ASP.NET MVC – Part 3

Agenda

- Partial Views
- Using CSS/JS files

Tools

- Visual Studio 2013

Pre-requisite

- ASP.NET MVC – Part 2
- Source code of Part 2 will be used

Version	Last updated	Comments	Modified By
V1.0	10-05-2016		Bilal Shahzad

Brief Introduction

Partial Views

There are many cases when we need to use same HTML (or HTML with functionality) on multiple places. In these cases instead of duplicating the HTML (or logic), we can create partial views. Partial views are not useable if we use them alone. They are supporting views which are used with other views.

Helper class

MVC provides some helper classes to be used in views. “Html” is an example of such classes. This provides many utility functions. “Url” is another such example. It provides functions to play with URLs.

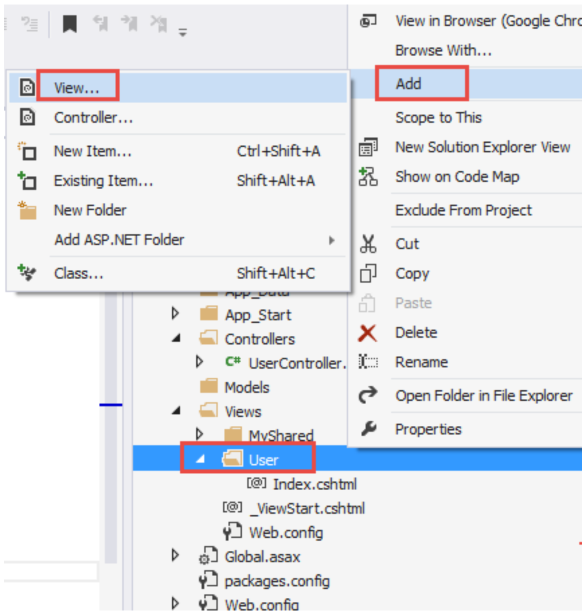
In ASP.NET, paths starting with **~** are called virtual path e.g. “~/Scripts/LoginManager.js”. Here ~ is referring to the virtual directory (if any) where web site is deployed. By default in Visual Studio, ~ maps to / as there is no virtual directory. But in actual production environment (or even in Visual studio), you can have some “virtual” directory. In that case, ~ will map to that virtual directory.

“Url.Content()” function is a helper function which converts virtual path to relative path. So let say, our website is deployed in a virtual directory “BilalWeb” and assume that we’ve a folder “Scripts” in this which further contains “LoginManager.js” file in it. Now in normal scenario, path of our JS file will be “/Scripts/LoginManager.js” but this will not work if we’ve some virtual directory. So we should always use virtual paths (with ~ sign at start).

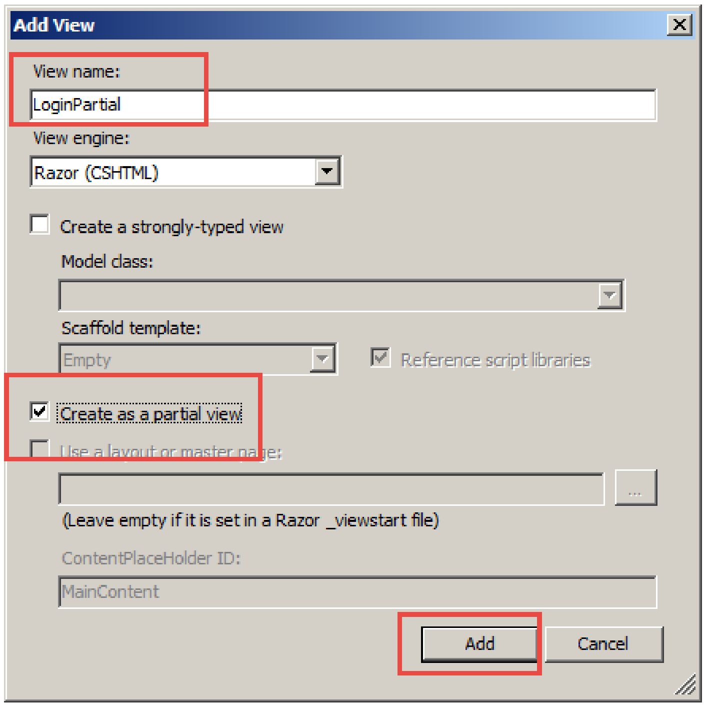
Url.Content(“~/Scripts/LoginManager.js”) will produce “/BilalWeb/Scripts/LoginManager.js” for above example.

Step by Step Walkthrough

1- Let’s add a new view in “User” folder by right clicking on it => Add => View.



