Learn MVC Project in 7 Days – Day 4

Marla Sukesh, 27 Jul 2015

This article is continuation of day 4 for Learn MVC Project in 7 Days.

Download Retain_Values_after_postback.zip
Download Client_Side_Validation.zip
Download Authentication.zip

Introduction

Welcome to day 4 of "Learn MVC Project in 7 days" series. If you are directly coming to this article then I strongly recommend you to first complete all the previous parts of the series and then proceed here.

Complete Series

- Day 1
- Day 2
- Day 3
- Day 4
- Day 5
- Day 6
- Day 7
- Bonus Day 1
- Bonus Day 2

We are pleased to announce that this article is now available as hard copy book you can get the same from www.amazon.com and www.flipkart.com

Agenda

Lab 15 – Retaining values on Validation Error

In Lab 13 we introduced Server side validation and in Lab 14 we took it to next level by adding a Custom Validation support.

I strongly recommend you to rerun to the Lab 14 once again. Execute it and understand the code and output both perfectly.

In the Lab 15 we will learn how to repopulate values on validation failure.

Step 1 - Create CreateEmployeeViewModel

Create a new class in ViewModel folder as follows.

```
public class CreateEmployeeViewModel
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Salary { get; set; }
}
```

Step 2 – Change SaveEmployee action method

For repopulation we will simply reuse the Employee object created by Model Binder. Change SaveEmployee Action method as follows.

```
public ActionResult SaveEmployee(Employee e, string BtnSubmit)
{
    switch (BtnSubmit)
    {
        case "Save Employee":
            if (ModelState.IsValid)
            {
                EmployeeBusinessLayer empBal = new EmployeeBusinessLayer();
                empBal.SaveEmployee(e);
                return RedirectToAction("Index");
            }
            else
            {
                CreateEmployeeViewModel vm = new CreateEmployeeViewModel();
                vm.FirstName = e.FirstName;
                vm.LastName = e.LastName;
                if (e.Salary.HasValue)
                {
                    vm.Salary = e.Salary.ToString();
                }
                else
                {
                    vm.Salary = ModelState["Salary"].Value.AttemptedValue;
                }
                return View("CreateEmployee", vm); // Day 4 Change - Passing e here
            }
        case "Cancel":
            return RedirectToAction("Index");
    }
    return new EmptyResult();
}
```

Step 3 – Repopulate values in View

Step 3.1 Make View a strongly typed view

Put following code in the top of CreateEmployee View.

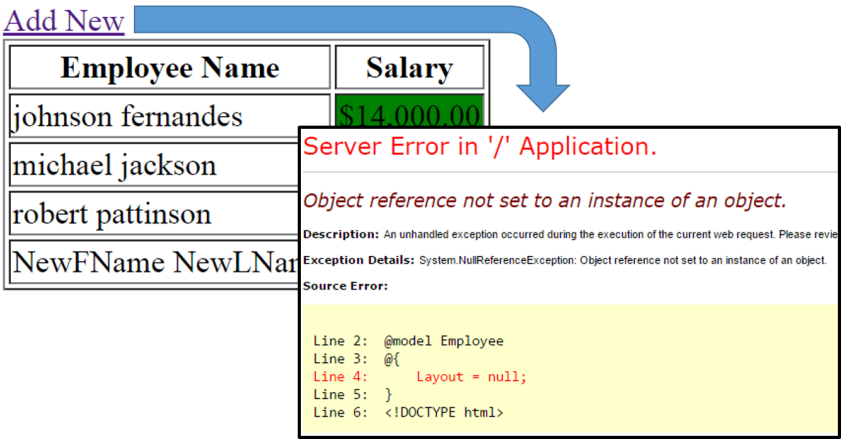
```
@using WebApplication1.ViewModels
@model CreateEmployeeViewModel
```

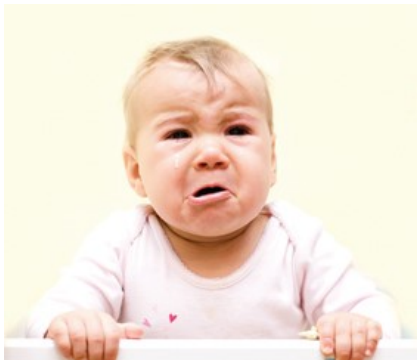
Step 3.2 Display values in corresponding controls from Model

```
...

...

...

<input type="text" id="TxtFName" name="FirstName" value="@Model.FirstName" />

...

...

...

<input type="text" id="TxtLName" name="LastName" value="@Model.LastName" />

...

...

...

<input type="text" id="TxtSalary" name="Salary" value="@Model.Salary" />

...

...

...
```

Step 4 – Execute and Test

Press F5 and execute the application. Navigate to the AddNew screen by clicking “Add New” link.





Reason for above error will be discussed at the end of this lab. Let's implement solution now.

Step 5 – Change AddNew Action method

```
public ActionResult AddNew()
{
    return View("CreateEmployee", new CreateEmployeeViewModel());
}
```

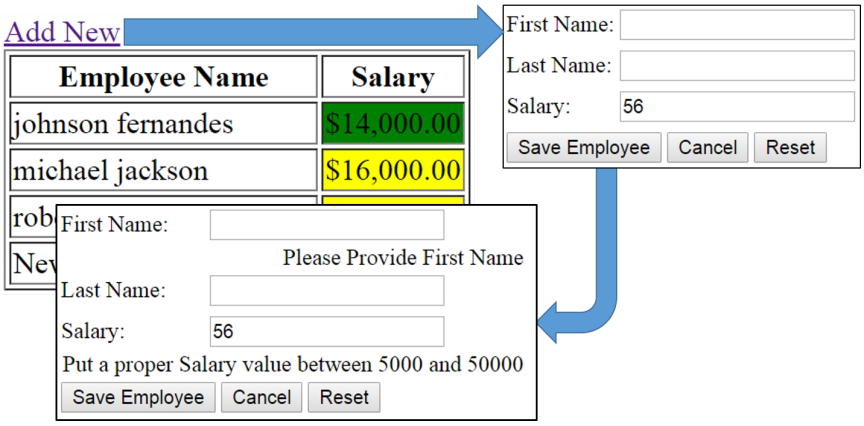
Step 6 – Execute and Test

Press F5 and execute the application.

