

# WEB524

WEB PROGRAMMING ON WINDOWS

## **WEEK 5 - LECTURE 1**

ITEM SELECTION AND BOOTSTRAP

# Previously

- You have learned:
  - Scaffolding a view - *the scaffolder provides an easy way to generate view code for common scenarios (list, details, create, edit, and delete)*
  - Hand-editing view code – *you will modify simple HTML in A2 to render an HTML Table for invoice line items.*
  - HTML Helpers – you will recall that HTML Helpers enable the Razor view engine to render an HTML element for the browser.
  - Associated entities – last week we discussed displaying of associated entities.

# Today

- We will explore the foundation required to implement *add* and *edit* patterns for associated entities.
  - Review
  - Item-selection elements for HTML Forms
  - Bootstrap introduction
  - SelectList object
  - HTML Helpers for dropdown and listbox

# Display-only views

- The **scaffolder** does a reasonably good job at rendering **display-only views**.
  - It renders a collection in an HTML Table and an object in an HTML Description List.
  - The data items are rendered in these elements and/or often in <div> or <span> elements.
  - In the first number of code examples, we deliberately constrained the scaffolder's activities, by working with a non-associated data entity (object or collection) on each view.
- If you have re-created last week's AssocOneToMany code example or worked on Assignment 2, you will notice that the scaffolder does NOT render associated objects or collections.
- When working with associated data, we often need to use an **item-selection element** (control).

# Data modification views

- The scaffolder does a reasonably good job at rendering views for the add-new, edit-existing, and delete-item use cases.
- As you have noticed, it renders input fields (for many kinds of data) and buttons.
- But, the scaffolder is limited in what it can render. For example, **the scaffolder will not render an item-selection element** (e.g. a dropdown list, or a checkbox group).

# HTML Form elements that are scaffolded

HTML Form element	Scaffolded?
Label	Yes
Text box	Yes
Multiline text area	Yes
Button	Yes
Hidden	Yes
Dropdown list (select)	No
Listbox (select, multiple)	No
Radio button group	No
Checkbox group	No

# Item-selection elements for HTML Forms

Size

Dropdown list  
<select> and <option>

Size   
Medium  
Large

Dropdown list  
<select> and <option>  
size=3 attribute

Size   
Medium  
Large

Listbox  
<select> and <option>  
size=4 attribute  
multiple attribute

# More item-selection elements

**Size**

☒ Small

☐ Medium

☐ Large

Radio button group  
`<input type=radio...`

**Size**

☐ Small

☒ Medium

☒ Large

Checkbox group  
`<input type=checkbox...`



# Problem?

- The scaffolder will not create all HTML form elements. For example, we may want to add an item-selection element to an HTML Form, especially when we are modifying associated entities. How can we work around this?
- A solution to this is **hand-coding**. *Probably not what you wanted to hear!*
  - Use good design and coding techniques.
  - Use custom HTML Helpers.
  - Use hand-written HTML.

# Hand-editing item-selection elements

- The strategy combines the use of the scaffolder with hand-coding:
  1. Continue to use the scaffolder to generate source code for views that will work with associated data.
  2. Plan on hand-editing the code, to add and configure the item-selection elements that you need.

# Generating data for the item-selection element

- All item-selection elements need data
  - For example, how do we populate the items in a select element.
- For very simple scenarios, it can be hard-coded and therefore static in the view
- Most often the data is passed into the view (from the controller)
  - The passed-in data will be, or have, a collection that can be used to render the items.
  - In the controller, we can create a simple collection of strings, for example a `List<string>`, and pass it to the view.
  - Use the collection to render the items (in a for or foreach loop).

# Passing data to the view

- Best practice is to create another view model class that has a property for the collection.
- In Assignment 1, you learned how to create a view model class named “EmployeeEditFormViewModel” which had the data needed on the HTML Form.
- Continue using this method for scenarios in which we need to send data to the HTML Form.

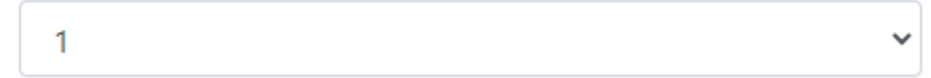
# Remember

- We create and use two view model classes for “add new” and “edit existing” use cases.
- For example, if we are coding the “add new” use case, they are:
  - **EmployeeAddFormViewModel** - used to package and send data to the HTML Form in the browser.
  - **EmployeeAddViewModel** - used to describe the shape of the data submitted by the browser.