2- Give it some name (e.g. LoginPartial) and choose the options as shown in the screenshot below. Click Add. Note that as soon as you will choose “Create as partial view” option, “Layout related options will be disabled”. That is the main difference: A partial view has only some HTML (no html/body etc. tags) and doesn’t have any Layout.



Add View

View name: LoginPartial

View engine: Razor (CSHTML)

☐ Create a strongly-typed view

Model class:

Scaffold template: Empty ☑ Reference script libraries

☑ Create as a partial view

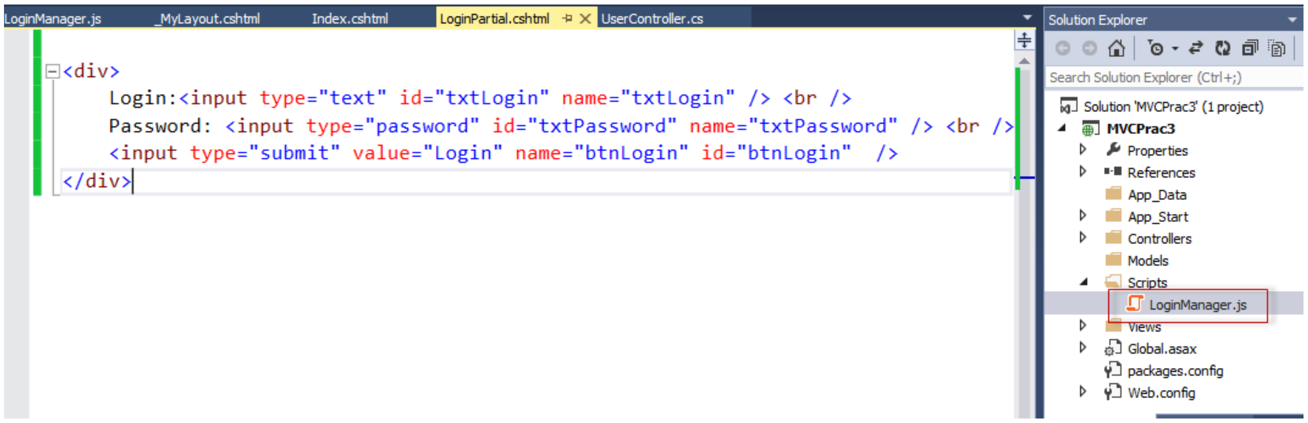
☐ Use a layout or master page:

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID: MainContent

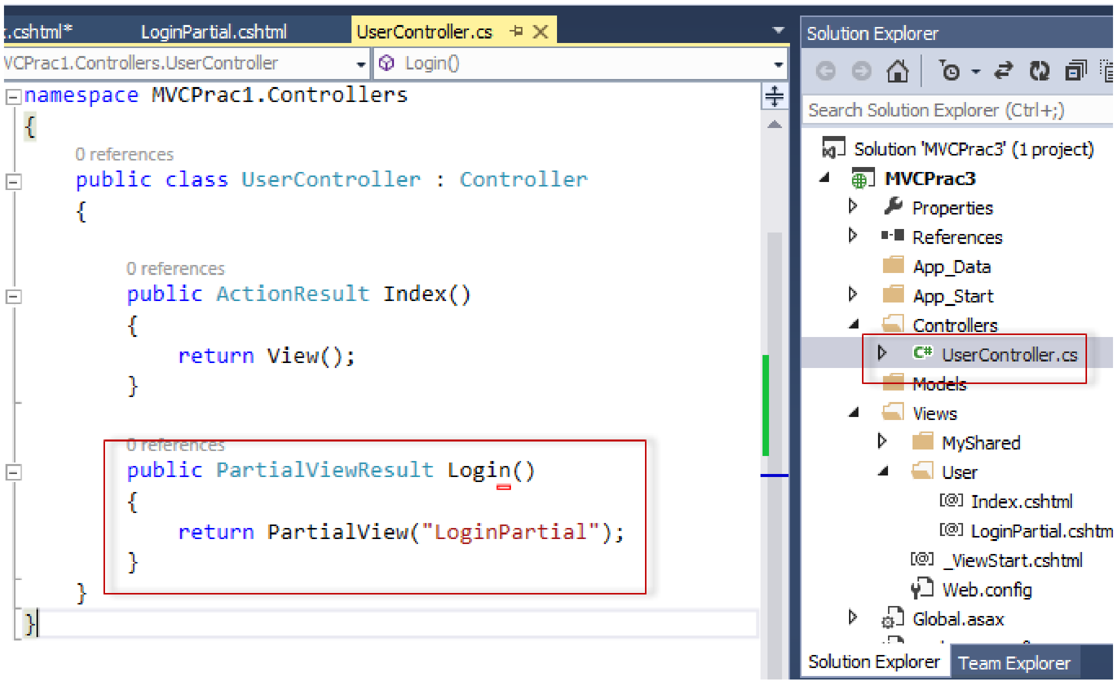
Add Cancel

3- Provide following html to “LoginPartial.cshtml” view. As we are creating this view as partial view therefore we hadn’t set “Layout” property here.



