

Web Engineering (CS-666/CS-723)

Mr. Aftab Khan
Mr. Amir Rashid

Email id: aftab@biit.edu.pk WhatsApp# 0344-8184704
Email id: amir@biit.edu.pk, WhatsApp# 0302-5698539

(Week 10) Lecture 19-20

Objectives: Learning objectives of this lecture are

- **Understanding State Management**
- **Client Side State Management**
- **Understanding Hidden Field**
- **Query String**
- **View Data, ViewBag and Temp Data**
- **Cookies**

Text Book & Resources: Professional ASP.NET MVC 5 by Jon Galloway, Brad Wilson, K. Scott Allen, David Matson

Video Links:

https://www.youtube.com/watch?v=3ARacDDNCn8&list=PLjVp2I_4DPy_4CvU-mdB_vfHfT8k7o18N&index=22&t=8s

https://www.youtube.com/watch?v=OsHLBVqLMH8&list=PLjVp2I_4DPy_4CvU-mdB_vfHfT8k7o18N&index=23&t=15s

https://www.youtube.com/watch?v=T-q81Fc9kWY&list=PLjVp2I_4DPy_4CvU-mdB_vfHfT8k7o18N&index=24&t=7s

https://www.youtube.com/watch?v=ya0-02mvZqc&list=PLjVp2I_4DPy_4CvU-mdB_vfHfT8k7o18N&index=26&t=9s

Introduction to State Management

As a web developer we all know, HTTP is a stateless protocol, i.e., each HTTP request does not know about the previous request. So [HTTP requests](#) are independent messages that don't retain user values or app states. We need to take additional steps to manage state between the requests. If we want to pass data from one page to another page or even on multiple visits of the same page, then we need to use the State management techniques provided by ASP.NET. In this lecture, we are going to look at various approaches to the HTTP state management that we can use in our application.

State management is a mechanism by which developers can maintain state and page information over multiple request for the same or different pages in web application.

There are two types of state management that ASP.NET provides to maintain page state.

- 1) Client side state management
- 2) Server side state management

1) Client Side State Management

In client side state management the information is stored on client system. This information will travel back and forth with every request and response to and from server. The major advantages of having this kind of state management is that it saves a lot of server memory. We relieve server from the burden of keeping the state related information. But it takes more bandwidth as considerable amount of data is traveling back and forth. Due to this, web page becomes slow to load. The main drawback is it creates security issue for sensitive information like passwords, credit card numbers etc.

ASP.NET MVC provides following types of client side methods to manage state in web applications.

1. Hidden Field
2. Query Strings
3. View Data
4. View Bag
5. Temp Data
6. Cookies

The above objects help us to persist our data on the same page or when moving from “Controller” to “View” or “Controller” to Controller”. Let us start practical implementation of these so that we can understand in a better way.

1) Hidden Field

It is not new, we all know it from HTML programming. Hidden field is used to hide your data on client side. It is not directly visible to the user on UI but we can see the value in the page source. So, this is not recommended if you want to store a sensitive data. It's only used to store a small amount of data which is frequently changed. We can save data in hidden form fields and send back in the next request. Sometimes we require some data to be stored on the client side without displaying it on the page. Later when the user takes some action, we'll need that data to be passed on to the server side. This is a common scenario in many applications and hidden fields provide a good solution for this. Let us implement this concept in asp.net MVC.

First of all. Create a new Model Class named User. Here is the code for User class.

```

using System;

using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace MVCDemo.Models
{
    public class User
    {
        public int Id { get; set; }

        public string UserName { get; set; }
        public string emailId { get; set; }

        public string Password { get; set; }
    }
}

```

Create UserController class with following code.

```

using MVCDemo.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCDemo.Controllers
{
    public class UserController : Controller
    {
        // GET: User
        public ActionResult Index()
        {
            User user = new User
            {
                Id=101 // populating value for hidden field
            };

            return View(user);
        }
    }
}

```

```

[HttpPost]
public ActionResult Index(User user)
{
    int id = user.Id; // reading hidden value of id from Model
    string name = user.Name ;
    string emailId = user.EmailAddress;
    string password = user.Password;

    return View();
}
}
}

```

Now create view for index action as following.

```

@model MVCDemo.Models.User

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    @using (Html.BeginForm())
    {

        <div class="form-horizontal">
            <h4>User Form</h4>
            <hr />
            @Html.HiddenFor(model=>model.Id)

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

