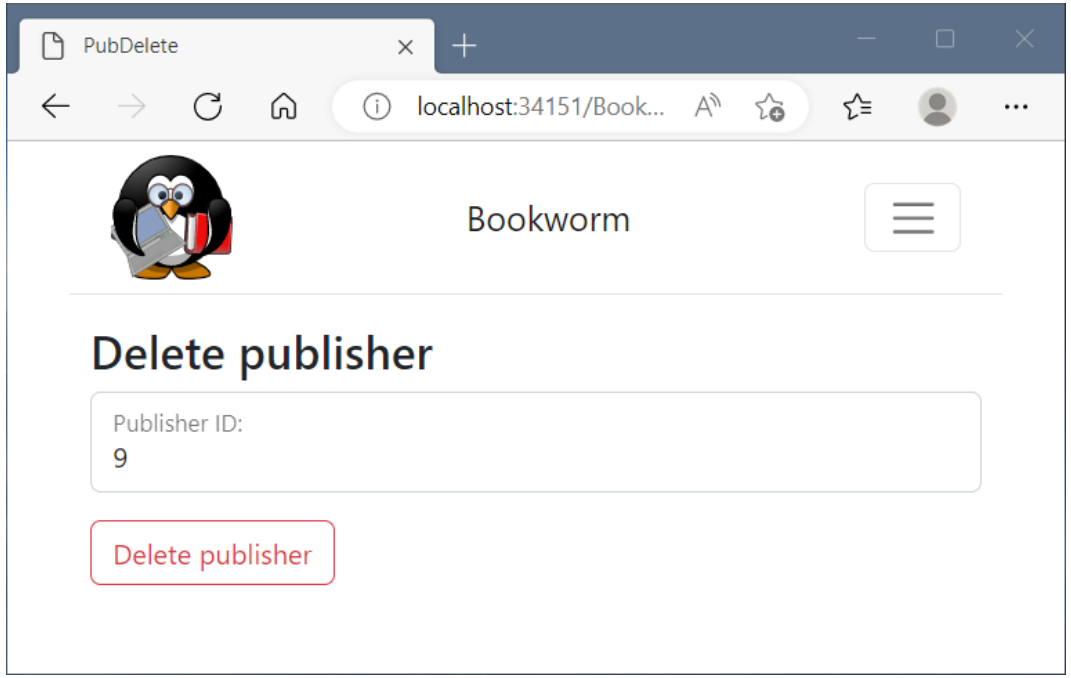
```
        string insert = String.Format(sql, pubid, pubname, pubaddr);

        int count = DBUtl.ExecSQL(insert);
        if (count == 1)
        {
            TempData["Message"] = "Publisher Successfully Added.";
            TempData["MsgType"] = "success";
            return RedirectToAction("Publishers");
        }
        else
        {
            ViewData["Message"] = DBUtl.DB_Message;
            ViewData["ExecSQL"] = DBUtl.DB_SQL;
            ViewData["MsgType"] = "danger";
            return View("PubAdd");
        }
    }
```

Deleting a Publisher

BookWormController.cs

- `PubDelete` Action
- `PubDeletePost` Action

PubDelete.cshtml



Bookworm/PubDeletePost

```
public IActionResult PubDelete()
{
    return View("PubDelete");
}

public IActionResult PubDeletePost()
{
    IFormCollection form = HttpContext.Request.Form;
    string pubid = form["Pubid"].ToString().Trim();

    if (!pubid.IsInteger())
    {
        ViewData["Message"] = "Pub ID must be an integer.";
        ViewData["MsgType"] = "warning";
        return View("PubDelete");
    }

    string sql = @"SELECT * FROM BwPublisher WHERE PubID={0}";
    string select = String.Format(sql, pubid);
    DataTable dt = DBUtl.GetTable(select);
    if (dt.Rows.Count == 0)
    {
        ViewData["Message"] = "Pub ID Not Found.";
        ViewData["MsgType"] = "warning";
        return View("PubDelete");
    }
```

Bookworm/PubDeletePost （cont.）

```
    sql = @"DELETE BwPublisher WHERE PubID='{0}'";
    string delete = String.Format(sql, pubid);
    int count = DBUtl.ExecSQL(delete);
    if (count == 1)
    {
        ViewData["Message"] = "Publisher Deleted.";
        ViewData["MsgType"] = "success";
    }
    else
    {
        ViewData["Message"] = DBUtl.DB_Message;
        ViewData["ExecSQL"] = DBUtl.DB_SQL;
        ViewData["MsgType"] = "danger";
    }

    return View("PubDelete");
}
```

Apprentice Activity

Experiment with Publisher Add/Delete

§

Solving The Problem

Required Data and the Web App UI

Thought Process §

Building the Forms

- Book
  - Insert Record
    - Data Needed : isbn, title, language, price, qty, type, pub id
    - Controls Needed : Labels, Text Fields, Buttons
  - Delete Record
    - Data Needed : isbn (Primary Key)
    - Controls Needed : Labels, Text Fields, Buttons

Thought Process §

Construct the SQL statements - **BwBook**

- Insert

```
string sql =
   @"INSERT INTO BwBook(Isbn, Title, Lang, Price, Qty, BkType, PubID)
     VALUES('{0}','{1}','{2}',{3},{4},'{5}',{6})";

string insert = String.Format(sql, isbn, title, lang,
                              price, qty, bkType, pubid);
```

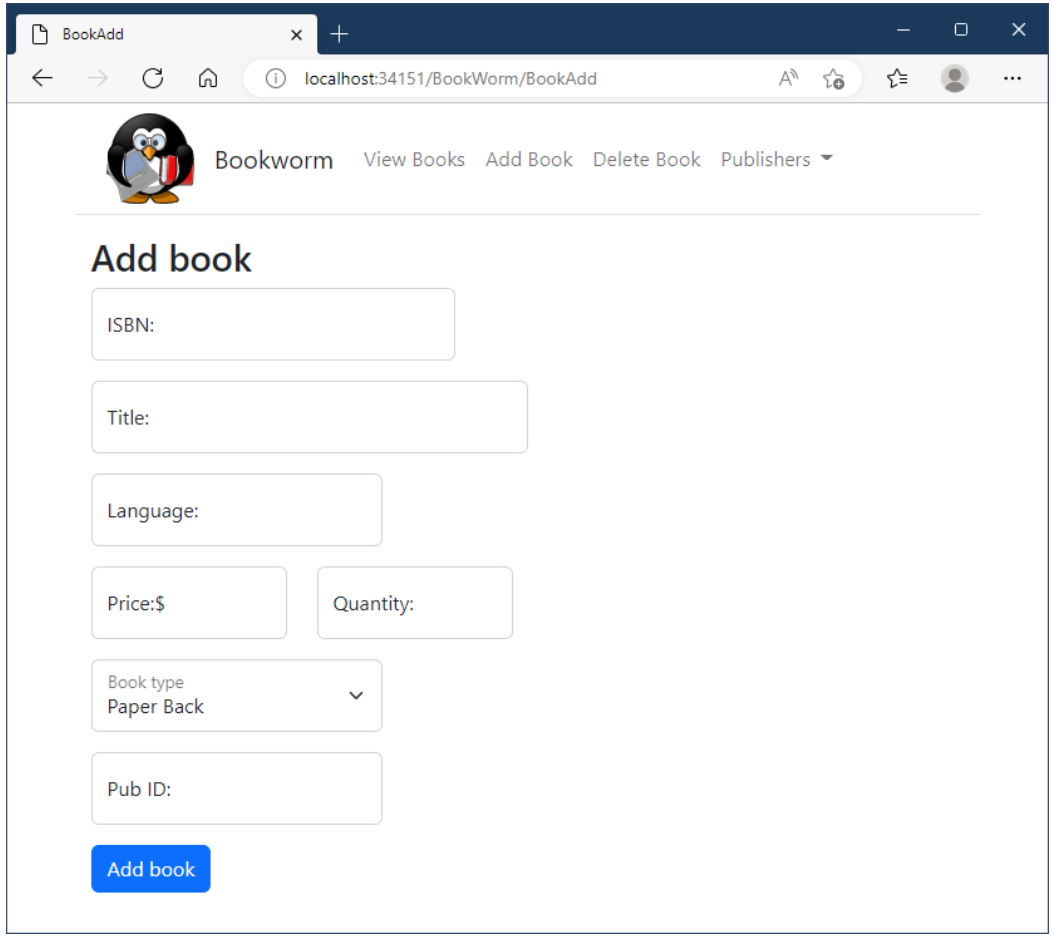
- Delete

```
sql = @"DELETE BwBook WHERE Isbn='{0}'";
string delete = String.Format(sql, isbn);
```

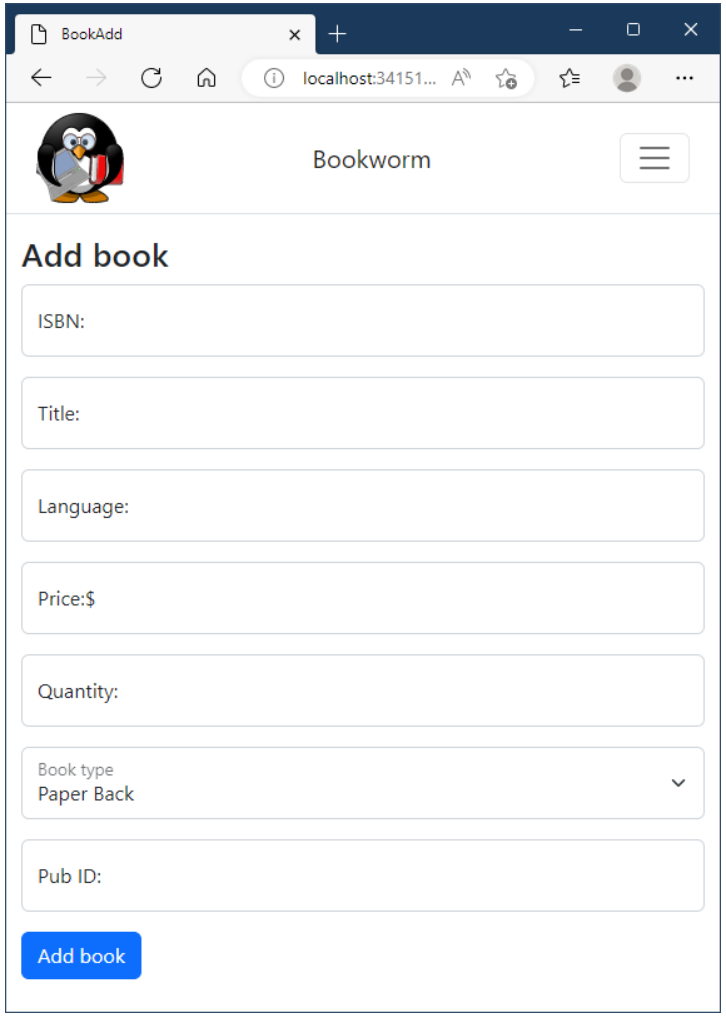
- Constraint
  - Cannot add a book without a publisher

Add New Book screen §

Medium Viewport



Small Viewport



BookAdd.cshtml §

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <title>BookAdd</title>
</head>
<body>
    @await Html.PartialAsync("BwNavBar")
    <div class="container">

        <div class="mt-3">
            <h2>Add book</h2>
        </div>

        <form method="post" action="~/BookWorm/BookAddPost">

            <div class="col-md-5 mb-3 form-floating">
                <input type="text" class="form-control" id="isbn" name="Isbn"
                       placeholder="isbn" />
                <label for="isbn">ISBN:</label>
            </div>
```

Calls the navigation bar which is stored in a separate view

The indicated action will be called when the submit button is clicked

In HTML 5, there are many types that can help ensure correct input

BookAdd.cshtml §

```
<div class="col-md-6 mb-3 form-floating">
    <input type="text" class="form-control" id="title" name="Title"
           placeholder="title" />
    <label for="title">Title:</label>
</div>

<div class="col-md-4 mb-3 form-floating">
    <input type="text" class="form-control" id="lang" name="Lang"
           placeholder="lang" />
    <label for="lang">Language:</label>
</div>

<div class="row">
    @*When using the "row" class, add class "ps-4" to the label
        to preserve correct spacing.*@
    <div class="col-md-3 mb-3 form-floating">
        <input type="number" class="form-control" id="price" name="Price"
               placeholder="$" />
        <label for="price" class="ps-4">Price:$</label>
    </div>

    <div class="col-md-3 mb-3 form-floating">
        <input type="number" class="form-control" id="qty" name="Qty"
               placeholder="#" />
        <label for="qty" class="ps-4">Quantity:</label>
    </div>
</div>
```

'form-floating' requires a placeholder. The value doesn't matter

'row' class required when more than one field per line.

Matches the value read in the controller (Case Sensitive)

Match each other. Can be same/different to the name attribute.

BookAdd.cshtml §

```
<div class="col-md-4 mb-3 form-floating">
    <select class="form-select" id="bktype" name="Bktype">
        <option value="Hard Back">Hard Back</option>
        <option value="Paper Back" selected="selected">Paper Back</option>
        <option value="Magazine">Magazine</option>
    </select>
    <label for="email">Book type</label>
</div>

<div class="col-md-4 mb-3 form-floating">
    <input type="number" class="form-control" id="pubid" name="Pubid"
           placeholder="pubid" />
    <label for="pubid">Pub ID:</label>
</div>

<div class="form-group mb-3">
    <input type="submit" value="Add book" class="btn btn-primary" />
</div>
```

Many bootstrap classes require the label to be after the control

Differing types have differing properties

BookAdd.cshtml §

```
@if (ViewData["Message"] != null)
{
    <div class="form-group mb-3">
        <div class="col-md-6">
            <div class="alert alert-@ViewData["MsgType"]">
                <b>Message: </b>@ViewData["Message"]<br />
                @if (!String.IsNullOrEmpty(ViewData["ExecSQL"]?.ToString()))
                {
                    <b>SQL: </b>@ViewData["ExecSQL"]?.ToString()
                }
            </div>
        </div>
    </div>
}
        </form>
    </div>
</body>
</html>
```

Message group. This group will print the message (if there is one) and optionally print the SQL statement if it was the cause of an error.

BookWormController.cs §

```
public IActionResult BookAdd()
{
    return View();
}

public IActionResult BookAddPost()
{
    IFormCollection form = HttpContext.Request.Form;

    string isbn = form["Isbn"].ToString().Trim();
    string title = form["Title"].ToString().Trim();
    string lang = form["Lang"].ToString().Trim();
    string price = form["Price"].ToString().Trim();
    string qty = form["Qty"].ToString().Trim();
    string bkType = form["Bktype"].ToString().Trim();
    string pubid = form["Pubid"].ToString().Trim();

    if (ValidUtl.CheckIfEmpty(isbn, title, lang, price, qty, bkType, pubid))
    {
        ViewData["Message"] = "Please enter all fields";
        ViewData["MsgType"] = "warning";
        return View("BookAdd");
    }
```

BookWormController.cs §

```
if (!qty.IsInteger())
{
    ViewData["Message"] = "Qty must be integer";
    ViewData["MsgType"] = "warning";
    return View("BookAdd");
}

if (!price.IsNumeric())
{
    ViewData["Message"] = "Price must be numeric";
    ViewData["MsgType"] = "warning";
    return View("BookAdd");
}

string sql =
   @"INSERT INTO BwBook(Isbn, Title, Lang, Price, Qty, BkType, PubID)
     VALUES('{0}','{1}','{2}',{3},{4},'{5}',{6})";
```

BookWormController.cs §

```
        string insert = String.Format(sql, isbn, title, lang, price, qty, bkType, pubid);

        int count = DBUtl.ExecSQL(insert);
        if (count == 1)
        {
            TempData["Message"] = "Book Successfully Added.";
            TempData["MsgType"] = "success";
            return RedirectToAction("Books");
        }
        else
        {
            ViewData["Message"] = DBUtl.DB_Message;
            ViewData["ExecSQL"] = DBUtl.DB_SQL;
            ViewData["MsgType"] = "danger";
            return View("BookAdd");
        }
    }
```

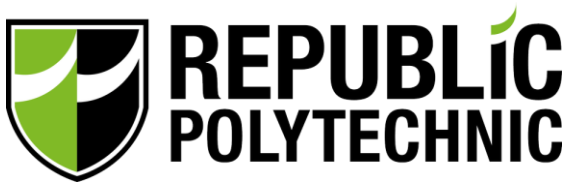
TempData for Redirect to Action

ViewData for return View

Summary

What you have learned

- Use Bootstrap navigation bar and alerts
- Reuse HTML code using Partial Views
- Transfer control from one action to another
- Pass data from one action to another using `TempData`
- Use the `ExecSQL` method to insert and delete records



Lesson 6

Modifying Data Part 2

Extension methods

Single Quotes in SQL

```
string uname = "O'Brian";
string sql = "UPDATE Student SET name='{0}' WHERE sno=6";
string update = String.Format(sql, uname);
// The SQL will result in an error
// UPDATE Student SET name='O'Brian' WHERE sno=6
```

Single Quote in VARCHAR columns must be escaped

Two single quotes is interpreted as one quote and stored as one quote.

```
string uname = "O'Brian";
string sql = "UPDATE Student SET name='{0}' WHERE sno=6";
string update = String.Format(sql, uname.EscQuote());
// sql is CORRECT
// UPDATE Student SET name='O''Brian' WHERE sno=6
// The SQL will result in the name O'Brian being correctly
// inserted into the database
```

`EscQuote()` Extension Method

Defined in DBUtl.cs

- Change every SINGLE quote into TWO SINGLE quotes so the data is stored correctly in the database as ONE SINGLE quote.

Extension method for strings

```
public static string EscQuote(this string line)
  {
     return line?.Replace("'", "''");
  }
```

If line is null, return null, otherwise call `Replace` method. Null conditional operator.

TravelUtl.cs

`Abbreviate()` extension method