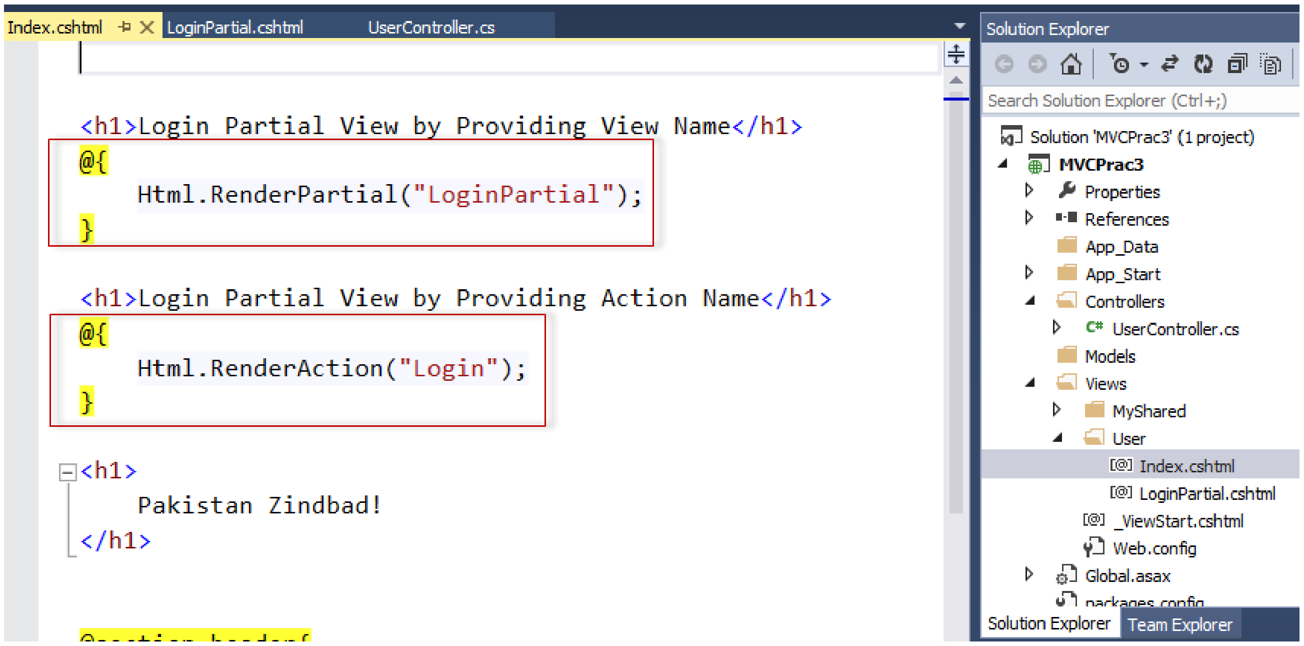
```
<div>
    Login:<input type="text" id="txtLogin" name="txtLogin" /> <br />
    Password: <input type="password" id="txtPassword" name="txtPassword" /> <br />
    <input type="submit" value="Login" name="btnLogin" id="btnLogin"  />
</div>
```

4- Add a new action “LoginPartial” for our partial view. Note that this time we use “PartialView” helper method and “PartialViewResult” as ActionResult. If we don’t provide view name in helper method, by default it is considered that view name is same as action name (in this case it would look for “Login” view if no name will be provided)



```
namespace MVCPrac1.Controllers
{
    0 references
    public class UserController : Controller
    {
        0 references
        public ActionResult Index()
        {
            return View();
        }

        0 references
        public PartialViewResult Login()
        {
            return PartialView("LoginPartial");
        }
    }
}
```

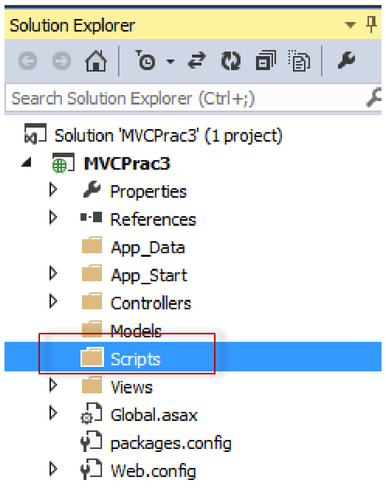
5- Now let's make following changes in your "Index.cshtml" file. "Html" is helper class which is available in view. Here we are rendering our view by two ways. "RenderPartial" method takes the name of partial view and renders it. "RenderAction" takes name of action (or action + controller) and renders the output of that action method.



