Articles » Web Development » ASP.NET » General

Learn MVC Project in 7 Days – Bonus Day 2

Marla Sukesh, 27 Jul 2015

This article is a continuation article of the series “Learn MVC project in 7 days.

Download Bonus_Day_2.zip

Introduction

This article is a continuation article of the series “Learn MVC project in 7 days”. As I promised here is the second bonus day article in the series. This is the last article in the series. I believe you have enjoyed the complete series. Thanks for all your support and wish you good luck.

Today we will speak about some topics on Asp.Net MVC which we have not covered in the project.

- Introduction
- Understand TempData in Asp.Net MVC
    - What is TempData?
    - When the TempData values get removed?
    - What will happen if the TempData values are not read?
    - Keep and Peek methods
- Avoid CSRF attacks in Asp.Net MVC
- MVC bundling and Minification
    - Understanding Bundling
    - Understanding Minification
    - Implementing bundling and minification in Asp.Net MVC
- Creating Customer Helper classes
- MVC Unit Testing
- Start with MVC 5
- Conclusion

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

Understand TempData in Asp.Net MVC

For this topic we won’t do any demo. TempData behaves differently in different situations. Incorporating all of the scenario in single project is not possible. That’s why we will just try to cover and understand all the scenarios step by step.

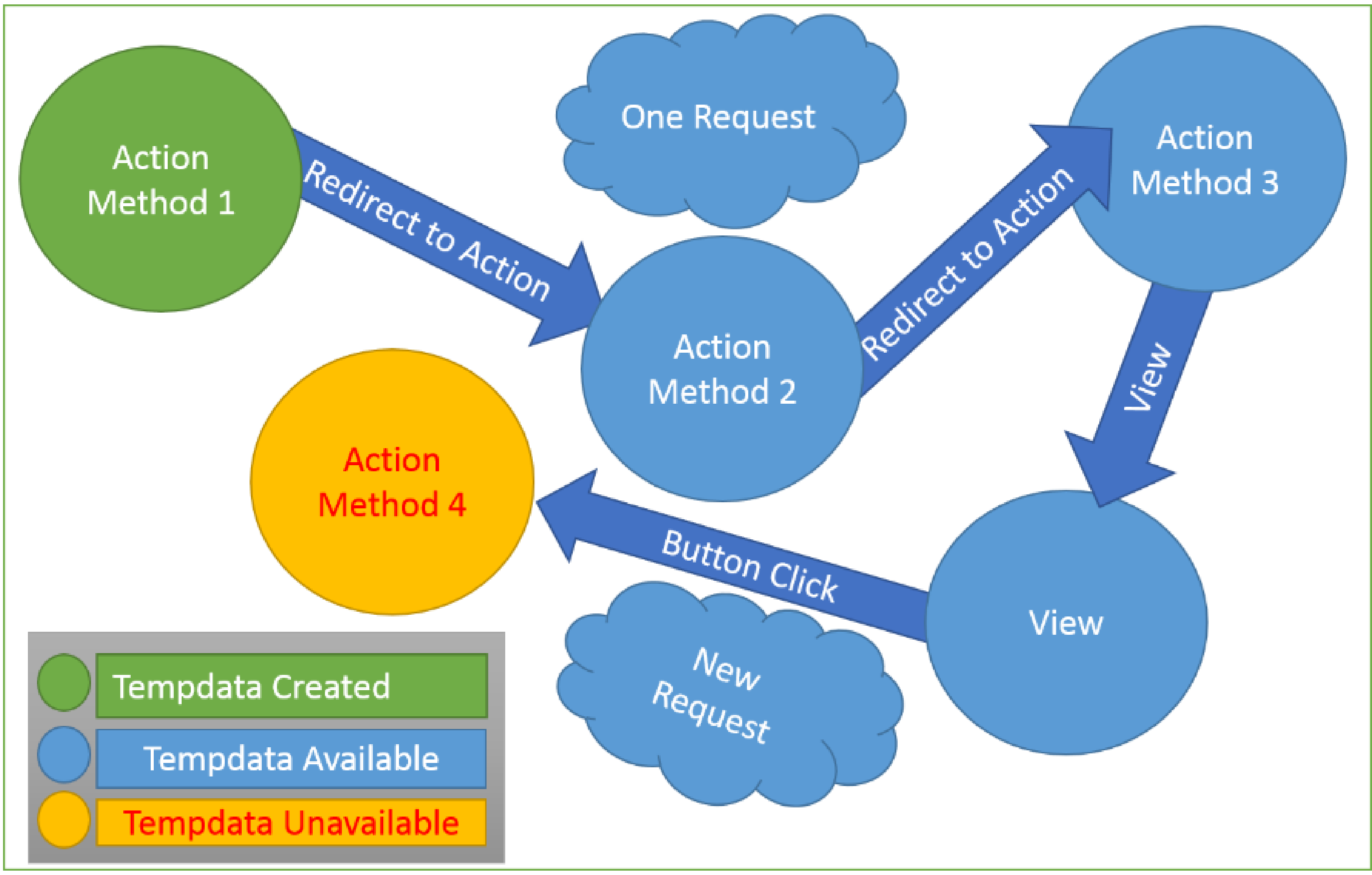
What is TempData?