Test 1

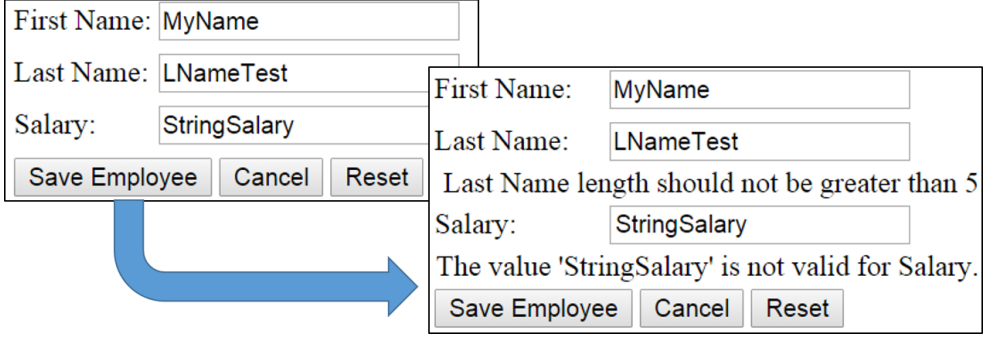
- Navigate to the AddNew screen by clicking "Add New" link.
- Keep First Name Empty
- Put Salary as 56.
- Click "Save Employee" button.

It will make two validations fail



As you can see 56 is maintained in Salary Textbox.

Test 2



As you can see FirstName and LastName textbox values are maintained.

Strange thing is Salary is not maintaining. We discuss the reason and solution for it in the end of the lab.

Talk on Lab 15

Are we really retaining values here?

No, we are not. Here we are actually repopulating the values from the posted data.

Why it's required to pass "new CreateEmployeeViewModel()" during initial request that is in "Add New" action method?

In the View we are trying to populate textbox with the values in model.

Example:

```
<input id="TxtSalary" name="Salary" type="text" value="@Model.Salary" />
```

As you can see, in the above code block we are accessing FirstName property of current Model. If Model is null, simply "Object reference not set to an instance of the class" exception will be thrown.

When "Add New" hyperlink is clicked, request will be handled by "Add New" action method. Earlier in this action method we wererereturning view without passing any data. It means Model property inside view be Null and Null.Something will throw "Object reference not set to an instance of the class". To solve this problems "new CreateEmployeeViewModel()" was passed during initial request.

Do we have any automated way to achieve same functionality?

Yes, we can use HTML helper classes for that. We will talk about this in one of the upcoming lab.

Lab 16 – Adding Client side validation

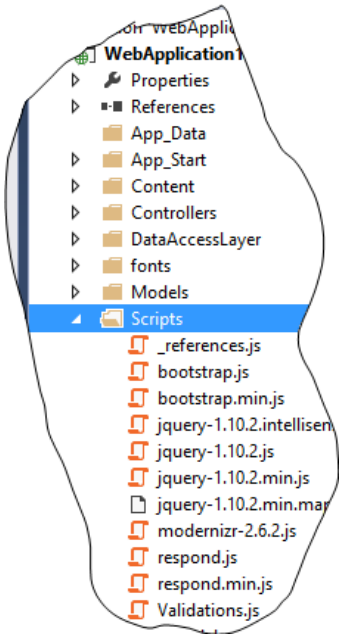
First of all let's list down what all validations we require here.

1. FirstName should not be empty.
2. LastName length should not be greater than 5.
3. Salary should not be empty.
4. Salary should be a proper number
5. FirstName should not contain "@" symbol

Let's see how to implement it.

Step 1 – Create JavaScript validation file

Create a JavaScript File called "Validations.js" and keep it inside Scripts folder



Step 2 – Create Validation functions

In "Validations.js" file create validation functions as follows

```
function IsFirstNameEmpty() {
    if (document.getElementById('TxtFName').value == "") {
        return 'First Name should not be empty';
    }
    else { return ""; }
}
```

```
function IsFirstNameInValid() {
    if (document.getElementById('TxtFName').value.indexOf("@") != -1) {
        return 'First Name should not contain @';
    }
    else { return ""; }
}
function IsLastNameInValid() {
    if (document.getElementById('TxtLName').value.length>=5) {
        return 'Last Name should not contain more than 5 character';
    }
    else { return ""; }
}
function IsSalaryEmpty() {
    if (document.getElementById('TxtSalary').value=="") {
        return 'Salary should not be empty';
    }
    else { return ""; }
}
function IsSalaryInValid() {
    if (isNaN(document.getElementById('TxtSalary').value)) {
        return 'Enter valid salary';
    }
    else { return ""; }
}
function IsValid() {

    var FirstNameEmptyMessage = IsFirstNameEmpty();
    var FirstNameInValidMessage = IsFirstNameInValid();
    var LastNameInValidMessage = IsLastNameInValid();
    var SalaryEmptyMessage = IsSalaryEmpty();
    var SalaryInvalidMessage = IsSalaryInValid();

    var FinalErrorMessage = "Errors:";
    if (FirstNameEmptyMessage != "")
        FinalErrorMessage += "\n" + FirstNameEmptyMessage;
    if (FirstNameInValidMessage != "")
        FinalErrorMessage += "\n" + FirstNameInValidMessage;
    if (LastNameInValidMessage != "")
        FinalErrorMessage += "\n" + LastNameInValidMessage;
    if (SalaryEmptyMessage != "")
        FinalErrorMessage += "\n" + SalaryEmptyMessage;
    if (SalaryInvalidMessage != "")
        FinalErrorMessage += "\n" + SalaryInvalidMessage;

    if (FinalErrorMessage != "Errors:") {
        alert(FinalErrorMessage);
        return false;
    }
    else {
        return true;
    }
}
```

Step 3 - Include Validation file in View

Simple put a reference of “Validations.js” file in the head section of “CreateEmployee” view as follows.

```
<script src="~/Scripts/Validations.js"></script>
```

Step 4 – Attach validations

Simply invoke IsValid function on SaveEmployee button click as follows.