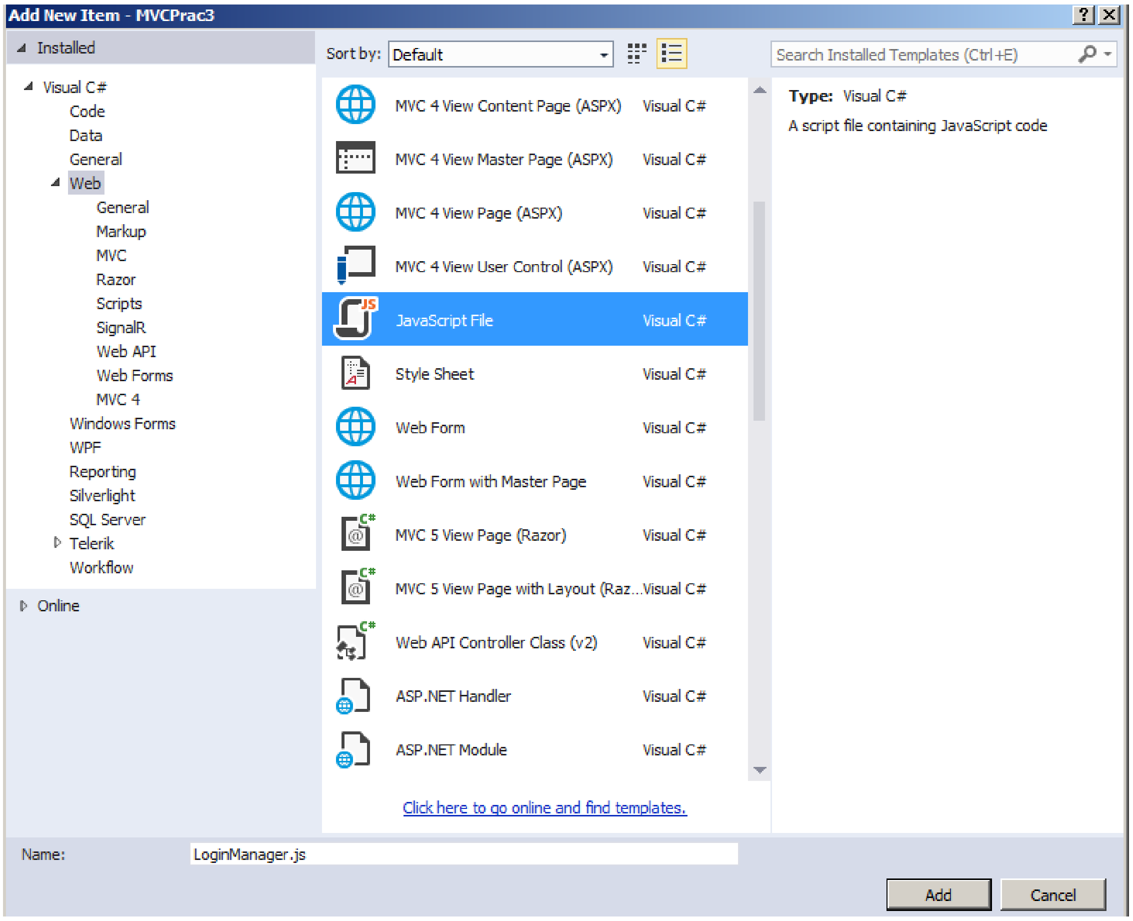
```
<h1>Login Partial View by Providing View Name</h1>
@{
    Html.RenderPartial("LoginPartial");
}

<h1>Login Partial View by Providing Action Name</h1>
@{
    Html.RenderAction("Login");
}

<h1>
    Pakistan Zindbad!
</h1>
```

6- Run the project and verify if everything is working as expected.

7- Now let’s create a folder to manage our script file. Right click on your project => Add => New Folder. Name it “Scripts”.



8- Now let’s create a JS file in “Scripts” folder. Right click on “Scripts” folder => Add New Item. Name it “LoginManager.js”. Click on Add.


