## Programmatic Logic

ViewStart.cshtml — It is used to specify common settings for all the views under a folder and sub-folders where it is created.

The Layout.html Page.

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8" />
        <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
        <title>@ViewBag.Title - My ASP.NET
Application</title>
        @Styles.Render("~/Content/css")

@Scripts.Render("~/bundles/modernizr")
//Razor – ASP.NET view engine that lets you embed server-
based code (C#) into web pages. All CSS files in the Content
folder are rendered.
    </head>
    <body>
        <div class="navbar navbar-inverse
navbar-fixed-top">
            <div class="container">
                <div class="navbar-header">
                    <button type="button"
class="navbar-toggle" data-toggle="collapse"
data-target=".navbar-collapse">
                        <span class="icon-
bar"></span>
                        <span class="icon-
bar"></span>
                        <span class="icon-
bar"></span>
                    </button>

@Html.ActionLink("Application name",
"Index", "Home", new { area = "" }, new {
@class = "navbar-brand" })
                </div>
                <div class="navbar-collapse
collapse">
                    <ul class="nav navbar-
nav">

<li>@Html.ActionLink("Home", "Index",
"Home")</li>

<li>@Html.ActionLink("About", "About",
"Home")</li>

<li>@Html.ActionLink("Contact", "Contact",
"Home")</li>
                    </ul>
                </div>
            </div>
        </div>
        <div class="container body-content">
            @RenderBody()
            <hr />
            <footer>
                <p>&copy; @DateTime.Now.Year
- My ASP.NET Application</p>
            </footer>
        </div>

        @Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
        @RenderSection("scripts", required:
false)
    </body>
```

Pseudo classes:A pseudo-class is used to define a special state of an element. For example, it can be used to:

Forms in View:

```
@*First version of the creation of a
Form where singe items are sent to
method/action called form 1 in the
HomeControler - in this version no
model is used.*@
<form action="form1" method="post">

        <label for="fname">First
name:</label><br>
        <input type="text" id="fname"
name="fname" value="John"><br>

        <label for="lname">Last
name:</label><br>
        <input type="text" id="lname"
name="lname" value="Doe"><br><br>

        <label
for="age">Age:</label><br>
        <input type="number" id="age"
name="age"><br><br>

        <label for="isAlive">Is
Alive:</label><br>
        <input type="checkbox"
name="isAlive" /><br >

           <input type="submit"
value="Submit Form" />

    </form>
```

**In the HomeController…**

form1 Action method, 1) receives these parameters, (2) assigns it to ViewBag objects (3) returns Contact View to (4) display these values.

The ViewBag in ASP.NET MVC is used to transfer temporary data (which is not included in the model) from the controller to the view.

```
    //Action called form1 accepts parameters
and assign it to ViewBag objects to be
dispayed in the Contact page
    [HttpPost]
    public ActionResult form1(string fName,
string lName, int age, string isAlive)
    {

        ViewBag.FirstName = fName;
        ViewBag.LastName = lName;
```

```
        ViewBag.Age = age;

        if (isAlive != null)

            ViewBag.IsAlive = "Alive";
        else
            ViewBag.IsAlive = "Not Alive";

        return View("Contact");
    }

@*For the display of the entered
values*@
<h4 style="color:purple">
    <b>First Name:</b>
@ViewBag.FirstName <br />
    <b>Last Name:</b>
@ViewBag.LastName <br />
    <b>Age:</b> @ViewBag.Age <br />
    <b>Is Alive:</b> @ViewBag.IsAlive
</h4>
```

My Understanding: The form of Contact page(for example) is created using scaffolding from the home controller Using the Model that's where the above code goes in.

HTML helper: the class renders HTML controls in the razor view.it binds the model object to the html controls to display the value of the model properties into those controls and also assign the the value of the controls to the model properties while submitting a web form.

```
@*second version of the creation of a
Form where an object is sent to
method/action called form2 in the
HomeControler. HtmlHelpers are used
in this example*@
@model
INF272Lecture4v1.Models.PersonModel
namespace INF272Lecture4v1.Models
{
    public class PersonModel
    {

        [Display(Name = "First
Name")]
        public string FirstName {
get; set; }

        [Display(Name = "Last Name")]
        public string LastName { get;
set; }

        [Display(Name = "Current
Age")]
        public int Age { get; set; }
= 0;

        [Display(Name = "Living
Status")]
        public bool IsAlive { get;
set; } = true;
    }
}
```

CSS REMINDER:

• **External** • Style rules are placed in a separate file with .css extension • HTML links to the style sheet by nesting a link tag (e.g. ) between the head tags.

• **Internal/Document** • style rules are placed within the head element of the HTML document between the tags that is nested in the … section.

• **Inline** • apply the style directly to the HTML element. The style rule becomes the value of the style attribute of the HTML element.

That css code for Prac 3:

```
@media screen and (max-width:700px){
    .carousel{
        display:none;
    }
}
```

Table in MVC using html

```
@model
List<Practical2_u23642425.Models.Pers
onModel>
@{
    ViewBag.Title = "Index";
}

<!--Add Container-->
<div class="container">
    <!--Row 1-->
    <div class="row">
        <div class="col-4 text-
start">
            <h2>List of People</h2>
        </div>
    </div>
    <!--Row 2-->
    <div class="row">
        <div class="col-12">
            <!--Add our Table-->
            <table class=" table
table-active">
                <thead>
                    <tr>
                        <th>Student
Number</th>
```

```
                        <th>First
Name</th>
                        <th>Last
Name</th>
                        <th>Email
Address</th>
                        <th>Link to
personal page</th>
                    </tr>
                </thead>
                <tbody>
                    <!--Checking  for
null list-->
                    @if (Model.Count
> 0)
                    {

                        foreach (var
person in Model) // loop throuygh
data object in Model list
                        {
                            <tr>

<td>@person.StudentNumber</td>

<td>@person.FirstName</td>

<td>@person.LastName</td>

<td>@person.EmailAddress</td>
                                <td>

<button class="btn btn-success"
onclick="window.location.href
='@Url.Content(person.MyLink)'">

LINK

</button>
```

## JavaScript Within MVC

• **JavaScript (abbreviated as JS)** is a cross-platform, object-oriented scripting language.

• Inside a host environment (for example, a web browser), JavaScript can be connected to the objects of its environment to provide programmatic control over them

. • Please note that JavaScript is not a subset of Java, it is a substantially different language. • JavaScript is interpreted language, while Java is compiled. • JavaScript is dynamically typed, while Java is statically typed.

• JS code can be placed between <script> </script> tags directly in the html. Place reference in HTML: <script src= "myscript.js"></script>• External scripts can be referenced with a full URL or with a path relative to the current web page. • You can place an external script reference in or

as you like. • The script will behave as if it was located exactly where the <script> tag is located .

Output • JavaScript can "display" data in different ways:

• Writing into an HTML element, using innerHTML.

Document.getElementById("demo").innerHTML =5+6;

• Writing into the HTML output using document.write().

Document.write(5+6);

• Writing into an alert box, using window.alert().

Window.alert(5+6);

• Writing into the browser console, using console.log().

Console.log(5+6);

### HTML Document Object Model (DOM)

• The HTML DOM is a standard object model and programming interface for HTML. • With the HTML DOM, JavaScript can access and change all the elements of an HTML document. (as well as their attributes, styles, and events) • When a web page is loaded, the browser creates a Document Object Model of the page.

HTML DOM methods are actions you can perform (on HTML Elements), for example:

• getElementById: Find an element by element id.

• getElementsByTagName: Find elements by tag name.

• getElementsByClassName:Find elements by class name.

• getElementsByName:Find elements by name.

• HTML DOM properties are **values** (of HTML Elements) that you can set or change, for example

• element.innerHTML: The contained HTML of a specified element.

• element.style: The style object (that contains a number of properties) of an HTML element.

### *Arrays – creating arrays*

• JavaScript arrays are used to store multiple values in a single variable

• Using an array literal is the easiest way to create a JavaScript Array:

var array_name = [item1, item2, ...]; or

var names = [“Batman”, ”Superman” , “Wonder Woman”];
• The following example a new empty array: var array_name = []; • var cars = [];

## JavaScript in MVC

• Internal placement is usually in an HTML or .cshtml page
You can simply add a <script>…</script> section but rather add a section named
“scripts” using @section directive.

```
@section scripts {
<script type="text/javascript">
//Add JavaScript code and razor code
</script>
}
```

## Prac4 Java code

Always remember that in the Home controller we make our list so we can use of it in the view.

```
public class PeopleController :
Controller
 {
      // GET: People
      public static
List<Models.PersonModel> people = new
List<Models.PersonModel>();
      public ActionResult ListPeople()
      {

          return View(people);
      }

      //There must be two action
result one for get which doesn't
expect data
      [HttpGet]

      public ActionResult Create()
      {
          return View();
      }
//This is the type of action result
that expects data.
      [HttpPost]
      public ActionResult
Create(Models.PersonModel pm)
      {
          people.Add(new
Models.PersonModel { FirstName =
pm.FirstName, LastName = pm.LastName,
Email = pm.Email,  StuNumber =
pm.StuNumber });
          return
RedirectToAction("ListPeople");