```
public static string Abbreviate(this string story)
{
    if (story.Length < 30)
    {
        return story;
    }
    else
    {
        return story.Substring(0, 20) + "...";
    }
}
```

Which class the method is extending? string

What is the purpose of this method? Shorten and add …

More about views

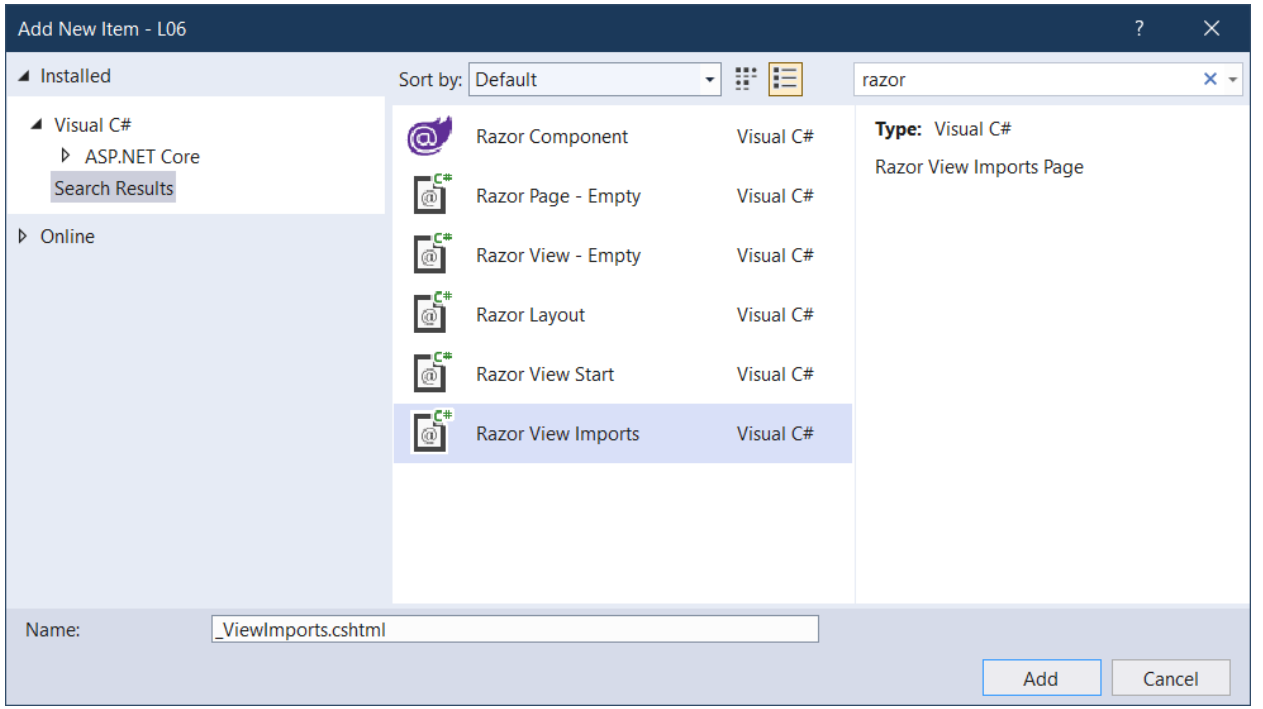
`_ViewImports.cshtml`

Shared Views

_ViewImports.cshtml

Right-click on the folder **Views > Add > New Items...**

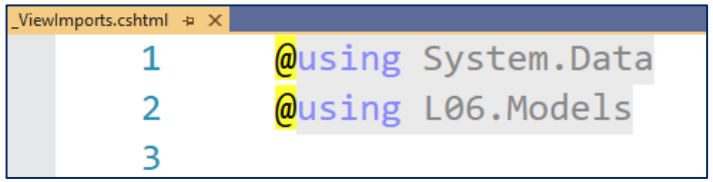
- Search for **razor**
- Click **Razor View Imports**
- Check the name `_ViewImports.cshtml` is in the Name field.



_ViewImports.cshtml

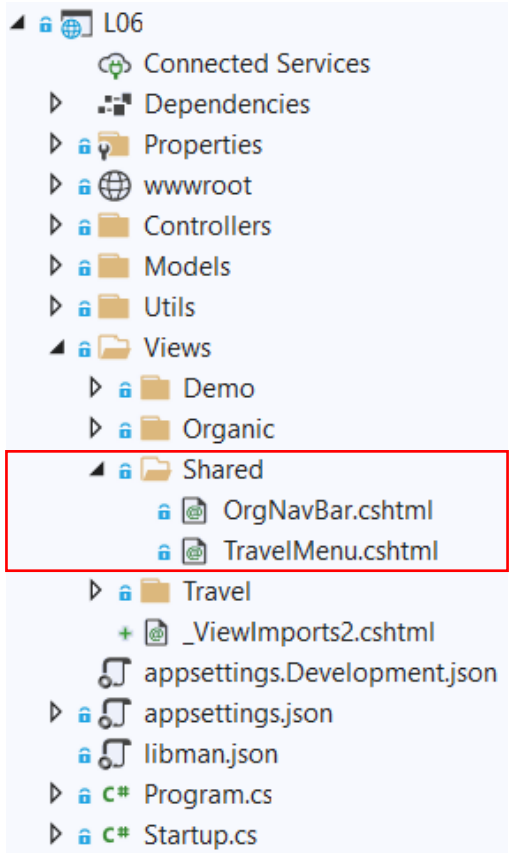
The `_ViewImports` file supports the following directives

- `@addTagHelper`
- `@removeTagHelper`
- `@tagHelperPrefix`
- **`@using`** ← For lesson 06
- `@model`
- `@inherits`
- `@inject`
- No need to repeat **`@using System.Data`** and **`@using Lesson06.Models`** in all views that use the Models, DataTable or DataRowCollection.



Views/Shared Folder

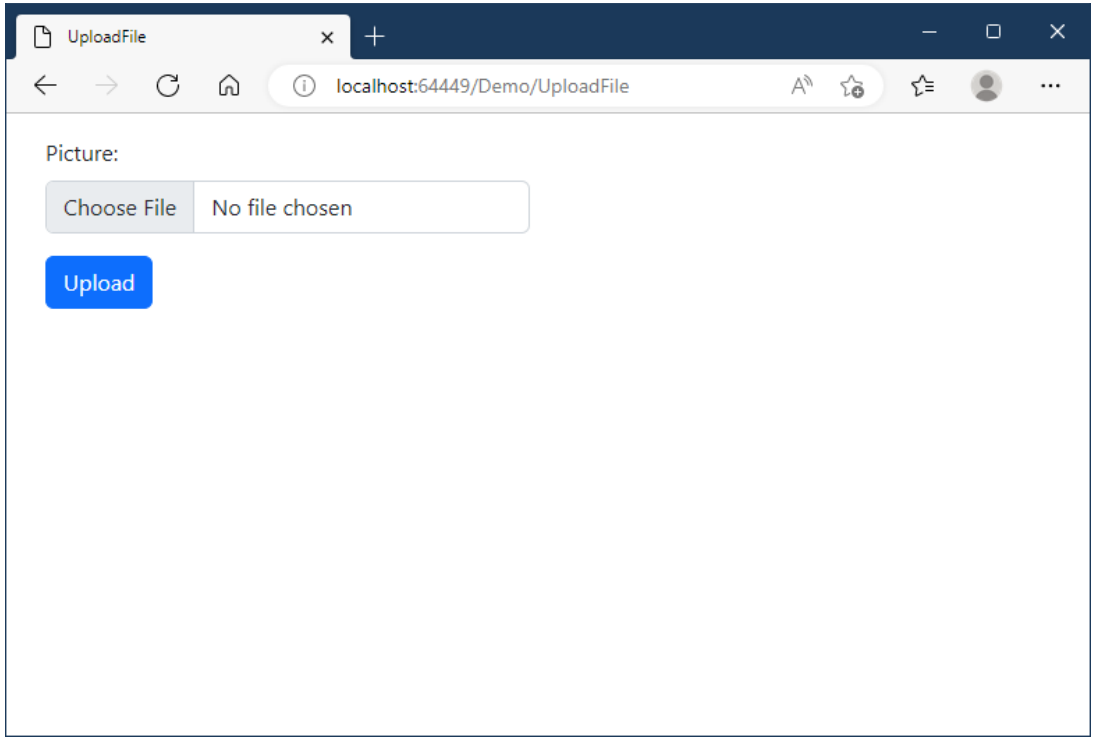
Any view in the **Views/Shared** folder is available to **ALL** controllers and views

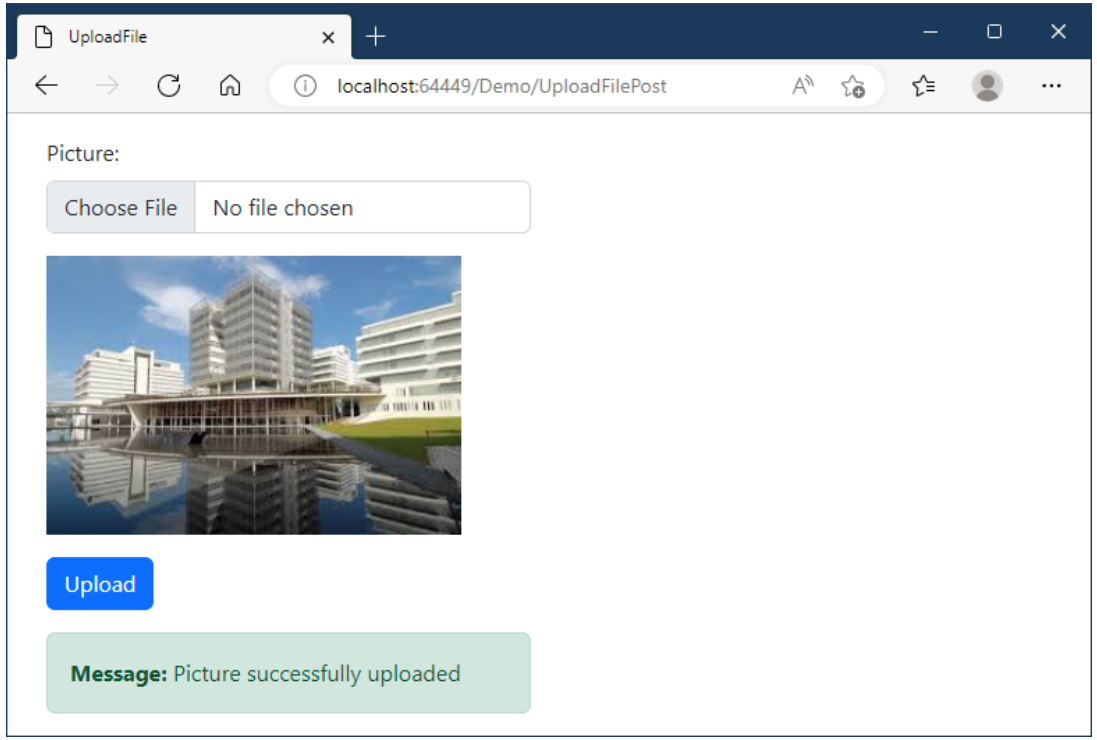


Uploading files

Demonstration

Demo/UploadFile





UploadFile.cshtml view

```
<body>
    <div class="container m-3">
        <form method="post"
              action="~/Demo/UploadFilePost"
              enctype="multipart/form-data">

            <div class="col-6 mb-3">
                <label for="formFile" class="form-label">Picture:</label>
                <input type="file" class="form-control" id="formFile"
                       name="Picture">
            </div>

            @if (ViewData["Picture"] != null)
            {
                <div class="form-group">
                    <div class="mb-3">
                        <img src="~/photos/@ViewData["Picture"]" height="200" />
                    </div>
                </div>
            }

            <div class="form-group">
                <div class="mb-3">
                    <input type="submit" value="Upload" class="btn btn-primary" />
                </div>
            </div>
```

Must include enctype for views performing file upload.

Control. The prompt 'Choose File' indicate only one file can be selected. Prompt is not easily modified

If a picture has been uploaded it will be displayed

Globally Unique Identifier (GUID)

- A GUID is a 128-bit number used to identify information.
- GUID's generated within a system are practically unique.
  - The probability of duplicate GUID's is very close to zero.
- In C#, GUID's can be generated using the **`Guid.NewGuid()`** method.
- Examples:
  - eb85a6d1-00a5-4aad-af4d-5265a4c1d255
  - 3c264870-4349-4dbf-a842-76e06866cffe

Controller

```
public class DemoController : Controller
  {
    private IWebHostEnvironment _env;
    public DemoController(IWebHostEnvironment environment)
    {
        _env = environment;
    }

    #region "File Upload"
    private string DoPhotoUpload(IFormFile photo)
    {
        string fext = Path.GetExtension(photo.FileName);
        string uname = Guid.NewGuid().ToString();
        string fname = uname + fext;
        string fullpath = Path.Combine(_env.WebRootPath, "photos/" + fname);
        FileStream fs = new FileStream(fullpath, FileMode.Create);
        photo.CopyTo(fs);
        fs.Close();
        return fname;
    }
```

The environment is injected into the controller by the ASP.NET Core Framework

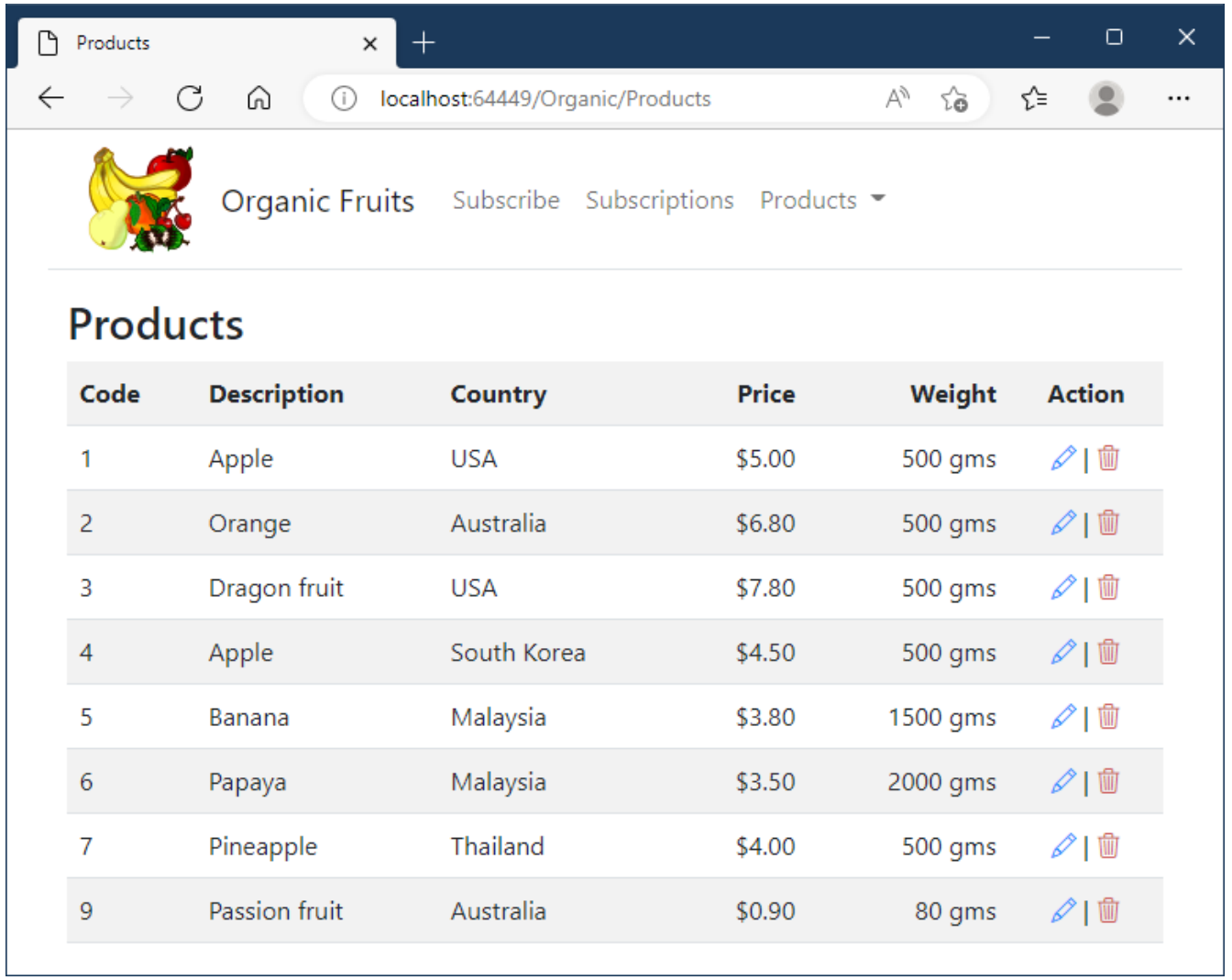
Injected environment is used to set the full path for photo and perform the upload (copy).

Controller

```
public IActionResult UploadFile()
      {
         return View();
      }

      public IActionResult UploadFilePost(IFormFile picture)
      {
         if (picture == null)
         {
            ViewData["Message"] = "Please select image to upload";
            ViewData["MsgType"] = "warning";
            return View("UploadFile");
         }

         string fname = DoPhotoUpload(picture);
         ViewData["Picture"] = fname;
         ViewData["Message"] = "Picture successfully uploaded";
         ViewData["MsgType"] = "success";

         return View("UploadFile");
      }
      #endregion
```

Organic Fruits

Organic/Products



Passing `dt.Rows` to a view

```
public IActionResult Products()
{
    string sql = "SELECT * FROM OrgProduct";
    DataTable dt = DBUtl.GetTable(sql);
    return View(dt.Rows);
}
```

Here we pass `dt.Rows` instead of the entire `DataTable`.

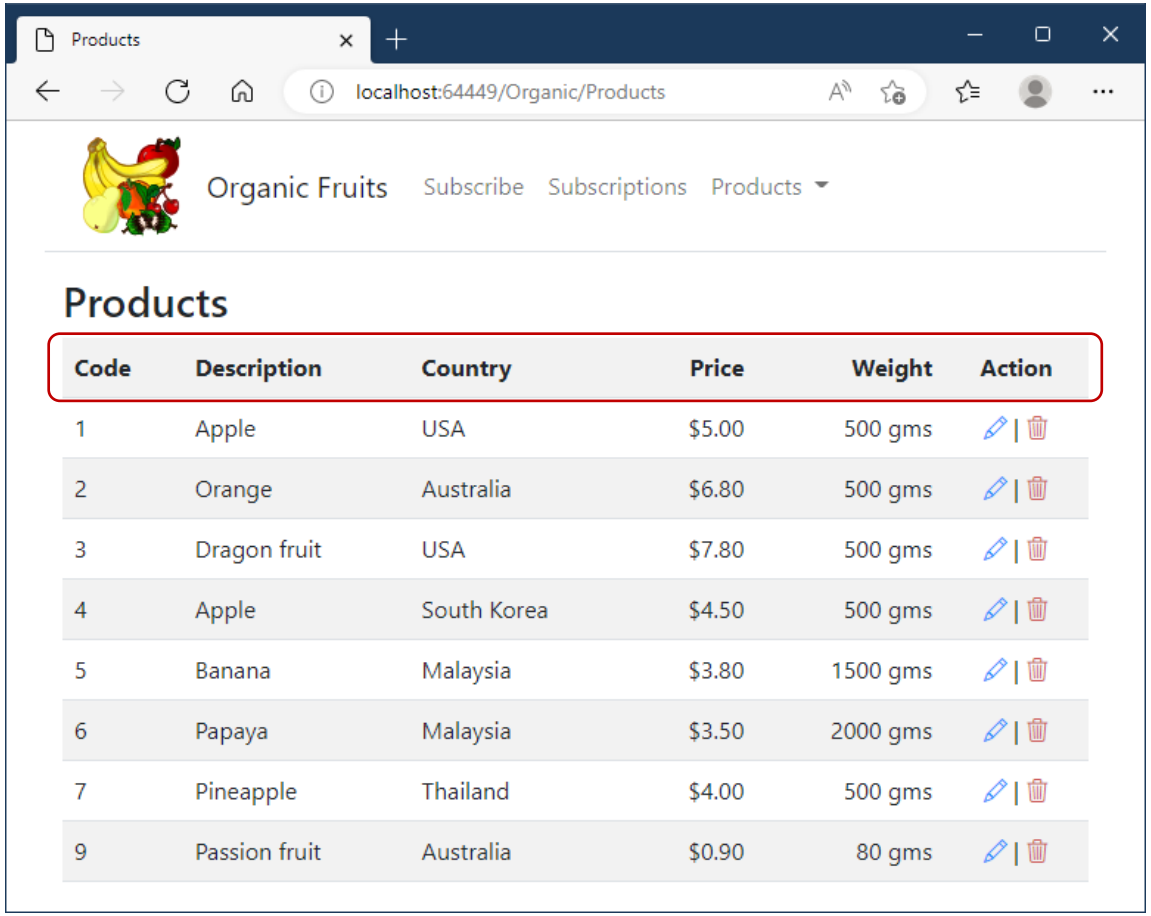
```
@model DataRowCollection
<!DOCTYPE html>
<html>
...
<body>

    @await Html.PartialAsync("OrgNavBar")

    <div class="container">

        @if (TempData["Message"] != null)
        {
            <div class="alert alert-@TempData["MsgType"]">
                @TempData["Message"]
            </div>
        }
        ...
            @foreach (DataRow row in Model)
            {...
            }
```

The @model directive is now for a `DataRowCollection` rather than a `DataTable`.

Use a `foreach` loop to retrieve each row.

Products.cshtml

```
<table class='table table-condensed table-striped'>
    <tr>
        <th>Product Code</th>
        <th>Description</th>
        <th>Price</th>
        <th>Weight</th>
        <th>Country</th>
        <th>Action</th>
    </tr>
```



Products.cshtml

```
@foreach (DataRow row in Model)
{
    <tr>
        <td>@row["OrgCode"]</td>
        <td>@row["OrgDesc"]</td>
        <td>@row["Country"]</td>
        <td style="text-align:right">@String.Format("{0:c}", row["Price"])</td>
        <td style="text-align:right">@String.Format("{0} gms", row["Gram"])</td>
        <td style="text-align:center">
            <a href="/Organic/ProductEdit/@row["OrgCode"]"
               style="text-decoration:none; color:dodgerblue">
                <text class="bi bi-pencil"></text>
            </a> |
            <a href="/Organic/ProductDelete/@row["OrgCode"]"
               style="text-decoration:none; color:indianred"
               onclick="return confirm('Delete product @row["OrgDesc"]?')">
                <i class="bi bi-trash3"></i>
            </a>
        </td>
    </tr>
}
```

1	Apple	USA	$5.00	500 gms	✎ | 🗑
2	Orange	Australia	$6.80	500 gms	✎ | 🗑
3	Dragon fruit	USA	$7.80	500 gms	✎ | 🗑
4	Apple	South Korea	$4.50	500 gms	✎ | 🗑

Bootstrap icons

Free, high quality, open source icon library with over 1,800 icons. Include them anyway you like—SVGs, SVG sprite, or web fonts. Use them with or without Bootstrap in any project.

Bootstrap Icons

```
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <link href="~/lib/bootstrap/font/bootstrap-icons.min.css" rel="stylesheet" />
    <title>Products</title>
</head>
```

Bootstrap icons (cont.)

