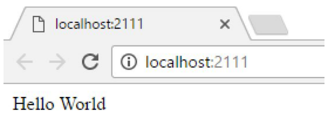
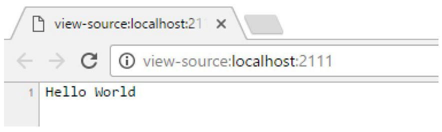
Views And Layouts

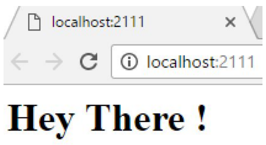
ASP.NET MVC C08



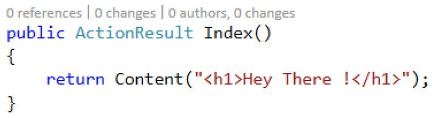
Hello World



```
Hello World
```



Hey There !

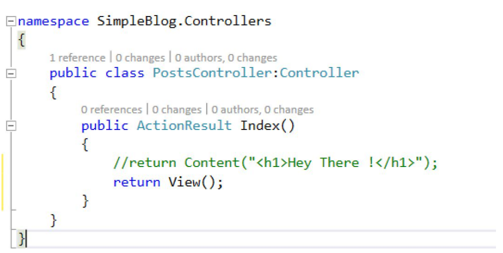


```
0 references | 0 changes | 0 authors, 0 changes
public ActionResult Index()
{
    return Content("<h1>Hey There !</h1>");
}
```

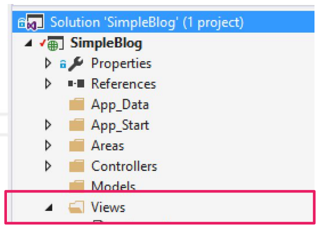
Write Some HTML

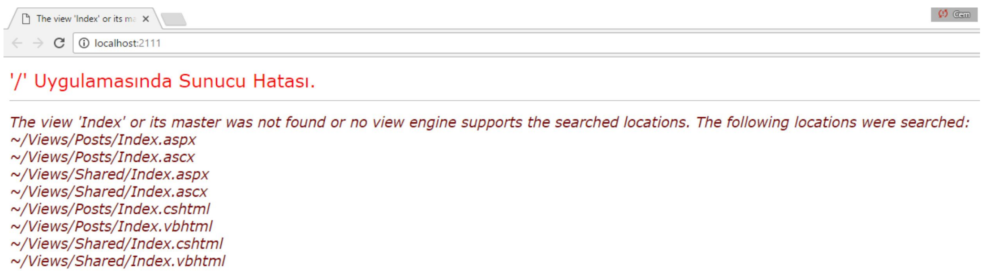
We talked about Routing, Controllers, Actions, Areas.
We actually start to write some HTML. If I right click on the Hello Wolrd browser window, and click View Source, we are getting just the text "Hello World".There is no HTML. Of course, that does not mean that, we can go our Controller , Posts Controller find the Index action and find Hello World and change it with <h1>Hey There </h1>.

But obviosuly this does not scale. We are not going to be writing all our view logic and html in Content. Instead of this we use taking advantage of Razor.



```
namespace SimpleBlog.Controllers
{
    1 reference | 0 changes | 0 authors, 0 changes
    public class PostsController:Controller
    {
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult Index()
        {
            //return Content("<h1>Hey There !</h1>");
            return View();
        }
    }
}
```





'/' Uygulamasında Sunucu Hatası.

The view 'Index' or its master was not found or no view engine supports the searched locations. The following locations were searched:
~/Views/Posts/Index.aspx
~/Views/Posts/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
~/Views/Posts/Index.cshtml
~/Views/Posts/Index.vbhtml
~/Views/Shared/Index.cshtml
~/Views/Shared/Index.vbhtml

Let's start using Razor View Engine

Razor is simply a templating language. It is actually very very interesting. It is realy cool. It does is, Razor allows us to write HTML and c# at the same time.

First I am gonna do is returning a View(). After rebuilding our project, refresh it and get an error. The engine searches for ~/Views/Posts/Index.aspx , ~/Views/Posts/Index.ascx,~/Views/Shared/Index.aspx,~/Views/Shared/Index.ascx ,~/Views/Posts/Index.cshtml,~/Views/Posts/Index.vbhtml,~/Views/Shared/Index.cshtml,~/Views/Shared/Index.vbhmtl files.

What are these file? These files are current valid locations and extensions for a view that is supported by the view engines allowed by ASP.NET MVC for this proejct.

aspx, and ascx are web form views. We are not using that view engine. It is still searching for them.

We wanna use Razor. We wanna use cshtml and vbhtml. So how Razor seperate c# and html code ? It also works with vb and seperate VB.Net code and HTML. Razor supports two extensions. cshtml, vbhtml.

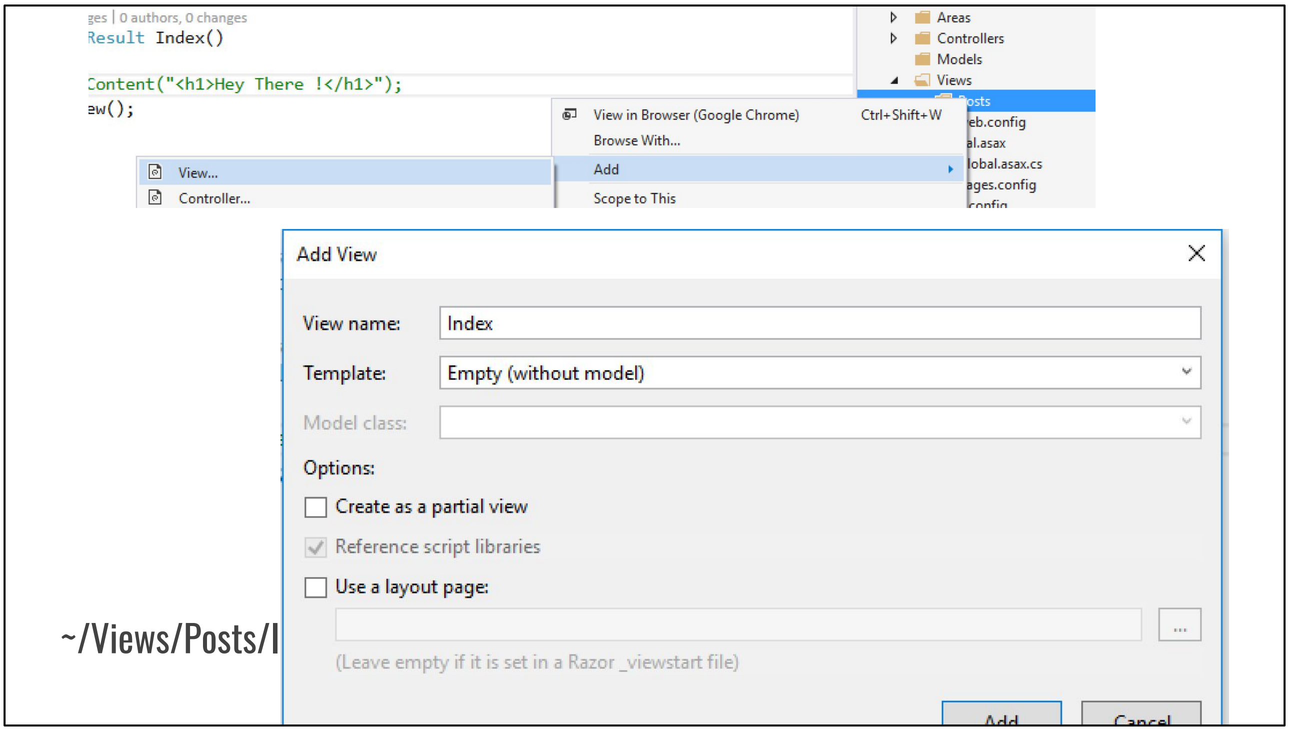
We are goint to be focusing on cshtml. Now we down in our paths to two. ~/Views/Posts/Index.cshtml and ~Views/Shared/Index.cshtml.
There is something interesting about this. What is interesting, it is searching based of the convention. It is searching based of the name of the Controller and the name of the action. So posts is the controller and index is the action.

It also searches under the Shared folder minus the controller. So, that means up we do not want to do it in the shared folder otherwise we wanna be able to differantiate between different actions that are similarly named between controllers. We want to insert it to ~Views/Posts/Index.cshtml file.

The error is “The view Index or its master was not found or no view engine supports the searched loacations. The following locations were searched ...“

This error is very important. Remeber, I said it does not matter where you put your models and controllers but it does matter where you put your views. That is because ASP.NET MVC automatically has a default path that it is searches for based of conventions to locate the view file that is gonna render when a controller returns a view.

And another thing there is a ~ . ~ means root folder of our web project. so ~/Views is the folder shown in left top figure.



```
ges | 0 authors, 0 changes
Result Index()

Content("<h1>Hey There !</h1>");
ew();
```

Add View

View name: Index

Template: Empty (without model)

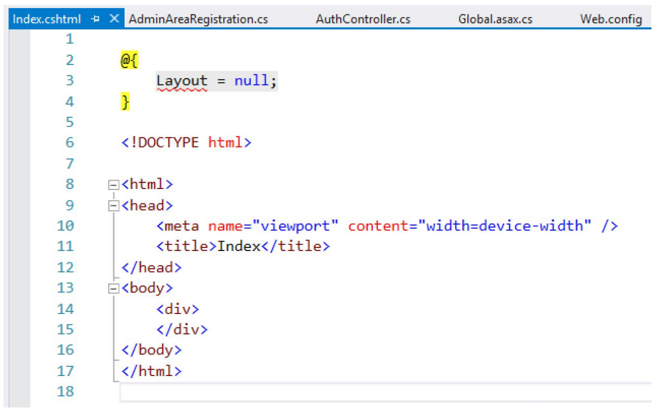
Model class:

Options:

- [ ] Create as a partial view
- [x] Reference script libraries
- [ ] Use a layout page:

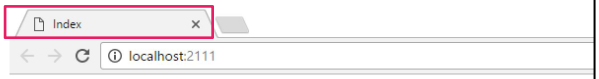
(Leave empty if it is set in a Razor _viewstart file)

This is the file we want. So that is the location we put our view. I am goint to Views. Create a new folder names Posts. Right click on the Posts and click View. I will call the view name Index.That is corresponds the file name. We do not use a template yet. We are not wary about Layout yet.