```

```

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

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

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

</body>
</html>

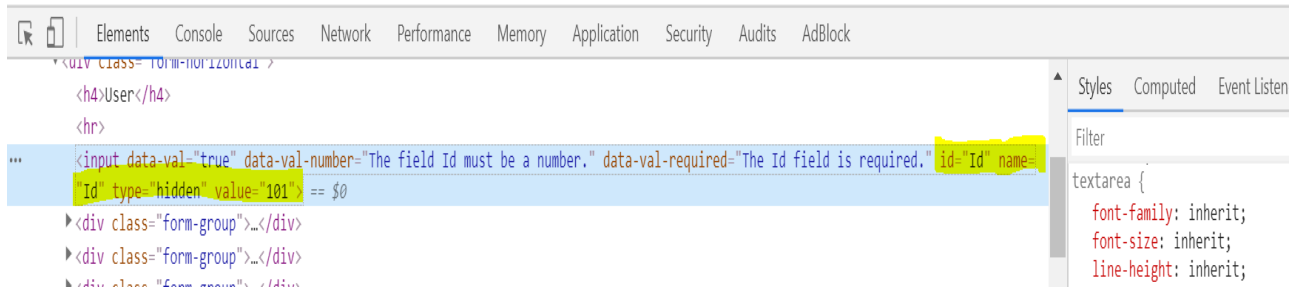
```

If you open the page source code for that page in the browser, you will find the following line of code, which is nothing but the HTML version of the above code with value. Just focus on last three attributes.

```

<input data-val="true" data-val-number="The field Id must be a number." data-val-required="The Id field is required." id="Id" name="Id" type="hidden" value="101">

```



2) Query String

Generally query string is one of client side state management techniques in ASP.NET in which query string stores values in URL that are visible to Users. We mostly use query strings to pass data from one page to another page in asp.net mvc.

In asp.net mvc routing has a support for query strings in **RouteConfig.cs** let's have a look on it. Our routing class file **RouteConfig.cs** will contain id as optional Parameter.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new
        {
            controller = "Home",
            action = "Index",
            id = UrlParameter.Optional
        });
}
```

In asp.net mvc we can pass query string with Optional Parameter and also without Optional Parameter.

Query strings without Optional Parameters

<https://localhost:44336/User/Index/101?Name=ali&emailId=ali@gmail.com>

Query strings with Optional Parameter

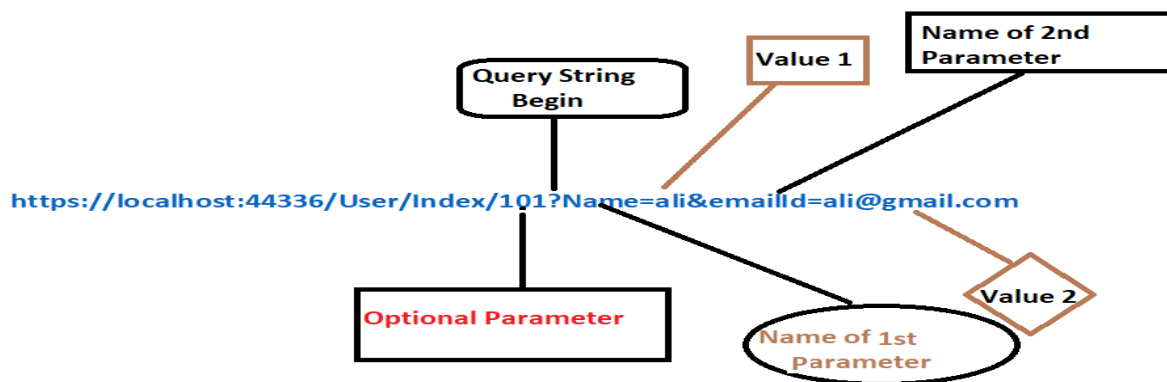
<https://localhost:44336/User/Index?Name=ali&emailId=ali@gmail.com>

Now let's add parameters to index action method in UserController. This action method has optional parameter (Id) with 2 other parameter Name and email.