```
@foreach (DataRow row in Model)
{
    <tr>
        <td>@row["Sno"]</td>
        <td>@row["UserName"]</td>
        <td>@row["SubEmail"]</td>
        <td style="text-align:center">
            @if (row["Gender"].ToString()!.Equals("F"))
            {
                <i style="color:deeppink; font-size:20px"
                class="bi bi-gender-female"></i>
            }
            else
            {
                <i style="color:blue; font-size:20px"
                class="bi bi-gender-male"></i>
            }
        </td>
        <td style="text-align:center">
            @if (Convert.ToBoolean(row["Recipes"]))
            {
                <i style="color: forestgreen; font-size:20px" class="bi bi-check-circle"></i>
            }
        </td>
        <td style="text-align:center">
            @if (Convert.ToBoolean(row["News"]))
            {
                <i style="color: forestgreen; font-size:20px" class="bi bi-check-circle"></i>
            }
        </td>
```

Subscriptions

#	User Name	Email	Gender	Recipes	News	Offers	Referral	Comments
1	Bruce Banner	bigguy@avengers.com	♂	✓		✓	Friend	Thor recommended me
2	Diana Prince	diana@jleague.com	♀		✓		Magazine	<3
3	Bruce Wayne	dknight@justice.com	♂		✓		Radio	Black or dark gray only

SQL quick recap

From C207 Database Systems

UPDATE Statement

```
UPDATE Contact
   SET last_name  = 'Goh',
       first_name = 'Ivan',
       email      = 'igoh@rp.edu.sg',
       phone      = '91237676'
 WHERE contact_id = 1
```

- The Table in which records are to be updated is specified after the **UPDATE** keyword.
- The **SET** clause specifies which columns are to be updated with the new values. Each assignment must be separated by comma.
- The **WHERE** condition determines which records are to be updated

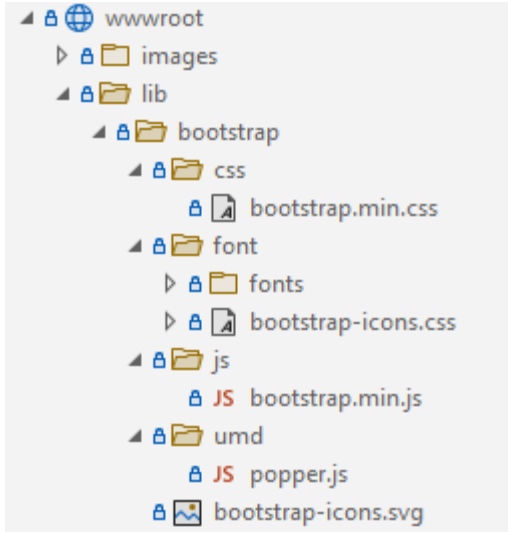
§

Problem solution

libman.json §

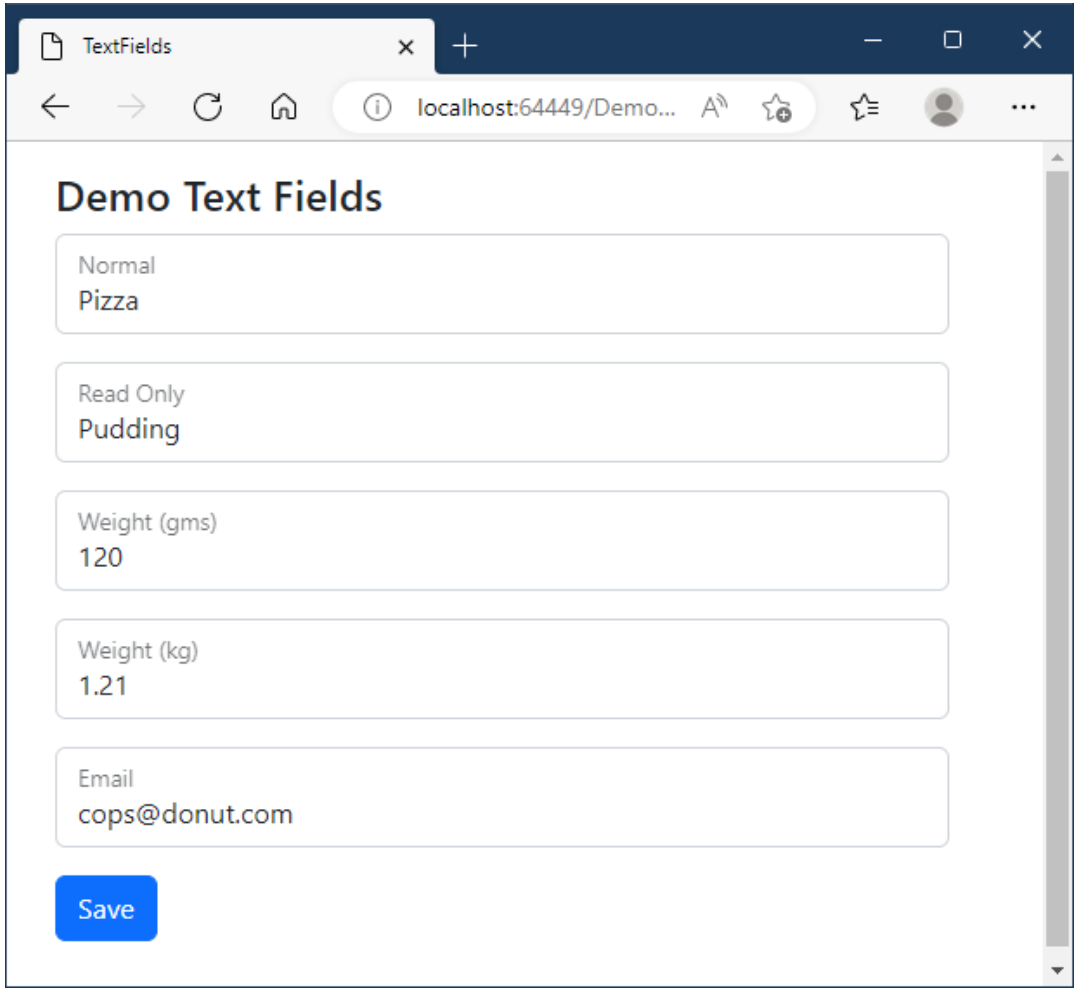
To configure which client-side libraries to be included in the project

```
{
    "version": "1.0",
    "defaultProvider": "cdnjs",
    "libraries": [
        {
            "destination": "wwwroot/lib/bootstrap/",
            "library": "popper.js@2.11.6",
            "files": [ "umd/popper.js" ]
        },
        {
            "destination": "wwwroot/lib/bootstrap/",
            "files": [
                "css/bootstrap.min.css",
                "js/bootstrap.min.js"
            ],
            "library": "bootstrap@5.2.2"
        },
        {
            "destination": "wwwroot/lib/bootstrap/",
            "files": [
                "bootstrap-icons.svg",
                "font/bootstrap-icons.css",
                "font/bootstrap-icons.min.css",
                "font/fonts/bootstrap-icons.woff"
            ],
            "library": "bootstrap-icons@1.10.2"
        }
    ]
}
```



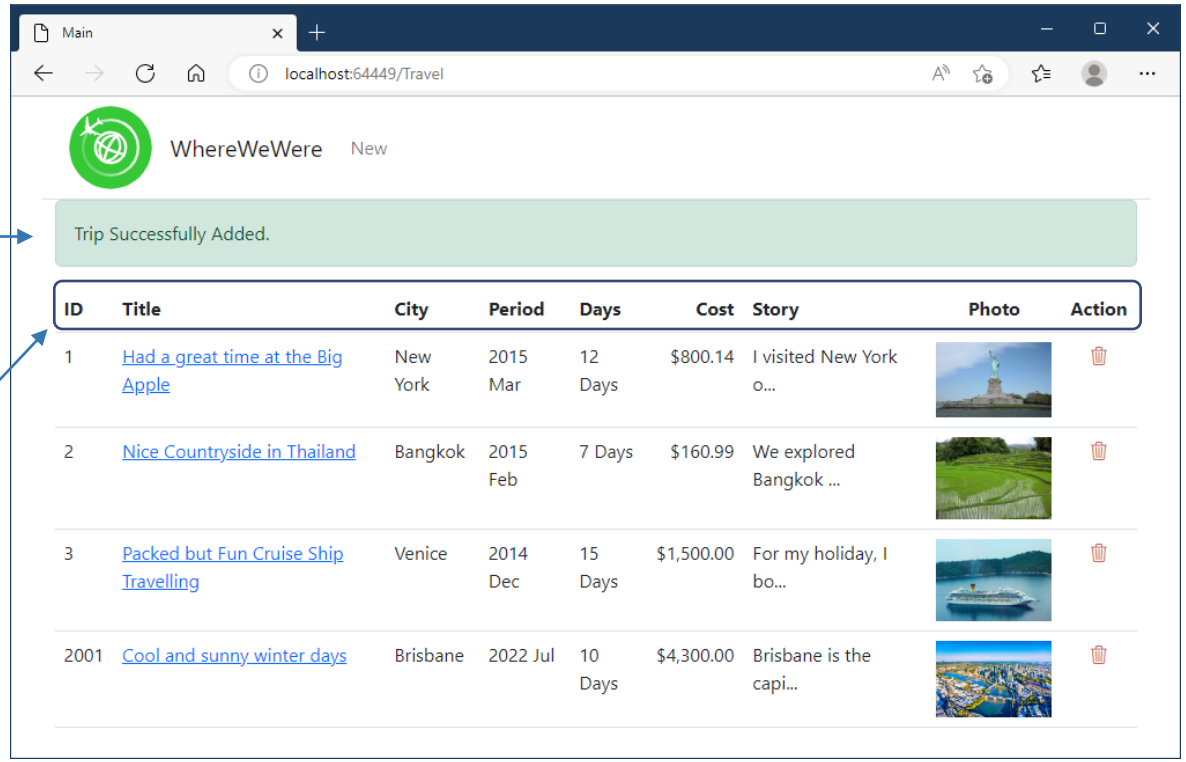
HTML Input Text Element §

```
<div class="col-md-6 mb-3 form-floating">
    <input type="text" class="form-control" id="text1" name="Text1"
           placeholder="text1" value="Pizza" required />
    <label for="name">Normal</label>
</div>

<div class="col-md-6 mb-3 form-floating">
    <input type="text" class="form-control" id="text2" name="Text2"
           placeholder="text2" value="Pasta" hidden />
    <label for="name">Hidden</label>
</div>

<div class="col-md-6 mb-3 form-floating">
    <input type="text" class="form-control" id="text3" name="Text3"
           placeholder="text3" value="Pudding" readonly />
    <label for="name">Read Only</label>
</div>

<div class="col-md-3 mb-3 form-floating">
    <input type="number" min="0" class="form-control" id="weight"
           name="Weight" placeholder="weight" required />
    <label for="name">Weight (gms)</label>
</div>

<div class="col-md-3 mb-3 form-floating">
    <input type="number" min="0" step="any" class="form-control" id="weightkg"
           name="Weightkg" placeholder="weightkg" required />
    <label for="name">Weight (kg)</label>
</div>

<div class="col-md-6 mb-3 form-floating">
    <input type="email" class="form-control" id="email" name="Email"
           placeholder="email" required/>
    <label for="email">Email</label>
</div>
```



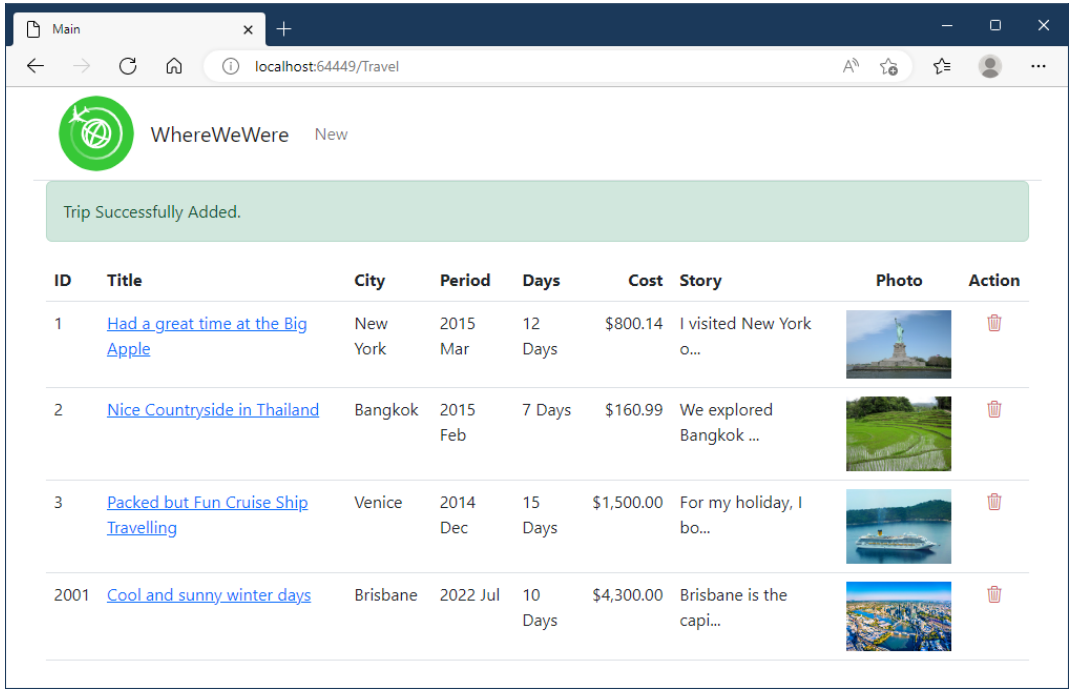
Views/Travel/Main.cshtml §

```
<body>
    @await Html.PartialAsync("TravelMenu")
    <div class="container">

        @if (TempData["Message"] != null)
        {
            <div class="alert alert-@TempData["MsgType"]">
                @TempData["Message"]
            </div>
        }
        <table class="table table-condensed table-hover">
            <tr>
                <th scope="col">ID</th>
                <th scope="col">Title</th>
                <th scope="col">City</th>
                <th scope="col">Period</th>
                <th scope="col">Days</th>
                <th scope="col">Spending</th>
                <th scope="col">Story</th>
                <th scope="col">Photo</th>
                <th scope="col">Action</th>
            </tr>
```



Views/Travel/Main.cshtml (cont.) §

```
@foreach (DataRow row in Model)
{
    <tr>
        <td>@row["ID"]</td>
        <td>
            <a href="~/Travel/Edit/@row["ID"]">@row["Title"]</a>
        </td>
        <td>@row["City"]</td>
        <td>@String.Format("{0:yyyy MMM}", row["TripDate"])</td>
        <td>@row["Duration"] Days</td>
        <td style="text-align:right">
            @String.Format("{0:C}", row["Spending"])
        </td>
        <td>
            @row["Story"].ToString()!.Abbreviate()
        </td>
        <td style="text-align:center">
            <img src="~/photos/@row["Picture"]" width="100" />
        </td>
        <td style="text-align:center">
            <a href="~/Travel/Delete/@row["ID"]"
           style="text-decoration:none; color:indianred"
           onclick="return confirm('Delete trip @row["ID"]?')">
                <text class="bi bi-trash3"></text>
            </a>
        </td>
    </tr>
}
```



Travel/**Add**, **AddPost** actions §

```
public IActionResult Add()
{
    return View();
}

public IActionResult AddPost(IFormFile photo)
{
    IFormCollection form = HttpContext.Request.Form;
    string title = form["Title"].ToString().Trim();
    string city = form["City"].ToString().Trim();
    string story = form["Story"].ToString().Trim();
    string tripDate = form["TripDate"].ToString().Trim();
    string duration = form["Duration"].ToString().Trim();
    string spending = form["Cost"].ToString().Trim();
    string picfilename = DoPhotoUpload(photo!);

    string sql = @"INSERT INTO Travel(Title, City, TripDate,
                                Duration, Spending, Story, Picture)
                        VALUES('{0}','{1}','{2}',{3},{4},'{5}','{6}')";

    string insert = String.Format(sql, title.EscQuote(), city.EscQuote(),
                                    tripDate, duration, spending,
                                    story.EscQuote(), picfilename);
```

Travel/**Add**, **AddPost** actions (cont) §

```
        if (DBUtl.ExecSQL(insert) == 1)
        {
            TempData["Message"] = "Trip Successfully Added.";
            TempData["MsgType"] = "success";
            return RedirectToAction("Index");
        }
        else
        {
            ViewData["Message"] = DBUtl.DB_Message;
            ViewData["MsgType"] = "danger";
            return View("Add");
        }
    }
```

Travel/`EditPost` Action §

```
public IActionResult EditPost()
{
    IFormCollection form = HttpContext.Request.Form;
    string id = form["ID"].ToString().Trim(); // Hidden Field
    string story = form["Story"].ToString().Trim();

    string sql = "UPDATE Travel SET Story='{1}' WHERE Id={0}";
    sql = String.Format(sql, id, story.EscQuote());
    if (DBUtl.ExecSQL(sql) == 1)
    {
        TempData["Message"] = "Trip Updated";
        TempData["MsgType"] = "success";
    }
    else
    {
        TempData["Message"] = DBUtl.DB_Message;
        TempData["MsgType"] = "danger";
    }
    return RedirectToAction("Index");
}
```

Travel/Delete (1) §

```
public IActionResult Delete(string id)
{
    if (id == null)
    {
        return RedirectToAction("Index");
    }

    string sql = String.Format("SELECT * FROM Travel WHERE id={0}", id);
    DataTable ds = DBUtl.GetTable(sql);
    if (ds.Rows.Count != 1)
    {
        TempData["Message"] = "Trip Record does not exist";
        TempData["MsgType"] = "warning";
    }
    else
    {
        string photoFile = ds.Rows[0]["picture"].ToString()!;
        string fullpath = Path.Combine(_env.WebRootPath, "photos/" + photoFile);
        System.IO.File.Delete(fullpath);
```

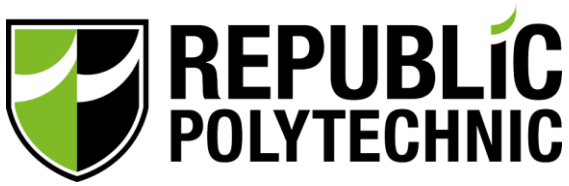
Travel/Delete (2) §

```
            int res = DBUtl.ExecSQL(String.Format("DELETE FROM Travel WHERE id={0}", id));
            if (res == 1)
            {
                TempData["Message"] = "Trip Record Deleted";
                TempData["MsgType"] = "success";
            }
            else
            {
                TempData["Message"] = DBUtl.DB_Message;
                TempData["MsgType"] = "danger";
            }
        }
        return RedirectToAction("Index");
    }
```

Summary

What you have learned

- HTML input types and uses
- Practical string extension methods
- File upload
- GUIDs
- _ViewImports.cshtml
- Bootstrap icons
- `GetTable()` method to `SELECT` database records
- `ExecSQL()` method to `INSERT`, `UPDATE` and `DELETE` database records



Lesson 7

Review and Consolidation

Concepts

Concepts commonly misunderstood - Revision

Capture data from a view

Http.Request.Form

```
public IActionResult SayHello5_Post()
{
    DoGreeting();
    string name = HttpContext.Request.Form["Name"];
    string salute = HttpContext.Request.Form["Gender"].ToString();
    string memship = HttpContext.Request.Form["Membership"];
    string smoke = HttpContext.Request.Form["Smoke"].ToString();

    ViewData["Message"] = $"Hello {salute} {name} ({memship}), Welcome!";
    if (String.Equals(smoke, "Smoking"))
        ViewData["Message"] += " Please smoke at the designated places.";

    return View("SayHello5");
}
```

```
public IActionResult Confirmation()
{
    // Use IFormCollection for shorter coding
    IFormCollection form = HttpContext.Request.Form;

    // Remove leading/trailing spaces for TextFields
    string name = form["Name"].ToString().Trim();
    string email = form["Email"].ToString().Trim();
    string refer = form["Refer"].ToString().Trim();
    string comments = form["Comments"].ToString().Trim();
    ...
}
```

The variable `form` is user-defined and can be called by any name. It holds a reference to `HttpContext.Request.Form`

The variable needs to be defined ***before*** it can be used.

name attribute

The "**name**" attribute links the view and controller

The "**id**" attribute links a label to the control

Controller

```
public IActionResult ProductEditPost(String id)
{
    IFormCollection form = HttpContext.Request.Form;
    string orgCode = form["Pcode"].ToString().Trim();
    string orgDesc = form["Description"].ToString().Trim();
    ...
}
```

View

```
<input type="hidden" name="Pcode" value="@Model.OrgCode" />

<div class="col-md-6 mb-3 form-floating">
    <input type="text" class="form-control" id="description" name="Description"
                       placeholder="description"
                       value="@Model.OrgDesc" readonly />
    <label for="description">Description</label>
</div>
```

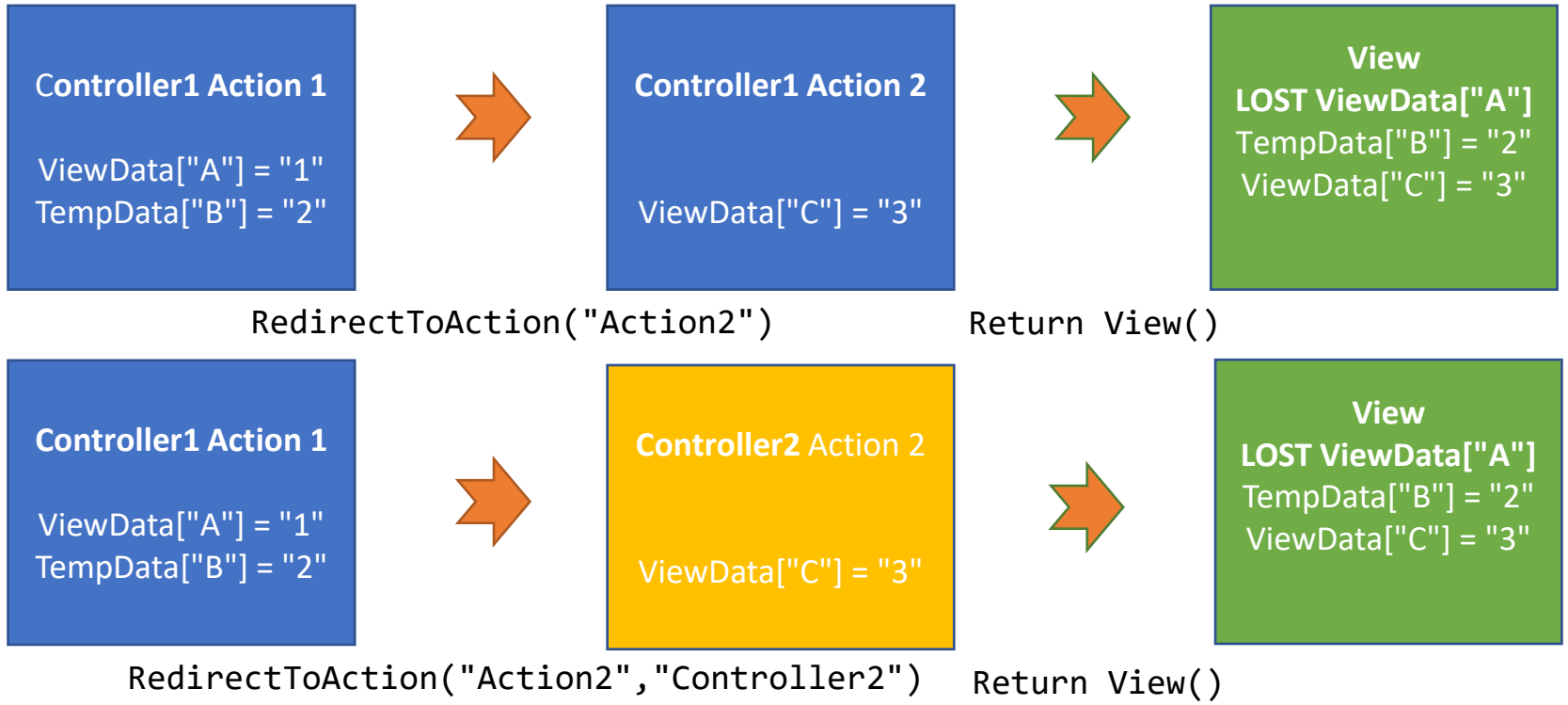
The 'for' attribute and the 'name' attribute in the view need not match.
The 'name' attribute and the value requested by HttpContext.Request.Form must match.

`ViewData` and `TempData`

Both are dictionaries

Both contain key-value pairs

Both are used to transfer data from an action to a view



Passing Data to View – DataTable