# HTML Form elements and the “name” attribute

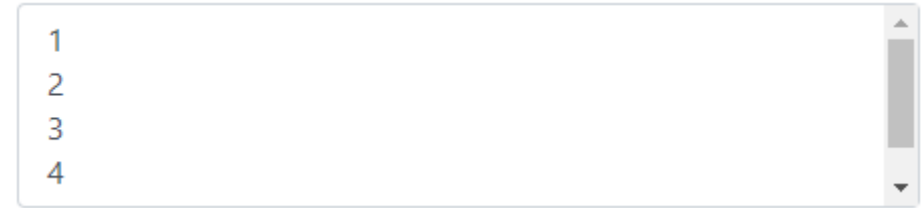
- The value of the “name” attribute of an HTML Form element is used to identify or “name” the data that is submitted (posted) by the browser. For example:
  - An `<input name=“Username” />` element will render a textbox.
  - When the form is submitted, your controller code has access to the data by using the “Username” property/variable.
  - The same idea applies to item-selection elements.
- You may open the HTMLFormFoundations code example while reviewing the following scenarios.

# Dropdown list



- This is rendered by a `<select>` element with `<option>` descendants.
- It permits the selection of one item.
- It can be rendered with a **DropDownList** HTML Helper, explained later.
- Each `<select>` item is configured with a “name” attribute and value. When the HTML Form is submitted, the controller code can access the data (described next) by using the variable name in the attribute’s value.
- Each `<option>` item is configured with a non-visible “value” attribute and text. The value is intended to be the item’s unique identifier. When the HTML Form is submitted, the controller code can use the data in the attribute’s value.

# Listbox

A screenshot of a web browser showing a listbox. The listbox is a rectangular box with a light gray border. Inside, the numbers 1, 2, 3, and 4 are listed vertically. The number 1 is at the top, followed by 2, 3, and 4. To the right of the listbox is a vertical scrollbar with a gray track and a white slider.

- Also rendered by a `<select>` element with `<option>` descendants.
- Adding a “size” attribute to the `<select>` element extends the vertical height of the list to include the specified number of items. Additional items are available by scrolling on the screen.
- It is not necessary for the user to select an item however if they do, they may select at most one item.
  - It can be rendered with a **DropDownList** HTML Helper.
- If you wish to allow the selection of multiple items (using the Ctrl or Command keys) then add a “multiple” attribute to the `<select>` element.
  - It can be rendered with a **ListBox** HTML Helper, explained later.
- The “name” and “value” attributes are configured in a manner similar to the dropdown list.



# Radio button group (aka “radio group”)

- ☒ Default radio
- ☐ Second default radio
- ☐ Disabled radio

- This is rendered by multiple `<input type="radio">` elements. It permits the selection of one item in the group.
- The “name” and “value” attributes are configured in a similar manner as listbox and dropdown items.
- A best practice is to wrap each in a `<label>` element.
- It can be rendered with a **RadioButton** HTML Helper.

# Checkbox group (aka “checkboxbo

☐ Default checkbox  
☐ Disabled checkbox

- This is rendered by multiple `<input type=“checkbox”>` elements. It permits the selection of multiple items in the group.
- The “name” and “value” attributes are configured in a manner similar to the elements above.
- A best practice is to wrap each in a `<label>` element.
- It can be rendered with a **CheckBox** HTML Helper.

# Submitting data from the browser

- Data entered on an HTML Form will enter a controller method when submitted (*posted back*) by the user.
- In PHP, you know that the data was available in the `$_POST` global predefined variable.
- In ASP.NET MVC, although the data was available in a **FormCollection** object (and therefore in a format similar to the PHP associative array), we almost always use strong typing by specifying a view model class as the parameter in the controller method.

# Submitting simple HTML elements

- For HTML Form elements, the value of the “name” attribute **MUST match a property name in the view model class.**
- When submitted, the body of the **POST request** will include the data as an **encoded name-value pair.**
- For example, an `<input name=“Email” />` textbox, in which the user enters “peter@example.com”, will submit **“Email=peter@example.com”.**
- **The ASP.NET MVC runtime will attempt to convert non-string data into usable values automatically** (e.g. int, double, DateTime).

# Submitting an item-selection element

- For a **single-select** scenario:
  - As you would expect, the “name” attribute MUST match a property name in the view model class. The property type could be **string** or **int** but an int is more typical.
- For **multiple-select** scenario:
  - Same approach as above but the view model class property type MUST be a collection with the appropriate data type. For view model you could use **IEnumerable<int>** initialized as **List<int>** in the constructor.