```
public ActionResult Index(string id, string Name, string emailId)
{
    User user = new User
    {
        Id=101 // populating value for hidden field
    };
    return View(user);
}
```

Now let's run application and access URL. For calling this URL first we are going to call Controller [User] then following with Action Method [Index] lastly with Optional Parameter [101] then query string will be like **[? Name=ali&&emailId=ali@gmail.com]**.

Below is Completed snapshot of how to pass Query string.



Now you can easily view all values given from URL while debugging.

You have learned concept of query string in asp.net mvc with accessing from URL. Now let's call Action Method from Action Link and pass Query string. Generate following link in the end of Index view.

With Optional Parameters

```
@Html.ActionLink("Detail Action", "Details", new { Id = 10, Name = "ali",  
emailId = "ali@gmail.com" })
```

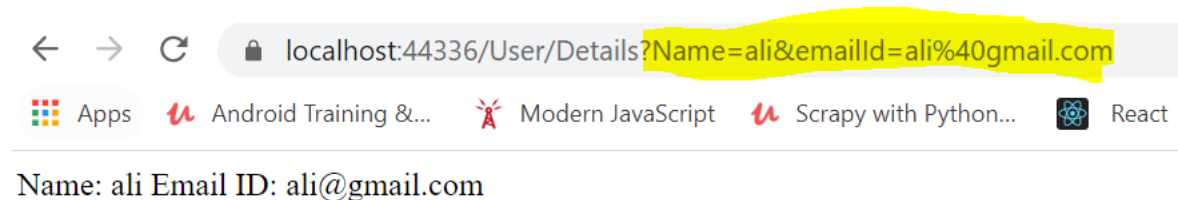
Without Optional Parameters

```
@Html.ActionLink("Details Action", "Details", new {Name = "ali", emailId  
= "ali@gmail.com" })
```

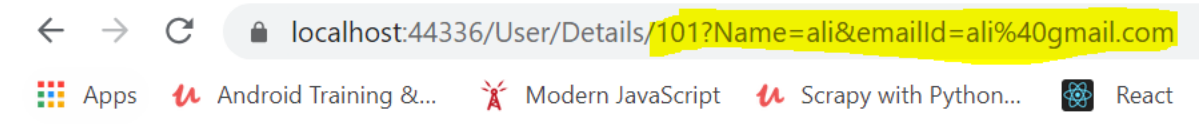
Add Details action in the User Controller.

```
public string Details(string id, string Name, string emailId)  
{  
    string userInfo = string.Empty;  
    if (!String.IsNullOrEmpty(id))  
    {  
        userInfo = "Id: " + id + "Name: " + Name + "Email ID:  
" + emailId;  
    }  
    else  
    {  
        userInfo = "Name: " + Name + "Email ID: " + emailId;  
    }  
    return userInfo;  
}
```

Here is the output of on click of Details Action Link without optional Parameter.



Here is the output of on click of Details Action Link with optional Parameter (Id).



Id: 101 Name: ali Email ID: ali@gmail.com

3) View Data

It helps us to maintain data when sending the data from Controller to View. It is a dictionary object and derived from ViewDataDictionary. As it is a dictionary object, it takes the data in a key-value pair.

Once you create ViewData object, pass the value, and make redirection; the value will become null. The data of ViewData is not valid for next subsequent request. Take care of two things when using ViewData, first, check for null and second, check for typecasting for complex data types. Let us try to understand this concept with the help of example.

Here is the modified code for Details Method in UserController.

```
public ActionResult Details()
{
    User user = new User
    {
        Id=101,
        Name="Ali",
        EmailAddress="Ali@gmail.com"
    };

    ViewData["Message"] = "Passing Message from ViewData!";
    ViewData["User"] = user;
    return View();
}
```

Here is the code for Details view.

```
@{
    Layout = null;
    ViewBag.Title = "Details Page";
}

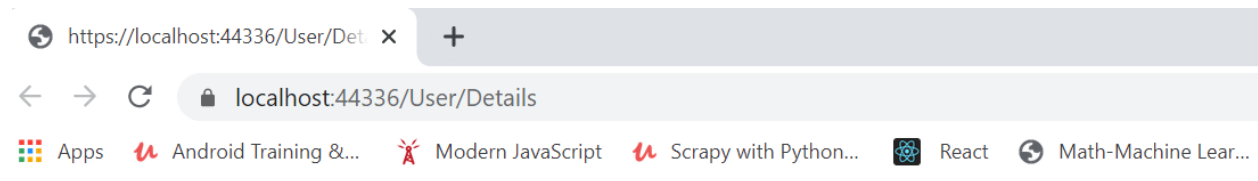
<div class="row">
    <div class="col-md-4">
```