TempData is a session with shorten life cycle. Just like session it's a key value pair used to hold user specific values. Difference between Session and TempData is, TempData let us maintain data only within single request cycle.

To understand TempData properly you must understand the meaning "Single Request" properly

- User make a request via browser's address bar which is handled by an action method. Let's call it Request 1.
- Now unless end user get a final response it will be called as request 1. Once end user get a response request ends there.
- Now by clicking a hyperlink, button or again with the help of browser's address bar end user can make a new request.

Look at the following image to understand "single request cycle" and TempData



Code Example:

```
public ActionResult M1()
{
    TempData["a"] = "Value";
    string s = TempData["a"].ToString();  // TempData will be available
    return RedirectToAction("M2");
}

public ActionResult M2()
{
    string s = TempData["a"].ToString(); //TempData will be available
    return RedirectToAction ("M3");
}
public ActionResult M3()
{
    string s = TempData["a"].ToString();// TempData will be available
    return view ("Some View"); // TempData will be available inside view alsoJ
}
```

It doesn't matter if it is a view or an action method. Unless request completes values in the TempData will be available.

When the TempData values get removed?

- At the end of the request, values in the TempData get removed automaticallyifvalues are marked for deletion.
- Values will be marked for deletion as soon as we read it.

Example 1

```
public ActionResult M1()
{
    TempData["a"] = "Value";
    return view("MyView");
}
```

```
...
...
...
View Code
...
...
@{
    string s = TempData["a"].ToString(); // TempData is marked for deletion
}
```

Example 2

```
public ActionResult M1()
{
    TempData["a"] = "Value";
    string s = TempData["a"].ToString(); // TempData is marked for deletion
    return view("MyView");  // MyView can access TempData because it will be deleted at the end of the request
}
```

Note: In above two examples values are marked for deletion as soon it get read. Now for the next request this values won't be available because at the end of the current request current TempData get deleted.

What will happen if the TempData values are not read?

As per the final explanation we had, TempData values will be deleted at the end of the current request considering they are marked for deletion in current request and they will get marked for deletion only if we read it. What will happened if we won't read it?

Simple answer, in that case TempData will be available to the next request as well. Now TempData will get removed at the end of the next request if it is marked for deletion in next request else this story continues.