      }
 }
```

## Create.cshtml page

```
@model S1P04.Models.PersonModel

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>PersonModel</h4>
        <hr />
        @Html.ValidationSummary(true,
"", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model =>
model.StuNumber, htmlAttributes: new
{ @class = "control-label col-md-2"
})
            <div class="col-md-10">
                @Html.EditorFor(model
=> model.StuNumber, new {
htmlAttributes = new { @class =
"form-control" } })

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

        <div class="form-group">
            @Html.LabelFor(model =>
model.FirstName, htmlAttributes: new
{ @class = "control-label col-md-2"
})
            <div class="col-md-10">
                @Html.EditorFor(model
=> model.FirstName, new {
htmlAttributes = new { @class =
"form-control" } })

@Html.ValidationMessageFor(model =>
model.FirstName, "", new { @class =
"text-danger" })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-
offset-2 col-md-10">
                <input type="submit"
value="Create" class="btn btn-
default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List",
"ListPeople")
</div>
```

**Table Code but this time is filled by the create button.**

```
<p>
    Search string: <input type="text"
id="search" size="20" name="search">
    <button
onclick="ST()">Submit</button>
</p>

<table class="table table-bordered">

    <tr style="background-color:
darkgreen; color: white; ">
        <th width="16%">

@Html.DisplayNameFor(model =>
model.StuNumber)
        </th>
        <th width="16%">

@Html.DisplayNameFor(model =>
model.FirstName)
        </th>
        <th width="16%">

@Html.DisplayNameFor(model =>
model.LastName)
        </th>
        <th class="text-center"
width="16%">

@Html.DisplayNameFor(model =>
model.Email)
        </th>

        <th class="text-center"
width="16%">
            Delete
        </th>
    </tr>

    <tbody id="info">
        @foreach (var item in Model)
//For each person
        {
            <tr class="rowcontent">
                <td width="16%">

@Html.DisplayFor(modelItem =>
item.StuNumber)
                </td>
                <td width="16%">

@Html.DisplayFor(modelItem =>
item.FirstName)
                </td>
                <td width="16%">

@Html.DisplayFor(modelItem =>
item.LastName)
                </td>
                <td class="text-
center" width="16%">

@Html.DisplayFor(modelItem =>
item.Email)
                </td>
                @*<td class="text-
center" width="20%">

@Html.DisplayFor(modelItem =>
item.myLink)
                    </td>*@

                <td class="text-
center" width="16%">
                    <button
type="button" class="btn btn-success
btn-sm"
onclick="delete_row(this)">Delete</bu
tton>
                </td>
            </tr>

        }
    </tbody>
</table>
```

Local Storage CODE:

```
<p>
    <button onclick="Save()">Save to
local storage</button>
    <button
onclick="Retrieve()">Retrieve from
local storage</button>
    <button onclick="Clear()">Clear
local storage</button>
</p>

@section scripts {
    <script type="text/javascript">


        var tableBody =
document.getElementById("info");
        //Store the string in
localStorage using the setItem()
method - first stringify the rows
        //-------------------------
---------------------
        function Save() {

            var tableBody =
document.getElementById("info");
            console.log(tableBody);
            const tableData =
JSON.stringify(tableBody.innerHTML);

localStorage.setItem("tableData",
tableData);

        }
//---------------------------------
-------------without JSON
```

```
 //var tableData = "";

 //for (var i = 0, row; row =
tableBody.rows[i]; i++) {
 //    for (var j = 0, col; col =
row.cells[j]; j++) {
 //        tableData += col.innerHTML
+ "|";

 //    }
 //    tableData += ",";

 //}

        //clearing local storage
        function Clear() {
            localStorage.clear();
        }

         //-------------------------
-----------------
        function Retrieve() {
            //        Retrieve the
string from localStorage using the
getItem() method.

            var tableData =
localStorage.getItem("tableData");
            //console.log(tableData);

            tableBody.innerHTML = "
";

            const tableHTML =
JSON.parse(tableData);
            tableBody.innerHTML =
tableHTML


        }

         //-------------------------
-----------------

        //search function getting the
rows
        var d =
document.getElementById("info");
        const x = d.children;
        console.log(x);

        //search function - one can
use string functions as well
        function ST() {
            var s =
document.getElementById("search").val
ue;
            const pattern = new
RegExp(s);
            console.log(pattern);

            for (let i = 0; i <
x.length; i++) {
                if
(pattern.test(x[i].cells[1].innerText
)) {
                    x[i].style.color
= "red";

setTimeout(function () {

x[i].style.color = "black";
                    }, 3000);
                }

            }
        }

         //-------------------------
-----------------

        //function to delete
specified row
        function delete_row(e) {

e.parentElement.parentElement.remove(
);
            //console.log("hi");
        }


    </script>
}
```