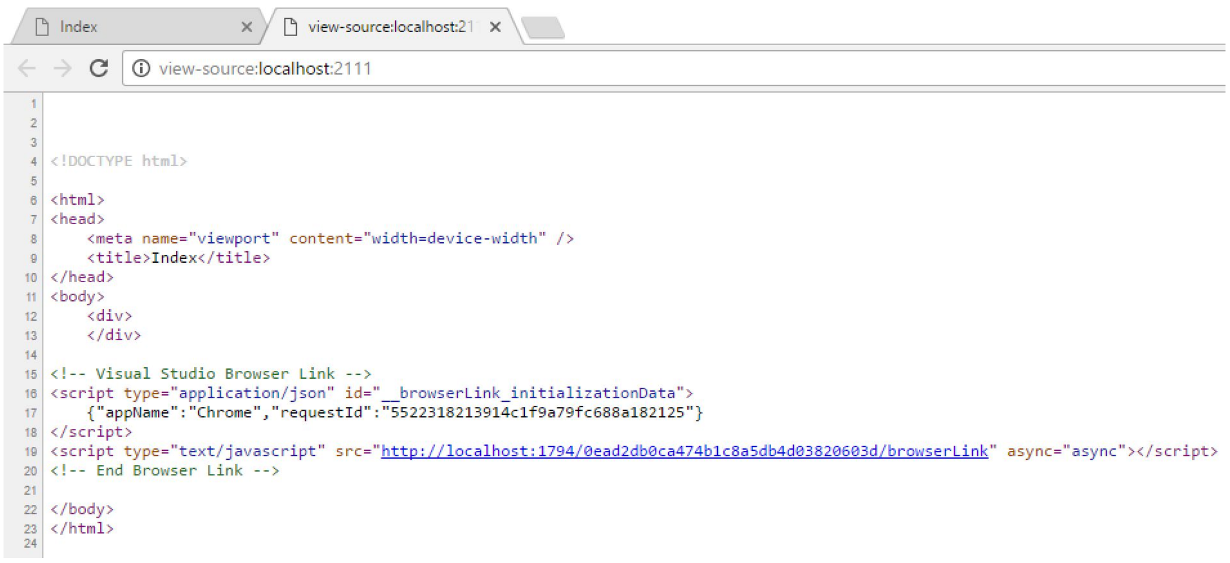
Index.cshtml AdminAreaRegistration.cs AuthController.cs Global.asax.cs Web.config

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```



Our First Razor File

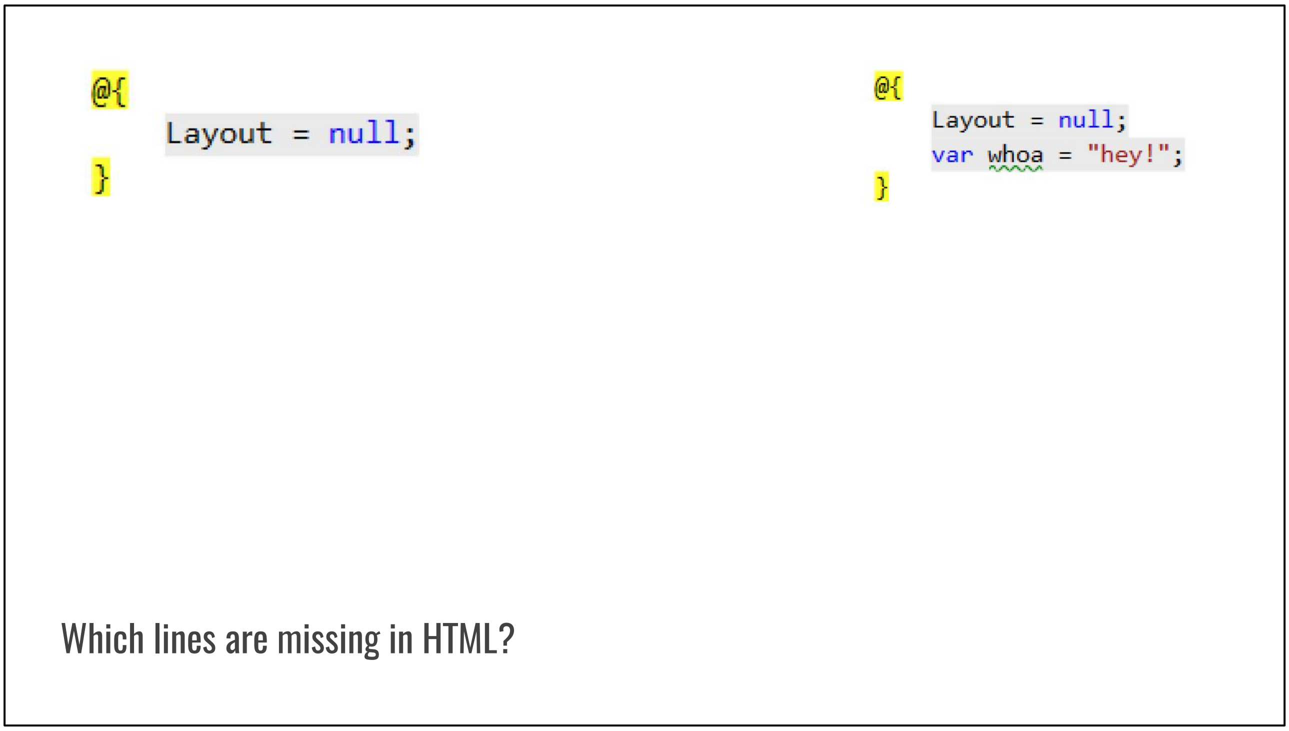
After clicking Add, what we get is our first razor file. I will rebuild my project and refresh homepage on browser.



```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>

<!-- Visual Studio Browser Link -->
<script type="application/json" id="__browserLink_initializationData">
    {"appName":"Chrome","requestId":"5522318213914c1f9a79fc688a182125"}
</script>
<script type="text/javascript" src="http://localhost:1794/0ead2db0ca474b1c8a5db4d03820603d/browserLink" async="async"></script>
<!-- End Browser Link -->

</body>
</html>
```

View Source Of Our First Razor File

If we look at our first Razor file view source, we get this. Which corresponds our index.cshtml file. There are some missing lines on the view source.

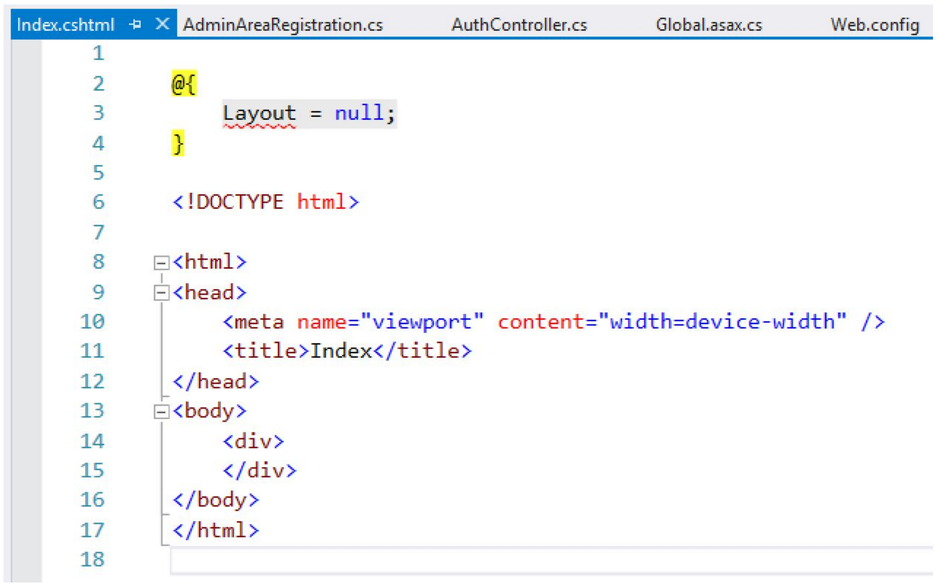


```
@{
    Layout = null;
}
```

```
@{
    Layout = null;
    var whoa = "hey!";
}
```

Which lines are missing in HTML?

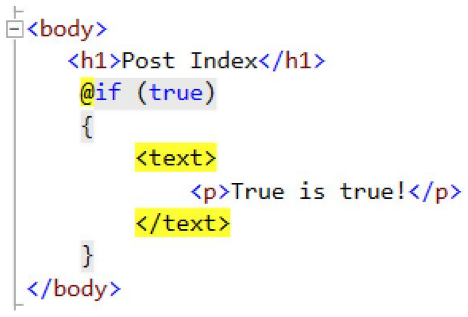
As we said, we can insert c# code in to HMTL. This right here, this blog is C# code. C# code is processed on the server and never sended to the client. So in that way, we can use C# to generate our HTML that send to the browser. We do note what the difference between C# code and HTML code with the @ sign. In this case, we are opening a block. This block can contain any arbitrary C# code.



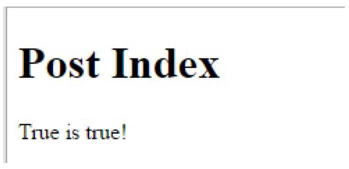
```

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
    </div>
</body>
</html>

```

Our Index.cshtml file

This is our Index.cshtml file. Note that it is a comple HTML file with the correct document type , even a viewport and a title. We do not have anything in it yet. We can delete div tags.