Code Example

```
public ActionResult M1()
{
   TempData["a"] = "Value";
   return view("MyView"); // TempData will be available inside view but let's assume MyView won't read the value
}

//Immediate next request after M1 made by end user
public ActionResult M2()
{
   string s = TempData["a"].ToString(); // TempData will be available and now its marked for deletion
   return view("MyView2"); // TempData will be available inside view also because request is not ended yet.
}
```

In the above sample,

- End user make a request to Action method M1. It creates TempData which will be available throughout that request, but neither inside M1 nor inside MyView values are read and hence values won't get removed at the end of the request.
- Next let's say end user makes a fresh request to M2 action method via hyperlink or some other way. Now TempData created during M1 action method will be available in M2 also. Inside M2 TempData values will be read and hence it get marked for deletion. Current request ends once end user get a final response. In our case end user get the final response only after view completes its rendering. Hence TempData will be available in the View also. Once the View rendering is done TempData get removed.

Keep and Peek methods

Keep

Keep method makes the values in the TempData persistent for next request.

Example 1

Action method

```
public ActionResult M1()
{
   TempData["a"] = "Value";
   TempData["b"] = "Value1";
   TempData["c"] = "Value2";
   TempData.Keep();
   return view("MyView");
}
```

View

```
...
...
```

```
@{
   string s= TempData["a"].ToString();
   string s1= TempData["b"].ToString();
   string s2= TempData["c"].ToString();
}
```

In above example all three TempData values will be available to the next request as well.

Example 2

Action method

```
public ActionResult M1()
{
   TempData["a"] = "Value";
   TempData["b"] = "Value1";

   TempData.Keep();

   TempData["c"] = "Value2";
   return view("MyView");
}
```

View

```
...
...
@{
   string s= TempData["a"].ToString();
   string s1= TempData["b"].ToString();
   string s2= TempData["c"].ToString();
}
```

In the above sample only TempData with key “a” and “b” will be available for next request. TempData with key “c” get removed at the end of current request.

Example 3

Action Method

```
public ActionResult M1()
{
   TempData["a"] = "Value";
   TempData["b"] = "Value1";
   TempData["c"] = "Value2";

   TempData.Keep("a");

   return view("MyView"); // TempData will be available inside view
}
```

View

```
...
...
@{
   string s= TempData["a"].ToString();
   string s1= TempData["b"].ToString();
   string s2= TempData["c"].ToString();
}
```

In the above sample only TempData with key “a” will be available in the next request.

TempData with key b and c will be removed the end of current request.

Peek

Peek will let us retrieve the TempData value without marking it for deletion.

Example without Peek

```
public ActionResult M1()
{
    TempData["a"] = "Value";
    return RedirectToAction("M2");
}
public ActionResult M2()
{
    string s = TempData["a"].ToString(); //TempData will be marked for deletion
    return view ("Some View");
}
```

```
.
.
.
//Next request
public ActionResult M3()
{
    string s = TempData["a"].ToString(); // Will get an Exception
    …
}
```

In above example TempData values (created inside Action method M1) won't be available for next request because it already used in current request (in Action method M2).

When user make a fresh request to action method M3, null reference exception will be thrown.

Example with Peek

```
public ActionResult M1()
{
    TempData["a"] = "Value";
    return RedirectToAction("M2");
}
public ActionResult M2()
{
    string s = TempData.Peek("a").ToString(); // TempData values will be read but will not be marked for deletion
    return view ("Some View");
}
.
.
.
public ActionResult M3()
{
    string s = TempData["a"].ToString(); // Will just work
    …
}
```

In above example TempData value will be available in next request also.

Avoid CSRF attacks in ASP.NET MVC

Now before we go and learn to avoid it, first let's learn about it.

CSRF stands for Cross site request forgery.

Here hacker will develop a two faced UI. Two faced UI means, a UI which will do something different according to end user but in reality it will do something different.

To understand it better let's do a small demo.

Step 1 – Open Day 7 Project

Open the project we have completed in Day 7

Step 2 – Execute the project

Press F5 and execute the project. Complete the login process with Admin user.