```
public IActionResult Publishers()
{
    DataTable dt = DBUtl.GetTable("SELECT * FROM BwPublisher");
    return View(dt);
}
```

```
@using System.Data
@model DataTable
...
<body>
    ...
      <table class='table'>
          <tr>
              <th>ISBN</th> <th>Title</th> <th>Language</th>
          </tr>
          @for (int row = 0; row < Model.Rows.Count; row++)
          {
             <tr>
                @for (int col = 0; col < Model.Columns.Count; col++)
                {
                   <td>@Model.Rows[row][col]</td>
                }
             </tr>
          } ...
</body>
</html>
```

Passing Data to View – DataRowCollection

```
public IActionResult Publishers()
{
    DataTable dt = DBUtl.GetTable("SELECT * FROM BwPublisher");
    return View(dt.Rows);
}
```

```
@using System.Data
@model DataRowCollection
...
<body>
    ...
      <table class='table'>
          <tr>
              <th>ISBN</th> <th>Title</th> <th>Language</th>
          </tr>
          @foreach (DataRow row in Model)
          {
              <tr>
                  <td>@row["isbn"]</td>
                  <td>@row["title"]</td>
                  <td>@row["lang"]</td>
               </tr>
           }...
</body>
</html>
```

Useful tips

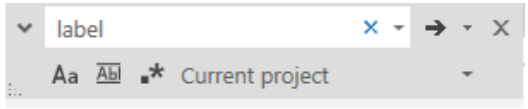
General tips

[CTRL]+[K] [D] to align your .cshtml file.

- If it doesn't align you may have an error that needs to be fixed first.

Use **[CTRL]+[F]** to bring up the FIND dialog.

- It will help you quickly identify elements




```
<form method="post"
      action="/Subscription/SubscriptionEditPost">

    <div class="col-md-5 mb-3 form-floating">
        <input type="text" id="subscriptionid" name="subscriptionId" class="form-control"
               placeholder="Subscription ID" value="@Model.SubscriptionId" readonly />
        <label for="subscriptionid">Subscription ID</label>
    </div>

    <div class="col-md-6 mb-3 form-floating">
        <select class="form-select" name="providerId" id="prov">
            @foreach (DataRow row in (DataRowCollection)ViewData["Providers"]!)
            {
                <option value="@row["provider_id"]"
                    selected="@(row["provider_id"].Equals(@Model.ProviderId))">
                    @row["name"]
                </option>
            }
        </select>
        <label for="prov">Provider </label>
    </div>
```

Recap

HTML, CSS, C#, Razor

HTML Elements

Document

- `<html>`
- `<head>`
- `<meta>`
- `<title>`
- `<style>`
- `<link>`
- `<body>`

Content & Navigation

- `<h1> ... <h6>`
- `<p>`
- `<div>`
- `<br>`
- `<img src="">`
- `<a href="">`
- `<nav>`

Formatting

- `<b> <u> <i>`
- `<span style="">`

Lists

- `<ul> <ol> <li>`

Table

- `<table>`
- `<tr>`
- `<th>`
- `<td>`

Form

- `<form method="post" action="">`
- `<label for="">`
- `<input type="text">`
- `<input type="radio">`
- `<input type="checkbox">`
- `<input type="submit">`
- `<input type="file">`
- `<select>`
- `<textarea>`

C#

Data Types

- `int, double, string, char, bool`
- `DateTime`
- `var`

Arrays

- `int[], string[]`

List<T> myList

Extension Methods

- `this`

Random rnd = new Random();

- `rnd.Next(10); // 0 to 9`
- `rnd.Next(5, 8); // 5, 6, 7`

Conversion

- `int x = Int32.Parse("123");`
- `double y = Double.Parse("-12.6");`

C#

String.Format

- Visit the the gym at http://c236dotnet.azurewebsites.net/StringFormat

Verbatim Strings

- `String v = @"Can span multiple lines without closing the quote and concatenating multiple lines";`

Interpolated Strings

- `int apple = 3`
- `String xyz = $"Can have {apple} inside";`

String Properties & Methods

- `Length`
- `Equals()`
- `Substring()`
- `ToUpper(), ToLower()`
- `Trim()`
- `Split()`
- `Replace()`

Useful Methods

ValidUtl.cs

- `ValidUtl.CheckIfEmtpy(….)`
- `String.IsInteger()`
- `String.IsNumeric()`
- `String.IsDate(string format)`
- `String.ToDate(string format)`

DBUtl.cs

- `String.EscQuote()`

Razor

`@model`

- Specifies the class of the model

`@Model`

- Used to reference the model

```
@if ... else
@for
@foreach

@await Html.PartialAsync("view")
```

DBUtl.cs and DataTable

`DBUtl.DB_Message`

- Stores any error message

`DBUtl.GetTable()`

- Only for `SELECT` statements

`DBUtl.ExecSQL()`

- For `INSERT, UPDATE, DELETE` statements

`DataTable dt`

- `dt.Rows`
- `dt.Columns`
- `dt.Rows.Count`
- `dt.Rows[ ][ ]`

§

Problem solution

Task 1 & 2 – ProviderAdd & ProviderAddPost §

```
#region "Provider Add"
// TODO: L07 Task 1 - Complete the ProviderAdd() action
public IActionResult ProviderAdd()
{
    return View();
}

// TODO: L07 Task 2 - Complete the ProviderAddPost() action.
public IActionResult ProviderAddPost()
{
    // Retrieve the text data from the form
    IFormCollection form = HttpContext.Request.Form;
    string pn = form["providername"].ToString().Trim();

    // Write the SQL to insert the record into the database
    string sql = @"INSERT INTO Provider (name) VALUES ('{0}')";
    string insert = String.Format(sql, pn.EscQuote());

    // Execute the SQL
    int res = DBUtl.ExecSQL(insert);
```

Task 1 & 2 – ProviderAdd & ProviderAddPost §

```
            // Check if the SQL was successful.
            //     If successful redirect to the 'Providers' action
            //     If failure show the DB_Message and return to the 'ProviderAdd' view
            if (res == 1)
            {
                TempData["Message"] = "Provider Record Added";
                TempData["MsgType"] = "success";
                return RedirectToAction("Providers");
            }
            else
            {
                ViewData["Message"] = DBUtl.DB_Message;
                ViewData["ExecSQL"] = DBUtl.DB_SQL;
                ViewData["MsgType"] = "danger";
                return View("ProviderAdd");
            }

        }
```

Task 3 & 4 – SubscriberAdd & SubscriberAddPost §

```
#region "Subscriber Add"
// TODO: L07 Task 3 - Create the SubscriberAdd() action
public IActionResult SubscriberAdd()
{
    return View();
}

// TODO: L07 Task 4 - Create the SubscriberAddPost() action
public IActionResult SubscriberAddPost()
{
    // Retrieve the text data from the form
    IFormCollection form = HttpContext.Request.Form;
    string username = form["uname"].ToString().Trim();
    string firstname = form["fname"].ToString().Trim();
    string familyname = form["familyname"].ToString().Trim();
    string dateofbirth = form["dob"].ToString().Trim();

    string publicprofile = form["pp"].ToString();
    string autofriends = form["aaf"].ToString();
    string broadcastposts = form["bp"].ToString();

    // Write the SQL to insert the record into the database
    string sql = @"INSERT INTO Subscriber ( username, first_name, family_name, dob,
                    public_profile_flag, auto_accept_friends_flag, broadcast_posts_flag)
                    VALUES ('{0}','{1}','{2}','{3}','{4}','{5}','{6}')";
    string insert = String.Format(sql, username, firstname.EscQuote(), familyname.EscQuote(),
                     dateofbirth, publicprofile, autofriends, broadcastposts);
```

Task 3 & 4 – SubscriberAdd & SubscriberAddPost §

```
            // Execute the SQL
            int res = DBUtl.ExecSQL(insert);

            // Check if the SQL was successful.
            //     If successful redirect to the 'Subscribers' action
            //     If failure show the DB_Message and return to the 'SubscriberAdd' view
            if (res == 1)
            {
                TempData["Message"] = "Subscriber Record Added";
                TempData["MsgType"] = "success";
                return RedirectToAction("Subscribers");
            }
            else
            {
                ViewData["Message"] = DBUtl.DB_Message;
                ViewData["ExecSQL"] = DBUtl.DB_SQL;
                ViewData["MsgType"] = "danger";
                return View("SubscriberAdd");
            }

        }
```

Task 5 – Subscriptions.cshtml §

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <link href="~/lib/bootstrap/font/bootstrap-icons.min.css" rel="stylesheet" />
    <title>Subscription</title>
</head>
<body>

    @await Html.PartialAsync("SocNavBar")

    <div class="container">

        <div class="mt-3">
            <h3>All Subscriptions</h3>
        </div>

        @if (TempData["Message"] != null)
        {
            <div class="alert alert-@TempData["MsgType"]">
                @TempData["Message"]
            </div>
        }
```

New View - SubscriberAdd.cshtml (1) §

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <link href="~/lib/bootstrap/font/bootstrap-icons.min.css" rel="stylesheet" />
    <title>SubscriberAdd</title>
</head>
<body>
    @await Html.PartialAsync("SocNavBar")

    <div class="container">

        <div class="my-3">
            <h2>New Subscriber</h2>
        </div>

        <form method="post"
              action="/Subscriber/SubscriberAddPost">

            <div class="col-md-4 mb-3 form-floating">
                <input type="text" id="uname" name="uname" class="form-control"
                       placeholder="uname" required />
                <label for="uname">Username</label>
            </div>
```

New View - SubscriberAdd.cshtml (2) §

```
<div class="col-md-5 mb-3 form-floating">
    <input type="text" id="fname" name="fname" class="form-control"
           placeholder="fname" required />
    <label for="fname">First Name</label>
</div>

<div class="col-md-5 mb-3 form-floating">
    <input type="text" id="familyname" name="familyname" class="form-control"
           placeholder="familyname" required />
    <label for="familyname">Family Name</label>
</div>

<div class="col-md-3 mb-3 form-floating">
    <input type="date" name="dob" id="dob" class="form-control"
           placeholder="YYYY-MM-DD" required />
    <label for="dob">Date of Birth</label>
</div>
```

New View - SubscriberAdd.cshtml (3) §

```
<div class="mb-3 form-floating">
    <div class="form-control border-0 h-auto">
        <div class="form-check form-check-inline">
            <input type="checkbox" class="form-check-input" name="pp" id="pp" value="1" />
            <label class="form-check-label" for="pp">Public Profile</label>
        </div>
        <div class="form-check form-check-inline">
            <input type="checkbox" class="form-check-input" name="aaf" id="aaf" value="1" />
            <label class="form-check-label" for="aaf">Auto-accept Friends</label>
        </div>
        <div class="form-check form-check-inline">
            <input type="checkbox" class="form-check-input" name="bp" id="bp" value="1" />
            <label class="form-check-label" for="bp">Broadcast Posts</label>
        </div>
    </div>
    <label>Settings</label>
</div>

<input type="submit" class="mb-3 btn btn-primary" value="Save" />
```

New View - SubscriberAdd.cshtml (4) §

```
            <input type="submit" class="mb-3 btn btn-primary" value="Save" />

            @if (ViewData["Message"] != null)
            {
                <div class="form-group row">
                    <div class="col-md-6">
                        <div class="alert alert-@ViewData["MsgType"]">
                            <b>Message: </b>@ViewData["Message"]<br />
                            @if (!String.IsNullOrEmpty(ViewData["ExecSQL"]?.ToString()))
                            {
                                <b>SQL: </b>

                                @ViewData["ExecSQL"]?.ToString()
                            }
                        </div>
                    </div>
                </div>
            }

        </form>
    </div>
</body>
</html>
```

Task 5 – Subscriptions.cshtml §

```
@if (Model != null)
{
    <table class='table table-striped table-hover'>
        <tr>
            <th>Subscription</th>
            <th>Family Name</th>
            <th>First Name</th>
            <th>Username</th>
            <th>Provider</th>
            <th>Date Subscribed</th>
            <th>Action</th>
        </tr>

        @foreach (DataRow row in Model)
        {
            <tr>
                <td>@row["subscription_id"]</td>
                <td>@row["family_name"]</td>
                <td>@row["first_name"]</td>
                <td>@row["username"]</td>
                <td>@row["name"]</td>
                <td>@String.Format("{0:yyyy-MM-dd}", @row["date_subscribed"])</td>
```

Task 5 – Subscriptions.cshtml §

```
                    <td>
                        <a href="/Subscription/SubscriptionEdit/@row["subscription_id"]"
                    style="text-decoration:none; color:dodgerblue">
                            <i class="bi bi-pencil"></i>
                        </a>
                        |
                        <a href="/Subscription/SubscriptionDelete/@row["subscription_id"]"
                    style="text-decoration:none; color:indianred"
                    onclick="return confirm('Delete Subscription [@row["subscription_id"]]')">
                            <i class="bi bi-trash3"></i>
                        </a>
                    </td>
                </tr>
            }
        </table>
    }
    else
    {
        <p class="alert alert-danger">No Subscriptions found.</p>
    }
</div>

</body>
</html>
```

Task 6 – SubscriptionDelete §

```
public IActionResult SubscriptionDelete(string id)
{
    if (id == null)
    {
        return RedirectToAction("Subscriptions");
    }

    string sql = String.Format("SELECT * FROM Subscription WHERE subscription_id={0}", id);
    DataTable ds = DBUtl.GetTable(sql);
    if (ds.Rows.Count != 1)
    {
        TempData["Message"] = "Subscription Record does not exist.";
        TempData["MsgType"] = "warning";
    }
    else
    {
        sql = "DELETE FROM Subscription WHERE subscription_id={0}";
        string delete = string.Format(sql, id);
        int res = DBUtl.ExecSQL(delete);
        if (res == 1)
        {
            TempData["Message"] = "Subscription Record Deleted";
            TempData["MsgType"] = "success";
        }
        else
        {
            TempData["Message"] = DBUtl.DB_Message;
            TempData["MsgType"] = "danger";
        }
    }
    return RedirectToAction("Subscriptions");
}
```

Task 7 – SubscriptionEdit §

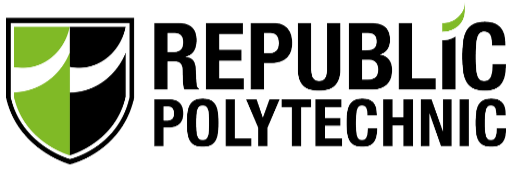
```
public IActionResult SubscriptionEdit(String id)
{
    // TODO: L07 Task 7 - Complete the SubscriptionEdit(string id) action
    string select = $"SELECT * FROM Subscription WHERE subscription_id={id}";
    DataTable dt = DBUtl.GetTable(select);
    if (dt.Rows.Count == 0)
    {
        TempData["Message"] = "Subscription Not Found";
        TempData["MsgType"] = "warning";
        return RedirectToAction("Subscriptions");
    }
    else
    {
        PopulateViewData();
        Subscription model = new()
        {
            SubscriptionId = (int)dt.Rows[0]["subscription_id"],
            SubscriberId = (int)dt.Rows[0]["subscriber_id"],
            ProviderId = (int)dt.Rows[0]["provider_id"],
            DateSubscribed = (DateTime)dt.Rows[0]["date_subscribed"],
        };
        return View(model);
    }
}
```

Task 8 – SubscriptionEditPost §

```
public IActionResult SubscriptionEditPost()
{
    IFormCollection form = HttpContext.Request.Form;
    string sid = form["subscriberId"].ToString().Trim();
    string pid = form["providerId"].ToString().Trim();
    string subID = form["subscriptionId"].ToString().Trim();
    string subDate = form["dateSubscribed"].ToString().Trim();

    // Update Record in Database
    string sql = @"UPDATE Subscription
                   SET subscriber_id = {1}, provider_id = {2}, date_subscribed ='{3}'
                   WHERE subscription_id = {0} ";

    string update = String.Format(sql, subID, sid, pid, subDate);

    int count = DBUtl.ExecSQL(update);
    if (count == 1)
    {
        TempData["Message"] = "Subscription Updated";
        TempData["MsgType"] = "success";
    }
    else
    {
        TempData["Message"] = DBUtl.DB_Message;
        ViewData["ExecSQL"] = DBUtl.DB_SQL;
        TempData["MsgType"] = "danger";
    }
    return RedirectToAction("Subscriptions");
}
```

End

- Enjoy your term break
- Be safe. See you all again in July
- 



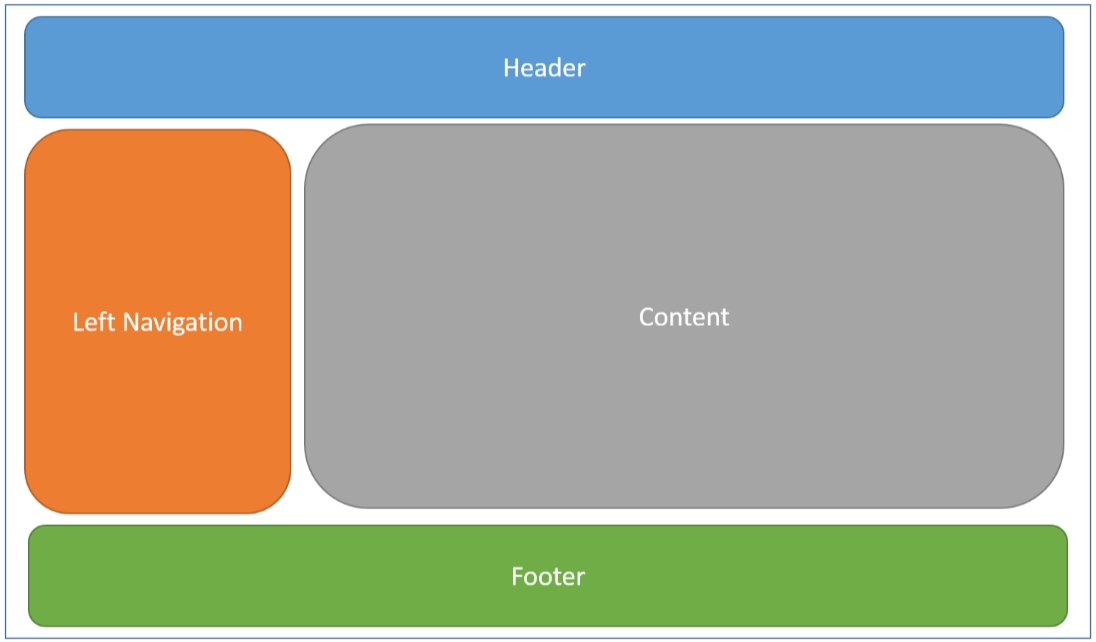
C236 - Web App Development in .NET

Lesson 08 - Security (Part 1)

Layouts

Layouts

- Web apps usually have a common layout to define a **site** template.
- Provides consistent look and feel from page to page.
- Includes common user interface elements such as the app header, navigation or menu elements, and footer.



Layout justifications

- Shared elements within a web app are frequently used by many pages with an app.
  - Scripts
  - Stylesheets
- These shared elements may be defined in a layout `.cshtml` file.
- The name usually starts with an underscore. E.g.,
  - `_Layout.cshtml`
  - `_Travel.cshtml`
  - `_Candidate.cshtml`
  - `_Organic.cshtml`
- Layouts reduce duplicate code in views.
- A layout can be referenced by any view used within the app.

Using a layout

Specify a Layout in individual views

```
@{
    // Inside each .cshtml view file
    Layout = "_Candidate";
}
```

Specify a layout for all views

Use the `Views/_ViewStart.cshtml` file

```
@{
    // Inside _ViewStart.cshtml file
    Layout = "_Travel";
}
```

Including scripts and stylesheets in a layout (e.g., `_Candidate.cshtml`)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    <script src="~/lib/bootstrap/umd/popper.js"></script>
    <script src="~/lib/bootstrap/js/bootstrap.min.js"></script>
    <link href="~/lib/bootstrap/font/bootstrap-icons.min.css" rel="stylesheet" />
    <title>Candidate</title>
</head>
<body>
    <div class="container border-bottom">
        <nav class="navbar navbar-expand-md">
            <div class="container-fluid">

                <a class="navbar-brand" asp-controller="Candidate" asp-action="Index">
                    <img src="~/images/applicant.svg" height="72" />
                    Candidates
                </a>
```

Note:

The order of the inclusions in the `head` element is important. Include the `bootstrap.min.css`, `popper.js`, `bootstrap.min.js` and `bootstrap-icons.min.css` in the order shown for predictable results.