Add New Item - MVCPrac3
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
MVC 4
Windows Forms
WPF
Reporting
Silverlight
SQL Server
Telerik
Workflow
Online
Search Installed Templates (Ctrl+E)
MVC 4 View Content Page (ASPX) Visual C#
MVC 4 View Master Page (ASPX) Visual C#
MVC 4 View Page (ASPX) Visual C#
MVC 4 View User Control (ASPX) Visual C#
JavaScript File Visual C#
Style Sheet Visual C#
Web Form Visual C#
Web Form with Master Page Visual C#
MVC 5 View Page (Razor) Visual C#
MVC 5 View Page with Layout (Raz... Visual C#
Web API Controller Class (v2) Visual C#
ASP.NET Handler Visual C#
ASP.NET Module Visual C#
Click here to go online and find templates.
Add
Cancel

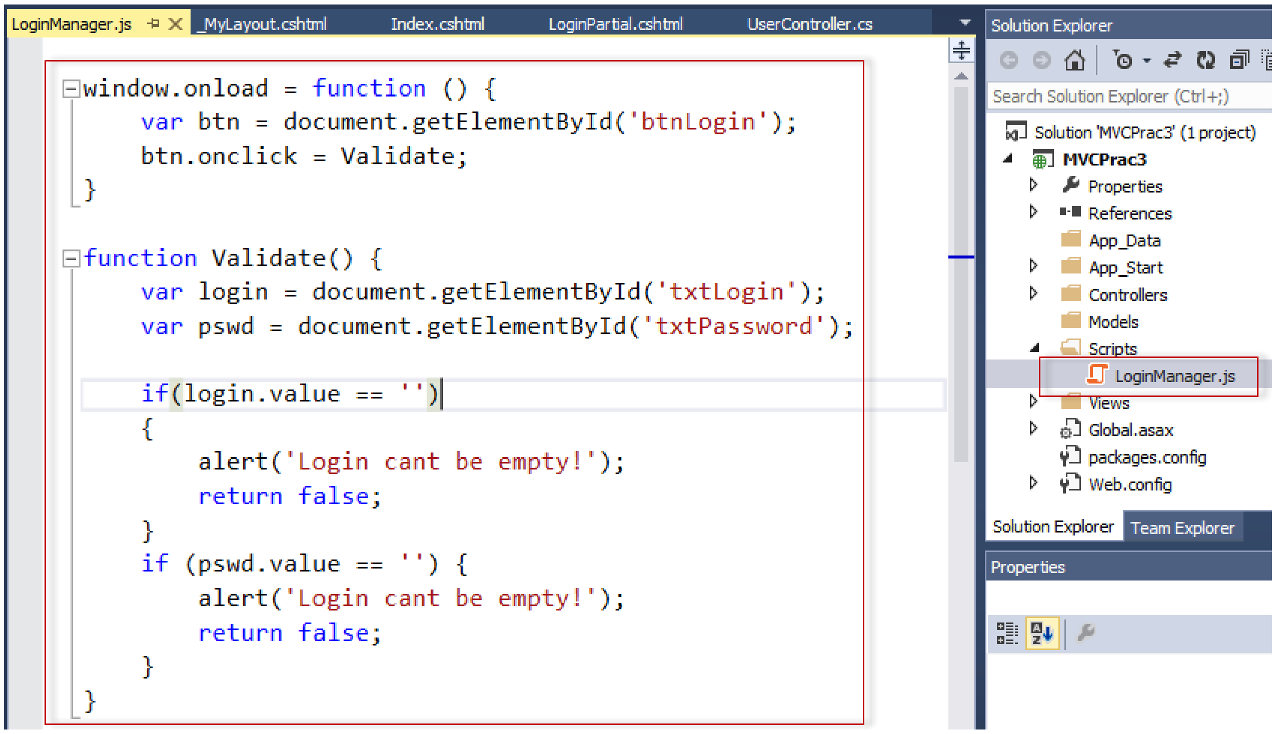
Sort by: Default

Type: Visual C#