Step 3 – Create forgery project

Open a fresh Visual studio. Select File >> New >> Project. Select Web section from left panel and "Asp.Net Web Application" from right. Call it "ForgeryProject" and click Ok.

Select Empty template and Web Forms references and click Ok.

(In this demo you are creating this forgery project but in real time this will be created by someone else who want to hack into your system.)

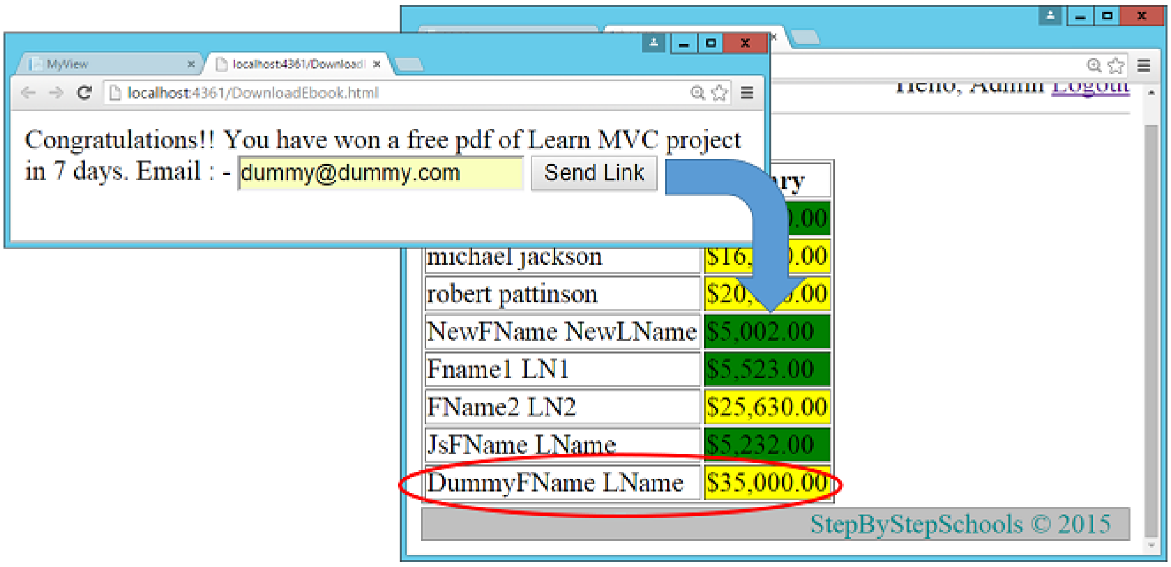
Step 4 – Create a DownloadEbook option

Create a HTML page called DownloadEbook.html in the newly created project as follows.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <form action="http://localhost:8870/Employee/SaveEmployee" method="post" id="EmployeeForm">
        <input type="hidden" name="FirstName" value="DummyFName" />
        <input type="hidden" name="LastName" value="LName" />
        <input type="hidden" name="Salary" value="35000" />
        <input type="hidden" name="BtnSubmit" value="Save Employee" />
        Congratulations!! You have won a free pdf of Learn MVC project in 7 days.
        Email : - <input type="text" name="Whatver" value="" />
        <input type="submit" name="MyButton" value="Send Link" />
    </form>
</body>
</html>
```

Step 4 – Execute the project

Press F5 and execute the application.



Congratulations!! You have won a free pdf of Learn MVC project in 7 days. Email : - dummy@dummy.com

michael jackson	$16, .00
robert pattinson	$20, .00
NewFName NewLName	$5,002.00
Fname1 LN1	$5,523.00
FName2 LN2	$25,630.00
JsFName LName	$5,232.00
DummyFName LName	$35,000.00

StepByStepSchools © 2015

Explanation

You were expecting to get a download pdf link but it actually created a new employee.

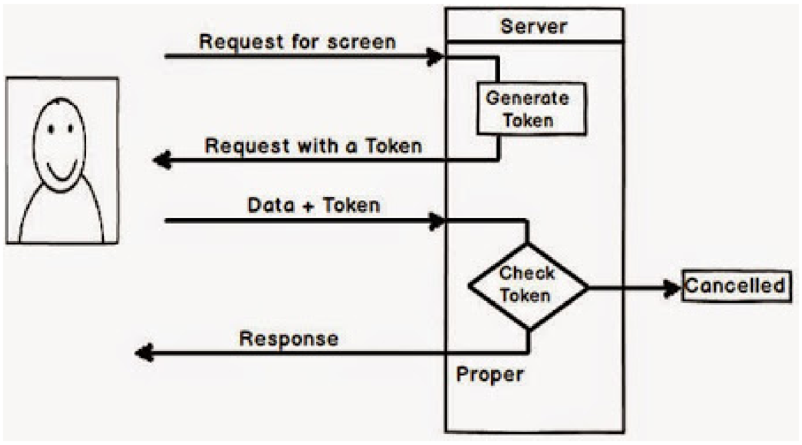