```

<h2>User Details</h2>
<br />
<p>
    @if (ViewData["Message"] != null)
    {
        <b>@ViewData["Message"].ToString();</b>
    }
</p>
<br />
@if (ViewData["User"] != null)
{
    var user = (MVCDemo.Models.User)ViewData["User"];
    <table>
        <tr>
            <td>
                ID :
            </td>
            <td>
                @user.Id
            </td>
        </tr>
        <tr>
            <td>
                Name :
            </td>
            <td>
                @user.Name
            </td>
        </tr>
        <tr>
            <td>
                Email Id :
            </td>
            <td>
                @user.EmailAddress
            </td>
        </tr>
    </table>
}
</div>
</div>

```

Following is the output for view Data related code.



User Details

Passing User Information from ViewData

ID : 101
Name : Ali
Email Id : Ali@gmail.com

4) ViewBag

The task of ViewBag is same as that of ViewData. It is also used to transfer the data from Controller to View. However, the only difference is that ViewBag is an object of Dynamic property. It is a wrapper around ViewData. If you use ViewBag rather than ViewData, you will not have to do typecasting with the complex objects and do not need to check for null.

If we consider the same above code with ViewBag, the output will be same.

Here is the modified code for Details Method in UserController.

```
public ActionResult Details()
{
    User user = new User
    {
        Id=101,
        Name="Ali",
        EmailAddress="Ali@gmail.com"
    };

    ViewBag.Message = "Passing Message from ViewBag!";
    ViewBag.User = user;
    return View();
}
```

Here is the code for Details view.

```

@{
    Layout = null;
    ViewBag.Title = "Details Page";
}

<div class="row">
    <div class="col-md-4">
        <h2>User Details</h2>
        <br />
        <p>

            <b>@ViewBag.Message</b>

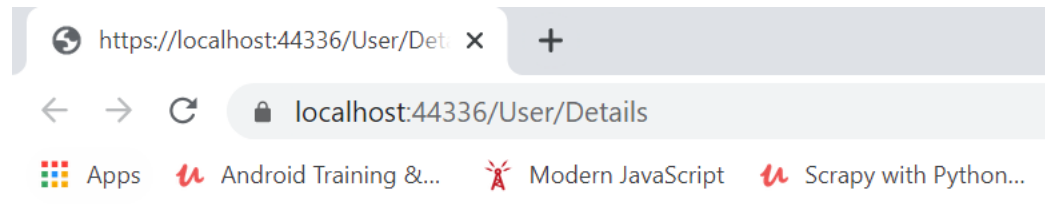
        </p>
        <br />
    </div>
    @
    {
        var user = ViewBag.User;

        <table>
            <tr>
                <td>
                    ID :
                </td>
                <td>
                    @user.Id
                </td>
            </tr>
            <tr>
                <td>
                    Name :
                </td>
                <td>
                    @user.Name
                </td>
            </tr>
            <tr>
                <td>
                    Email Id :
                </td>
                <td>
                    @user.EmailAddress
                </td>
            </tr>
        </table>
    }
}

```

```
</div>  
</div>
```

Following is the output for view Data related code.



User Details

Passing User Information from ViewBag

ID : 101
Name : Ali
Email Id : Ali@gmail.com

Points to Remember

1. If you are using ViewData that is not defined on Controller, then it will throw an error; but with ViewBag, it will not.
2. Do not use ViewBag and ViewData with the same name, otherwise, only one message will display. See the following code in the controller is using both ViewData and ViewBag with same name "Message".

```
public ActionResult Details()  
{  
    ViewData["Message"] = "This is Message from ViewData";  
    ViewBag.Message = "This is Message from ViewBag";  
  
    return View();  
}
```

On view defined both as following.

```

<b>@ViewBag.Message</b>
    @if (ViewData["Message"] != null)
    {
        ViewData["Message"].ToString();
    }

```

The output will show only one message and that will be the last one [in this case, message will be ["This is Message from ViewBag"].

5) TempData

TempData is also a dictionary object as ViewData and stores value in key/value pair. It is derived from TempDataDictionary. It is mainly used to transfer the data from one request to another request or we can say subsequent request. If the data for TempData has been read, then it will get cleaned. To persist the data, there are different ways. It all depends on how you read the data. Let us understand this concept with the help of example.

Change the Details Actions code as following.