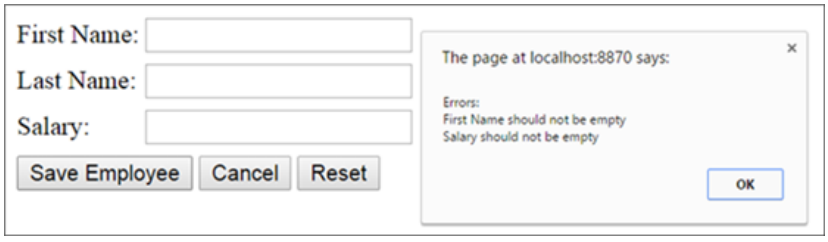
```
<input type="submit" name="BtnSubmit" value="Save Employee" onclick="return IsValid();" />
```

Step 5 – Execute and Test

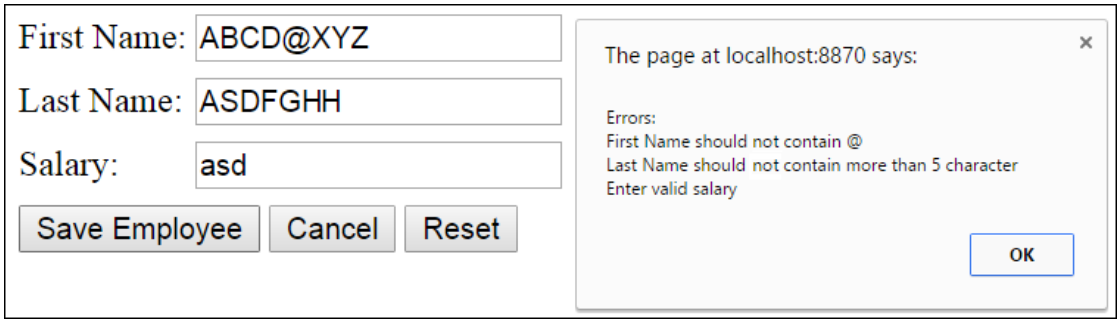
Press F5 and execute the application

Navigate to the AddNew screen by clicking “Add New” link.

Test 1



Test 2



Talk on Lab 16

Why return keyword is required in onclick of SaveEmployee button click?

As we discussed in Lab 9, submit button will make a request to server when clicked. There is no point on making server request when validation fails. By writing "return false" in the onclick of submit button, we can stop the default server request.

In our case IsValid function will return false when validation fails and thus we achieve desired functionality.

Instead of alert can we show the error messages in the page itself?

Yes we can. Simply create a span tag for each error. Make it invisible (using css) in the beginning and on submit button click if validation fails make it visible using JavaScript.

Is there any way to get this client side validation automatically?

Yes, when we use HTML helpers we get automatic client side validation based on server side validation. We will discuss this in one of the future lab

Does server side validation is required anymore?

Yes, In case someone disables JavaScript, Server side validation keep everything in place.

Lab 17 – Adding Authentication

In this lab we will make our GetView action method secured. We will make sure that only valid user will be able to access the action method.

In the Day 1 of this series we understood the real meaning of word ASP.NET and MVC. We understood that ASP.NET MVC is part of ASP.NET. Most of the features of ASP.NET are inherited in ASP.NET MVC. One of the feature is Forms Authentication.

Before we start with lab first let's understand how Forms Authentication works in ASP.NET

1. End user make a request to Forms authentication enabled application with the help of browser.
2. Browser will send all the associated cookies stored in the client machine with request.
3. When request is received as server end, server examines the request and check for the special cookie called "Authentication Cookie".
4. If valid authentication cookie is found, server confirms the identity of the user or in simple words, consider user as a valid user and make him go further.
5. If valid authentication cookie is not found server considers user as anonymous (or unauthenticated) user. In this case if the requested resource is marked as protected/secured user will be redirected to login page.

Step 1 – Create AuthenticationController and Login action method.

Right click controller folder and Select "Add New Controller" and create a controller called "Authentication". In this case full class name will be "AuthenticationController".

Inside controller create and Action method called Login as follows.

```
public class AuthenticationController : Controller
{
    // GET: Authentication
    public ActionResult Login()
    {
        return View();
    }
}
```

Step 2 – Create Model

Create new Model class called UserDetails inside Models folder as follows.

```
namespace WebApplication1.Models
{
    public class UserDetails
    {
        public string UserName { get; set; }
        public string Password { get; set; }
    }
}
```

Step 3 – Create Login View

Crete a new view called Login inside "~/Views/Authentication" folder. Make it a strongly typed view of type UserDetails.

Put following HTML inside View

```
@model WebApplication1.Models.UserDetails

@{

    Layout = null;

}

<!DOCTYPE html>

<html>

<head>

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

    <title>Login</title>

</head>

<body>

    <div>

        @using (Html.BeginForm("DoLogin", "Authentication", FormMethod.Post))

        {

            @Html.LabelFor(c=>c.UserName)

            @Html.TextBoxFor(x=>x.UserName)
```

```
            <br />

            @Html.LabelFor(c => c.Password)

            @Html.PasswordFor(x => x.Password)

            <br />


            <input type="submit" name="BtnSubmit" value="Login" />

        }

    </div>

</body>

</html>
```

As you can see, this time for generating View instead of Pure HTML we are using HtmlHelper class.

- In view we will get a readymade object of HtmlHelper class called “Html”
- HtmlHelper class functions simply returns html string.

Example 1:

```
@Html.TextBoxFor(x=>x.UserName)
```

Above code will generate following HTML.

```
<input id="UserName" name="UserName" type="text" value="" />
```

Example 2:

```
@using (Html.BeginForm("DoLogin", "Authentication", FormMethod.Post))
{
}
```

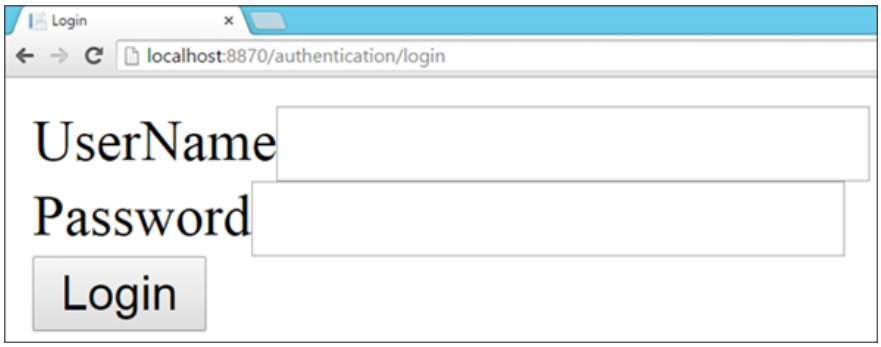
Above code will generate following HTML.

```
<form action="/Authentication/DoLogin" method="post">
</form>
```

<form action="/Authentication/DoLogin" method="post"> <

Step 4 – Execute and Test

Press F5 and execute the application. In the address put the URL of Login action method. In my case it will be“http://localhost:8870/Authentication/Login”.