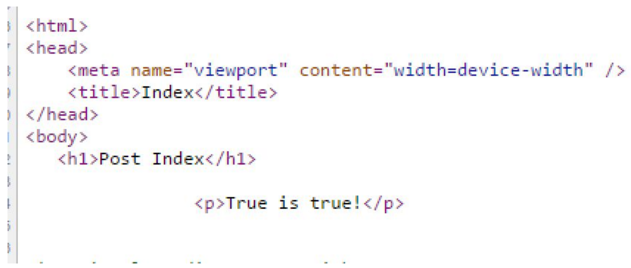


```
<body>
    <h1>Post Index</h1>
     @if (true)
     {
         <text>
             <p>True is true!</p>
         </text>
     }
</body>
```



Post Index

True is true!



```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
   <h1>Post Index</h1>

                <p>True is true!</p>
```

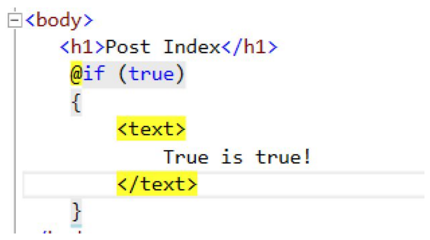
If Statement In Razor

HTML does not have if statements. HTML is a declarative language that only allows us to write data. Does not allows us to add logic. So you write the if statement in C#. And we open up C# code.

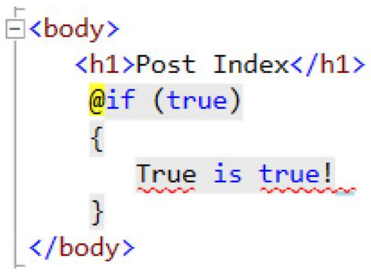
We do not need to Rebuild our project. Because views are not actually compiled by Visual Studio but compiled by the web server. By modifying the view you do not have to tell Visual Studio to recompile your DLL.

Remember if you modify .cs files, you have to recompile. If you modify cshtml files, yo do not. Those are gone automatically reloaed by the web server.

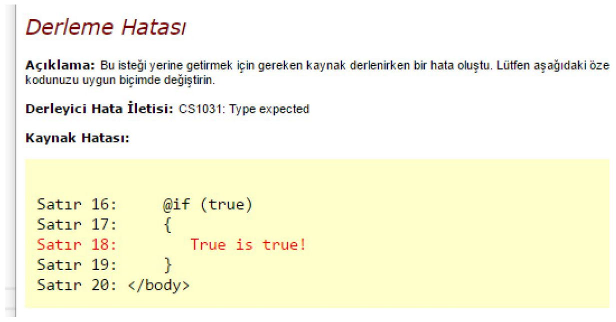
We see that our if statements are not actually sended to the browser. Again C# code is executed on the server, and not actually sended to the browser for processing. So C# codes does not display on our web browser source code.



```
<body>
    <h1>Post Index</h1>
    @if (true)
    {
        <text>
            True is true!
        </text>
    }
```



```
<body>
    <h1>Post Index</h1>
    @if (true)
    {
        True is true!
    }
</body>
```



Derleme Hatası

Açıklama: Bu isteği yerine getirmek için gereken kaynak derlenirken bir hata oluştu. Lütfen aşağıdaki öze kodunuzu uygun biçimde değiştirin.

Derleyici Hata İletisi: CS1031: Type expected

Kaynak Hatası:

```
Satır 16:     @if (true)
Satır 17:     {
Satır 18:         True is true!
Satır 19:     }
Satır 20: </body>
```

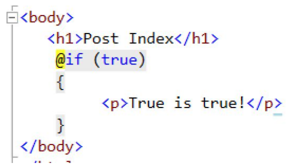
<text> tag

If I removed the <p> tag and write only True is true! in if block what will happen? There is no error when viewing our home page.

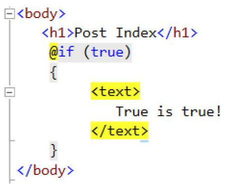
The text tag is important. Because we need a method to seperate html code in C# block. If I remove text blocks what we get. We get an complation error.

This is happening because Razor was confused. Because we opended a C# block but put HTML in this block.

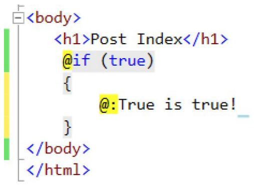
Open up a HTML Tag


```
<body>
    <h1>Post Index</h1>
     @if (true)
     {
          <p>True is true!</p>
     }
</body>
```

Using text


```
<body>
    <h1>Post Index</h1>
     @if (true)
     {
          <text>
             True is true!
          </text>
     }
</body>
```

@:


```
<body>
    <h1>Post Index</h1>
     @if (true)
     {
         @:True is true!
     }
</body>
</html>
```

Tree Ways To Tell Razor We Are Working With HTML

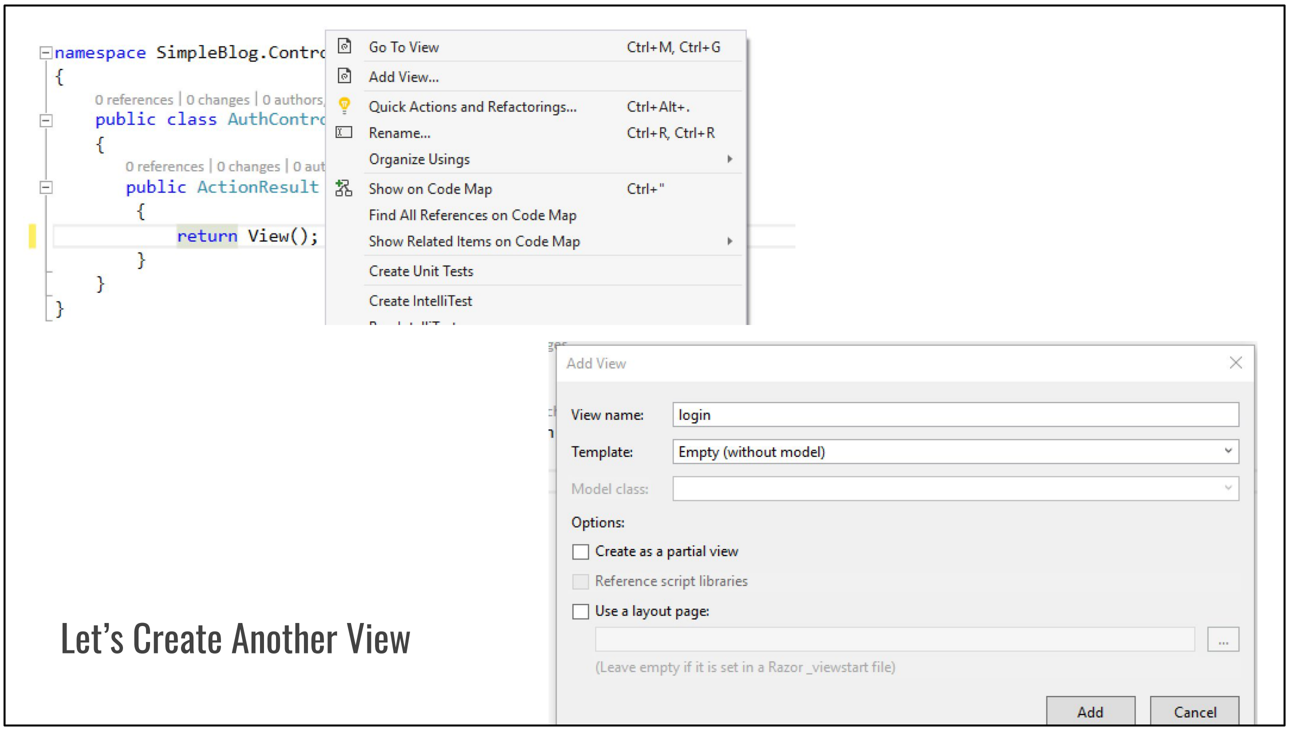
There is tree ways to tell Razor we are working with HTML.

The first one is to open up a HMTL tag.

Second way what we did before. With the text. The text tells Razor to go out of the context of C# and go back into the context of HTML.

Third way, @:

@: only applies to the single line as the same effect with text. It instructs Razor, that we are no longer looking at C# code.



```
namespace SimpleBlog.Contro
{
    0 references | 0 changes | 0 authors,
    public class AuthContro
    {
        0 references | 0 changes | 0 aut
        public ActionResult
        {
            return View();
        }
    }
}
```

Go To View Ctrl+M, Ctrl+G
Add View...
Quick Actions and Refactorings... Ctrl+Alt+.
Rename... Ctrl+R, Ctrl+R
Organize Usings
Show on Code Map Ctrl+"
Find All References on Code Map
Show Related Items on Code Map
Create Unit Tests
Create IntelliTest
Add View

View name: login

Template: Empty (without model)

Model class:

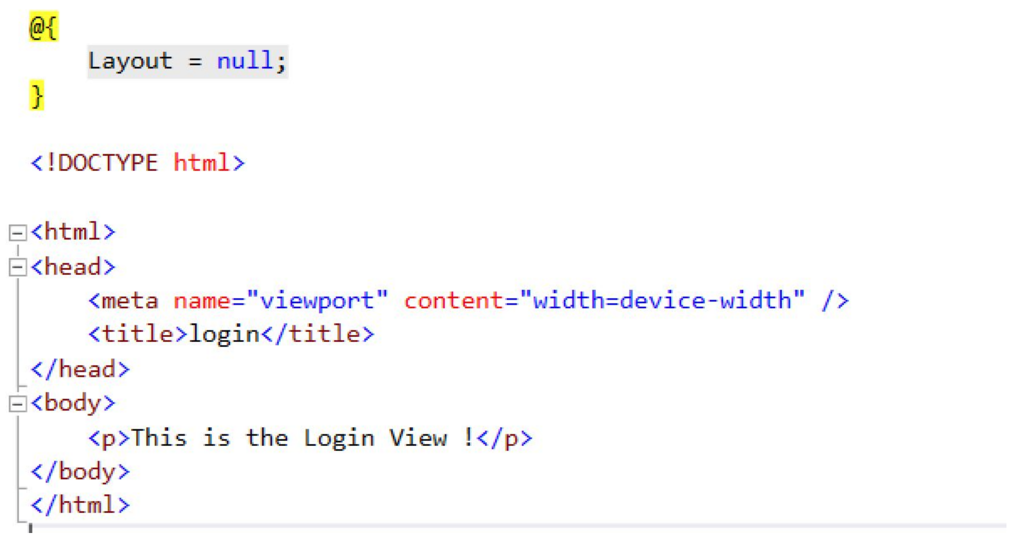
Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [ ] Use a layout page:

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

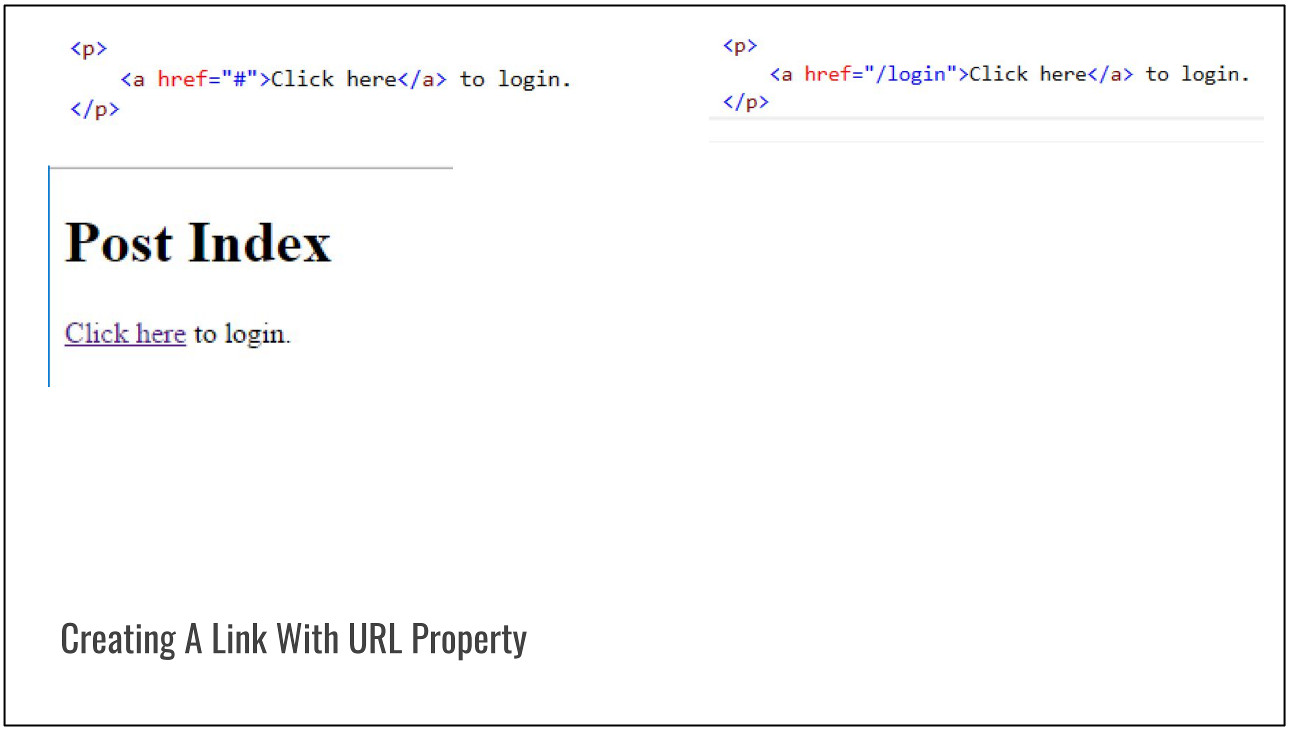
Let's Create Another View

I am goint to my AuthController. In my AuthController, I am goint to return a View. Now, instead of manually creating this View, I will use a shortcut. Right click when I am in Login action and click Add View to add a view for this action. It automatically creates our Auth folder in Views folder and add login.cshtml file in it.



```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>login</title>
</head>
<body>
    <p>This is the Login View !</p>
</body>
</html>
```

Our Login View



```
<p>
    <a href="#">Click here</a> to login.
</p>
```

```
<p>
    <a href="/login">Click here</a> to login.
</p>
```

Post Index

Click here to login.

Creating A Link With URL Property

I want a link between Login.cshtml and Index.cshtml.
So I am gonna do is creating a paragraph and write Click Here To Login. I am going to mark up Click Here inside the a tag.
<a href=”#”>Click here </a> to login.

Now we have a problem. What goes into the href. If you are not familiar with HMTL so much a tag is the link.

It is easy. We can write “/login”.

Click on this link and we will be in login.cshtml file. It works.

We want things to be changeble and maintainable down the road. So the links must not be hard coded.

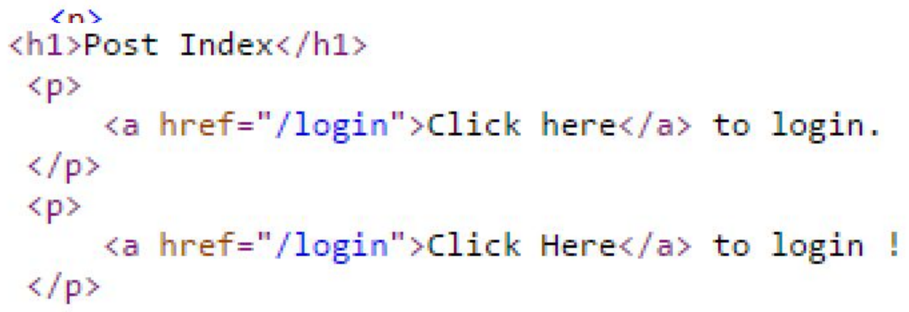
Razor and ASP.NET MVC provide many many ways to generate the URL to whatever route you may possibly be able to come up with.

What I wanna do is instead write “/login” directly open up @ , open up C# code, type URL which is property of Razor view, .RouteURL (“login”). We type login because in RouteConfig we write the name for the route Login.

If we refresh our page, the link href is still /login but the difference is it is now going through my routing engine. So if I change my routes, this link will be updated automatically.

With Url property, you can go directly to and action.

HTML.RouteLink (string linkText, string routeName)



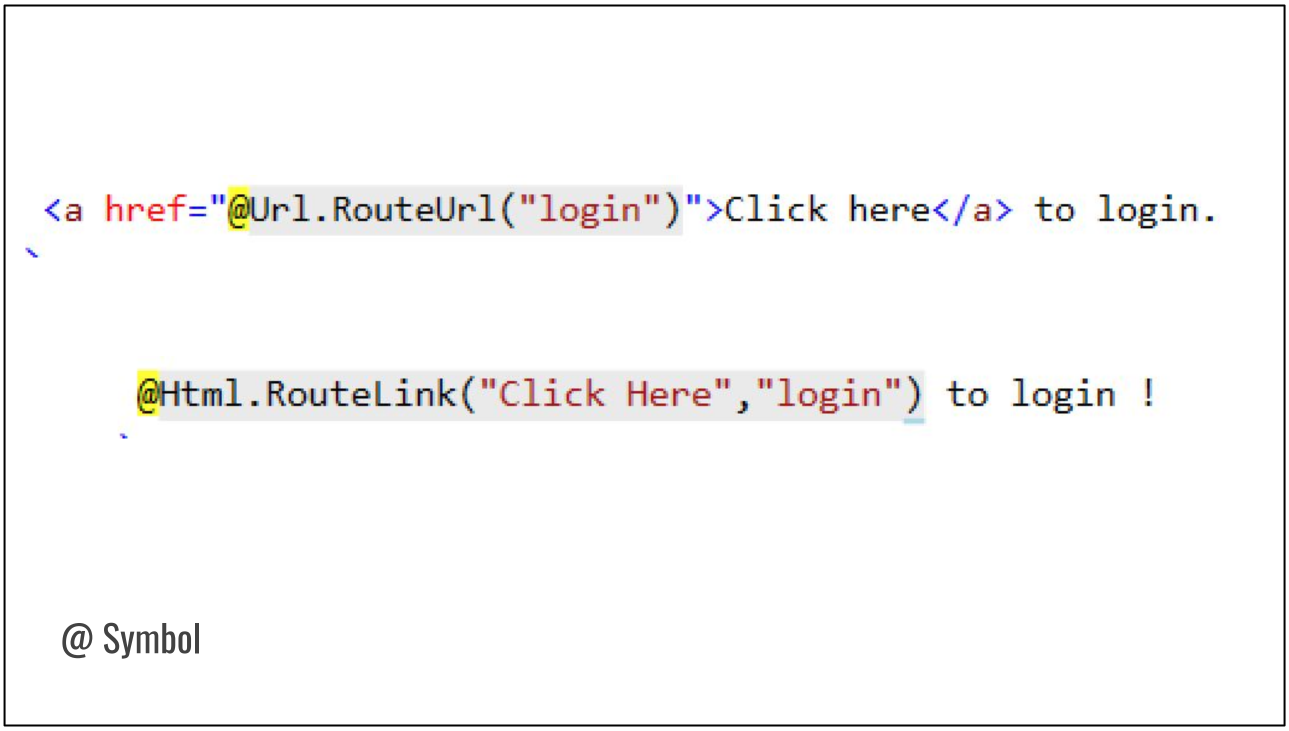
```
<h1>Post Index</h1>
 <p>
     <a href="/login">Click here</a> to login.
 </p>
 <p>
     <a href="/login">Click Here</a> to login !
 </p>
```

Html.ActionLink or HTML.RouteLink creates Links

There is one more way to create links. That is through the HTML property.
HTML property gives us methods, that generate HTML for us. So we can type in ActionLink or RouteLink.

RouteLink has a lot of programmables. We use the one that takes two parameters. Link text and Route Name.
So basically instead of creating a link manually, we are going to have ASP.NET MVC generate the actual a element for us. It still generates an a element. If we click wiev source in browser window. We can see that, it generates an a tag for us.