# Learning from the code example

- If you run the HTMLFormFoundations code example you will notice that the item-selection elements were coded with:
  - hand-coded statically-defined data
  - data in a simple collection of (for example) strings
  - data in a collection of objects that had more complexity than strings
- You will also notice that the item-selection elements were rendered in the browser without 'pretty' styling.
- These HTML Forms do not use the same styling that we have seen on other scaffolded forms. The spacing is wrong and alignment is off.

## Simple Input

This page has simple input fields.

User name:

Age:

Password:

## Dropdown Listbox Object

This page has item-selection controls.

A collection of objects will be used to render the items.

Size:

Teacher:

Teacher (single item select):

Teachers (multi item select):



## Radio Check Simple

This page has radios and checkboxes.

A collection of strings will be used to render the items.

Size:

☐ Small ☐ Medium ☐ Large

Colour:

☐ Red ☐ Green ☐ Blue

Colours:

☐ Red ☐ Green ☐ Blue

Login

# Introduction to Bootstrap

- Bootstrap is a popular framework that helps us develop responsive and mobile-compatible web apps.
- It was created by Mark Otto and Jacob Thornton (of Twitter).
- It is an open source project.
- You can read more about Bootstrap in this [Wikipedia](#) article.
- All ASP.NET MVC web apps include the Bootstrap framework by default.
- The current stable release is 4.x however MVC ships with 3.x. There are many breaking changes and therefore you are asked not to update the NuGet package on assignments.

# Bootstrap is a Framework

- At its core, it is **a set of CSS rules**. Most are implemented as element and class selectors.
- It enables us to have visual consistency on web pages. It's Bootstrap that affects the appearance of our web app.
- The black-background navigation menu, layout characteristics, comfortable spacing, nice clear buttons, rounded-corner text input fields on HTML Forms, and responsiveness for different screen sizes – all come from Bootstrap.

# How to use Bootstrap

- After learning a small set of “need-to-know” topics, use Bootstrap by adding class names to the “class” attribute of HTML elements.
  - Grid system
  - HTML Forms
  - Tables
  - Buttons
- Organize content by using <div> elements.
- We’ll look at each need-to-know item.

# Grid system

- “Bootstrap includes a responsive, mobile first fluid grid system that appropriately scales up to 12 columns as the device or viewport size increases. It includes predefined classes for easy layout option”.
- Open the “Grid system” [documentation](#), and read/skim these sub-sections:
  - Introduction
  - Grid options
  - Example: Mobile and desktop
- The **\_Layout.cshtml** source code file includes a `<div class="container">` element that encloses the grid system.
- We can use classes to size and position content and HTML Forms.

# HTML Forms

- “Individual form controls automatically receive some global styling. Wrap labels and controls in **.form-group** for optimum spacing.”
- Add the **.form-control** class when necessary
- Open the “Forms” [documentation](#), and read/skim these sub-sections:
  - Basic example
  - Horizontal form
  - Supported controls
  - Control sizing

# Horizontal HTML Forms

- We use horizontal forms frequently in this course. The important take-aways from the documentation are:
  - Always use a `<label>` with an HTML Form element/control.
  - Package the label and control inside a `<div class="form-group">` element.
- By following this rule, we get visual consistency and nice spacing and appearance.
- Look at the result in the next two images, from another code example, **SelectListIntro**.

## Add Vehicle

Complete the form, and then click the Create button

Model	<input type="text"/>
Trim	<input type="text"/>
ModelYear	<input type="text" value="2016"/>
ClassificationList	<input type="text" value="Passenger car: compact"/>
ManufacturerList	<div>Ford Motor Company Toyota Motor Company Volkswagen AG</div>
<input type="button" value="Create"/>	

[Back to List](#)



## Plan Courses

Complete the form, and then click the Create button

Student name

Academic term

☒ 2016 - Summer

☐ 2016 - Fall

☐ 2017 - Winter

Desired course(s)

☐ BTR490 - Research Internship

☐ BTP500 - Data Structures

☐ BTB520 - Canadian Business Environment

☐ BTS530 - Project Planning

☐ BTH540 - User Interface Design

☐ BTP600 - Design Patterns

☐ BTE620 - Ethics, Law, Professionalism

☐ BTS630 - Project Implementation

☐ BTC640 - Multimedia Design

Create

[Back to List](#)

# Mobile Viewports

- Earlier, it was noted that Bootstrap makes your web app mobile-friendly.
- The following image shows the “Add Vehicle” page in a narrow viewport that’s typical of a hand-held smartphone.
- Notice that the navigation menu is collapsed to the widely-used “hamburger” icon.
- Notice that the HTML Form element labels are above the element and the HTML Form elements themselves are left-aligned.