Step 5 – Enable Forms Authentication

Open Web.config file. Navigate to System.Web section. Find a child section called Authentication. If it won’t exist create it. Set Authentication mode to Forms and Login URL to “Login” action method created in step 1.

```
<authentication mode="Forms">
<forms loginurl="~/Authentication/Login"></forms>
</authentication>
```

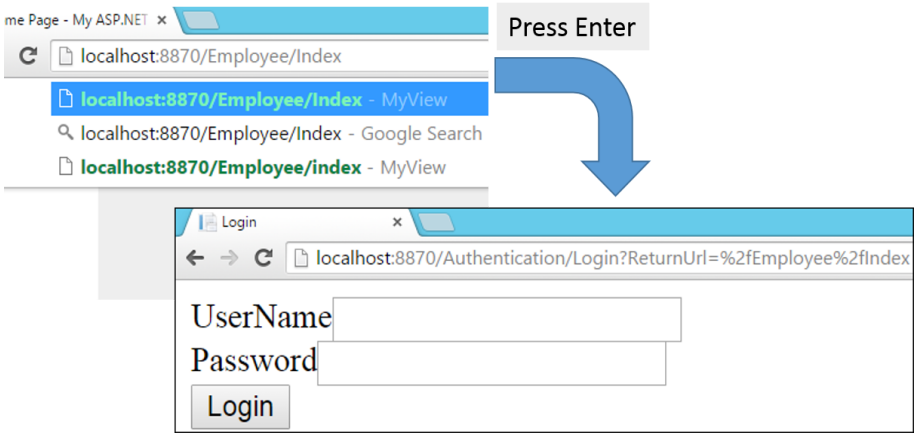
Step 6 – Make action method secured.

Open EmployeeController and attach Authorize attribute to Index action as follows.

```
[Authorize]
public ActionResult Index()
{
    EmployeeListViewModel employeeListViewModel = new EmployeeListViewModel();
......
```

Step 7 – Execute and Test

Press F5 and execute the application. In the address bar put URL of Index action ofEmployeeController. In my case it will be “http://localhost:8870/Employee/Index”.



As you can see, request to Index action automatically redirected to login action.

Step 8 – Create business layer functionality

Open EmployeeBusinessLayer class and create a method called IsValidUser as follows.

```
public bool IsValidUser(UserDetails u)
{
    if (u.UserName == "Admin" && u.Password == "Admin")
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Note: In business layer we are comparing username and password with hardcoded values. In real time we can make call to Database layer and compare it with real time values.

Step 9 – Create DoLogin Action method

Open AuthenticationController class and create a new action method called DoLogin.

This DoLogin action method will be invoked when Login button in Login view is clicked (Check Step 3).

Now let's list down the points need to be done in DoLogin

1. Check for validity of user by invoking business layer function.
2. If user is a valid user create an authentication cookie. It makes futures requests authenticated request.
3. If user is invalid, add a new error to current ModelState. This error will be displayed in View.