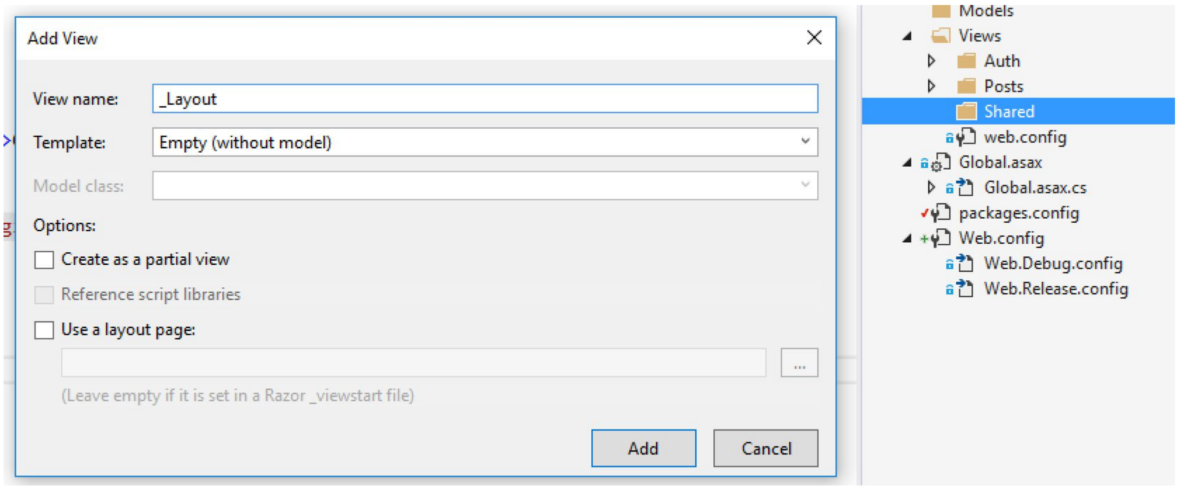
Most commonly we use @Url property. Typically that is because, if i want to mark up other attributes on my a tag, it is easier to do.



```
<a href="@Url.RouteUrl("login")">Click here</a> to login.

    @Html.RouteLink("Click Here","login") to login !
```

@ Symbol

With @symbol, the result of C# code is printed directly to the HTML.


Add View

View name: _Layout

Template: Empty (without model)

Model class:

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [ ] Use a layout page:

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

Add Cancel

We have an Issue - Dublication - Solution is **Layouts**

We have an issue. Dublication.

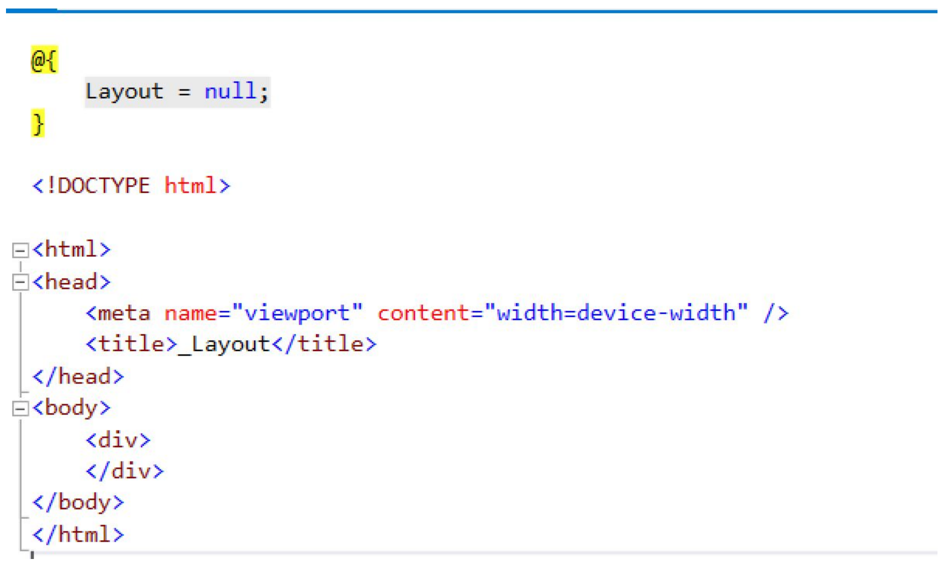
I am completly repeating all of my HTML. Both the login and index.
Imagine, I have a sidebard both in login and index. I have to add sidebar both these views.

There is a way to handle it. The souliton is layouts.

A layout is a way in Razor to create a reusable container of HTML that will allows to inject HTML specific views into the HTML of entire layout.

Let’s go ahead and create a layout. Inside of my views folder, create a new Folder named Shared. This is again a convention on ASP.NET MVC. Putting my layout in Shared folder is a convention.

Inside of shared I am goint to crete a new View. The name of my View will be _Layout.



```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>_Layout</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

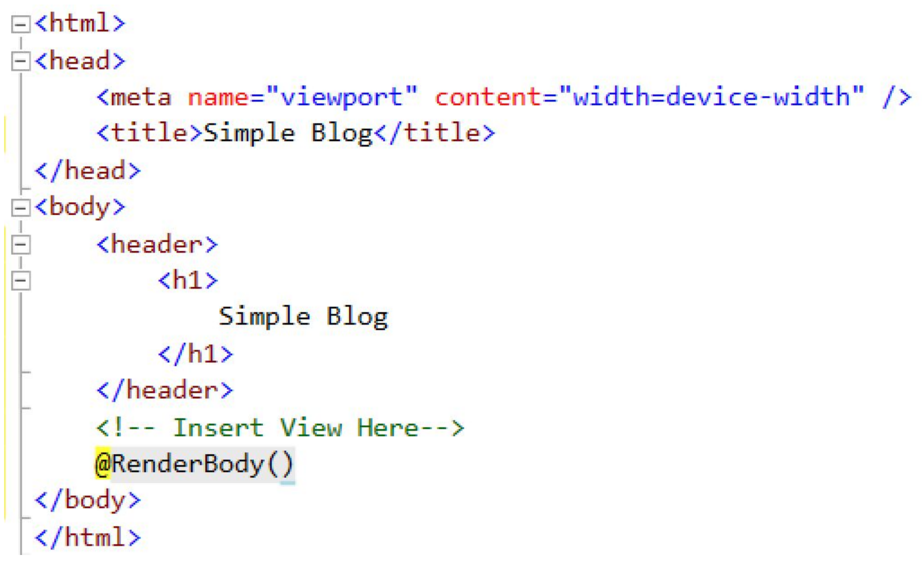
_Layout File

I will leave Layout=null. Leaving the Layout=null.

Layouts can actually be nested. Layouts can have layouts. So I want this layout to be the top level layout so I am explicitly telling it do not use any other layout.

Inside of it we have HTML codes. That code is standart HTML codes.

The first question that we have to answer , how we tell the layout to basically inject the result of the views of our particular actions.

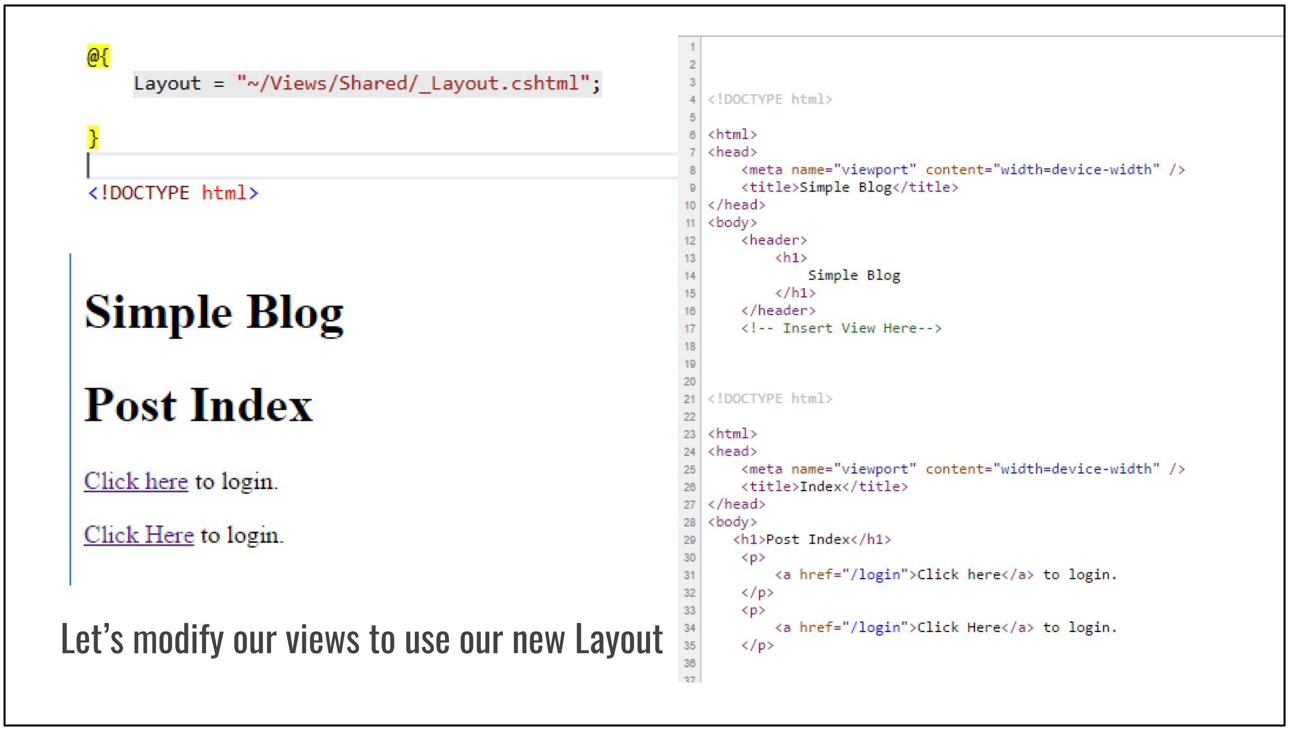


```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Simple Blog</title>
</head>
<body>
    <header>
        <h1>
            Simple Blog
        </h1>
    </header>
    <!-- Insert View Here-->
    @RenderBody()
</body>
</html>
```

Modify our Layout - RenderBody method

First, I will write some HTML codes, that will appear all our web pages that uses this layout.

@RenderBody method will actually go out and put the result of the view into where appears.



```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";

}

<!DOCTYPE html>
```

Simple Blog

Post Index

Click here to login.

Click Here to login.

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Simple Blog</title>
</head>
<body>
    <header>
        <h1>
            Simple Blog
        </h1>
    </header>
    <!-- Insert View Here-->


<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
   <h1>Post Index</h1>
    <p>
        <a href="/login">Click here</a> to login.
    </p>
    <p>
        <a href="/login">Click Here</a> to login.
    </p>
```

Let’s modify our views to use our new Layout

In index.cshtml, I am gonna have Index to take advantage of Layout by setting it layout property.