```

public ActionResult Details()
{
    User user = new User
    {
        Id = 101,
        Name = "Ali",
        EmailAddress = "Ali@gmail.com"
    };
    ViewBag.Message = "Message from View Data";
    ViewData["Info"] = "Passing Info from ViewData!";

    TempData["User"] = user;
    return RedirectToAction("Delete");
}

```

Here is the code for Delete Action.

```

public ActionResult Delete()
{
    var user = TempData["user"]; /* TempData["user"] is available here*/
    var Message = ViewBag.Message; /* ViewBag.Message is Not available here*/
}

```

```
var Info = ViewData["Info"];/* ViewData["Info"] is Not available here
too*/
```

```
        return View();
    }
```

Here is the code for Delete View.

```
@{
    Layout = null;
    ViewBag.Title = "Details Page";
}

<div class="row">
    <div class="col-md-4">
        <h2>Delete Users</h2>
        <br />
        <p>
            @**View Bag is null here*@
            <b>@ViewBag.Message </b>

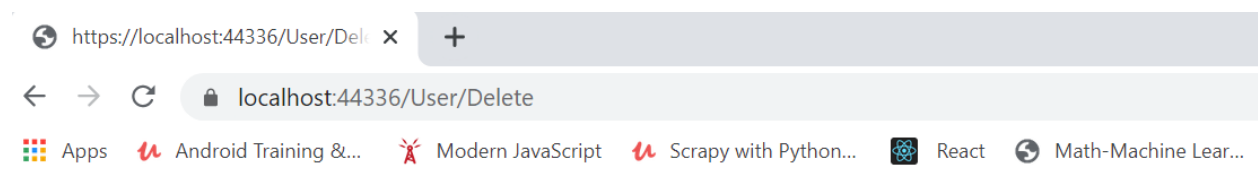
            @**View Data is null here too*@
            @if (ViewData["Info"] != null)
            {
                @ViewData["Info "]
            }
        </p>
        <p>
            Do you Really Want to Delete Following Record!
        </p>
        <br />
        @**Temp Data is available in this 2nd Request *@
        @if (TempData["User"] != null)
        {
            var user = (MVCDemo.Models.User)TempData["User"];
            <table>
                <tr>
                    <td>
                        ID :
                    </td>
                    <td>
                        @user.Id
                    </td>
                </tr>
            </table>
        }
    </div>
</div>
```

```

        </td>
    </tr>
    <tr>
        <td>
            Name :
        </td>
        <td>
            @user.Name
        </td>
    </tr>
    <tr>
        <td>
            Email Id :
        </td>
        <td>
            @user.EmailAddress
        </td>
    </tr>
    <tr>
        <td><input type="submit" value="OK" /></td>
        <td><input type="button" value="Cancel" /></td>
    </tr>
</table>
    }
</div>
</div>

```

Following is the output of Delete View.



Delete Users

Do you Really Want to Delete Following Record!

ID : 101
 Name : Ali
 Email Id : Ali@gmail.com

6) Cookies

Cookies are used for storing the data but that data should be small. It is like a small text file where we can store our data. The good thing is that a cookie is stored on client side memory in the browser. Most of the times, we use a cookie for storing the user information after login with some expiry time. Basically, a cookie is created by the server and sent to the browser in response. The browser saves it in client-side memory.

A cookie is basically a physical, plain-text file stored by the client (usually a browser), tied to a specific website. The client will then allow this specific website to read the information stored in this file on subsequent requests, basically allowing the server (or even the client itself) to store information for later use.

One advantage of cookies is that they work transparently without the user being aware that information needs to be stored. They also can be easily used by any page in your application and even be retained between visits, which allows for truly long-term storage.

They suffer from some of the same drawbacks namely, they're limited to simple string information, and they're easily accessible and readable if the user finds and opens the corresponding file. These factors make them a poor choice for complex or private information or large amounts of data.

Also some users disable cookies on their browsers, which will cause problems for web applications that require them. Also, users might manually delete the cookie files stored on their hard drives. But for the most part, cookies are widely adopted and used extensively on many websites.

The important trick to remember is that we retrieve cookies from the Request object, and we set cookies using the Response object.

To set a cookie, just create a new `HttpCookie` object. We can then fill it with string information and attach it to the current web response:

```
// Create the cookie object.  
HttpCookie cookie = new HttpCookie("FavouriteCookie");  
// Set a value in it.  
cookie["FavouriteLanguage"] = "Urdu";  
// Add another value.  
cookie["Country"] = "Pakistan";  
// Add it to the current web response.  
Response.Cookies.Add(cookie);
```

A cookie added in this way will persist until the user closes the browser and will be sent with every request. To create a longer-lived cookie, you can set an expiration date.