Including a `navbar` in a layout (e.g., `_Candidate.cshtml`)

```
<div class="container border-bottom">
    <nav class="navbar navbar-expand-md">
        <div class="container-fluid">

            <a class="navbar-brand" asp-controller="Candidate" asp-action="Index">
                <img src="~/images/applicant.svg" height="72" />
                Candidates
            </a>

            @*Button below (hamburger menu) appears when viewport too small.*@
            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
                    data-bs-target="#navbarSupportedContent">
                <span class="navbar-toggler-icon"></span>
            </button>

            <ul class="navbar-nav mr-auto">
                <li class="nav-item">
                    <a class="nav-link"
                       asp-controller="Candidate"
                       asp-action="Create">New Candidate</a>
                </li>
            </ul>

            <div class="collapse navbar-collapse" id="navbarSupportedContent">
            </div>
        </div>
    </nav>
</div>>
```

Note:

When included in a layout, the `navbar` is only specified once, then used throughout the application.

Including a `RenderBody()` in a layouts (e.g., `_Candidate.cshtml`)

```
    <div class="container m-3">
        @RenderBody()
    </div>

</body>
</html>
```

Note:

The code from a view using a layout is inserted at the position of `@RenderBody()`. Therefore the view using the layout is only required to render the specifics of that view.

Referencing the layout in a view

```
@{
    Layout = "_Candidate";
}

@model Candidate
<h1>@Model.CName</h1>
<img src="~/candidates/@Model.PicFile">

<dl class="dl-horizontal">
    <dt>RegNo : </dt>
    <dd>@Model.RegNo</dd>
    <dt>Gender : </dt>
    <dd>@Model.Gender</dd>
    <dt>Height : </dt>
    <dd>@Model.Height </dd>
    <dt>Birthdate : </dt>
    <dd>@string.Format("{0:yyyy-MM-dd}", Model.BirthDate)</dd>
    <dt>Race : </dt>
    <dd>@Model.Race </dd>
    <dt>Clearance : </dt>
    <dd>@Model.Clearance </dd>
</dl>
```

Notes:

- The **Razor Layout Directive** is specified in the top of the view.
- The code below is replaces the call to the **Razor RenderBody Method** `@RenderBody()` in the layout.
- The layout enables the removal of "boiler plate" text (e.g., `<html>`, `<head>`, `<body>`, navigation bar and scripts) from the view. (i.e., DRY - Don't Repeat Yourself)

`_ViewImports.cshtml` & Tag helpers

`_ViewImports.cshtml`

The `_ViewImports file` supports the following Razor directives:

- `@using` 📌
- `@addTagHelper` 📌
- `@removeTagHelper`
- `@tagHelperPrefix`
- `@model`
- `@inherits`
- `@inject`

The following code is used in `_ViewImports.cshtml` to simplify the code for Lesson 08.

```
@using System.Data
@using Lesson08.Models

@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Advantages:

- Eliminates `@using Lesson08.Models` in all views that use a Model.
- Eliminates `@using System.Data` in all views that use a `DataRowCollection`.
- `@addTagHelper` **enables** common tag helpers.

Tag helpers

- Tag helpers
  - Enable server-side code to easily create and render HTML elements in `.cshtml` files.
  - Look similar to standard HTML **elements** and **attributes**.
  - Target HTML elements based on **element name**, **attribute name**, or **parent tag**.
- Tag helpers used in today's Lesson
  - `asp-for`
  - `asp-controller`
  - `asp-action`
  - `asp-route-id`
  - `asp-validation-for`
  - `asp-validation-summary`
- Tag Helpers may be **enabled** in `_ViewImports.cshtml` 📌

For a more details and additional help please reference Tag Helpers in MVC

Tag helpers enable code simplification

Before

```
<!DOCTYPE html>
<head>...</head>
<body>
   @if (User.Identity.IsAuthenticated)
   {
        @*For authenticated users*@
        @await Html.PartialAsync("TravelUser")>
   }
   else
   {
        @*For unauthenticated (guest) users*@
        @await Html.PartialAsync("TravelGuest")
    }

   <div class="container mt-10">
      @RenderBody()
   </div>
```

After

```
<!DOCTYPE html>
<head>...</head>
<body>
   @if (User.Identity.IsAuthenticated)
   {
        @*For authenticated users*@
        <partial name="TravelUser">
   }
   else
   {
        @*For unauthenticated (guest) users*@
        <partial name="TravelGuest">
   }

   <div class="container mt-10">
      @RenderBody()
   </div>
```

Tag helpers enable code clarity

Before

```
@foreach (Candidate cdd in Model)
{
  <div class="col">
    <div class="card align-items-center">
      <img src="~/candidates/@cdd.PicFile"
           width="100" class="mx-3 mt-3">
      <div class="card-body">
        <div class="card-title">
          <a href="~/Candidate/Display/@cdd.RegNo">
            @cdd.CName
          </a>
        </div>
      </div>
    </div>
  </div>
}
```

After

```
@foreach (Candidate cdd in Model)
{
  <div class="col">
    <div class="card align-items-center">
      <img src="~/candidates/@cdd.PicFile"
           width="100" class="mx-3 mt-3">
      <div class="card-body">
        <div class="card-title">
          <a asp-controller="Candidate"
            asp-action="Display"
            asp-route-id="@cdd.RegNo">
            @cdd.CName
          </a>
        </div>
      </div>
    </div>
  </div>
}
```

Tag helpers enable code generation

You write

```
<div class="col-md-3 mb-3 form-floating">
  <input type="number"
         min="0"
         class="form-control"
         asp-for="RegNo"
         placeholder="RegNo"
         required />
  <label asp-for="RegNo">Reg#</label>
</div>
```

Framework creates

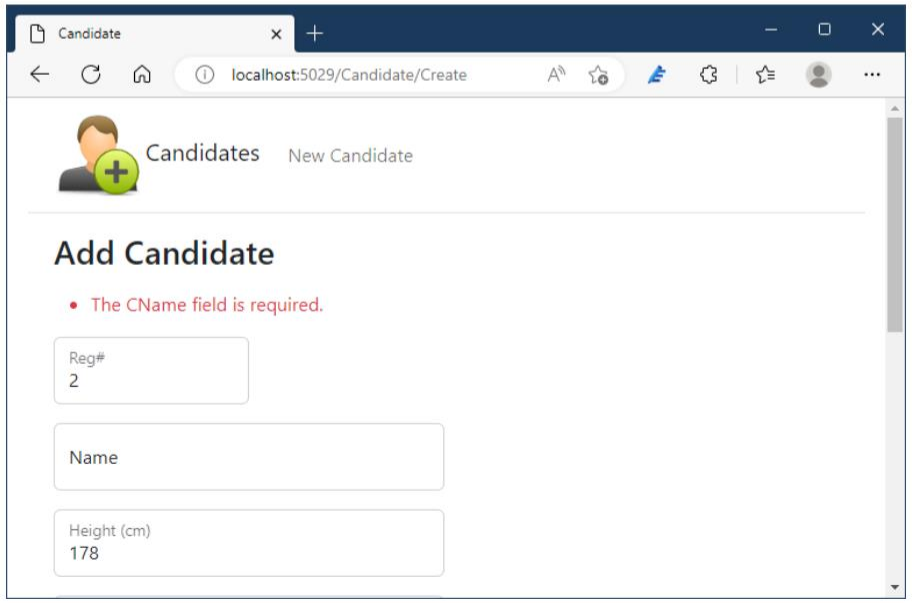
```
<div class="col-md-3 mb-3 form-floating">
  <input type="number"
         min="0"
         class="form-control"
         placeholder="RegNo"
         required data-val="true"
         data-val-required="The RegNo field is required."
         id="RegNo"
         name="RegNo"
         value="" />
  <label for="RegNo">Reg#</label>
</div>
```

Tag helpers enable a validation summary

- The validation summary tag helper is used to show a list of form **validation errors**.
- The validation summary tag helper is usually at the **top of the form**.
- Simple validation errors are derived from the **column names**.
- A sample from the `Create.cshtml` view is shown below:

```
<div asp-validation-summary="All" class="text-danger"></div>
```

Sample output is:



Model Binding & C# Attributes

Introduction to model binding

Model binding in ASP.NET Core maps data from **HTTP requests** to **action method parameters**.

- Parameters may be simple types such as `string`, `int`, or `float`, or they may be complex types.
- Model binding creates enormous productivity gains by simplifying
  - Data Conversion
  - Data Validation

Before and after model binding

Before

```
public IActionResult AddPost(IFormFile photo)
  {
    IFormCollection form = HttpContext.Request.Form;
    string title    = form["Title"].ToString().Trim();
    string city     = form["City"].ToString().Trim();
    string story    = form["Story"].ToString().Trim();
    string tripDate = form["TripDate"].ToString().Trim();
    string duration = form["Duration"].ToString().Trim();
    string spending = form["Cost"].ToString().Trim();
    ...
```

After

```
public IActionResult Create(Trip trip, IFormFile photo)
  {
  ...
```

Note:

Model binding is achieved by the addition of a parameter `trip` of type `Trip` in the action.

C# Attributes

- Attributes are used in C# to convey **declarative information** or **metadata** about various code elements such as **methods**, **assemblies**, **properties**, **types** and other elements.
- Attributes convey information on the **behaviour** of the element to which they are attached.
- A declarative tag is depicted by square brackets placed **above** the method or class for which behaviour is to be **modified**.

Examples:

```
[HttpPost]
public IActionResult Create(Candidate cdd, IFormFile photo)
{ ...
```

```
[AllowAnonymous]
[HttpPost]
public IActionResult Login(UserLogin user)
{...
```

```
public class Candidate
{
    [Required]
    public string CName { get; set; } = null!;
    ...
    [Required]
    public string Race { get; set; } = null!;
    ...
```

Controller attributes

[HttpGet]

- Return a view (form) to the user for input.
- The view can be empty or pre-populated with data for editing.
- A model is be used to pass data to the form for editing.

```
[HttpGet]
public IActionResult Create()
{
  return View();
}
```

[HttpPost]

- Receive user entered data from a view (form).
- Input data is passed from view to controller using a model.
- Data is used by the action method.

```
[HttpPost]
public IActionResult Create(Candidate cdd, IFormFile photo)
  {
  if (!ModelState.IsValid)
    {..
```

Model attributes

- `string` properties that are required have the `[Required]` attribute attached. **Recall**: Nullability and the null forgiving operator are applied in this model for `CName and` Gender`.
- `int` can never be null, only zero, so the `[Required]` attribute is not necessary.
- Similarly, `double` and `DateTime` can never be null, so the `[Required]` attribute is not necessary.

```
public class Candidate
{
    public int RegNo { get; set; }
    [Required]
    public string CName { get; set; } = null!;
    [Required]
    public string Gender { get; set; } = null!;
    public double Height { get; set; }
    public DateTime BirthDate { get; set; }
    ...
    // bool's Default is [False]
    public bool Clearance { get; set; }
    ...
```

DataType attribute:

Attributes other than `[Required]` can be added to the model.

- In the `Trip` class, the datatype `DateTime` matches the database.
- However, the system only requires the **date** component. The **time** component is of no interest.
- Bootstrap 5 will add **both** date and time pickers to the form.
- An additional `[DataType(DataType.Date)]` attribute placed on `TripDate` helps resolve this matter and ensures Bootstrap will only create a picker for the date component.

```
public class Trip
{
    ...
    [Required]
    [DataType(DataType.Date)]
    public DateTime TripDate { get; set; }
    ...
}
```

Validation using a model

- There is no longer any requirement to check for null attributes, because that is now completed by the model.
- In place, simply check the `ModelState`. If the state is invalid return the model to the view.
- The view can show a validation summary using the tag helper `asp-validation-summary`.
- If validation is successful, the model properties can be used directly in an **SQL** statement

Successful validation defined by `ModelState`

The model (`ccd`) is returned to the view.

```
[HttpPost]
public IActionResult Create(Candidate cdd, IFormFile photo)
{
  if (!ModelState.IsValid)
    {
      ViewData["Message"] = "Invalid Input";
      ViewData["MsgType"] = "warning";
      return View(cdd);
    }
else ...
```

Unsuccessful validation

A validation summary (`asp-validation-summary`) is shown to the user. This validation summary contains all errors that were found when validating the user input against the model.

```
<div class="form-group row">
  <div class="offset-sm-2 col-sm-4">
    <div asp-validation-summary="All" class="text-danger">
    </div>
  </div>
</div>
```

Turning off validation in a controller

- Occasionally it is useful/necessary to **override** the default validation on a property given by the model. This requirement may occur when
  - Checking validation prior to retrieving additional information (e.g., the username from the claim) to populate the property
  - Updating a record where the property is not changed
  - Updating/inserting a record where the property can be derived from another property.
- In this case use `ModelState.Remove` as demonstrated in the **Travel/Create** [POST] action, a portion of which appears below.

```
...
ModelState.Remove("Picture");     // No Need to Validate "Picture" - Derived from "Photo".
ModelState.Remove("SubmittedBy"); // Ignore "SubmittedBy". See claim below.
if (!ModelState.IsValid)
{
    return View("Create");
}
else
{ ...
```

Passwords & Authentication

Storing passwords

- A **hash** function is used to **encrypt** passwords before they are stored in a database.
- Encrypting passwords ensures that, even if the database is breached, an individual's passwords remain private.
- The function used is **one-way**, meaning passwords **cannot be decrypted**.

```
CREATE TABLE TravelUser (
  UserId    VARCHAR(10) PRIMARY KEY,
  UserPw    VARBINARY(50) NOT NULL,
  FullName  VARCHAR(50) NOT NULL
);

INSERT INTO TravelUser (UserId, UserPw, FullName) VALUES
('john',    HASHBYTES('SHA1', 'password1'), 'John Lim'),
('pauline', HASHBYTES('SHA1', 'password3'), 'Peter Chan');
```

UserId	UserPw	FullName
john	0xE38AD214943DAAD1D64C102FAEC29DE4AFE9DA3D	John Lim
pauline	0x1119CFD37EE247357E034A08D844EEA25F6FD20F	Pauline Chan

Authentication user

- The same function that was used to encrypt the password originally is used to encrypt the value the user enters.
- The two encrypted values, one entered by the user and the other in the database, are compared.
- If the two encrypted values (hashes) match, then the user is authenticated.

```
private static bool AuthenticateUser(string uid, string pw,
                              out ClaimsPrincipal principal)
{
    principal = null!;
    string sql = @"SELECT * FROM TravelUser
                   WHERE UserId = '{0}' AND UserPw = HASHBYTES('SHA1', '{1}')";
    string select = string.Format(sql, uid, pw);
    DataTable ds = DBUtl.GetTable(select);
    if (ds.Rows.Count == 1)
    {
        principal =
           new ClaimsPrincipal(
              new ClaimsIdentity(
                 new Claim[] {
                  new Claim(ClaimTypes.Name, ds.Rows[0]["FullName"].ToString()!)
                 },
                 CookieAuthenticationDefaults.AuthenticationScheme));
        return true;
    }
    return false;
}
```

Authentication implementation

To fully implement authentication, the following files need to be modified. The code for the implementation is not included in this slide deck. **Please refer to Lesson 08 code for details**.

- Code entry point (`Program.cs`)
  - `builder.Services.AddAuthentication(...)`
  - `app.UseAuthentication();`
  - `app.UseAuthorization();`
- Controllers (`AccountController.cs`)
  - `AccountController.cs`
    - `Login [HttpGet],[AllowAnonymous]`
    - `Login [HttpPost],[AllowAnonymous]`
    - `Logoff [Authorize]`
- Models (`UserLogin.cs`)
  - `UserID [Required]`
  - `Password [Required]`
- Views (`Login.cshtml`)
  - `UserId`
  - `Password`

Authentication check in a view

The authentication check for Lesson 08 is placed in the shared layout called `_Travel.cshtml` By placing the authentication check in this file, the code is able to render the correct view.

```
<!DOCTYPE html>
<html>
<head> .... </head>
<body>
    @if (User.Identity!.IsAuthenticated)
    @*For authenticated users*@
    {
        <partial name="TravelUserNav" />
    }
    else
    @*For unauthenticated (guest) users*@
    {
        <partial name="TravelGuestNav" />
    }

    <div class="container mt-10">
        @RenderBody()
    </div>

</body>
</html>
```

Note:

- If the username and password are validated, then the navigation for an authenticated user is displayed.
- If the username and password are **not** validated, then the navigation for a guest user is displayed.

Authentication check in a controller

- The authorization check in a controller is performed by using an annotation.
- Note: The `userid` is retrieved from the **claim**.

```
[Authorize]
public IActionResult MyTrips()
{
  string userid = User.FindFirst(ClaimTypes.NameIdentifier)!.Value;
  string select = string.Format(@"SELECT * FROM TravelHighlight
                                  WHERE UserId = '{0}'", userid);
  List<Trip> list = DBUtl.GetList<Trip>(select);
  return View("MyTrips", list);
}
```

`DBUtl.cs` additional functionality

DBUtl.GetList<ModelClass>

- New function returns a `List<ModelClass>` based on the **SQL**.
- **Important**: The property names in the model class **must match** the column names in the database table. These names are **case sensitive**.
- Mismatches between database and model are not tolerated by the framework.

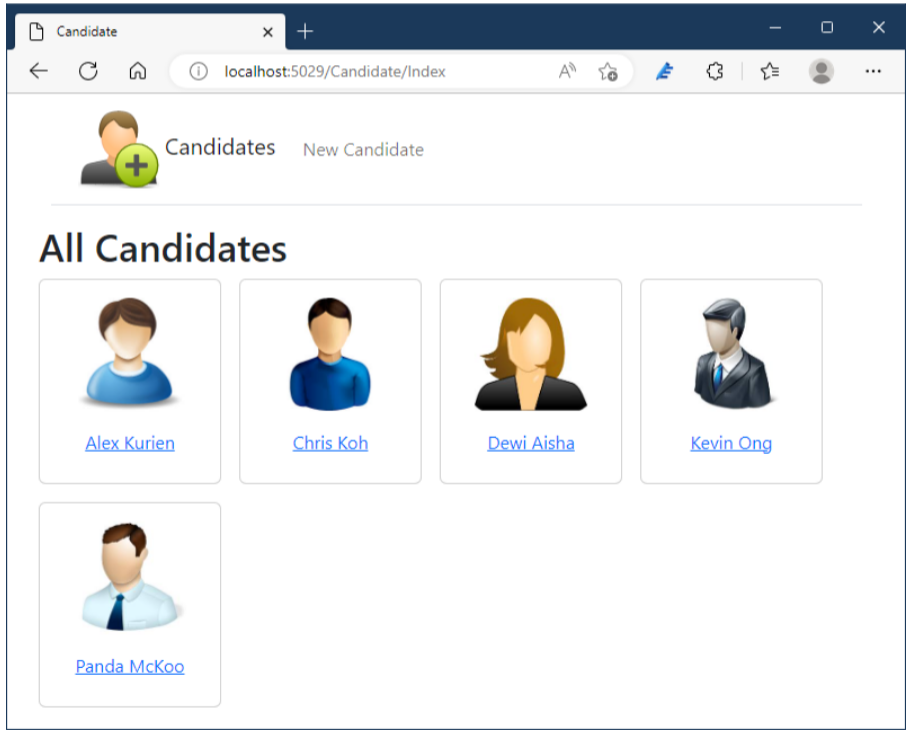
```
public class Candidate
{
    public int RegNo { get; set; }
    public string CName { get; set; }
    public string Gender { get; set; }
    public double Height { get; set;
    public DateTime BirthDate { get; set; }
    public string Race { get; set; }
    public bool Clearance { get; set; }
    public string PicFile { get; set; }
}
```

```
CREATE TABLE Candidate (
    RegNo       INT PRIMARY KEY,
    CName       VARCHAR(50) NOT NULL,
    Gender      CHAR        NOT NULL,
    Height      FLOAT       NOT NULL,
    BirthDate   DATE        NOT NULL,
    Race        VARCHAR(5)  NOT NULL,
    Clearance   BIT         NOT NULL,
    PicFile     VARCHAR(30) NOT NULL
);
```

⚠️ **Note:** After you have created the class, you must consider **nullability**, and modify the class accordingly.
E.g., `public string CName { get; set; } = null!;`

Calling `DBUtl.GetList<ModelClass>`

```
public IActionResult Index()
{
  List<Candidate> lstCandidate =
    DBUtl.GetList<Candidate>("SELECT * FROM Candidate ORDER BY CName");
  return View(lstCandidate);
}
```



Display models with multiple rows

- Use the correct razor model directive `@model List<Candidate>`.
- Loop through using a `foreach` statement.

```
...
@model List<Candidate>
<h1>All Candidates</h1>
...
<div class="row row-cols-sm-4 g-3">
  @foreach (Candidate cdd in Model)
  {
    <div class="col">
      <div class="card align-items-center">
        <img src="~/candidates/@cdd.PicFile" width="100" class="mx-3 mt-3">
        <div class="card-body">
          <div class="card-title">
            <a asp-controller="Candidate"
               asp-action="Display"
               asp-route-id="@cdd.RegNo">
               @cdd.CName
            </a>
          </div>
        </div>
      </div>
    </div>
  }
</div>
```

Retrieve a single value

- Example can be found in the **Candidate** controller, **Display** action.
- Select the record based on **id**.
- Take the **first** record returned, which will always be record `[0]`.