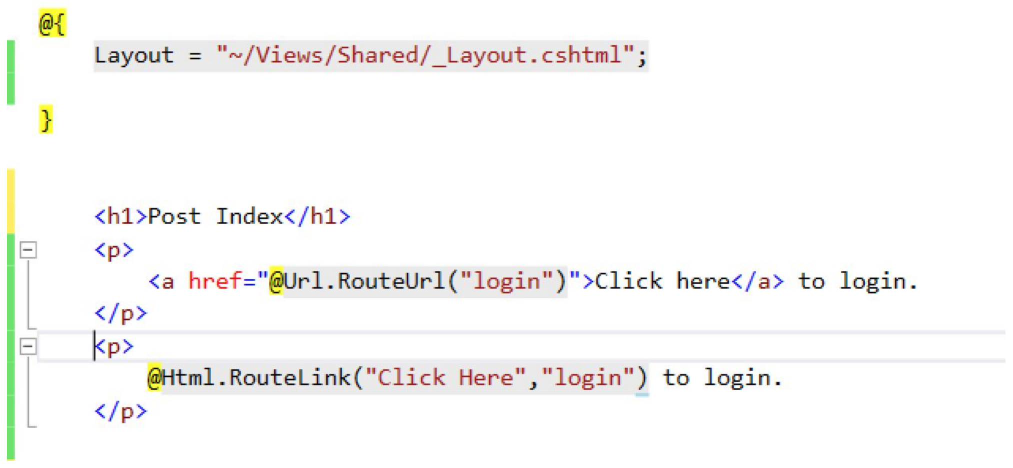
Remember this particular Layout has not been used anywhere.

By setting Layout property of our Index file ~/Views/Shared/_Layout.cshtml.

Come back our page in browser Refresh and look there. We have a problem in source code, not in the brower view.

We have two DocType, two html tags. That is not a valid HTML.

The common HTML code staff is handle by layout.

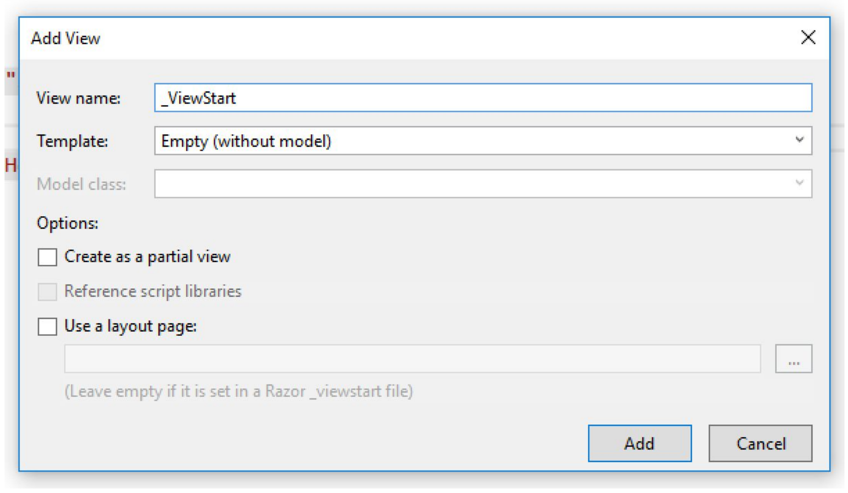


```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";

}


    <h1>Post Index</h1>
    <p>
        <a href="@Url.RouteUrl("login")">Click here</a> to login.
    </p>
    <p>
        @Html.RouteLink("Click Here","login") to login.
    </p>
```

Our modified Index.cshtml file

Now ,our modified Index.cstml file is in the figure. It just has the content that I want to appear inside of the Layout.
If I refresh the page on the browser the appears is the same but HTML code is valid now. In the source we do not have repeated HTML tags.

Our Index.cshtml file uses layout but our Login.cshtml file does not use Layout.


Add View

View name: _ViewStart

Template: Empty (without model)

Model class:

Options:

- [ ] Create as a partial view
- [ ] Reference script libraries
- [ ] Use a layout page:

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

Make Login.cshtml using Layout file - _ViewStart

There is two ways to make login use Layout.

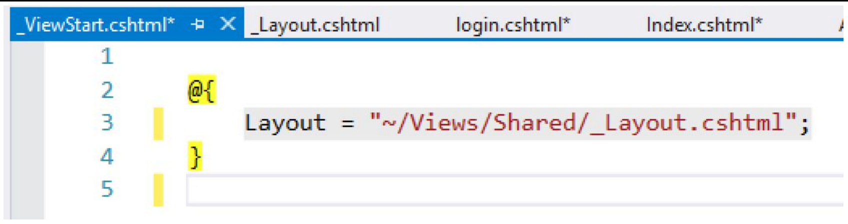
We can go into Login.cshtml file and change Layout property. Or we can do something better.

I do not want to specify the layout manually for every single view I am gonno put in the FrontEnd. That is annoying.

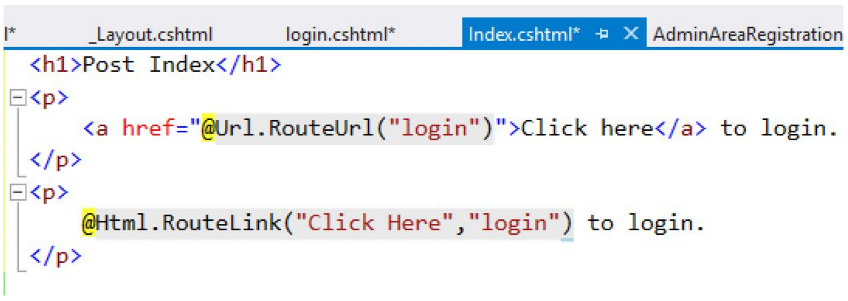
I gonno effectivly setup a default layout that used for everyone except if it is overriden for a reason.

To do that, I am goint to right click Views (The folder in root) and add a view. The name is important.

The name must be _ViewStart.

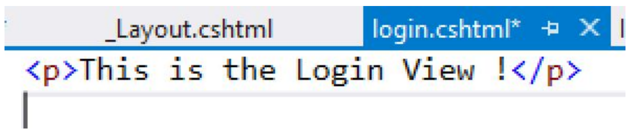


```

@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

```



```
<h1>Post Index</h1>
<p>
    <a href="@Url.RouteUrl("login")">Click here</a> to login.
</p>
<p>
    @Html.RouteLink("Click Here","login") to login.
</p>
```

_ViewStart.cshtml file



```
<p>This is the Login View !</p>
```

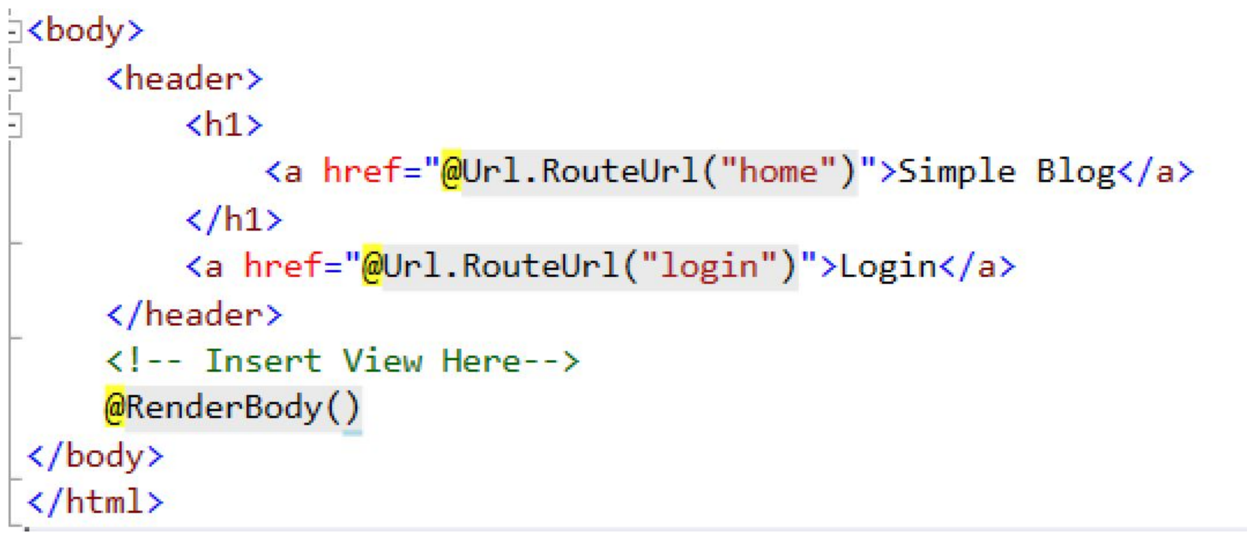
Then I am going to delete all the HTML.

Only C# codes will be here. The ViewStart is now going to be run before every other View that is in the subdirectory of the folder that ViewStart is in. Meaning I can run arbitrary C# codes now before my layouts are runned.

So, I can go in my Index.cshtml and copy Layout declaration and delete the entire code in index.cshtml. Then paste the code _ViewStart.

The effect of these will have is that now every single view by default will indeed have the layout.

Now we have index and login. They are so clear.