```
// This cookie lives for one year.  
cookie.Expires = DateTime.Now.AddYears(1);
```

we retrieve cookies by cookie name using the Request. Cookies collection:

```
HttpCookie cookie = Request.Cookies["FavouriteCookie "];  
// Check to see whether a cookie was found with this name.  
string language;  
if (cookie != null)  
{  
    language = cookie["FavouriteLanguage"];  
}
```

Let us implement Cookies concepts in ASP.NET MVC application.

Create a new Model class Question.cs in your MVC Models folder.

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel.DataAnnotations;  
using System.Linq;  
using System.Web;  
  
namespace MVCDemo.Models  
{  
    public class Question  
    {  
        [Required(ErrorMessage = "Please enter your Favourite  
Fruit.")]  
        [Display(Name = "Favourite Fruit : ")]  
        public string Fruit { get; set; }  
  
        [Required(ErrorMessage = "Please enter your Favourite Day.")]  
        [Display(Name = "Favourite Day : ")]  
        public string Day { get; set; }  
  
        [DataType(DataType.EmailAddress)]  
        [Required(ErrorMessage = "Please enter your Email Address.")]  
        [Display(Name = "EmailAddress : ")]  
        public string EmailAddress { get; set; }  
  
        [Required(ErrorMessage = "Please enter your Favourite  
Color.")]  
        [Display(Name = "Favourite Color : ")]
```

```

        public string Color { get; set; }
    }
}

```

Create a new Controller with name **Questions** and edit it as following.

```

using MVCDemo.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MVCDemo.Controllers
{
    public class QuestionsController : Controller
    {
        // GET: Question
        public ActionResult Index()
        {
            return View();
        }

        [HttpGet]
        public ActionResult Questions()
        {
            return View();
        }

        [HttpPost]
        public ActionResult Questions(Question questions)
        {
            if (ModelState.IsValid)
            {
                //Creating Cookie
                HttpCookie PreferencesCookie = new
                HttpCookie("Preferences");

                //Adding value for day
                PreferencesCookie["FavouriteDay"] = questions.Day;

                //Adding value for color
                PreferencesCookie["FavouriteColor"] = questions.Color;

                //Adding value for EmailAddress
            }
        }
    }
}

```

```

        PreferencesCookie["EmailId"] = questions.EmailAddress;

        //Adding value for Fruit
        PreferencesCookie["FavouriteFruit"] = questions.Fruit;

        Response.Cookies.Add(PreferencesCookie);
        //Cookie will persist for 3 months
        PreferencesCookie.Expires = DateTime.Now.AddMonths(3);
        return RedirectToAction("MyAnswers", "Questions");
    }

    return View(questions);
}

public ActionResult MyAnswers()
{
    //We can read cookie in controller as well as view
    HttpCookie cookie = Request.Cookies["Preferences"];
    // Check to see whether a cookie was found with this name.

    if (cookie != null)
    {
        string color = cookie["FavouriteColor"];
    }
    return View();
}
}
}

```

Here is code for View of Questions.

```
@model MVCDemo.Models.Question
```

```
@{
    Layout = null;
}
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <meta name="viewport" content="width=device-width" />
```

```
    <title>Questions</title>
```

```
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
```

```
</head>
```

```

<body>
    @using (Html.BeginForm())
    {
        @Html.AntiForgeryToken()

        <div class="form-horizontal">
            <h4>Question</h4>
            <hr />

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

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

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

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

```

```

        @Html.EditorFor(model => model.Color, new {
htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.Color,
"", new { @class = "text-danger" })
    </div>
    </div>

    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Save Preferences"
class="btn btn-default" />
        </div>
    </div>
</div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
</body>
</html>

```

Here is code for View of MyAnswers.

```
@model MVCDemo.Models.Question
```

```

@{
    Layout = null;
    HttpCookie cookie = Request.Cookies["Preferences"];
}

```

```
<!DOCTYPE html>
```

```

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>MyAnswers</title>
    <link href="~/Content/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div>
        <h4>Your Prefernces</h4>
        <hr />
        <dl class="dl-horizontal">