## Add Vehicle

Complete the form, and then click the Create button

Model

Trim

ModelYear

ClassificationList

ManufacturerList

Create

[Back to List](#)

# Tables

- It is easy to style a table with Bootstrap classes.
- Open the “Tables” [documentation](#), and read/skim these sub-sections:
  - Basic example
  - Striped rows

# Buttons

- We can use the button classes on an `<a>`, `<button>` or `<input>` element.
- Open the “Buttons” [documentation](#), and read/skim these sub-sections:
  - Button tags
  - Options
  - Sizes

# How do I learn more?

- Don't get lost in the Bootstrap framework documentation. Start simple and in the future dig deeper when you need to.
- It is safe to read/skim the Bootstrap CSS [documentation](#) because you will get the most value from it during the middle part of the course.
- It is safe to read/skim the Getting started [documentation](#) however Bootstrap is already installed in all ASP.NET MVC projects so you won't have to do any of the install/configure tasks that are described there.
- Later in the course, we'll touch on some of the Bootstrap components and some of its JavaScript helpers and add-ins. We will also look at themes, to change the overall look of your web app.

# Checkpoint

- You have learned of some design and coding techniques for working with item-selection elements in ASP.NET MVC web apps:
- 1. We use a “...Form” view model class to package the data that will be needed in the item-selection element.
- 2. We must hand-edit views, to add item-selection elements.
- 3. Each item-selection element must have a <label> element. Both are packaged inside a <div class=form-group> element. (Radio and checkbox elements also need their own <label> element.)
- 4. A separate class is then used to describe the shape of the data submitted by the browser user. A single-select value will be in an int property. Multiple-select values will be in an int collection property (e.g. List<int>).
- Now, we will learn about **SelectList** objects so that we can make progress on 1, 2, and 3.

# SelectList Introduction

- As you are learning about the **SelectList** object, open the **SelectListIntro** code example.
- **SelectList** is a class for packaging the data needed by an item-selection element and works with the typical HTML elements (dropdown, listbox, radios, checkboxes).
- We create a SelectList object in a controller method.
- We assign the new SelectList object to a property in a view model object that gets passed to a view.
- Best practice is to name a SelectList property in the view model class using “List” as the suffix. For example, “ManufacturerList”

```
// SelectList for Manufacturer  
public SelectList ManufacturerList { get; set; }
```



# SelectList Sample Output

```
<select name="ManufacturerId">  
  <option value="1">Ford Motor Company</option>  
  <option value="2">Toyota Motor Company</option>  
  <option value="3">Volkswagen AG</option>  
</select>
```

# SelectList Data Requirements

- After reviewing the HTML on the previous slide, we see that we will need:
  - A value for the “name” attribute, such as **ManufacturerId**, which will be used as the name in the name-value pair that is submitted by the browser.
  - Values for each item’s “value” attribute, which will be used as the value in the name-value pair that is submitted by the browser. For example, “2”.
  - Visible text that will be displayed to the user. For example, “Volkswagen AG”.

# SelectList structure

- A SelectList class is available in two forms:
  - SelectList, intended for single-select use cases
  - MultiSelectList, intended for multiple-select use cases
- When you create a SelectList object, its variable or property name becomes the “name” attribute in the HTML Form item-selection element.
- The most-commonly-used constructor is used with a collection of objects. For example, a collection of Products or a collection of Employees.

# SelectList Common Constructor

- The constructor has these three parameters:
  1. **IEnumerable items** – A collection, which will be used for the items in the item-selection element. Most often, the collection will be fetched by a call to a manager method (which returns a collection).
  2. **string dataValueField** – The name of the property of each object in the collection that has the data to be used for the value of the “value” attribute in the item-selection element. Many times, this will be the object’s unique identifier (e.g. “Id” or “CustomerId”) and therefore, will be an int.
  3. **string dataTextField** – The name of the property of each object in the collection that has the data which will be used for the visible text in the item-selection element. Many times this will refer to a name or other descriptive information.

The SelectList object is passed along with the other properties in the view model class to the view.

# SelectList Less Common Constructor

- A less-commonly-used constructor is one that takes a collection of simple strings or ints. For example, a string collection of sizes (“Small”, “Medium”, “Large”) or birth years (1995, 1996, 1997...)
- The constructor only needs one parameter, an IEnumerable of items.
- The Razor view engine will not render a “value” attribute in the HTML document instead it renders each item in the collection as visible text.
- HTML Form standards will then cause the selected item’s visible text to be submitted by the browser.