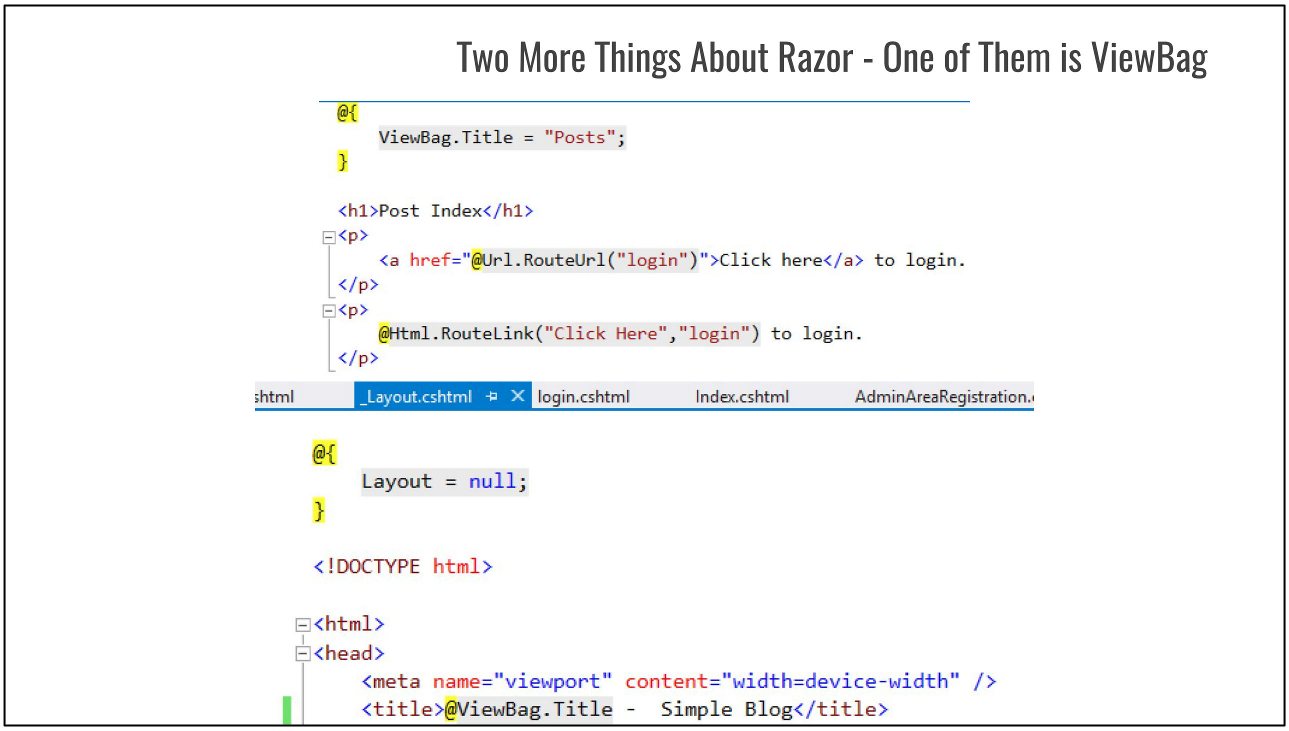


```
<body>
    <header>
        <h1>
            <a href="@Url.RouteUrl("home")">Simple Blog</a>
        </h1>
        <a href="@Url.RouteUrl("login")">Login</a>
    </header>
    <!-- Insert View Here-->
    @RenderBody()
</body>
</html>
```

Adding Links to Layout

Let’s go ahead and I am going on _Layout.cshtml file and add a link to Login and Index that is avaliable to entire site. To do that we add two a tags in _Layout.

Becuse these link are on _Layout , this all will be shown in both pages.



Two More Things About Razor - One of Them is ViewBag

```
@{
    ViewBag.Title = "Posts";
}

<h1>Post Index</h1>
<p>
    <a href="@Url.RouteUrl("login")">Click here</a> to login.
</p>
<p>
    @Html.RouteLink("Click Here","login") to login.
</p>
```

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title -  Simple Blog</title>
```

There is two more things we need to talk about with Razor really quickly we before remove on.

First thing is a ViewBag.
ViewBag is an interesting concept that effectively allows me to share data between Views or between Controllers and Views.

I am not going to use ViewBag to share data between Controllers and Views for a while. By using ViewBag to share data between Controllers and Views code become difficult to maintain.

But a best practive for using ViewBag is for communication between Views.

Login.cshtml and Index.cshtml files are compiled before they inserted to Layout. Any code inside the view, run before the code in Layouts.

These makes us to use programmers and override values on Layout.
For example I am gonne set some data in the ViewBag. What I am gonna do.

First I open a C# code block. Then I am gonna say ViewBag.Title=”Posts”

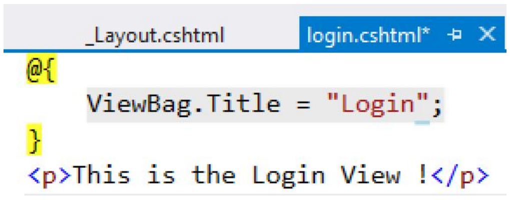
It is important to know that ViewBag is dynamic. If you are unfamiliar with dynamic keyword in C#, it basically means that this particular property has the type of dynamic you can add any property into it.

I add this arbitrary Title property into ViewBag dynamically.

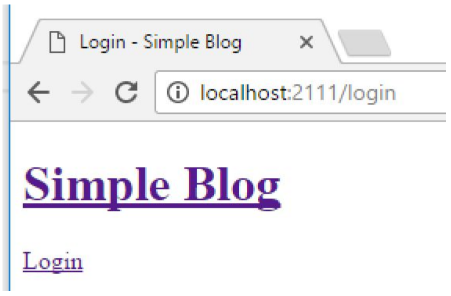
Remember, this particular View executed before the layout. Which means, If I use the result of ViewBag.Title inside of Layout, it will be set to whatever the child action set it to or null if the child action not set it.

I gonno in _Layout.cshtml and look at my title tag. I want the child view to determine what the title should be.

What effectivly be done. Now, I am using the variable that I am setting inside my index.cshtml to set something inside of the layout. Now, also do the same thing for the login.


_Layout.cshtml login.cshtml*

```
@{
    ViewBag.Title = "Login";
}
<p>This is the Login View !</p>
```



Simple Blog

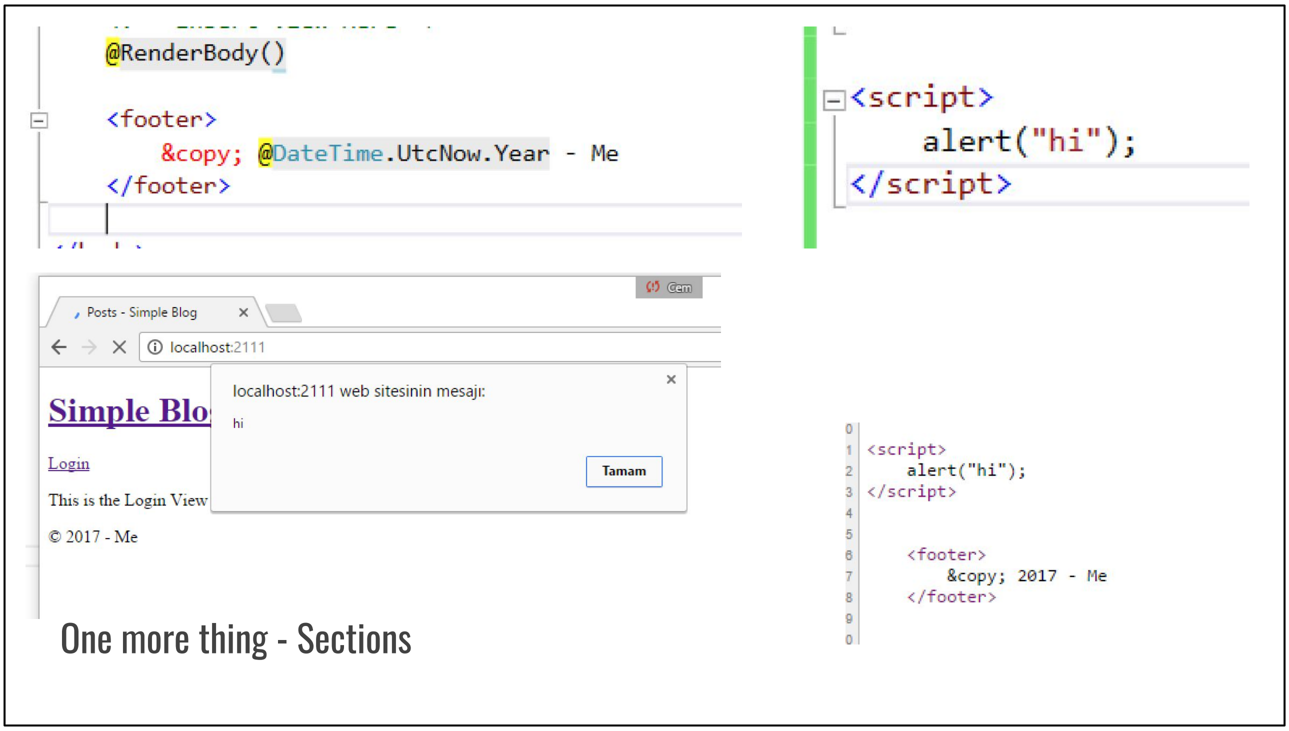
Login

Change Login.cshtml for setting ViewBag.Title

It is an example of how moving data between Views and Layouts.

Indivial wievs are setting ViewBag.Title programmable and then ViewBag.Title programmable is being used in Layout.

So in this way, we can effectively configure the layout to do something on a per view basis which is the very powerful technique.



```
@RenderBody()

<footer>
    &copy; @DateTime.UtcNow.Year - Me
</footer>
```

```
<script>
    alert("hi");
</script>
```

Posts - Simple Blog

localhost:2111

Simple Blo

Login

This is the Login View

© 2017 - Me

localhost:2111 web sitesinin mesajı:

hi

Tamam