```

```
<dt>
    @Html.DisplayNameFor(model => model.Fruit)
</dt>

<dd>
    @if (cookie != null)
    {
        @cookie["FavouriteFruit"];
    }

</dd>

<dt>
    @Html.DisplayNameFor(model => model.Day)
</dt>

<dd>
    @if (cookie != null)
    {
        @cookie["FavouriteDay"];
    }
</dd>

<dt>
    @Html.DisplayNameFor(model => model.EmailAddress)
</dt>

<dd>
    @if (cookie != null)
    {
        @cookie["EmailId"];
    }
</dd>

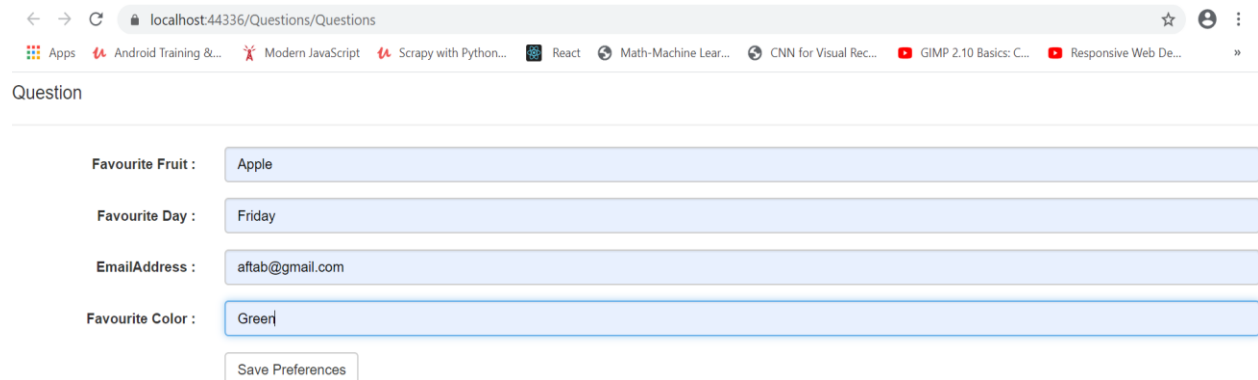
<dt>
    @Html.DisplayNameFor(model => model.Color)
</dt>

<dd>
    @if (cookie != null)
    {
        @cookie["FavouriteColor"];
    }
</dd>
```

```
        </dl>
    </div>

</body>
</html>
```

Here is output for Questions Form.



localhost:44336/Questions/Questions

Apps Android Training &... Modern JavaScript Scrapy with Python... React Math-Machine Lear... CNN for Visual Rec... GIMP 2.10 Basics: C... Responsive Web De...

Question

Favourite Fruit : Apple

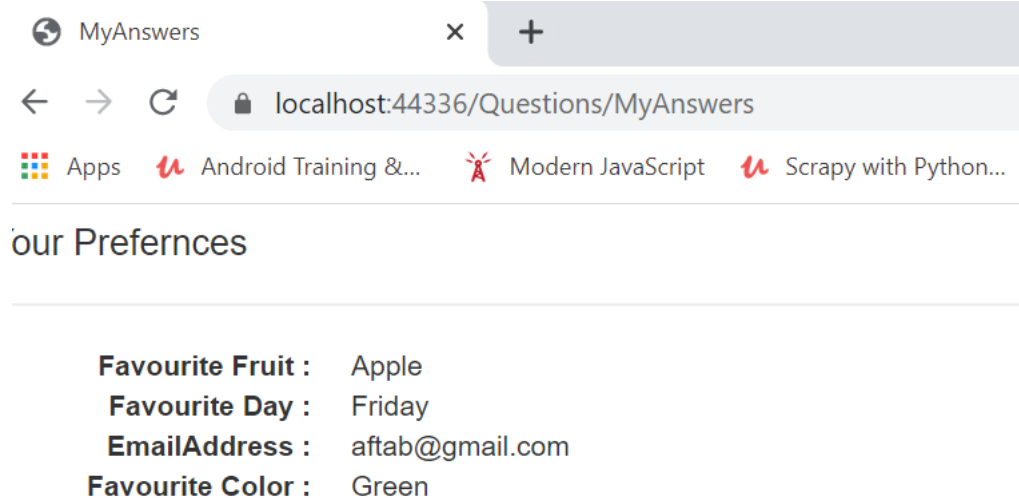
Favourite Day : Friday

EmailAddress : aftab@gmail.com

Favourite Color : Green

Save Preferences

Here is output for Answers View After reading values from Cookies stored on Local Machine.