```
[HttpPost]
public ActionResult DoLogin(UserDetails u)
{
    EmployeeBusinessLayer bal = new EmployeeBusinessLayer();
    if (bal.IsValidUser(u))
    {
        FormsAuthentication.SetAuthCookie(u.UserName, false);
        return RedirectToAction("Index", "Employee");
    }
    else
    {
        ModelState.AddModelError("CredentialError", "Invalid Username or Password");
        return View("Login");
    }
}
```

Let's understand the above code block.

- If you remember in "Day 3 – Lab 13" we spoke about ModelState and understood that it encapsulates current state of the model. It contains all the errors related to current model. In above code snippet we are adding a new error when user is an invalid user (new error with key "CredentialError" and message "Invalid Username or Password").
- FormsAuthentication.SetAuthCookie will create a new cookie in client's machine.

Step 10 – Display message in view

Open Login View and add following code just above the @Html.BeginForm

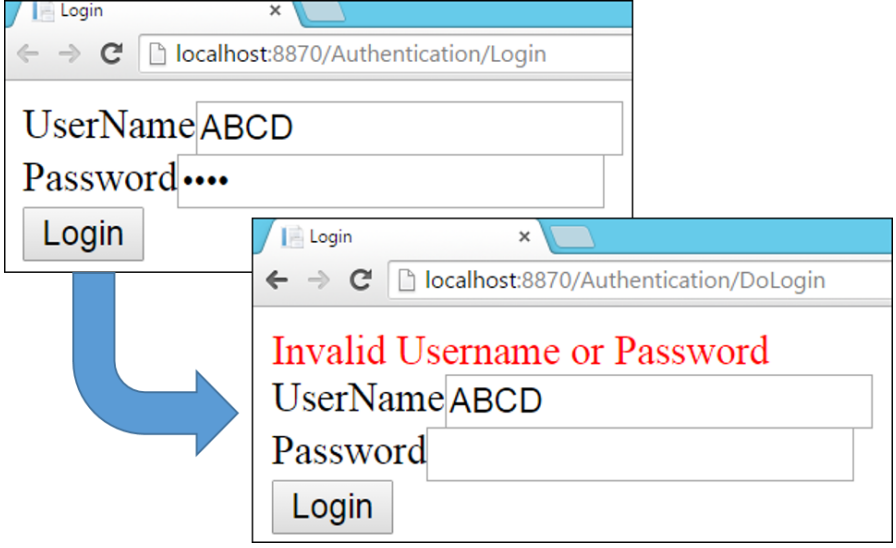
```
@Html.ValidationMessage("CredentialError", new {style="color:red;" })
@using (Html.BeginForm("DoLogin", "Authentication", FormMethod.Post))
{
```

Step 11 – Execute and Test

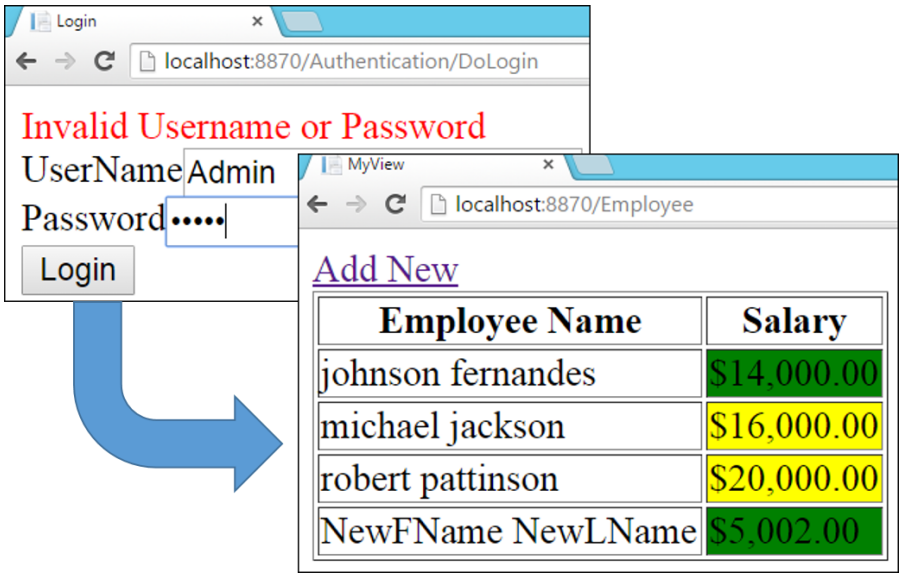
Press F5 and execute the application. Directly make a request to Login Action. I believe you know how to do it now.

Note: If you want you can make request to Index action of EmployeeController but it ultimately redirect you to Login Action.

Test 1



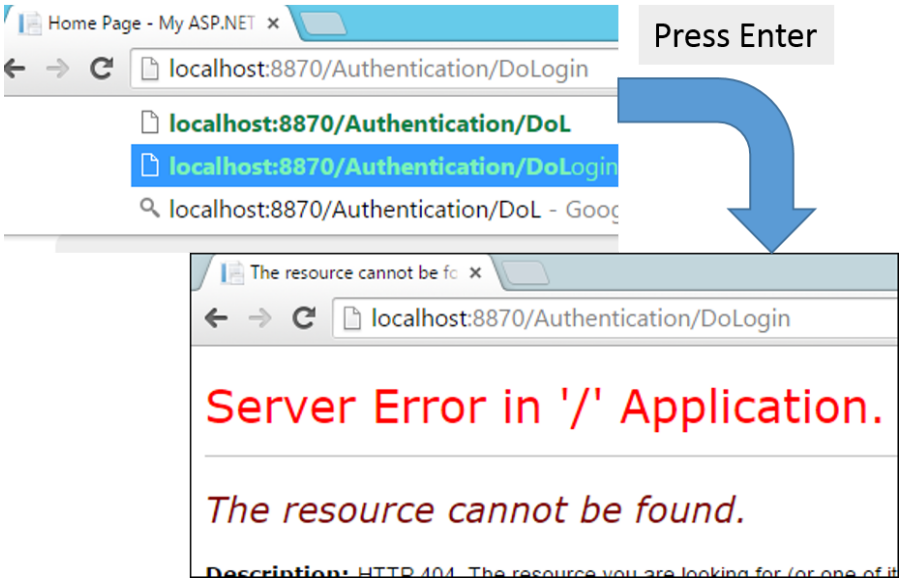
Test 2



Talk on Lab 17

Why DoLogin is attached with HttpPost attribute?

This attribute makes DoLogin action method open for only Post request. If someone try to make a get request to DoLogin it won'twork out.



Do we have more such attributes?

Yes. We have HttpGet,HttpPut and HttpDelete. As a best practice every action method must be attached with such attributes.

Note: To keep coding and learning simple and easy we have not followed such practice everywhere in this series but I recommend you to strictly follow it in your project.

As we move along we will keep on talking about such best practices.

Is it must to write FormsAuthentication.SetAuthCookie?

Yes.

Let's understand a small process.

- Client make a first request to server with the help of browser.
- When request is made via browser, all the associated cookies will be sent with the request.
- Server receives the request and prepares the response.
- Now as we know request-response happens via HTTP protocol and HTTP is a stateless protocol. For server every request will be a new request hence when the same client makes second request server won't recognize it. To solve this issue Server will add a cookie in the prepared response and send back.
- When client's browsers receives the response with cookies, it creates cookies in client's machine.
- Now if client make a request once again server will recognize him/her because request contains cookies.

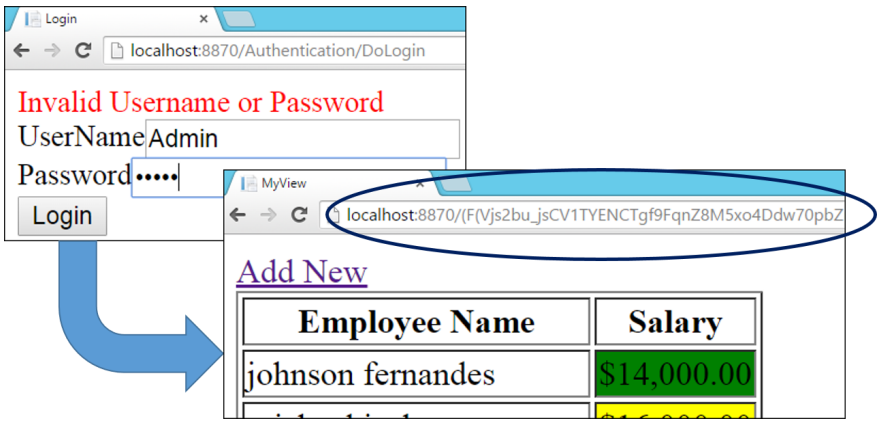
FormsAuthentication.SetAuthCookie will add a special cookie called "Authentication" cookie to the response.

Does it means FormsAuthentication won't work without cookies?

No. We have an alternative for it. We can use URI instead of cookies.

Open Web.Config and change "Authentication/Forms" section as follows.

```
<forms cookieless="UseUri" loginurl="~/Authentication/Login"></forms>
```



As you can see, now authentication cookie is passed in URL itself.

By default cookieless property is set to "AutoDetect". It means authentication works via cookies and in case cookies are not supported URL will do the required work.

What does second parameter of FormsAuthentication.SetAuthCookie do?

It will decide whether we want to create a persistent cookie or not. Non persistent cookies will get deleted automatically when browser is closed. Persistent cookies wont deleted automatically. We have to remove it manually either via coding or by using browser settings.

How to do logout via coding?

We will learn it in one of the future lab.

How come the UserName textbox is repopulated when credentials are wrong?

That's the magic of HTML helper classes. They will repopulate the values in the controls from the posted data. This is one of the advantage of using Helper classes.

What does @Html.ValidationMessage does?

We already discussed it in Lab 13 talks. It displays ModelState error based on key.

What does Authorize attribute do?

In Asp.net MVC there is a concept called Filters. Which will be used to filter out requests and response. There are four kind of filters. We will discuss each one of them in our 7 daysjourney. Authorize attribute falls under Authorization filter. It will make sure that only authenticated requests are allowed for an action method.

Can we attach both HttpPost and Authorize attribute to same action method?

Yes we can.

Why there is no ViewModel in this example?

As per the discussion we had in Lab 6, View should not be connected to Model directly. We must always have ViewModel in between View and Model. It doesn't matter if view is a simple "display view" or "data entry view", it should always connected to ViewModel. Reason for not using ViewModel in our project is simplicity. In real time project I strongly recommend you to have ViewModel everywhere.

Is it required to attach Authorize attribute to each and every action method?

No. We can attach it Controller level or Global level also. When attached at controller level, it will be applicable for all the action methods in a controller. When attached at Global level, it will be applicable for all the action method in all the controllers.

Controller Level