```
public IActionResult Display(int id)
{
   string sql = String.Format(@"SELECT * FROM Candidate
                                WHERE RegNo = {0}", id);
   List<Candidate> lstCandidate = DBUtl_orig.GetList<Candidate>(sql);
   if (lstCandidate.Count == 0)
   {
      TempData["Message"] = $"Candidate #{id} not found";
      TempData["MsgType"] = "warning";
      return RedirectToAction("Index");
   }
   else
   {
      // Get the FIRST element of the List
      Candidate cdd = lstCandidate[0];
      return View(cdd);
   }
}
```

Solving the problem

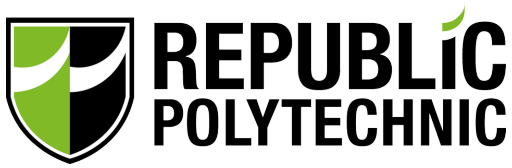
- Layouts are to be used in the solution to Lesson 08.
- Lesson 08 moves from HTML attributes to ASP.NET **tag helpers**. Because of this change, HTML tags such as `min`, `step` and `required` have been removed.
- The use of **attributes** will be extended in Lesson 09, so it is important to understand their use in models and controllers.
- Please use the **tag helpers** methods in your code. You can use the **Candidates** system for reference.

Special Note for 2022 Classes:

Full solutions to problems will no longer be released.

Please use your time in class to complete the problem to the best of your ability.

End of Lesson 08



C236 - Web App Development in .NET

Lesson 09 - Security (Part 2)

Table of Contents

SQL injection

SQL injection

> From Wikipedia, the free encyclopedia.
>
> In computing, SQL injection is a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[1][2] SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed.
>
> Source: Link to article

Anatomy of an SQL injection attack

- Consider the following SQL statement, which may be embedded within your application.

```
INSERT INTO Fruit(name) VALUES('{0}')
```

- SQL uses the single quote to surround text strings (or `varchar` columns).
- You can type your own quote within the application, modifying the intended SQL.
- For example if you typed the following text into a text field ...

```
'); DELETE FROM Fruit --
```

then, when your input is substituted into the SQL, it will become ...

```
INSERT INTO Fruit(name) VALUES(''); DELETE FROM Fruit --)
```

This code is **valid** and **executable**.

SQL injection attack analysis

- The code below repeated from the previous slide, for analysis.

```
INSERT INTO Fruit(name) VALUES(''); DELETE FROM Fruit --)
```

- This code contains **two** SQL statements, which will **both** be executed, one after the other.
- The first is `INSERT INTO Fruit(name) VALUES('');` This code will insert a blank string into the `name` column of the `Fruit` table.
- The second statement is the attack. The `DELETE FROM FRUIT --)` code will delete all records within the `Fruit` table.
- The `--` is a comment in SQL. By placing the `--` after the injected code, the attacker makes the SQL engine treat any code occurring after the injected code as a **comment**. In other words the remainder of the code is ignored.

SQL injection attack discussion

- SQL injection attacks can be used:
  - To gain unauthorized access to **sensitive data**, such as customer information or intellectual property.
  - To **modify or delete data**, potentially causing damage or disruption to the targeted organization.
  - To **bypass security controls**, such as login pages or authentication requirements.
  - To **execute arbitrary commands** on the database server, potentially allowing the attacker to **gain access** to other systems or networks.
  - To **launch further attacks**, such as **distributing malware** or setting up a network of compromised machines for use in a **botnet**.

Mitigation of an SQL injection attack.

- Stop the exploitation of the single quote by **escaping the single quote**.
- Replace each single quote with **two** single quotes.
- Good side effect, valid quotes will no longer have adverse effects. E.g., You can now enter the name **O'Brian**.
- The code is very simple and can be automatically called if placed inside the `DbUtl` helper class.

```
public static string EscQuote(this string line)
{
  return line?.Replace("'", "''")!;
}
```

Changes to `DbUtl` class to prevent SQL injection (`GetTable`)

- The `GetTable` method accepts two arguments.
  - The **`sql`** with placeholders. E.g., `'{1}'`
  - The list of values (**`params`**) to substitute into those placeholders. E.g., Value of the person's ID.
- Each value is then examined. If it is a `string` value, then the `string` **extension method** is called which replaces any **single** quote by **two single** quotes.
- `string.Format` is then called to make the substitutions, before the completed SQL statement is executed.

```
public static DataTable GetTable(string sql, params object?[] list) {
    for (int i = 0; i < list.Length; i++) {
        if (list[i] is string parameter)
        {
            list[i] = parameter?.EscQuote(); // prevent SQL Injection
        }
    }
    DB_Message = "";
    DB_SQL = string.Format(sql, list);
    DataTable dt = new();
    using SqlConnection dbConn = new(DB_CONNECTION);
    using SqlDataAdapter dAdptr = new(DB_SQL, dbConn);
    ...
```

Changes to `DbUtl` class to prevent SQL injection (`ExecSQL`)

- The `ExecSQL` method is similarly modified.

```
public static int ExecSQL(string sql, params object?[] list) {
    for (int i = 0; i < list.Length; i++) {
        if (list[i] is string parameter)
        {
            list[i] = parameter?.EscQuote(); // prevent SQL Injection
        }
    }
    DB_Message = "";
    DB_SQL = string.Format(sql, list);
    int rowsAffected = 0;
    using (SqlConnection dbConn = new(DB_CONNECTION))
    using (SqlCommand dbCmd = dbConn.CreateCommand())
    ...
```

Calling `GetTable` from controller code

- The controller code that creates the SQL statement remains unchanged.

```
string sel = @"SELECT * FROM TravelUser
               WHERE UserId = '{0}'
               AND UserPw = HASHBYTES('SHA1', '{1}')";
```

- Calling `GetTable` using the previous **insecure** method involved first calling `string.Format`.

```
string sql = string.Format(sel, uid, pw);
DataTable dt = DBUtl.GetTable(sql);
```

- Calling `GetTable` using the modified **secure** method is **simpler**, as the `string.Format` code is moved to the helper class.

```
DataTable dt = DBUtl.GetTable(sel, uid, pw);
```

Data validation

Data validation

- The process of ensuring that a program operates on clean, correct and useful data.
- Code that performs data validation is sometimes referred to as:
  - Validation rules
  - Validation constraints
  - Check routines
- Two main types will be considered:
  - Input validation
  - Model validation

Input validation

💡 All data entered into a web application must be validated in order to ensure accuracy.

- Types of validation checks:
  - Completeness check
  - Format check
  - Range check
  - Check digit check
  - Consistency check
  - Database check
- Validation can be performed on both the **client** 💻 and the **server** ☁️.

Completeness check

- Check to ensure several, possibly all, fields are entered before the form is processed.
- Ensure that all required (mandatory) data have been entered.
- If required information is missing, the form is returned unprocessed to the user.

Format check

- Use when fields are numeric or contain coded data.
- A format check ensures that data are of the right **type**.
  - Numeric
  - Alphabetic
  - Alpha-numeric
  - Currency
- A format check ensures that data are of the right **format**.
  - Dates (e.g., YYYY-MM-DD)
  - Times (e.g., HH:MM:SS)
  - Credit Card Numbers
  - Email addresses
  - IPV4/IPV6 addresses

Format check (cont'd)

- Ideally **numeric** fields will not permit users to input non-numeric data.
- If the ideal cannot be achieved, data entered must be checked to ensure that it is numeric.
- Sample format check - Car license plate:
  - Starts with three letters and first letter must be "S".
  - Followed by a maximum of four digits.
  - Ends with a checked letter.

Range check

- A range check ensures that **numeric data**, inclusive of **date** and **time** are within correct minimum and maximum values.
- Ideally, use range checks with all numeric data if possible to prevent arithmetic overflow in Web Server or Database Server.
- A range check permits only numbers between correct values.
- It can be used to screen unreasonable data, such as birth dates prior to 1920.

Check-digit check

- This type of check uses special numeric codes to prove the sequence is correct.
- International Bank Account Numbers (IBANs): IBANs are used to identify bank accounts internationally, and they include a check digit as the last two digits of the number. The check digit is calculated using a formula based on the other digits in the IBAN. This check helps to ensure the accuracy of the IBAN and to prevent errors in **data entry** or **transmission**.
- Most commonly, the final number or two numbers are calculated by a mathematical formula based on the preceding numbers.
- More examples of data with a check digit usage:
  - NRIC
  - Credit Card numbers
  - Shipping Container numbers
  - Vehicle registration plates

📌 Multiple checks can be applied, as in the case of vehicle registration plates which employ both **format** and **check-digit** checks.

Consistency check

- This type of check is used when data are related.
- The check ensures that combinations of data are valid.
- Data fields are often related. Examples:
  - Someone's birth year should precede the year in which they were married. While it may not be possible to determine whether the birth year is correct or the marriage year is correct, the relationship between them can be checked.
  - The title (e.g., Miss, Mrs) should match the gender female.

Database check

- When data are available in the database to be checked, the application may compare data entered into a form against a database to ensure validity.
- Existence check: E.g., A `ProductID` must exist in the `Product` table before that product can be added to the `Order` table.
- Non-existence check: E.g., A `UserID` must not exist in the `AppUser` table before it can be allocated to a new user.
- Business rules: E.g., A customer cannot withdraw funds from an account which exceed deposits available in that account.

Model validation

Model code

- Within a model class, add a the following to the top of the class:

```
using System.ComponentModel.DataAnnotations;
```

- Model validation can include an error message.
- The error message will be shown to the user when an error occurs on the property (field).

```
using System.ComponentModel.DataAnnotations;

namespace Lesson09.Models;

public class Trip {
  public int Id { get; set; }

  [Required(ErrorMessage = "Please enter a title.")]
  [StringLength(100, ErrorMessage = "Max 100 chars")]
  public string Title { get; set; } = null!;

  [Required(ErrorMessage = "Please enter a city.")]
  [StringLength(70, ErrorMessage = "Max 70 chars")]
  public string City { get; set; } = null!;
  ...
```

Validation attributes

`[Required]`

Mandatory with custom message.

```
[Required(ErrorMessage = "Please enter Product Code")]
public string ProductCode { get; set; } = null!;
```

`[StringLength]`

Specify number of characters in a string field.
Must be used together with `[Required]`.

```
[Required(ErrorMessage = "Please enter Product Code")]
[StringLength(12, MinimumLength = 6, ErrorMessage = "6-12 chars")]
public string ProductCode { get; set; } = null!;
```

Validation attributes (cont'd)

`[Range]`

Specify minimum and maximum values for number field.

```
[Range(13, 19, ErrorMessage = "Values from thirteen to nineteen only.")]
public int Teenager { get; set; }
```

`[Compare]`

Check two values match exactly. Below `UserPw` must exactly match `UserPw2`.

```
[Compare("UserPw", ErrorMessage = "Passwords do not match.")]
[DataType(DataType.Password)]
public string UserPw2 { get; set; } = null!;
```

Validation attributes (cont'd)

Other useful standard attributes include:

- `[CreditCard]` : Validates the property has a credit card format.
- `[EmailAddress]` : Validates the property has an email format.
- `[Phone]` : Validates the property has a telephone format.
- `[Url]` : Validates the property has a URL format.
- `[RegularExpression]` : Validates that the data matches the specified regular expression.

DataType attributes

- DataType attributes provide **display** information to the views.
- DataType attributes **do not** validate if the input conforms to the specification.
- Useful DataType attributes include:
  - `[DataType(DataType.DateTime)]`
  - `[DataType(DataType.Date)]`
  - `[DataType(DataType.Currency)]`
  - `[DataType(DataType.Password)]`
  - `[DataType(DataType.Text)]`
  - `[DataType(DataType.MultiLineText = 4)]`

Custom validation attributes

- It is not always possible to find the precise **validation attribute** you need. In this case you can create a **custom validation attribute**.
- A custom validation attribute lets you create metadata that you can use in the data model to validate data fields. You must derive the custom attribute from the `ValidationAttribute` base class.
- A sample is given in the code for `DateGreaterThan`, which is inside the `RP.SOI.DotNet.Utils` namespace on the slide that follows.
  - Inherits from `ValidateAttribute`
  - The comparison is made near the end in the code

```
if (currentValue > comparisonValue)
  return ValidationResult.Success!;
else
  return new ValidationResult(ErrorMessage);
```

  - As is standard practice in this module, you will not be expected to write or explain code in the `RP.SOI.DotNet.Utils`, although you are **expected to be able to use that code**. Students are encouraged to further explore custom validation in ASP.NET MVC by themselves.

Custom validation attributes (cont'd)

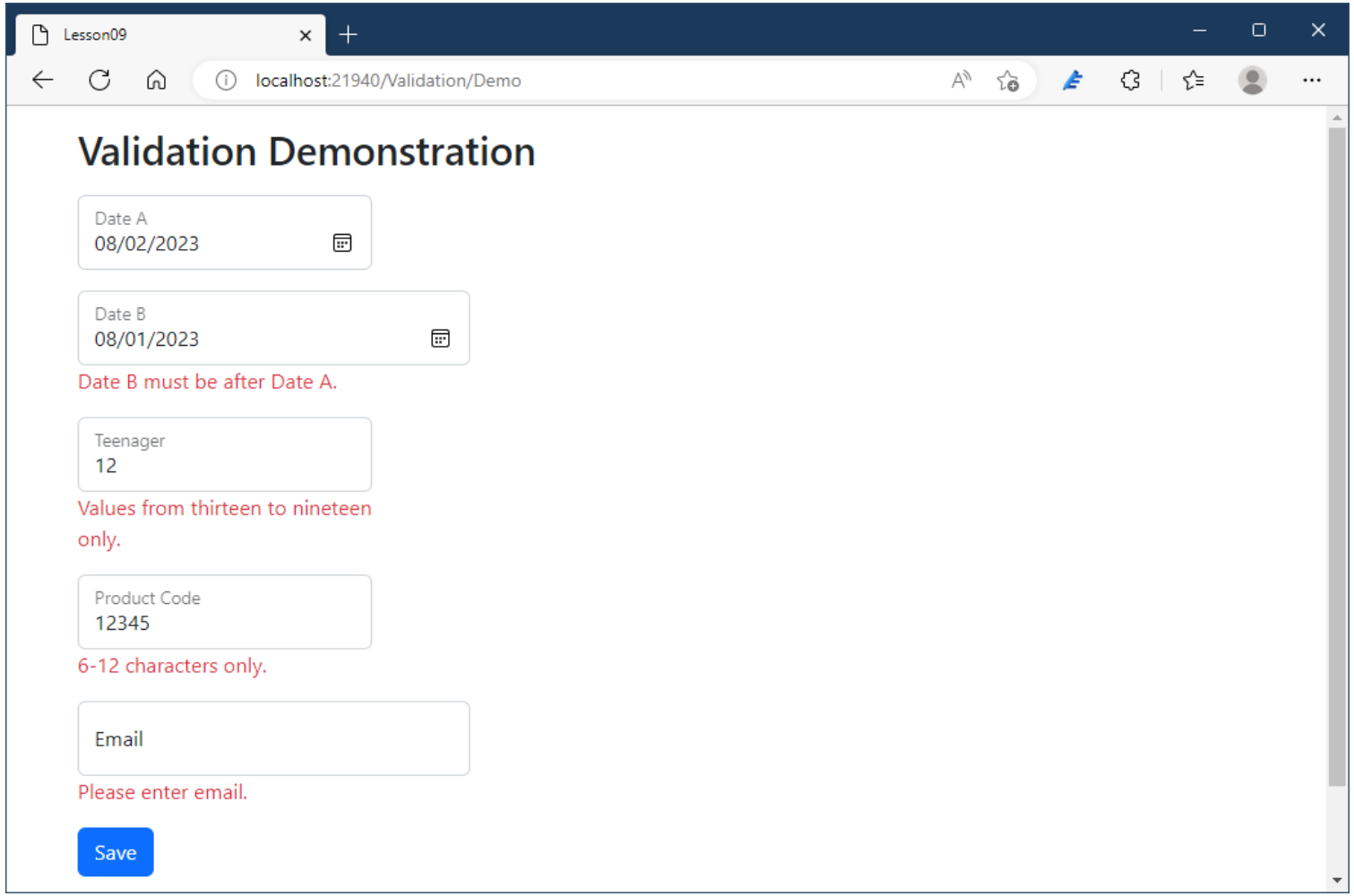
```
using System.ComponentModel.DataAnnotations;

namespace RP.SOI.DotNet.Utils;

public class DateGreaterThan : ValidationAttribute
{
    private readonly string _comparisonProperty;

    public DateGreaterThan(string comparisonProperty)
    {
        _comparisonProperty = comparisonProperty;
    }

    protected override ValidationResult IsValid(object? value, ValidationContext validationContext)
    {
        ErrorMessage = ErrorMessageString;
        var currentValue = (DateTime?)value;

        if (currentValue == null)
            return ValidationResult.Success!;

        var property = validationContext.ObjectType.GetProperty(_comparisonProperty);
        if (property == null)
            throw new ArgumentException("Property with this name not found");

        var comparisonValue = (DateTime?)property.GetValue(validationContext.ObjectInstance);

        if (comparisonValue == null)
            return ValidationResult.Success!;

        if (currentValue > comparisonValue)
            return ValidationResult.Success!;
        else
            return new ValidationResult(ErrorMessage);
    }
}
```

Inline validation messages

- Previously `asp-validation-summary` has been used to show error messages.
- The validation messages can be improved upon. Specifically they can be placed in context, beside the field to which they relate.
- Placing an error message **next** to the field where it occurred is a common practice in user interface design, and it is often referred to as **inline error messaging** or **inline validation**.
- Inline error messaging can improve the user experience by making it easier for users to understand and fix any errors in their input.
- The `asp-validation-for` **tag helper** is used to display validation messages for a specific model property in a view. It is typically used in conjunction with data annotations, discussed earlier.
- Note the positioning of the `asp-validation-for` **tag helper** in the following code:

```
<div class="col-md-6 mb-3 form-floating">
    <input type="date" asp-for="DateFieldB" class="form-control"
           placeholder="DateFieldB" />
    <label asp-for="DateFieldB">Date B</label>
    <span asp-validation-for="DateFieldB" class="text-danger"></span>
</div>
```

Inline validation messages



Controller code

- The controller checks model validation by querying the `ModelState.IsValid` property of the `ModelStateDictionary` class.
- If **all** validations within the model pass, then the `ModelState.IsValid` property returns `True`, and processing continues.
- If not all validations pass, the property returns `False` and the view can be returned to the user to correct the errors, together with the data that they have entered. (i.e., `trip`)
- As we have seen, it is possible to remove certain validations from the query. In the example below, we remove the validations for `Photo` and `SubmittedBy`, so those validations are not considered when the `ModelState.IsValid` property is next evaluated.

```
[Authorize(Roles = "user, admin")]
[HttpPost]
public IActionResult Update(Trip trip)
{
    ModelState.Remove("Photo");       // No Need to Validate "Photo"
    ModelState.Remove("SubmittedBy"); // Ignore "SubmittedBy". See claim below.
    if (!ModelState.IsValid)
    {
        return View("Update", trip);
    }
    else
    {...
```

Fine-grained authorization code

Controller code can be made more fine-grained than simply `[Authorize]`. You can add the user's role.

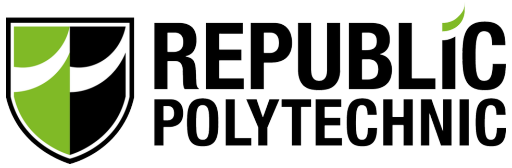
Doing so enables you to specify which user has access to which actions. Perhaps you want only admin to access a particular function, that can be achieved.

Below is an example from `TravelController.cs` that enables both normal logged-in users (`user`) and administrators (`admin`) access to the action.

```
[Authorize(Roles = "admin, user")]
public IActionResult MyTrips()
{
    string userid = User.FindFirst(ClaimTypes.NameIdentifier)!.Value;
    string select = @"SELECT * FROM TravelHighlight
                      WHERE UserId = '{0}'";
    List<Trip> list = DBUtl.GetList<Trip>(select, userid);
    return View("MyTrips", list);
}
```

End of Lesson 09

Back to Lesson 09



C236 - Web App Development in .NET

Lesson 10 - Security (Part 3)

Table of Contents

Email

Sending email

`SmtpClient` Class

- Enables applications to send email using the Simple Mail Transfer Protocol (SMTP)

```
using System.Net.Mail;
```

- To initialize an instance of the SmtpClient class that sends email by using Microsoft's or Google's service, use the following code and port numbers.