MyAnswers

localhost:44336/Questions/MyAnswers

Apps Android Training &... Modern JavaScript Scrapy with Python...

our Preferences

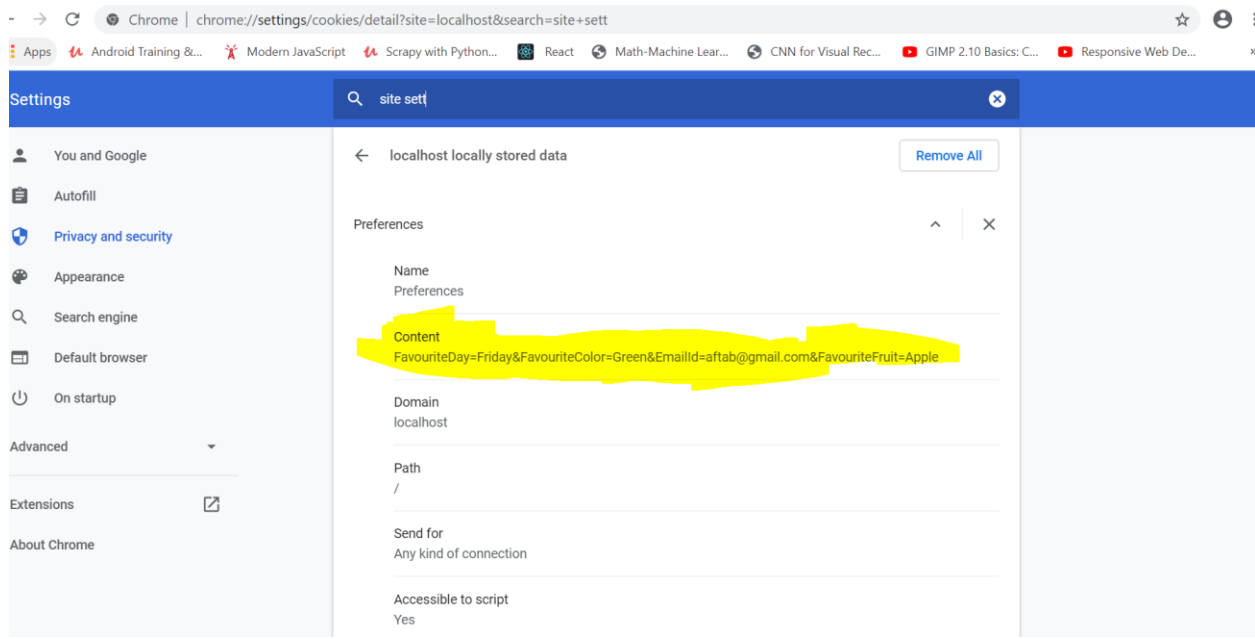
Favourite Fruit : Apple

Favourite Day : Friday

EmailAddress : aftab@gmail.com

Favourite Color : Green

To check Cookies in chrome go to setting and write site settings in search bar. Select site settings and click on cookies and site data tab. Click on All cookies and site data and search for localhost. Select preferences and here is the screenshot for preferences cookie.



Assignment #8

Dear students read the given lectures carefully as per the course objectives mentioned on the top and carryout the assignment as per following instructions

1. Submission Date: Sunday 03-May-2020 at 11:59 PM. This will also count as your Attendance for this week.
2. You must prepare handwritten Assignment with implementation in the form of project.
3. Send it to respective course teacher (after scanning it) for assessment by email only.

Create a form on top of previously developed Mobile List View and do as following on **save Preferences** Button.

Part 1: Form should consist of a text box to take input for preference of brand/OS (android or IOS). Next time, Records must be already filtered according to user preferred brand. (Display only android mobiles in case of android and vice versa)

Part 2: form must have another text box to take input from user regarding number of records per page (3, 5, 10 etc.). Next time on page load, paging criteria must be updated as per user preferences.

Main Requirement: User preference should be saved on local Machine for at least 1 month and he/she must be able to change preferences at any time using above developed form.

Note: Read Lecture and above questions carefully to solve assignment. Use proper technique.