```
[Authorize]
public class EmployeeController : Controller
{
....
```

Global level

Step 1 - Open FilterConfig.cs file from App_start folder.

Step 2 - Add one more line RegisterGlobalFilters as follows.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());//Old line
    filters.Add(new AuthorizeAttribute());//New Line
}
```

Step 3 - Attach AllowAnonymous attribute to Authentication controller.

```
[AllowAnonymous]
public class AuthenticationController : Controller
{
```

Step 4 – Execute and Test the application in the same way we did before.

What does filters.Add(new HandleErrorAttribute()) does?

We will discuss this in detail in future lab.

Why AllowAnonymous attribute is required for AuthenticationController?

We have attached Authorize filter at global level. That means now everything is protected including Login and DoLogin action methods. AllowAnonymous opens action method for non-authenticated requests.

How come this RegisterGlobalFilters method inside FilterConfig class invoked?

It was invoked in Application_Startevent written inside Global.asax file.

Lab 18 – Display UserName in the view.

In this lab we will display currently logged in User Name in View.

Step 1 – Add UserName in ViewModel

Open EmployeeListViewModel and add a new property called UserName as follows.

```
public class EmployeeListViewModel
{
    public List<EmployeeViewModel><employeeviewmodel> Employees { get; set; }
    public string UserName { get; set; }
}
</employeeviewmodel>
```

Step 2 – Set value to ViewModel UserName

Open EmployeeController and change Index as follows

```
public ActionResult Index()
{
    EmployeeListViewModel employeeListViewModel = new EmployeeListViewModel();
    employeeListViewModel.UserName = User.Identity.Name; //New Line
......
```

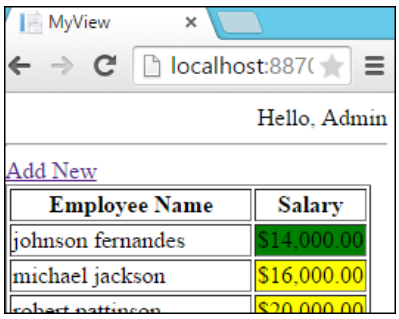
Step 3 – Display UserName in View

Open Index.cshtml view and display UserName as follows.

```
<body>

  <div style="text-align:right"> Hello, @Model.UserName </div>

  <hr />

  <a  href="/Employee/AddNew">Add New</a>

    <div>

       <table border="1"><span style="font-size: 9pt;">
</span>
```

Step 4 – Execute and Test

Press F5 and execute the application. Complete the login process and you will get to see following output.



Lab 19 – Implement Logout

Step 1 – Create Logout link

Open Index.cshtml and create Logout link as follows.

```
<body>

    <div style="text-align:right">Hello, @Model.UserName

    <a href="/Authentication/Logout">Logout</a></div>

    <hr />

    <a  href="/Employee/AddNew">Add New</a>

    <div>

        <table border="1">
```

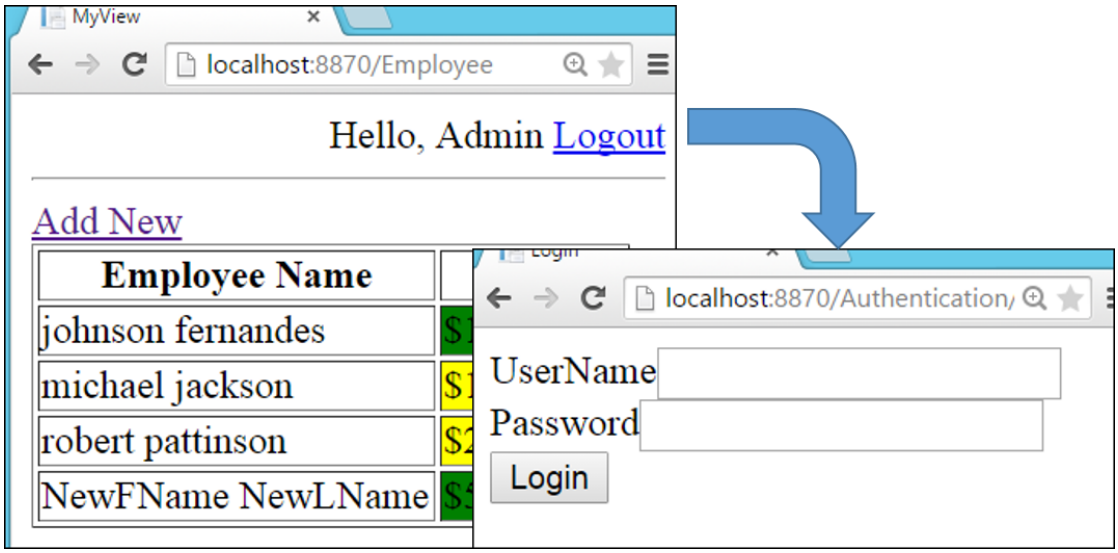
Step 2 – Create Logout Action method.

Open AuthenticationController and add new action method called Logout as follows.

```
public ActionResult Logout()
{
    FormsAuthentication.SignOut();
    return RedirectToAction("Login");
}
```

Step 3 – Execute and Test

Press F5 and execute the application



Lab 20 – Implementing validation in Login Page

Step 1 – Add data annotations

Open UserDetails.cs and add Data Annotation as follows.

```
public class UserDetails
{

[StringLength(7,MinimumLength=2, ErrorMessage = "UserName length should be between 2 and 7")]
    public string UserName { get; set; }
    public string Password { get; set; }
}
```

Step 2 – Display error messages in view

Change Login.cshtml to display error messages.

```
@using (Html.BeginForm("DoLogin", "Authentication", FormMethod.Post))
{
    @Html.LabelFor(c=>c.UserName)
    @Html.TextBoxFor(x=>x.UserName)
    @Html.ValidationMessageFor(x=>x.UserName)
......
```

Note: This time instead of Html.ValidationMessage we have used Html.ValidationMessageFor. Both will do same thing. Html.ValidationMessageFor can be used only when the view is strongly typed view.