A script file containing JavaScript code

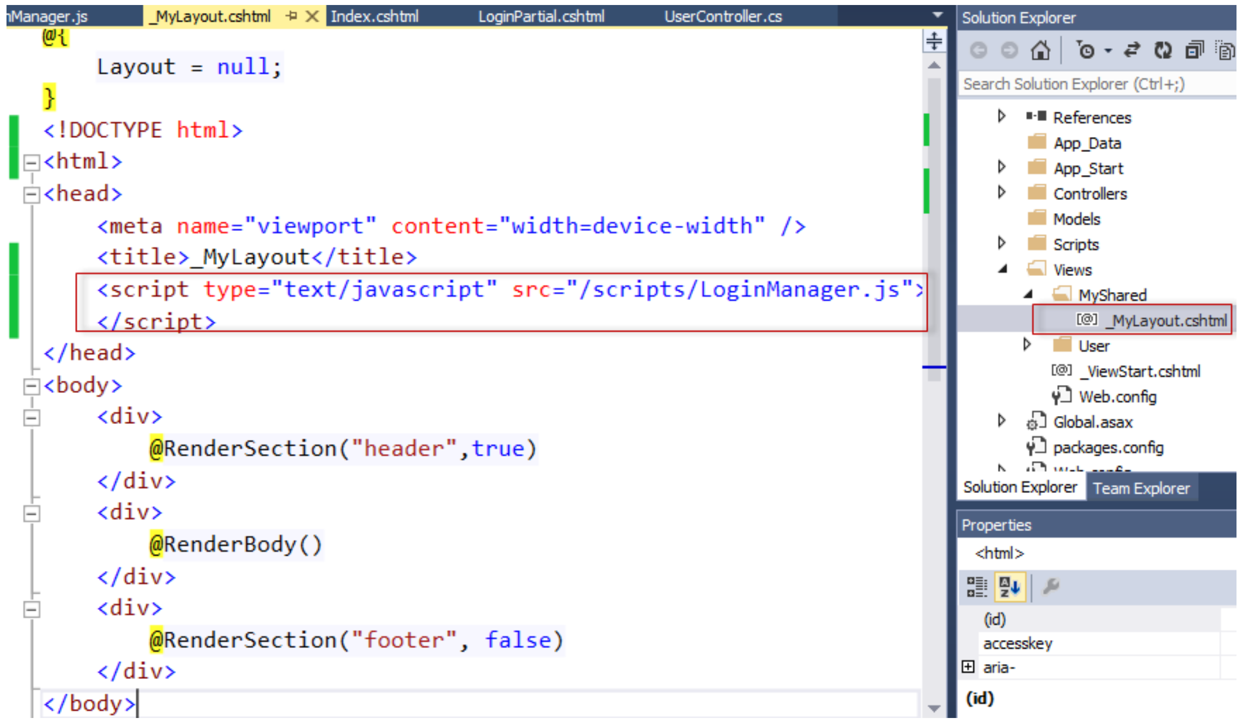
Name: LoginManager.js

9- Add following code in this JS file. This is simple code which validates login form inputs when ‘btnLogin’ button is clicked. Save the changes.



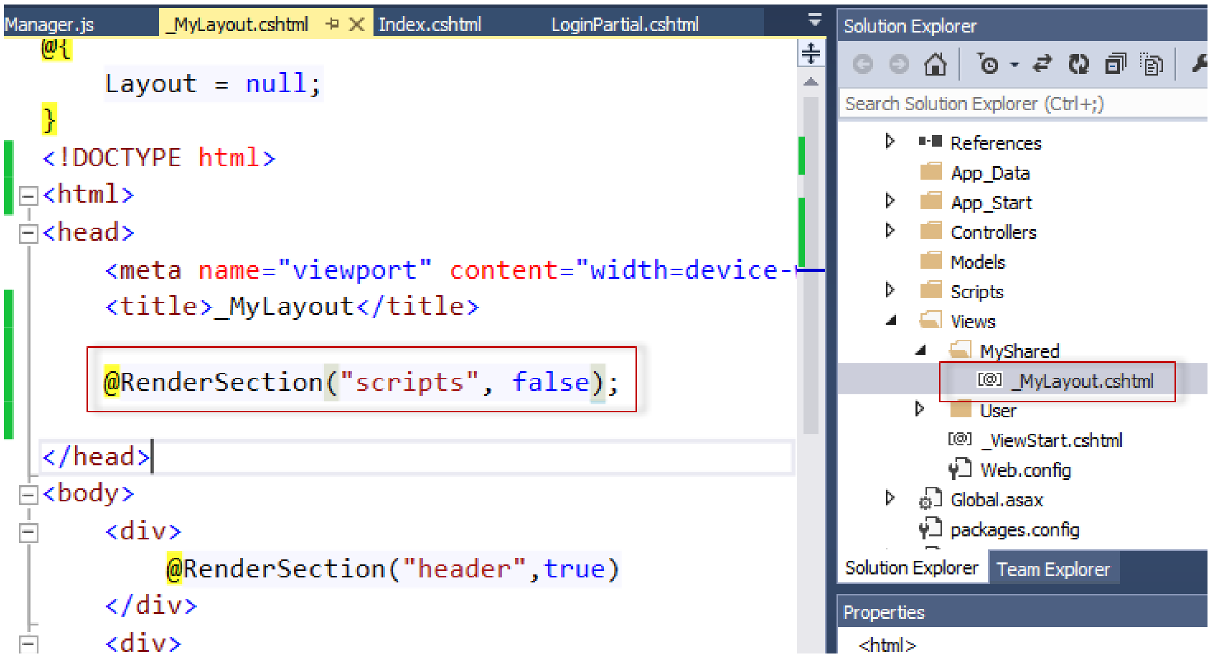
```
window.onload = function () {
    var btn = document.getElementById('btnLogin');
    btn.onclick = Validate;
}

function Validate() {
    var login = document.getElementById('txtLogin');
    var pswd = document.getElementById('txtPassword');

    if(login.value == '')
    {
        alert('Login cant be empty!');
        return false;
    }
    if (pswd.value == '') {
        alert('Login cant be empty!');
        return false;
    }
}
```