Let's understand what exactly happened.

- End user is able to create an employee only if he is an Authenticated Admin user.
- In our case Admin user will login using his admin credentials and after sometime he will go and open a new browser instance (or browser tab) and start doing some surfing. (Remember his current session is active and he is still authenticated user for our application)
- All of the sudden above forgery page (created by some enemy) comes in front of him and he/she click the "Send link" button.
- "Send link" button will actually send a post request to our application. You know what happens next.

This is called CSRF attack.

Solution for this problem in tokens.

- In actual data entry screen server will inject a secret token in the form of hidden field.
- It means when post request is made from actual data entry screen, secret token will be sent as a part of posted data.
- At server side received token will be validated against generated token and if they are not same request won't be served.



Now let's implement solution for it.

Step 1 – Inject token in original data entry screen

Open CreateEmployee.cshtml from "~/Views/Employee" folder in the Day 7 project and inject secret token as follows.

```
<td colspan="2">
    @Html.AntiForgeryToken()
    <input type="submit" name="BtnSubmit" value="Save Employee" onclick="return IsValid();" />
    <input type="submit" name="BtnSubmit" value="Cancel" />
    <input type="button" name="BtnReset" value="Reset" onclick="ResetForm();" />
</td>
```

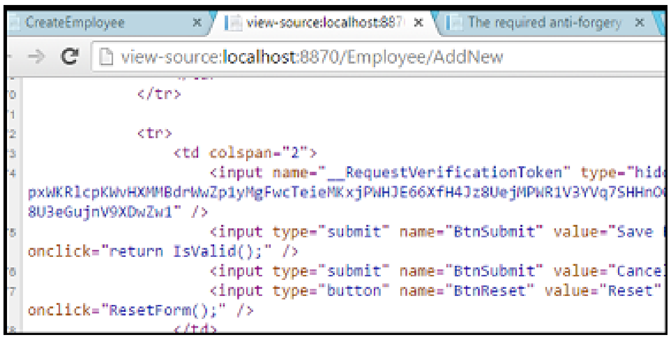
Step 2 –Enable forgery validation on Action Method

Attach ValidateAntiForgeryToken attribute to SaveEmployee action method of EmployeeController as follows.