```
// Using Microsoft
SmtpClient client = new SmtpClient("smtp-mail.outlook.com", 587);
// Using Google
SmtpClient client = new SmtpClient("smtp.gmail.com", 587);
```

Creating an email

To create a message use the `MailMessage` class. This class represents an email message that can be sent using the `SmtpClient` class.

Create the message

```
MailMessage message =
   new MailMessage("donotreply@domain.com", // From
                   "student@myrp.edu.sg",   // To
                   "Subject Title",         // Subject
                   "Dear Student, …");      // Message
```

Send the message

```
client.Send(message);
```

`EmailUtl.cs` helper class

`EmailUtl.cs`

```
public static class EmailUtl
{
  ...
  private static readonly IConfiguration config =
     new ConfigurationBuilder()
        .SetBasePath(Directory.GetCurrentDirectory())
        .AddJsonFile("appsettings.json")
        .AddJsonFile($"appsettings.{env}.json", optional: true)
        .Build()
        .GetSection("EmailSettings");

  private static readonly string EMAIL_ID = config.GetValue<String>("OutlookID");
  private static readonly string EMAIL_PW = config.GetValue<String>("OutlookPW");
  ...
```

Changes to `appsettings.Development.json`

```
"EmailSettings": {
    "OutlookID": "<your outlook account>",
    "OutlookPW": "<non-2FA password>"
}
```

`EmailUtl.cs` helper class (cont'd)

`SendEmail` method

```
public static bool SendEmail(string recipient,
                             string subject, string msg,
                         out string error)
{
    SmtpClient client = new(HOST, PORT);
    client.EnableSsl = true;
    client.Timeout = 20000;
    client.Credentials = new System.Net.NetworkCredential(EMAIL_ID, EMAIL_PW);

    MailMessage mm = new(EMAIL_ID, recipient, subject, msg);
    mm.IsBodyHtml = true;
    bool success = true;
    error = "";
    try
    {
        client.Send(mm);
    }
    catch (Exception e)
    {
        error = e.Message;
        success = false;
    }
    return success;
}
```

Model code for email, with annotations

```
using System.ComponentModel.DataAnnotations;

namespace Lesson10.Models;

public class Email
{
    [Required(ErrorMessage = "A customer name is required.")]
    public string CustomerName { get; set; } = null!;

    [Required(ErrorMessage = "A customer email address is required.")]
    [EmailAddress(ErrorMessage = "Email address is not valid.")]
    public string CustomerEmail { get; set; } = null!;

    [Required(ErrorMessage = "A subject is required.")]
    public string Subject { get; set; } = null!;

    [Required(ErrorMessage = "Message cannot be null.")]
    public string Message { get; set; } = null!;
}
```

Controller code to send email using a template

```
[HttpPost]
public IActionResult SendEmail(Email email)
{
    string template = "Dear {0} \n\r" +
                      "<p>{1}</p> \n\r" +
                      "<p>Your redemption code for free gift is <b>{2}</b>.</p> \n\r" +
                      "Marketing Manager.";
    string giftcode = Guid.NewGuid().ToString()[..12];
    string body = String.Format(template, email.CustomerName, email.Message, giftcode);

    if (EmailUtl.SendEmail(email.CustomerEmail, email.Subject, body, out string result))
    {
        ViewData["Message"] = "Email Successfully Sent";
        ViewData["MsgType"] = "success";
    }
    else
    {
        ViewData["Message"] = result;
        ViewData["MsgType"] = "warning";
    }
    return View();
  }
```

Remote server validation

Remote server validation

`[Remote]` attribute

- Implements client-side validation that requires **calling a method - on the server** to determine whether field input is valid.
- E.g., the app may need to verify if a user name is already in use.
- **Controller** code

```
public IActionResult VerifyEmail(string userId)
{
    if (!_userService?.VerifyEmail(userId))
    {
        return Json($"Email {userId} is already in use.");
    }
    return Json(true);
}
```

- **Model** code

```
// Short version: [Remote("VerifyEmail", "Users")]
[Remote(action: "VerifyEmail", controller: "Users")]
public string userId { get; set; } = null!;
```

Enabling remote server validation

Remote server validation currently relies on a javascript library called jQuery. You must include relevant jQuery libraries in `libman.json` and also in the relevant view or layout, e.g., `_Layout.cshtml`.

`libman.json`

```
...
{
    "destination": "wwwroot/lib/jquery/",
    "library": "jquery-validate@1.19.5",
    "files": [
        "jquery.validate.min.js"
    ]
},
{
    "destination": "wwwroot/lib/jquery/",
    "library": "jquery-validation-unobtrusive@4.0.0",
    "files": [
        "jquery.validate.unobtrusive.min.js"
    ]
}
...
```

`_Layout.cshtml`

```
...
<script src="~/lib/jquery/jquery.min.js"></script>
<script src="~/lib/jquery/jquery.validate.min.js"></script>
<script src="~/lib/jquery/jquery.validate.unobtrusive.min.js"></script>
...
```

Remote server validation code

The following example code checks an `amount` against the current `balance` to determine if a withdrawal is possible.

Model code

```
public class VData
{
    [Remote("CheckBalance", "Demo")]
    public double Amount { get; set; }
    ...
}
```

View code

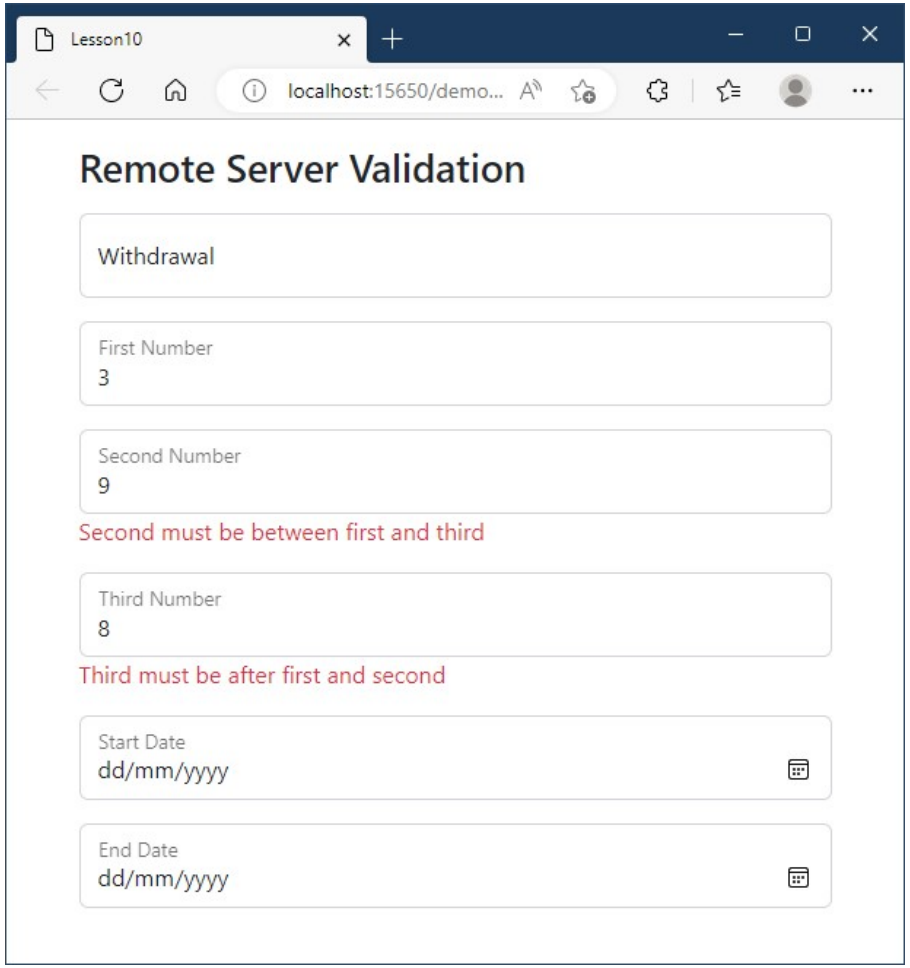
```
<div class="form-group row">
    <label class="control-label col-sm-2" asp-for="Amount">Withdrawal :</label>
    <div class="col-sm-3">
        <input asp-for="Amount" class="form-control" />
    </div>
    <div class="col-sm-4">
        <span asp-validation-for="Amount" class="text-danger"></span>
    </div>
</div>
```

Controller code

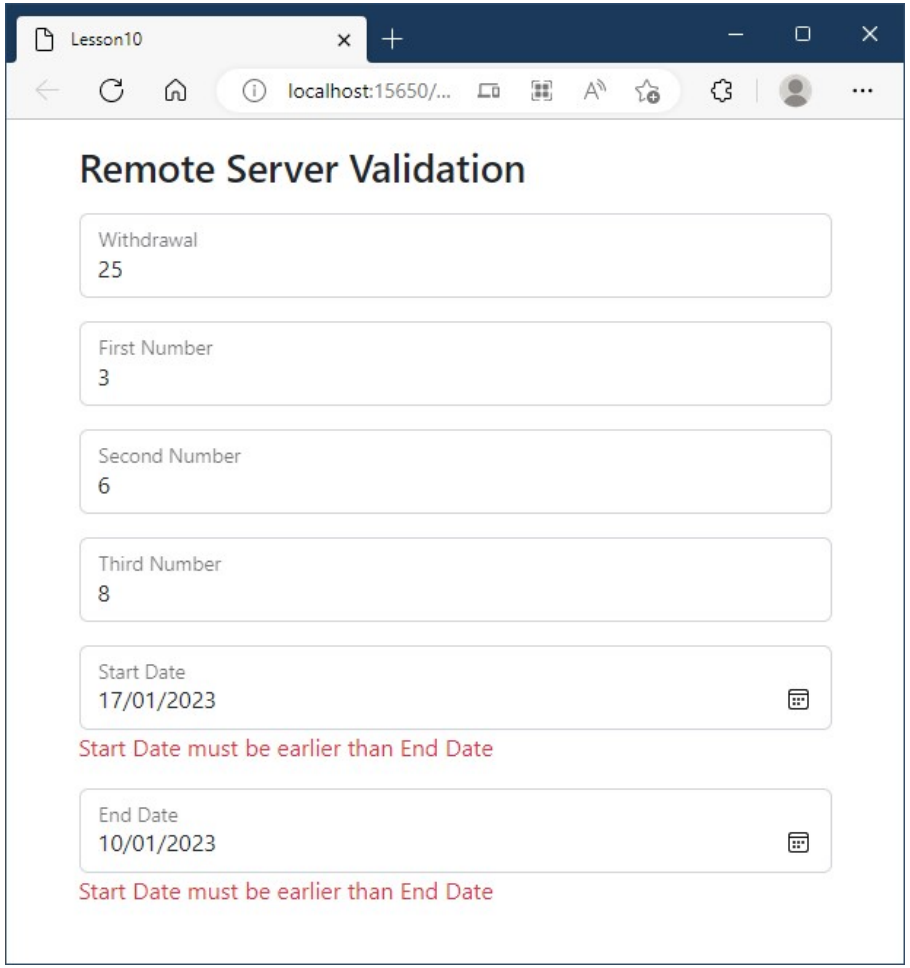
```
public IActionResult CheckBalance(double amount)
{
    double balance =
        Double.Parse(DBUtl.GetTable("SELECT 99.9").Rows[0][0].ToString()!);
    if (amount > balance)
    {
        return Json($"Not enough balance for withdrawal");
    }
    return Json(true);
}
```

Remote server validation error messages

Numbers not in order message



Dates not in order message



Regular Expressions

Regular expressions

> From Wikipedia, the free encyclopedia.
>
> A regular expression (shortened as regex or regexp; sometimes referred to as rational expression) is a sequence of characters that specifies a **search pattern** in text. Usually such patterns are used by string-searching algorithms for **find** or **find and replace** operations on strings, or for **input validation**. Regular expression techniques are developed in theoretical computer science and formal language theory.
>
> Source: Link to article

- Regular expressions are supported by most IDEs (e.g., Visual Studio, Visual Studio Code, Android Studio, Eclipse, NetBeans, Visual).
- Regular expressions are supported by most programming languages (e.g., C#, Python, JavaScript).
- Many regular expression cheat sheets exist e.g., DataCamp.

Regular expressions (cont'd)

Regular expressions are commonly classified into groups. From the DataCamp cheat sheet we will examine in detail

- Matching types of character and character classes
- Repetition
- Alternation
- Literal matches and modifiers

Regular expressions important note

The square brackets used in regular expressions signify **one character** only.

You may see the following expressions that signify only **one character**. Do not be confused.

- `[ABC]` matches either A **or** B **or** C, **but not** AB BC AC or other variation
- `[1-9]` matches a **single** number between 1 and 9. So 1, 4, 7 are matched. However `[1-39]` **does not match** all numbers between 1 and 39. Rather it matches only a number **between** 1 and 3 **or** 9, in other words one of 1, 2, 3, 9.

📌 The examples above are the **most common** student errors.

Regular expressions - Types of character & character classes

Regular expression	Matched	Not matched	Remember
[abc]	a or b or c	A, B, d, z	**Character**, lower case
[A-Z]	A, B, Z	1, 3, a, z	`[` and `]` define a **class**
[0-9]	0, 1, 2, 3	A, a, z	Digit
\d	0, 1, 2	A, a, z	Digit
\D	A, a, z	0, 1, 2	Not a digit
\w	A, a, b, 0, 3, _	?, -. !	Word character
\W	?, -, !	A, a, b, _	Not a word character
\s	<whitespace>		Whitespace
\S	<not whitespace>		Not whitespace
.	Any character		Anything
[^A-C]	a, b, z, D, Z	A, B, C	Negation. Not A-C

Regular expressions - Samples of character and character classes

RexEx	Data	Matched?
\d\d\d\d	1980	Yes
	23456	Partial
	999	No
[cfp]at	cat	Yes
	fat	Yes
	phat	No
	at	No
	path	Partial
[^abc]\w\w\w	a123	No
	d_99	Yes
	ABCD	Yes

Regular expressions - Repetition

Regular expression	Matched	Not matched	Remember
x*	x, xx, xxx		**Zero** or more
x+	x, xx, xxx		**One** or more
x?	x		**Zero** or **one** only
x{3}	xxx		Match only 3 'x'
x{n}			Match n 'x'
x{n,m}			Match between n and m 'x'

Regular expressions - Samples of repetition

RexEx	Data	Matched?
\d{4,5}	2023	Yes
	54321	Yes
	567	No
[cfph]?at	cat	Yes
	fat	Yes
	phat	No
	at	Yes
	path	Partial
[A-X]\w*	X	Yes
	A1_B2-C3	No
	A_BCDefg	Yes

Regular expressions - Literals, alternation and braces

Regular expression	Matched	Not matched	Remember
(Apple|Orange)	Apple	Banana	Apple is a **literal**
(Mr|Dr|Miss|Mrs)	Dr, Miss	Prof	The | symbol gives **alternatives**
(Ms|Dr) Jeckyl	Dr Jeckyl	Mr Jeckyl	Group by **braces**
([abc]|x|2)	a, 2	a2, x2	
([abc]|x|2){2}	aa, ax, x2	jx, x, 2	
(Mr\.|Dr\.) Jeckyl	Mr. Jeckyl, Dr. Jeckyl	Mrs Jeckyl	Special characters need to be escaped
(Mr.|Dr.) Jeckyl	Mrs Jeckyl, Mr2 Jeckyl		

Regular expressions - Summary

- Regular Expressions or RegEx are used for **matching**
- Refer to internet cheat sheets
- All characters in regular expressions must be escaped before they can be used literally. E.g., if you want to match the following pattern `[Required]` the regular expression will be `\[Required\]`.
- Parenthesis for grouping `(` and `)`
- Square brackets for defining a **single** character `[` and `]`
- Curly brackets for quantifying `{` and `}`

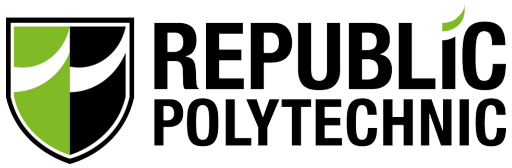
Regular expressions - Class discussion

Write regular expressions to match the following

- 'iOS or 'Andriod' or 'Win10'
- 'iPhone 12' or 'iPhone 13 Pro'
- Monday, Tuesday, Friday
- 24 hour time (e.g., 1830 hrs)

End of Lesson 10

Back to Lesson 10



C236 - Web App Development in .NET

Lesson 11 - Tabulating Data

Persisting login sessions

Remember me feature

The ability to maintain a user's login data between sessions cannot be done in HTTP, since HTTP is stateless. The mechanism therefore needs to involve files, in particular browser cookies. Much of the functionality is handled by the .net framework, however three files need to be changed.

- Login view: Add a `CheckBox`.
- User login model: Add a `bool` to hold the user's decision.
- `AccountController.cs`: Add a new authentication property.

Remember me checkbox

In the `UserLogin.cshtml` add the following code between the `Password` field and the error `Message`. This code enables a user to register their decision to be remembered, or not remembered.

```
<div class="form-control border-0 h-auto">
    <div class="form-check form-check-inline">
        <input class="form-check-input" asp-for="RememberMe" />
        <label class="form-check-label" asp-for="RememberMe">Remember Me?</label>
    </div>
</div>
```

Remember me model and controller changes

Add the following property to the `UserLogin.cs` model to hold the user's decision.

```
public bool RememberMe { get; set; }
```

Change the persistent behaviour by setting the `IsPersistent` cookie property to the user's decision (`user.RememberMe`) in `AuthenticationProperties` .

```
[AllowAnonymous]
[HttpPost]
public IActionResult Login(UserLogin user)
{
    if (!AuthenticateUser(user.UserID, user.Password, out ClaimsPrincipal principal))
    {
        ...
    }
    else
    {
        HttpContext.SignInAsync(
            CookieAuthenticationDefaults.AuthenticationScheme, principal,
            new AuthenticationProperties
            {
                //ExpiresUtc = DateTime.UtcNow.AddMinutes(20)
                IsPersistent = user.RememberMe
            });
        ...
```

jQuery data tables

jQuery DataTables

jQuery is a fast, small, and feature-rich JavaScript library. While it's popularity is on the decline, it is still used by many other established and established libraries.
jQuery DataTables is one such library, which enables advanced interaction controls on HTML tables.
Top features of jQuery DataTables include

- Pagination
- Search
- Multi-column ordering
- Responsive display

Additional scripts

- Since not all views use tables, not all views require the additional scripting that makes jQuery DataTables work.
- Developers can use the .Net framework to add **additional scripting** to selected views.
- First modify the shared `_layout.cshtml` of the view to position the scripts within the `<head>`. In the example below the additional script section is called `"BootstrapTableScripts"`, but the name can be anything that makes sense.

```
<head>
    <meta charset="utf-8" />
    ...
    @RenderSection("BootstrapTableScripts", false)

    <title>Lesson11</title>
<head>
```

Additional scripts (cont'd)

- The next step is to add in the additional scripts required.
- **jQuery DataTables** requires three scripts, which are added to `Libman.json`, together with the configuration javascript. In the code below we add the necessary scripts to the Demo `ListCadets.cshtml` view, just below the `@model` directive.

```
@section BootstrapTableScripts {
    <link href="~/lib/datatables/css/jquery.dataTables.min.css" rel="stylesheet" />
    <script src="~/lib/datatables/js/jquery.dataTables.min.js"></script>
    <script src="~/lib/datatables/js/dataTables.bootstrap.min.js"></script>

    <script>
        $(document).ready(function () {
            $('#jsUsersTable').DataTable({
                scrollY: 285,
                stateSave: true,
                lengthMenu: [15, 30, 60, 150]
            });
        });
    </script>
}
```

Additional scripts (cont'd)

The next step is to ensure the referenced table id in view `id="jsUsersTable"`, exactly matches the javascript `$('#jsUsersTable').DataTable({`

Table

```
<table id="jsUsersTable" class="table">
```

exactly matches the id referenced in the javascript.

Javascript

```
$('#jsUsersTable').DataTable({
```

Configure jQuery Datatable

The final step in the process is to configure the DataTable to ensure it is displayed as desired. There are many possibilities. You will need to refer to the jQueryDataTable documentation.

Links to the options used in `ListCadets.cshtml` are given below.

- `ScrollY: 285`: Enable vertical scrolling. Specify a height of 285 pixels for the visible region. Can be used as an alternative to paging, although both can be used.
- `stateSave: true`: DataTables store state information such as pagination position, display length, filtering and sorting. When this initialisation option is active and the end user reloads the page, the table's state will be altered to match that which was previously set.
- `lengthMenu: [15, 30, 60, -1],[15, 30, 60, "All"]`: This parameter allows you to readily specify the entries in the length drop down select list that DataTables shows when pagination is enabled.
- **Note:** Many other options are also available.

Access control in views

Access control

Views can be modified using Razor syntax to **enable role based security**.

Table `<head>`

```
<thead>
    <tr>
        <th>Title</th>
        <th>Artist</th>
        <th>Date/Time</th>
        <th>Duration</th>
        <th>Price</th>
        <th>Chamber</th>
        @if (User.IsInRole("manager"))
        {
            <th>Action</th>
        }
    </tr>
</thead>
```

Table `<body>`

```
<tbody>
    @foreach (DataRow row in Model)
    {
        <tr>
            <td>@row["Title"]</td>
            ...
            <td>@row["Chamber"]</td>
            @if (User.IsInRole("manager"))
            {
                <td>
                <a asp-controller="Performance"
                asp-action="Update"
                asp-route-id="@row["Pid"]">
                Update </a> |
                <a asp-controller="Performance"
                asp-action="Delete"
                asp-route-id="@row["Pid"]"
                onclick="return confirm('Delete Performance <@row["Title"]>')">
                Delete </a>
                </td>
            }
        </tr>
    }
</tbody>
```

Droplists and formatting in edit views

Using `asp-items` to populate a drop list

- The demo system illustrates how to populate values for a drop list and save it to `ViewData["ClassList"]` in the `DemoController.cs`. The stored values can then be referenced by reading `ViewData["ClassList"]` in `EditCadet.cshtml`.
- A new `using` statement is required in the controller: `using Microsoft.AspNetCore.Mvc.Rendering;`
- A new tag helper called `asp-items` is used to hold the options for the `<select>`.

Populate the list in the controller

```
// cList is used to populate the View's dropdown list
List<SelectListItem> cList =
    DBUtl.GetList<SelectListItem>(
      @"SELECT DISTINCT
               ClassGrp As Value,
               ClassGrp As Text
        FROM Cadet
        ORDER BY ClassGrp");
ViewData["ClassList"] = cList;
```

Reference the list in the view

```
@{
    List<SelectListItem> clist =
    ViewData["ClassList"] as List<SelectListItem> ??
    new List<SelectListItem>();
}

<div class="col-md-3 mb-3 form-floating">
    <select class="form-select"
            asp-for="ClassGrp" asp-items="clist">
    </select>
    <label asp-for="ClassGrp">Class Group</label>
    <span asp-validation-for="ClassGrp"
          class="text-danger"></span>
</div>
```

Creating static value lists in a controller

From time to time it will be useful to generate a **static**, or **selected values** only, list. This situation occurs when timing for a performance is rounded to the nearest 30 minute duration. There are two simple methods for creating such a list shown below. (Ref: `PerformanceController/Update`)

Method 1 - Create the values from a database `SELECT`

```
using Microsoft.AspNetCore.Mvc.Rendering;
...
List<SelectListItem> dlist = DBUtl.GetList<SelectListItem>(
  @"SELECT *
    FROM (VALUES('0.5', '0.5'), ('1', '1'),
                ('1.5', '1.5'), ('2', '2'),
                ('2.5', '2.5'), ('3', '3'),
                ('3.5', '3.5'), ('4', '4'))
    AS MyTable(Value, Text)");
```

Method 2 - Create the values by adding to a `List`

```
using Microsoft.AspNetCore.Mvc.Rendering;
...
List<SelectListItem> dlist = new()
  {
  new SelectListItem("0.5","0.5"), new SelectListItem("1","1"),
  new SelectListItem("1.5","1.5"), new SelectListItem("2","2"),
  new SelectListItem("2.5","2.5"), new SelectListItem("3","3"),
  new SelectListItem("3.5","3.5"), new SelectListItem("4","4")
  };
```

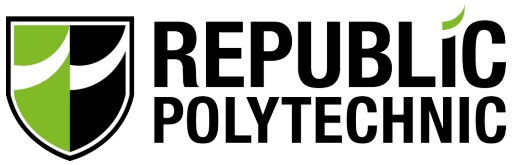
Applying a `DisplayFormat` to a model property

The following sample code illustrates how to apply a numeric format string to always show two decimal points in a numeric field. The code may be found applied in `Performance.cs`

```
[Required(ErrorMessage = "Please enter Price")]
[Range(0.0, 1000.0, ErrorMessage = "Price 0-1000")]
[DisplayFormat(DataFormatString = "{0:N2}", ApplyFormatInEditMode = true)]
public double Price { get; set; }
```

End of Lesson 11

Back to Lesson 11



C236 - Web App Development in .NET

Lesson 12 - Constructive Project Upgrades

Query strings

Query strings

- Query strings are sets of characters tacked onto the end of a URL after the question mark (?)of a URL.
- They assign values to specified parameters.
- Query strings can be used for useful purposes including filtering, searching and pagination.
- Query strings can also be used for UTM tracking and other marketing purposes.
- The query string comprises key-value pairs separated by an equals sign (=) and each pair is separated by an ampersand (&).
- The query string is not part of the URL path and is not used to identify a resource.

Query strings (cont'd)

Encoding is required for query strings to be used in a URL.

- SPACES are encoded as `+` or `%20`
- Letters and numbers are not encoded
- Other characters are encoded as `%XX` where `XX` is the hexadecimal value of the character.

Examples:

```
http://www.google.com/search?q=hello+world
http://www.fyp.org/project/register?name=frankie&gender=male
http://localhost:65000/Demo/One?keyA=value1&keyB=value2
```

Query strings (cont'd)

- Query Strings can be accessed in the controller using the `HttpContext.Request.Query` property.

  `http://www.fyp.org/project/register?name=frankie&gender=male`

  `http://localhost:65000/Demo/One?keyA=value1&keyB=value2`

- Corresponding controller code

```
IQueryCollection qry = HttpContext.Request.Query;

//First Example
string firstString = qry["name"];     // "frankie"
string secondString = qry["gender"]   // "male"

// Second Example
string ka = qry["keyA"];    // "value1"
string kb = qry["keyB"];    // "value2"
```

🍿 **Lecturer Demo**

Query strings (cont'd)

Alternative methods to access query strings values:

- The `HttpContext.Request.Query` can be used to access query strings.