10- Now we want to add this JS file. Simplest option can be to add this in our layout file (i.e. _MyLayout.cshtml) as shown below. Note that this JS file will be included in all pages which will use “_MyLayout.cshtml” as layout. What if we want to include this file for some views and want to skip for other views?



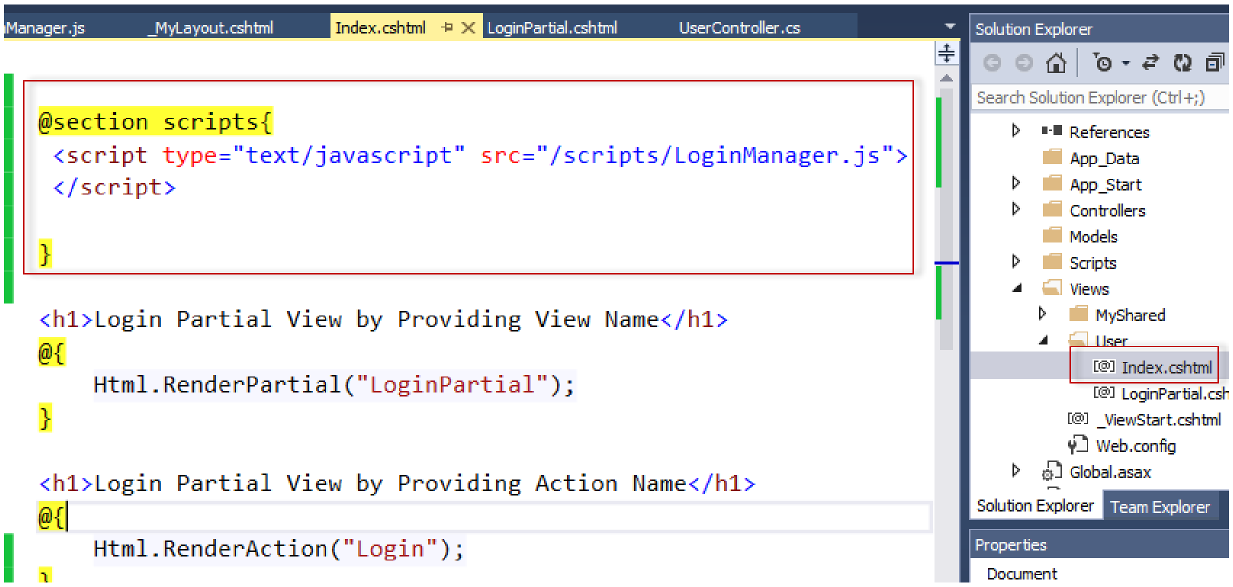
```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>_MyLayout</title>
    <script type="text/javascript" src="/scripts/LoginManager.js">
    </script>
</head>
<body>
    <div>
        @RenderSection("header",true)
    </div>
    <div>
        @RenderBody()
    </div>
    <div>
        @RenderSection("footer", false)
    </div>
</body>
```

11- Run the project and verify if everything is working as expected.
12- Now let's see how to include some file from child view instead of hard coding the file in Layout. For this purpose, we can create a "section" in Layout head and then child view can fill that section. Make following change in "_MyLayout.cshtml" file.



```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-
    <title>_MyLayout</title>

    @RenderSection("scripts", false);

</head>
<body>
    <div>
        @RenderSection("header",true)
    </div>
    <div>
```

13- Now let's make following change in "Index.cshtml" to provide content of "scripts" section.