# Using a SelectList object in a view

- We can loop through the collection of items in the SelectList object using **for** or **foreach** and render the items for any kind of item-selection element.
- Following are two examples of rendering the single-selection elements by hand.

# Drop-Down List Example

```
<div class="form-group">
  <label for="ManufacturerList" class="control-label col-md-2">
    Manufacturer List
  </label>
  <div class="col-md-10">
    <select name="ManufacturerId" class="form-control">
      @foreach (var item in Model.ManufacturerList)
      {
        <option value="@item.Value">@item.Text</option>
      }
    </select>

    @Html.ValidationMessageFor(model => model.ManufacturerList, "",
      new { @class = "text-danger" })
  </div>
</div>
```

# Radio Buttons Example

```
<div class="form-group">
  <label for="ManufacturerList" class="control-label col-md-2">Manufacturer List</label>
  <div class="col-md-10">
    @foreach (var item in Model.ManufacturerList)
    {
      <div class="radio">
        <label>
          <input type="radio" name="ManufacturerId" value="@item.Value" />@item.Text
        </label>
      </div>
    }
    @Html.ValidationMessageFor(model => model.ManufacturerList, "",
      new { @class = "text-danger" })
  </div>
</div>
```



# Naming the “name” Attribute

- In the HTML Form in the view, the value of the “name” attribute in the item-selection element MUST match the name of a property in the view model class that describes the data submitted by the user.
- For example, in a VehicleAdd view model class, assume that there is an int property named “ManufacturerId”. This MUST match the value used for the “name” attribute.
- For single items, use the singular word form, like “ManufacturerId”.
- For multiple items (multiple select listbox or checkbox group), use the plural word form, like “CourseIds”.

# HTML Helpers for item-selection elements

- The textbook has good coverage of this topic in chapter 5, “Forms and HTML Helpers”.
- By now you have already seen how to use HTML Helpers when coding views.
- HTML helpers are code expressions that begin with “@Html” and are intended to make some of the HTML easier to write.
- HTML helpers offer many benefits:
  - Link (URI) resolution at runtime.
  - Less code required to render some HTML Forms elements.
  - Model binding.
  - User interface support for interaction and error message display.

# Simple helpers

- ActionLink – i.e. `HTML.ActionLink()`

# Helpers used with view model data

- Helpers offer more benefits when used with view model data (specifically model binding and user interface support).
- You can use these helpers:
  - Textbox
  - Password
  - Label
  - TextArea
  - DropDownList
  - Listbox

# Helpers for item-selection elements

- If you want to render a dropdown list or a listbox (single or multiple selection), you can use an HTML Helper.
- The ASP.NET MVC class library does not (yet) have HTML Helpers for a radio button group or a checkbox group.
  - We can create our own custom HTML Helper to support groups of radio or checkboxes however this is an advanced topic beyond the scope of this course.
  - There may be NuGet packages that someone else has created.
- Let's rework our previous examples using HTML helpers instead.

# Drop-Down List (HTML Helper)

```
<div class="form-group">
    @Html.LabelFor(model => model.ManufacturerList,
        htmlAttributes: new { @class = "control-label col-md-2" })

    <div class="col-md-10">
        @Html.DropDownList("ManufacturerId", Model.ManufacturerList,
            htmlAttributes: new { @class = "form-control" })

        @Html.ValidationMessageFor(model => model.ManufacturerList, "",
            new { @class = "text-danger" })
    </div>
</div>
```

# Multi-Line Single/Multi-Selection

- If you want to display a multi-line single-selection listbox, the DropDownList HTML Helper will still work.
  - Simply add a “size” attribute to the method call.
- If you want to display a multi-line multiple-selection listbox, the ListBox HTML Helper will still work.
  - It is a best practice to include a “size” attribute.
  - An example follows.

# ListBox (HTML Helper)

```
<div class="form-group">
    @Html.LabelFor(model => model.CourseList,
        htmlAttributes: new { @class = "control-label col-md-2" })

    <div class="col-md-10">
        @Html.ListBox("CourseIds", Model.CourseList,
            htmlAttributes: new { @class = "form-control col-md-2", size = 10 })

        @Html.ValidationMessageFor(model => model.CourseList, "",
            new { @class = "text-danger" })
    </div>
</div>
```



# The big picture and workflow

- Posted with this lecture is a workflow diagram that attempts to capture today's topic by showing the assets involved in rendering an item-selection element on an HTML Form in the browser.
- See the image: "Working with an HTML Forms Item-Select Element"