```
[AdminFilter]
[HeaderFooterFilter]
[ValidateAntiForgeryToken]
public ActionResult SaveEmployee(Employee e, string BtnSubmit)
{
   switch (BtnSubmit)
   {
```

Step 3 – Check the generated token in Data Entry screen

Press F5 and execute the application. Complete the login process. Navigate to AddNew screen and check the view source.

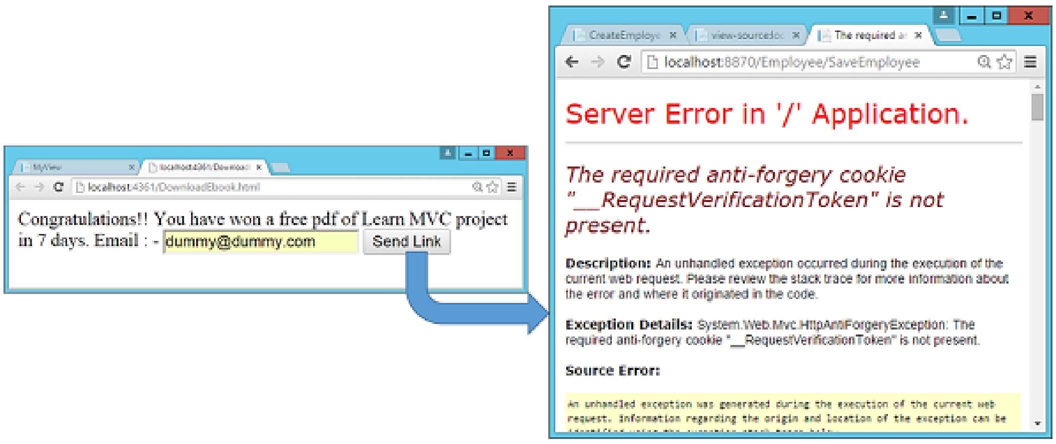

CreateEmployee
view-source:localhost:887
The required anti-forgery
view-source:localhost:8870/Employee/AddNew

```
            </tr>

            <tr>
                <td colspan="2">
                    <input name="__RequestVerificationToken" type="hid
pxWKRlcpKWvHXMMBdrWwZp1yMgFwcTeieMKxjPWHJE66XfH4Jz8UejMPWR1V3YVq7SHHnO
8U3eGujnV9XDwZw1" />
                    <input type="submit" name="BtnSubmit" value="Save 
onclick="return IsValid();" />
                    <input type="submit" name="BtnSubmit" value="Cance
                    <input type="button" name="BtnReset" value="Reset"
onclick="ResetForm();" />
                </td>
```

Step 4 – Check forgery

Keep the project login open and execute the forgery project and perform the same testing we did before.



Congratulations!! You have won a free pdf of Learn MVC project in 7 days. Email : - dummy@dummy.com

Server Error in '/' Application.

The required anti-forgery cookie "__RequestVerificationToken" is not present.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Web.Mvc.HttpAntiForgeryException: The required anti-forgery cookie "__RequestVerificationToken" is not present.

Source Error:

```
An unhandled exception was generated during the execution of the current web
request. Information regarding the origin and location of the exception can be
```

Now it's secured.

MVC bundling and Minification

Understanding Bundling

We cannot imagine a project without CSS and JavaScript file.

Bundling is concept of combining multiple JavaScript or CSS files into a single fileat runtime. Now the million dollar question is why should we combine?

To understand this better let's do a demo.

Step 1 – Execute the project

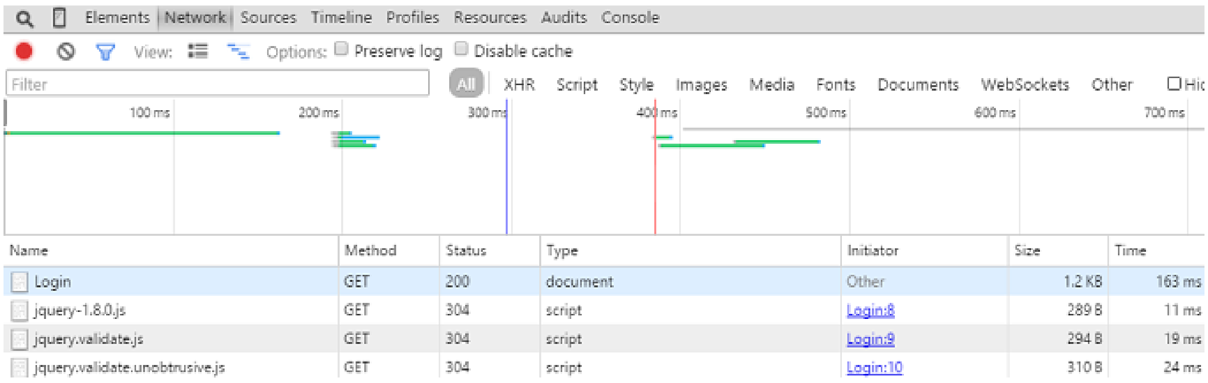
Execute our project again. Perform execution in chrome.

Step 2 – Open chrome developer tools

In chrome press “cntrl + shift + I” to open developer tools. Now navigate to “Network tab”

Step 3 – Open Login

Now make a request to Login page and check the network tab again.



Name	Method	Status	Type	Initiator	Size	Time
Login	GET	200	document	Other	1.2 KB	163 ms
jquery-1.8.0.js	GET	304	script	Login:8	289 B	11 ms
jquery.validate.js	GET	304	script	Login:9	294 B	19 ms
jquery.validate.unobtrusive.js	GET	304	script	Login:10	310 B	24 ms

If you remember we have implemented Unobtrusive client side validation in login view which require three JavaScript files. So when Login View is requested it makes four request calls:-

- One for the Login View.
- Three requests for JavaScript files “jQuery.js”, “jQuery.validate.js” and “jQuery.validate.unobtrusive.js”.

Just try to imagine a situation when we have lots of JavaScript file. It leads to multiple requests thus decreases performance.

Solution will be combine all the JS files into single bundle and request it in a single request as a single unit. This process is termed as bundling.

We will learn how to perform bundling using Asp.Net MVC soon.

Understanding Minification

Minification reduces the size of script and CSS files by removing blank spaces, comments etc. For example below is a simple JavaScript code with comments.

```
// This is test
var x = 0;
x = x + 1;
x = x * 2;
```

After implementing Minification the JavaScript code looks something as below. You can see how whitespaces and comments are removed to minimize file size and thus increases performance as the file size has get reduced.

```
var x=0;x=x+1;x=x*2;
```

Implementing bundling and minification in Asp.Net MVC

Step 1 – Create Script bundles

Open BundleConfig.cs file from App_Start folder. You will find a RegisterBundle method. Remove all existing code inside it and redefine it as follows.

```
public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jqueryValidation").Include(
                "~/Scripts/jquery-1.8.0.js").Include(
                "~/Scripts/jquery.validate.js").Include(
                "~/Scripts/jquery.validate.unobtrusive.js"));
}
```

Step 2 - Include bundle in Login View

Open Login.cshtml view from “~/Views/Authentication” folder.

Remove the references of three script files and include it has one bundle.

```
    <title>Login</title>
   @Scripts.Render("~/bundles/jqueryValidation")
</head>
<body>
```

Step 3 – Execute and test

Execute the application and check the network tab again.

Unfortunately you will not find any difference.

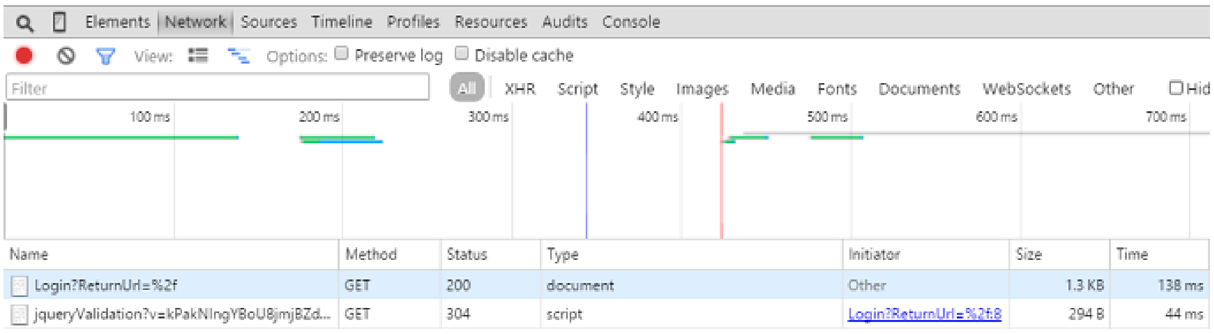
Step 4 – Enable bundling and minification

Open Global.asax file and in put following line in the top of Application_Start event.