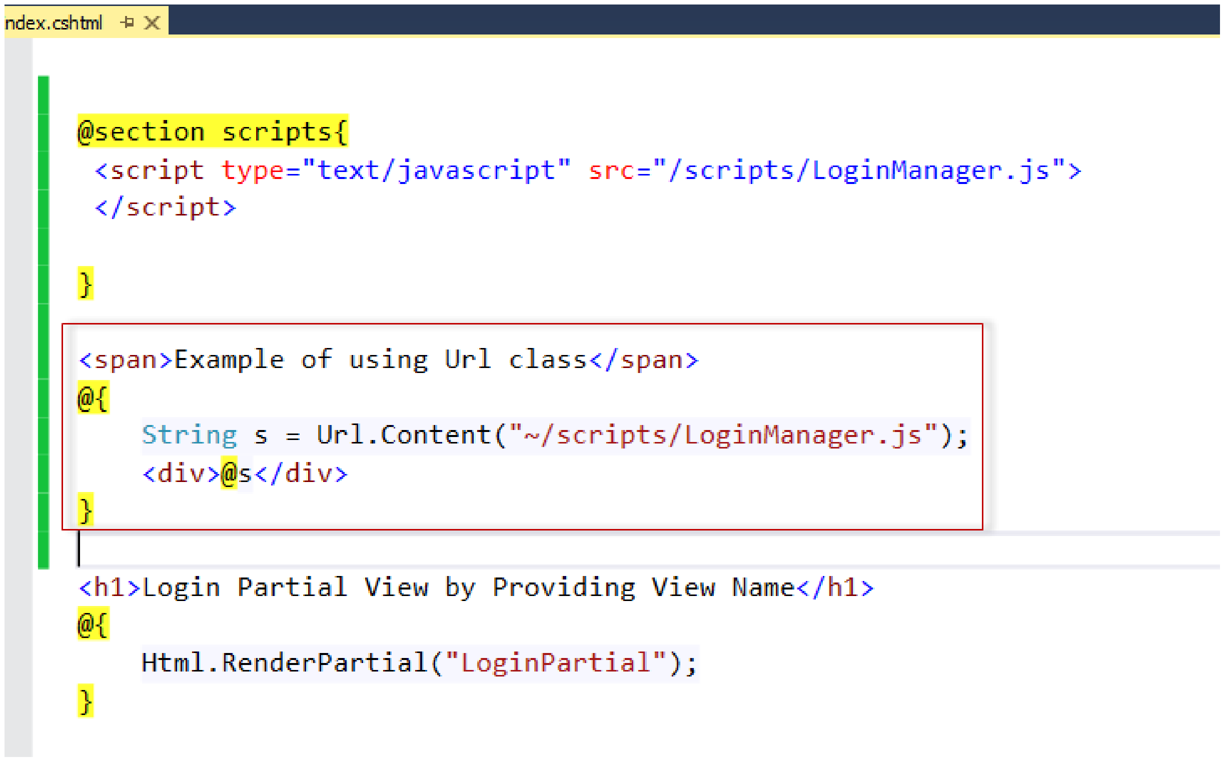


```
@section scripts{
 <script type="text/javascript" src="/scripts/LoginManager.js">
 </script>

}

<h1>Login Partial View by Providing View Name</h1>
@{
    Html.RenderPartial("LoginPartial");
}

<h1>Login Partial View by Providing Action Name</h1>
@{
    Html.RenderAction("Login");
```

14- Run the project and verify if everything is working as expected.

15- Same approach can be used to add a CSS file. You may add a folder to store your CSS files, related images etc. Then you may refer those files in Layout file or in specific views through section approach.

16- Here is an example of using "Url.Content()" function. Make following change in your "Index.cshtml" file. Run the project and check the output of this change. Here we are just converting a virtual path to relative path and showing on the screen.



```
@section scripts{
 <script type="text/javascript" src="/scripts/LoginManager.js">
 </script>

}
<span>Example of using Url class</span>
@{
    String s = Url.Content("~/scripts/LoginManager.js");
    <div>@s</div>
}

<h1>Login Partial View by Providing View Name</h1>
@{
    Html.RenderPartial("LoginPartial");
}
```

Tasks for Practice

Once you are done with above tutorial. Try to work on these tasks.

1- Try to check by providing a normal view name to “RenderPartial()” function.
2- Try to check by providing a normal action name to “RenderAction()” function.
3- Explore difference between “Html.RenderPartial()” and “Html.Partial()”.
4- Explore difference between “Html.RenderAction()” and “Html.Action()”.
5- Create a folder named “mycontent” in project.
6- Add some CSS file in it with sample styling in it.
7- Refer CSS file in your layout and apply style (created in file) on any element.
8- Check some helper methods on this link
    a. https://msdn.microsoft.com/en-us/library/dd410596(v=vs.100).aspx

Useful Links

http://www.asp.net/mvc/overview/views