```
public IActionResult One()
{
    IQueryCollection query = HttpContext.Request.Query;
    string ka = query["keyA"];    // "value1"
    string kb = query["keyB"];    // "value2"
}
```

- Method **arguments** can also be used to access query strings.

```
public IActionResult One(string keyA, string keyB)
{
    string ka = keyA;   // "value1"
    string kb = keyB;   // "value2"
}
```

Tag helpers

Tag helpers

We have seen how to use some tag helpers in the previous lessons. By adding a **tag helper attribute** to a HTML tag, we can add some functionality to the tag.

So far we have covered the following tag helper attributes:

- `asp-for`
- `asp-validation-for`
- `asp-controller`
- `asp-action`
- `asp-route-id`

However, there are many more tag helpers available. You can find a list of tag helpers here.

We can also define our own custom tag helpers.

Adding custom tag helpers

- `_ViewImports.cshtml` is a file that is automatically imported into all views in the project. It is therefore an ideal place to add **custom** tag helpers.

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper Lesson12.TagHelpers.QRCodeTagHelper, Lesson12
@addTagHelper Lesson12.TagHelpers.DemoTagHelper, Lesson12
```

- We can then use the custom tag helper in a view.

```
<div>
    <qrcode text="https://www.google.com" size="200" />
</div>
```

Adding custom tag helpers (cont'd)

To create a **custom** tag helper we need to create a class that inherits from `TagHelper` and override the `Process` method.

```
namespace Lesson12.TagHelpers;

[HtmlTargetElement("bigsmall")]
public class DemoTagHelper : TagHelper
{
    public async override void Process(TagHelperContext context,
                                       TagHelperOutput output)
    {
        var content = (await output.GetChildContentAsync()).GetContent();

        string newText = "";
        for (int i = 0; i < content.Length; i++)
        {
            if (i % 2 == 0)
                newText += "<i>" + content.Substring(i, 1).ToUpper() + "</i>";
            else
                newText += "<u>" + content.Substring(i, 1).ToLower() + "</u>";
        }

        output.TagName = "b"; // Replaces <bigsmall> with <b>"
        output.Content.SetHtmlContent(newText);
    }
}
```

Adding custom tag helpers (cont'd)

Code before rendering

```
<h2>Demonstration of Custom Tag</h2>

<bigsmall>The quick brown fox jumps over the lazy dog.</bigsmall>
```

Code after rendering

```
<h2>Demonstration of Custom Tag</h2>

<b><i>T</i><u>h</u><i>E</i><u> </u><i>Q</i><u>u</u><i>I</i><u>c</u><i>K</i><u>...</u><i>D</i><u>o</u><i>G</i><u>.</u></b>
```

Rendered result

QR Codes

(Quick Response Codes)

QR Codes

- A QR code is a type of matrix barcode (or two-dimensional code) first designed in 1994 for the automotive industry in Japan.
- QR codes are used for their fast readability and comparatively large storage capacity.
- They are used for tracking inventory, as well as in mobile phone applications such as location tagging, social media, and ticketing.
- Other uses include:
  - Wi-Fi access
  - Payments
  - Tracking (Remember COVID-19 contact tracing QR codes?)

QR Codes (cont'd)

- QR codes can be generated using a QR code generator.
- QRCode is given in the TagHelpers folder.
- The target HTML element is `<qrcode>`.
- Sample usage in view `/Demo/QRCode.cshtml`

```
<h1>Demonstration of QRCode</h1>
<p>
Republic Polytechnic  
<qrcode>http://www.rp.edu.sg</qrcode>
</p>
<p>
Microsoft  
<qrcode width="100" height="100">http://www.microsoft.com</qrcode>
</p>
<p>
C236  
<qrcode width="150" height="150">http://c236dotnet.azurewebsites.net/</qrcode>
</p>
```

🍿 **Lecturer Demo**

ChartJS

ChartJS

- ChartJS is a JavaScript library that can be used to create charts.
- ChartJs is an open source project and is available on GitHub
- **Eight** chart types including: Area, Bar, Bubble, Doughnut, Line, Pie, Polar Area, Radar and Scatter.
- Excellent rendering performance
- Responsive - Redraws charts on window resize for perfect graphs on any device.
- Demonstrations on front page here: ChartJS

🍿 Lecturer Demo through ChartJS Web Site.

ChartJS (cont'd)

Additional scripts required:

- `Chart.min.js` - The core ChartJS library
- `Chartjs-plugin-datalabels.min.js` - Plugin to display data labels on the chart

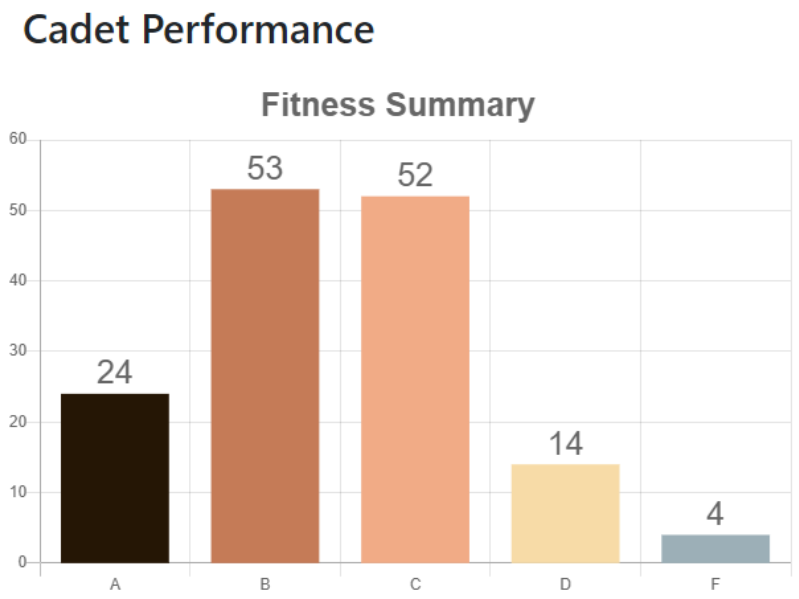
These libraries can be added directly to the `_Layout.cshtml` view or
Added in a custom scripts area, similar to `BootstrapTableScripts` we used in Lesson 12.

In the code below they are added directly to the `_Layout.cshtml` view.

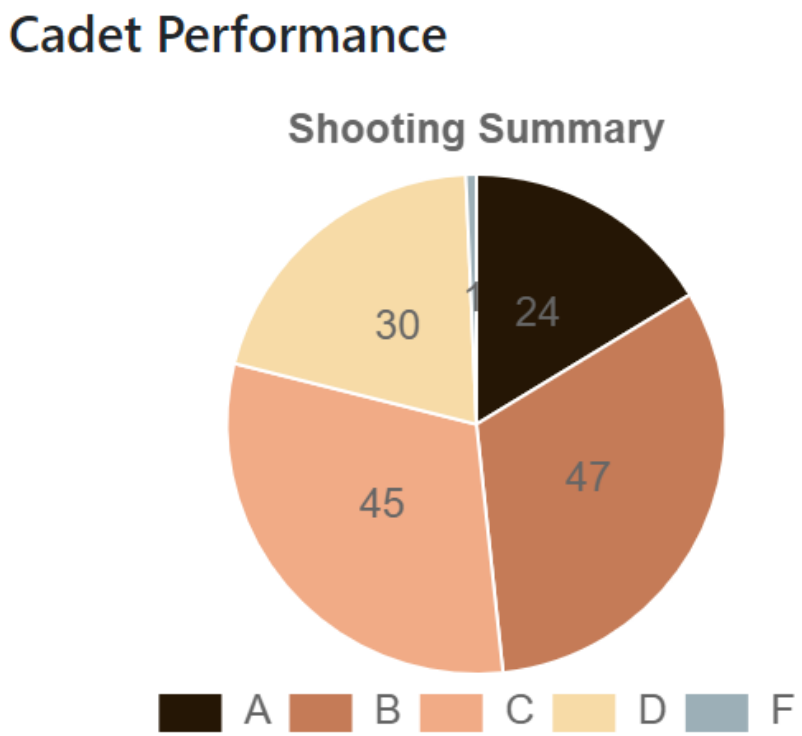
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <link href="~/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
    ...
    <script src="~/lib/chartjs/Chart.min.js"></script>
    <script src="~/lib/chartjs/chartjs-plugin-labels.min.js"></script>

    @RenderSection("BootstrapTableScripts", false)

    <title>Lesson12</title>

</head>
```

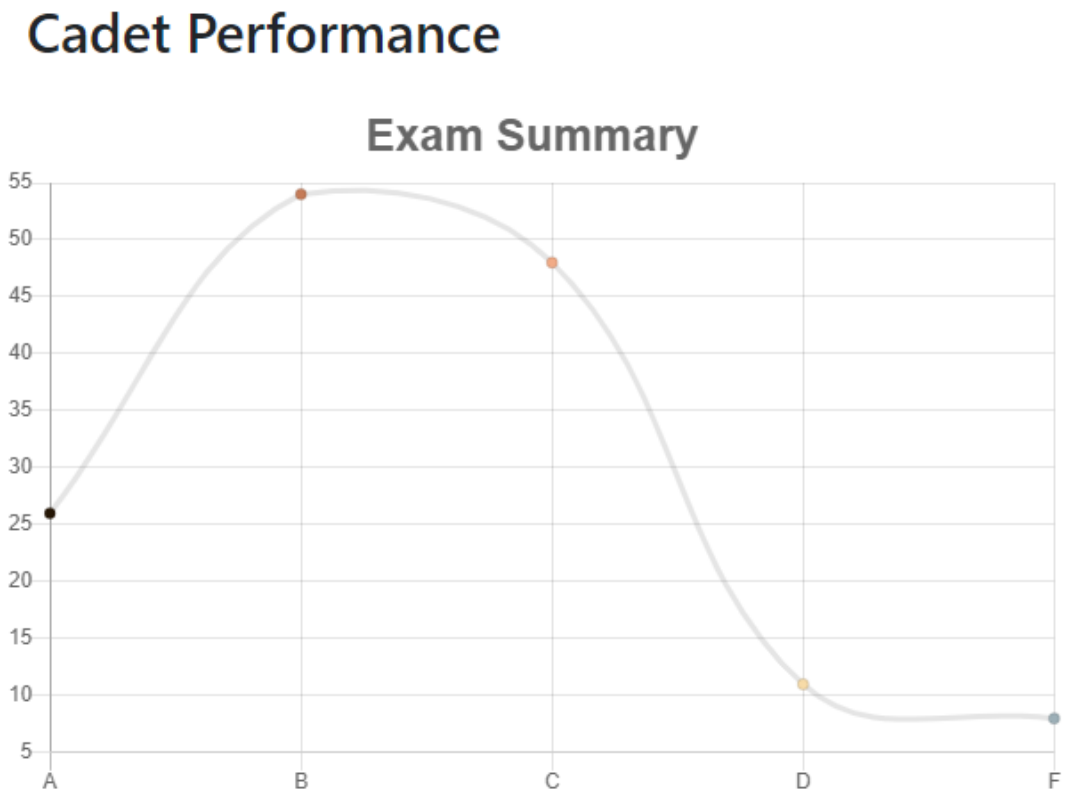
Cadet analysis

```
CREATE TABLE Cadet
(
   CNo          VARCHAR(10) PRIMARY KEY,
   ClassGrp     VARCHAR(6) NOT NULL,
   CName        VARCHAR(40) NOT NULL,
   Shooting     INT NOT NULL,
   Fitness      INT NOT NULL,
   Exam         INT NOT NULL
);
```



Cadet analysis (cont'd)





ChartController.cs

```
using Microsoft.AspNetCore.Mvc;

namespace L20.Controllers;

public class ChartController : Controller
{
    public IActionResult Bar()
    {
        PrepareData(1);
        ViewData["Chart"] = "bar";
        ViewData["Title"] = "Fitness Summary";
        ViewData["ShowLegend"] = "false";
        return View("Chart");
    }

    public IActionResult Pie()
    {
        PrepareData(0);
        ViewData["Chart"] = "pie";
        ViewData["Title"] = "Shooting Summary";
        ViewData["ShowLegend"] = "true";
        return View("Chart");
    }

    public IActionResult Line()
    {
        PrepareData(2);
        ViewData["Chart"] = "line";
        ViewData["Title"] = "Exam Summary";
        ViewData["ShowLegend"] = "false";
        return View("Chart");
    }
```

```
    private void PrepareData(int x)
    {
        int[] dataFitness = new int[] { 0, 0, 0, 0, 0 };
        int[] dataShooting = new int[] { 0, 0, 0, 0, 0 };
        int[] dataExam = new int[] { 0, 0, 0, 0, 0 };
        List<Cadet> list = DBUtl.GetList<Cadet>("SELECT * FROM Cadet");
        foreach (Cadet cdt in list)
        {
            dataFitness[CalcGrade(cdt.Fitness)]++;
            dataShooting[CalcGrade(cdt.Shooting)]++;
            dataExam[CalcGrade(cdt.Exam)]++;
        }

        string[] colors = new[] { "cyan", "lightgreen", "yellow",
            "pink", "lightgrey" };
        string[] grades = new[] { "A", "B", "C", "D", "F" };
        ViewData["Legend"] = "Cadets";
        ViewData["Colors"] = colors;
        ViewData["Labels"] = grades;
        if (x == 0)
            ViewData["Data"] = dataExam;
        else if (x == 1)
            ViewData["Data"] = dataFitness;
        else
            ViewData["Data"] = dataShooting;
    }

    private static int CalcGrade(int score)
    {
        if (score >= 80) return 0;
        else if (score >= 70) return 1;
        else if (score >= 60) return 2;
        else if (score >= 50) return 3;
        else return 4;
    }

}
```

Chart.cshtml

```
@section MoreScripts {
   <script language="javascript">

      $(document).ready(function () {
         new Chart(document.getElementById("chart"), {
            type: '@ViewData["Chart"]',
            data: {
               labels: @Json.Serialize(ViewData["Labels"]),
               datasets: [{
                  label: '@ViewData["Legend"]',
                  data: @Json.Serialize(ViewData["Data"]),
                  backgroundColor: @Json.Serialize(ViewData["Colors"]),
                  fill: false
               }]
            },
            options: {
               responsive: false,
                legend: {
                  display: @ViewData["ShowLegend"]
               },
               title: {
                  display: true,
                  text: '@ViewData["Title"]'
               },
               plugins: {
                  labels: {
                     render: 'value'
                  }
               }
            }
         });
      });

   </script>
}
<canvas id="chart" width="600" height="400"></canvas>
```

Asynchronous Javascript and XML (AJAX)

Asynchronous Javascript and XML (AJAX)

> From Wikipedia, the free encyclopedia.
>
> Ajax is a set of web development techniques that uses various web technologies on the client-side to create asynchronous web applications. With Ajax, web applications can send and retrieve data from a server **asynchronously** (in the background) without interfering with the display and behaviour of the existing page. By decoupling the data interchange layer from the presentation layer, Ajax allows web pages and, by extension, web applications, to change content dynamically without the need to reload the entire page. In practice, modern implementations commonly utilize JSON instead of XML.
>
> Source: Link to article

AJAX (cont'd)

Traditional HTTP model

- Every request results in loading an entire new page
- Dynamic pages are created on the server-side
- Data formatting and styling are set on the server

AJAX model

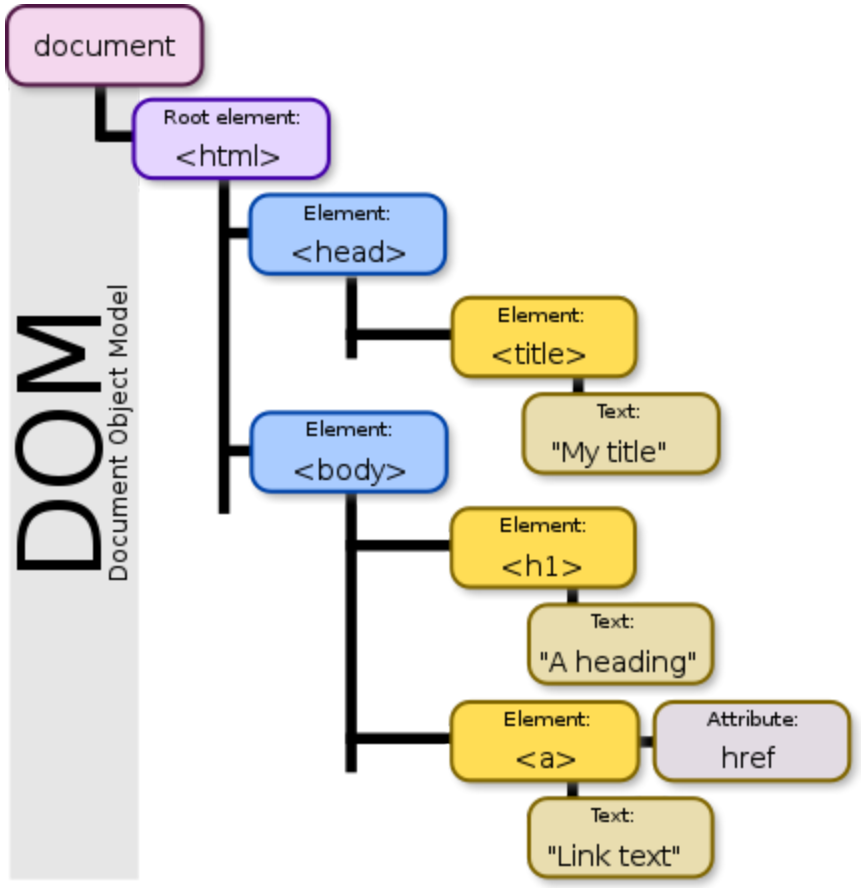
- The result of an AJAX call may only change a small part of an existing HTML page being refreshed (**Partial-page refresh**)
- Server creates the page structure, JavaScript code performs content creation
- Data formatting and styling is done via JavaScript, HTML5/CSS.

Document Object Model (DOM)

> From Wikipedia, the free encyclopedia.
>
> The Document Object Model (DOM) is a cross-platform and language-independent interface that treats an XML or HTML document as a **tree structure** wherein each node is an object representing a part of the document. The DOM represents a document with a logical tree. Each branch of the tree ends in a node, and each node contains objects. DOM methods allow programmatic access to the tree; with them one can change the structure, style or content of a document. Nodes can have event handlers attached to them. Once an event is triggered, the event handlers get executed.

AJAX and the DOM

AJAX uses the DOM to dynamically update a HTML page, making it interactive.



End of Lesson 12

Back to Lesson 12