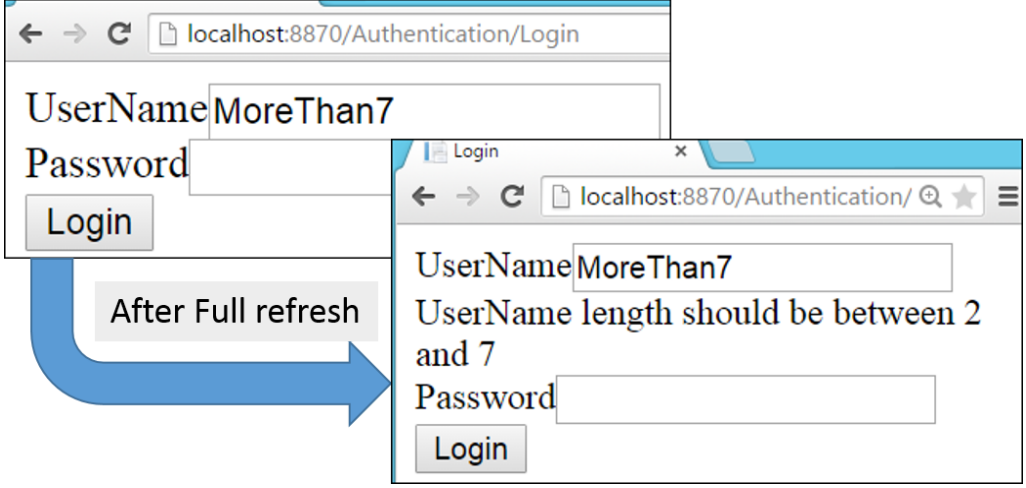
Step 3 – Change DoLogin

Change DoLogin action method as follows

```
[HttpPost]
public ActionResult DoLogin(UserDetails u)
{
    if (ModelState.IsValid)
    {
        EmployeeBusinessLayer bal = new EmployeeBusinessLayer();
        if (bal.IsValidUser(u))
        {
            FormsAuthentication.SetAuthCookie(u.UserName, false);
            return RedirectToAction("Index", "Employee");
        }
        else
        {
            ModelState.AddModelError("CredentialError", "Invalid Username or Password");
            return View("Login");
        }
    }
    else
    {
        return View("Login");
    }
}
```

Step 4- Execute and Test

Press F5 and execute the application.



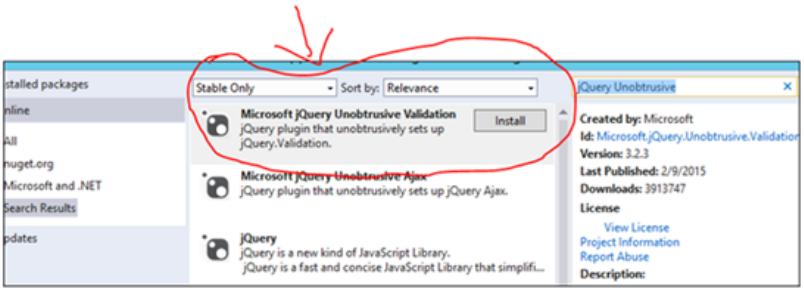
Lab 21 – Implementing Client side validation in Login Page

This time we will do client side validation in a different way.

Step 1 – Download jQuery unobtrusive Validation files.

Right click the project. Select "Manage Nuget packages".

Click on online and search for "jQuery Unobtrusive".



Install "Microsoft jQuery Unobtrusive Valiadtion

Step 2 – Include jQuery Validation files in View

Above steps adds three JavaScript files in Scripts folder.

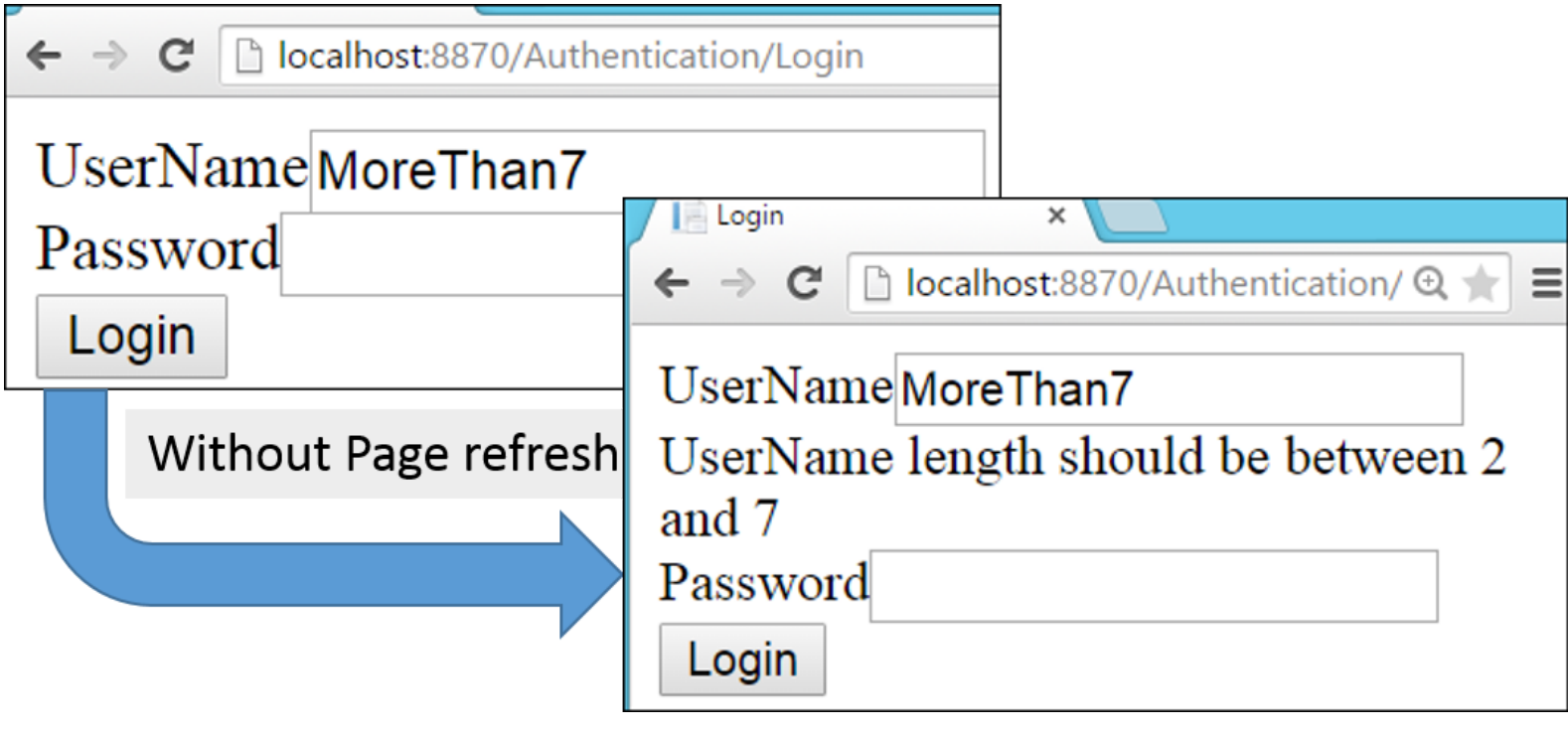
- jQuery-Someversion.js
- jQuery.valiadte.js
- jquery.validate.unobtrusive