```
BundleTable.EnableOptimizations = true;
```

Step 5 – Execute and Test

Perform the same test again.



Name	Method	Status	Type	Initiator	Size	Time
Login?ReturnUrl=%2f	GET	200	document	Other	1.3 KB	138 ms
jqueryValidation?v=kPakNIngYBoU8jmjBZd...	GET	304	script	Login?ReturnUrl=%2f:8	294 B	44 ms

As you can see a single file is requested instead of three. All three JavaScript files are bundled into one. Check the size. It is just 294B. Earlier it was (289+294+310). It proves that minification also did his work.

(Don't forget to check the validation. ☻)

In the same we create bundles for CSS files called StyleBundle.

Creating Customer Helper classes

It's going to be C# more than Asp.Net MVC.

In this article we will create a custom helper function for submit button.

Step 1 – Create a CustomerHelperClass

Create a new folder called Extensions in the project and create a new class called CustomHelperMethods inside it.

Step 2 – Make class a static class

Now make above class a static class because that's going to be the first requirement for extension methods.

```
namespace WebApplication1.Extenions
{
    public static class CustomHelperMethods
    {
    }
}
```

Step 3 –Create an extension method

Put using statement in the class follows

```
usingSystem.Web.Mvc;
```

Create a new static extension method called Submit inside above class as follows.

```
public static MvcHtmlString Submit(this HtmlHelper helper, string displayText)
{
    return MvcHtmlString.Create
        (
        string.Format("<input type='submit' name=Btn{0} id=Btn{0} value={0} />",displayText)
        );
}
```

Step 4 – Change Login View

Open Login View from "~/Views/Authenticate" folder and put using statement in the as the follows.

```
@using WebApplication1.Extenions
```

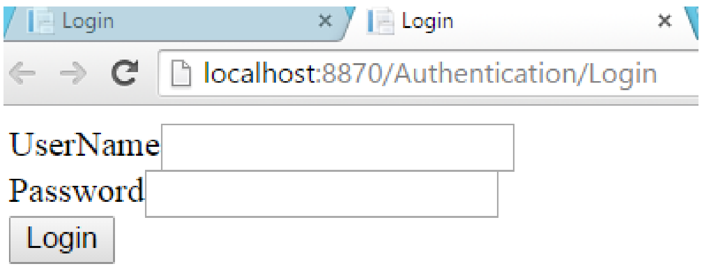
Now remove input type submit and put following code instead of that.

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

    @Html.Submit("Login")
}
...
...
```

Step 5 - Execute and Test

Press F5 and execute the application. Check the output.



MVC Unit Testing

For this we will not do a complete demo. I want you to take it as an assignment.

For reference check below article.

http://www.codeproject.com/Articles/763928/WebControls

Start with MVC 5

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

Learn ASP.NET MVC 5 (Model view controller) step by step in 2 days (16 hours)
Learn MVC in 2 Days

Conclusion

So thank you very much everyone for your wonderful comments and feedbacks.

Your comments, Mails always motivates us do more. Put your thoughts and comments below or send mail to SukeshMarla@Gmail.com

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

Have fun and keep learning. We will meet again with a new series ☻

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

3 messages have been posted for this article Visit **https://www.codeproject.com/Articles/1013142/Learn-MVC-Project-in-Days-Bonus-Day** to post and view comments on this article, or click **here** to get a print view with messages.