Open Login.cshtml and in head section include all three js files (in same sequence)/

```
<script src="~/Scripts/jquery-1.8.0.js"></script>
<script src="~/Scripts/jquery.validate.js"></script>
<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>
```

Step 3 – Execute and Test.

Press F5 and execute the application.



Talk on Lab 21

How come client side validation is implemented?

As you can see, without much effort client side validation is implemented. In Login view, HTML elements are generated using HTML helper classes. Helper functions while generating HTML markup attach some attributes based on the data annotation attributes used.

Example:

```
@Html.TextBoxFor(x=>x.UserName)
@Html.ValidationMessageFor(x=>x.UserName)
```

Above code will generate following HTML.

```
<input data-val="true" data-val-length="UserName length should be between 2 and 7" data-val-length-max="7" data-val-length-min="2" id="UserName" name="UserName"
type="text" value="" />
<span class="field-validation-error" data-valmsg-for="UserName" data-valmsg-replace="true"> </span>
```

These custom HTML attributes will be used by “jQuery Unobtrusive validation files” and thus validation get implemented at client side automatically.

Automatic client side validation is the second advantage of Html helper classes.

What is unobtrusive JavaScript means?

This is what Wikipedia says about it.

Unobtrusive JavaScript is a general approach to the use of JavaScript in web pages. Though the term is not formally defined, its basic principles are generally understood to include:

- Separation of functionality (the "behaviour layer") from a Web page's structure/content and presentation
- Best practices to avoid the problems of traditional JavaScript programming (such as browser inconsistencies and lack of scalability)
- Progressive enhancement to support user agents that may not support advanced JavaScript functionality

Let me define it in layman terms.

“Write your JavaScript in such way that, JavaScript won’t be tightly connected to HTML. JavaScript may access DOM elements, JavaScript may manipulate DOM elements but won’t directly connected to it.”

In the above example, jQuery Unobtrusive JavaScript simply used some input element attributes and implemented client side validation.

Can we use these JavaScript validation without HTML helper classes?

Yes, for that we have to manually attach such attributes to elements.

What is more preferred, Html helper functions or pure HTML?

I personally prefer pure HTML because Html helper functions once again take “full control over HTML” away from us and we already discussed the problems with that.

Secondly let’s talk about a project where instead of jQuery some other JavaScript frameworks/librariesare used. Some other framework like angular. In that case mostly we think about angular validation and in that case these custom HTML validation attributes will go invain.

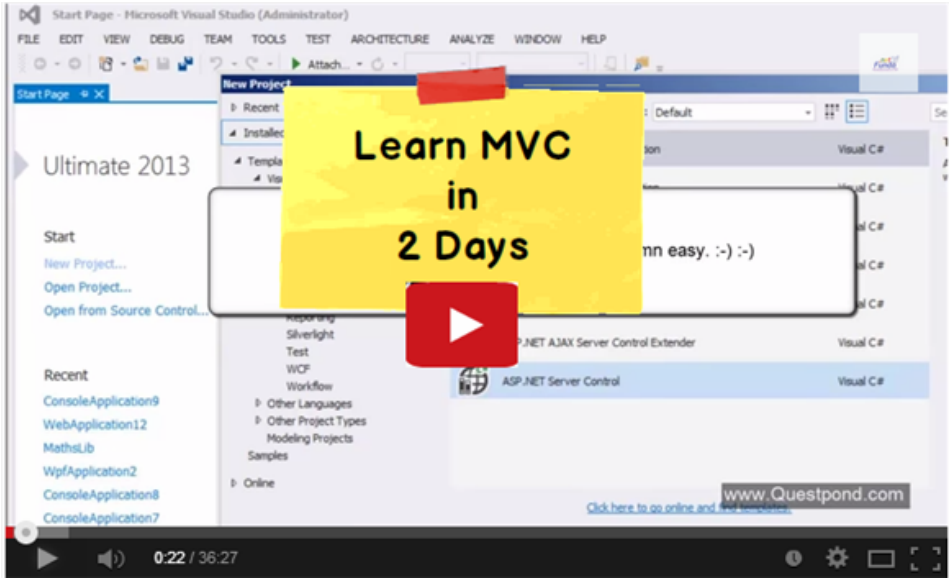
Conclusion

Here we complete our day 4. In Day 5 we will have more advanced stuff and more fun.

Connect us on Facebook, LinkedIn or twitter to stay updated about new releases.

For Offline Technical trainings in Mumbai visit StepByStepSchools.Net For Online Trainings visit JustCompile.com or www.Sukesh-Marla.com

In case you want to start with MVC 5 start with the below video Learn MVC 5 in 2 days.



License

About the Author

Marla Sukesh

Instructor / Trainer Train IT
India

Learning is fun but teaching is awesome.

Code re-usability is my passion ,Teaching and learning is my hobby, Becoming an successful entrepreneur is my goal.

By profession I am a Corporate Trainer.
I do trainings on WCF, MVC, Business Intelligence, Design Patterns, HTML 5, jQuery, JSON and many more Microsoft and non-Micrsoft technologiees.

Find my profile here

My sites

- **justcompile.com**
- **www.sukesh-marla.com**

@Twitter
@Facebook

Comments and Discussions

118 messages have been posted for this article Visit **https://www.codeproject.com/Articles/996832/Learn-MVC-Project-in-Days-Day** to post and view comments on this article, or click **here** to get a print view with messages.