```
<script>
    alert("hi");
</script>


    <footer>
        &copy; 2017 - Me
    </footer>
```

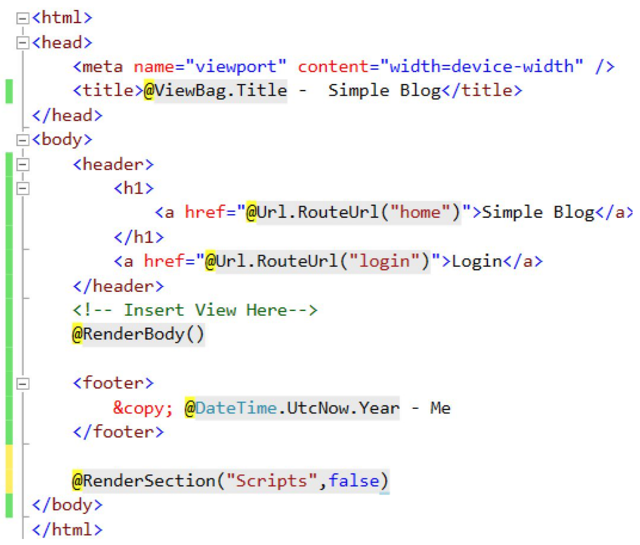
One more thing - Sections

Now, we are not going to have any javascript and css quite yet in our project. I want to talk about the technique that allows us to insert scripts properly in our web site.

We must put our scripts file at the end of our body.

We add a footer tag in _Layout. And add a script in index.cshtml file.

When we look at the source code script is located before footer but it must be after the footer.



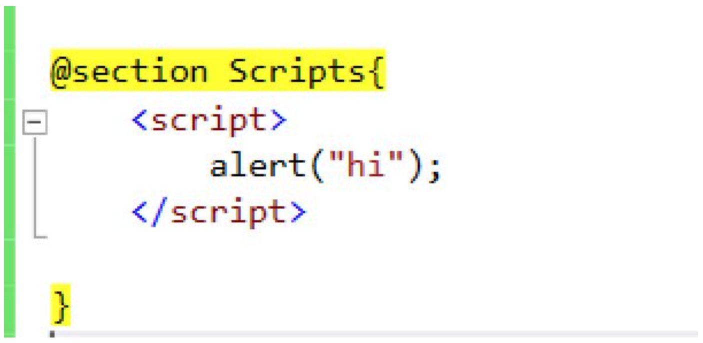
```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title -  Simple Blog</title>
</head>
<body>
    <header>
        <h1>
            <a href="@Url.RouteUrl("home")">Simple Blog</a>
        </h1>
        <a href="@Url.RouteUrl("login")">Login</a>
    </header>
    <!-- Insert View Here-->
    @RenderBody()

    <footer>
        &copy; @DateTime.UtcNow.Year - Me
    </footer>

    @RenderSection("Scripts",false)
</body>
</html>
```

Sections

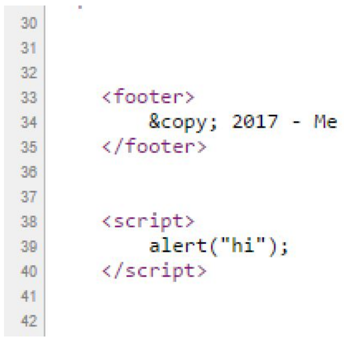
We will use sections. Section is effectively is another RenderBody. One view can specify one RenderBody and multiple secitons which is really cool.

In the layout, I want all my scripts located before the body close tag. @RenderSection("Scripts",false) The second parameter is required, that I set false. I do not want to force views to implement scripts section. That effects nothing yet.

We must modify index.cshtml file. I must tell in my view (index.cshtml) this code will be located in the section.



```
@section Scripts{
    <script>
        alert("hi");
    </script>

}
```



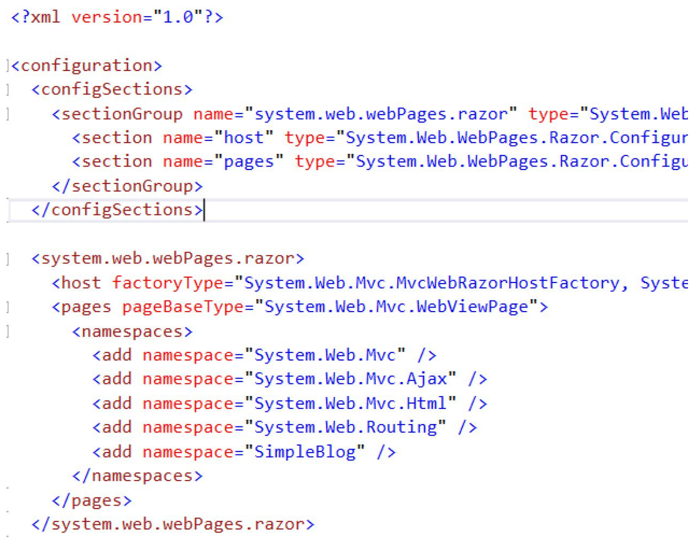
```
    <footer>
        &copy; 2017 - Me
    </footer>


    <script>
        alert("hi");
    </script>
```

index.cshtml - using section

The HTML codes in index.cshtml file that located in Scripts section will be rendered at the bottom after footer tag.

In index.cshtm codes in any section wil be rendered in the apporiate place in layout. Others are rendered for RenderBody. So it does not matter where you put your section in index.cshtml file.

We will create a new section later in the head for meta tags.

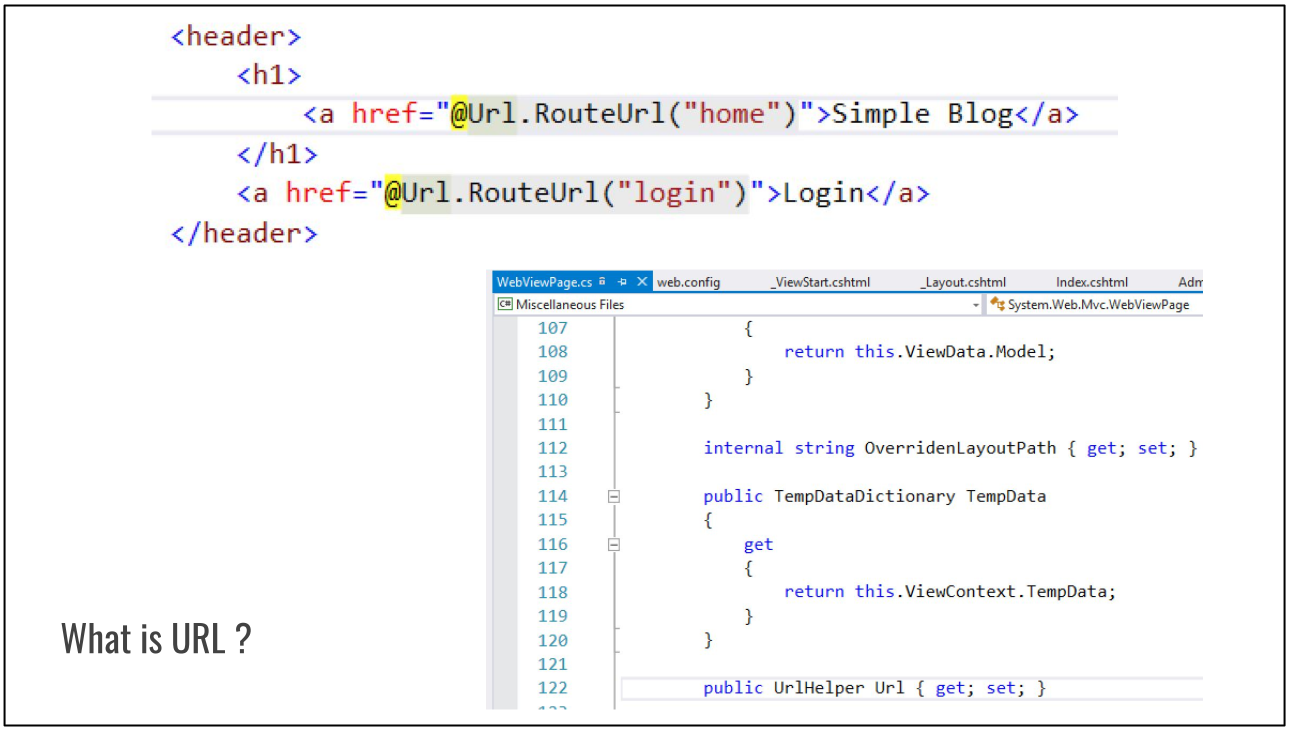


```
<?xml version="1.0"?>

<configuration>
  <configSections>
    <sectionGroup name="system.web.webPages.razor" type="System.Web
      <section name="host" type="System.Web.WebPages.Razor.Configur
      <section name="pages" type="System.Web.WebPages.Razor.Configu
    </sectionGroup>
  </configSections>

  <system.web.webPages.razor>
    <host factoryType="System.Web.Mvc.MvcWebRazorHostFactory, Syste
    <pages pageBaseType="System.Web.Mvc.WebViewPage">
      <namespaces>
        <add namespace="System.Web.Mvc" />
        <add namespace="System.Web.Mvc.Ajax" />
        <add namespace="System.Web.Mvc.Html" />
        <add namespace="System.Web.Routing" />
        <add namespace="SimpleBlog" />
      </namespaces>
    </pages>
  </system.web.webPages.razor>
```

web.config in Views Folder

We have a web.config in the root folder for the entire project.

And then, we have another web.config for each Views folder. This web.config is effectively telling .NET to use razor, in fact we see an <system.web.WebPages.razor> tag.



```
<header>
    <h1>
        <a href="@Url.RouteUrl("home")">Simple Blog</a>
    </h1>
    <a href="@Url.RouteUrl("login")">Login</a>
</header>
```

What is URL ?

```
            {
                return this.ViewData.Model;
            }
        }

        internal string OverridenLayoutPath { get; set; }

        public TempDataDictionary TempData
        {
            get
            {
                return this.ViewContext.TempData;
            }
        }

        public UrlHelper Url { get; set; }
```

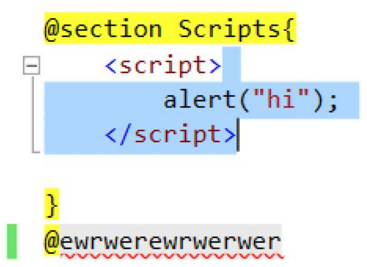
If you press F12 when you or on URL, It goes to defination. What is URL?

URL is a property.

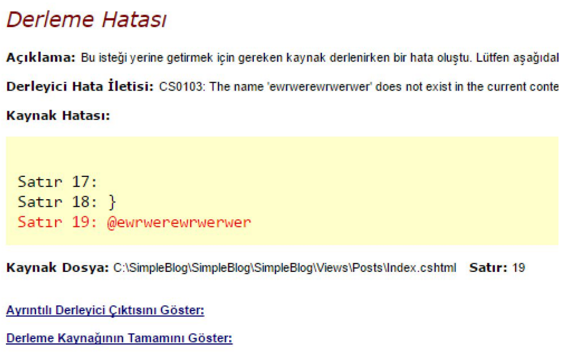
What is DateTime. DateTime is a class. It is accessible only in C# if you are writing a using statement. using system code is required for writing only DateTime.

It is important to understand that, in this particular case, System namespace is actually included in Razor itself.

If you want to access a class, not included in Razor itself, then you have to add it by writing fully qualified name or you have to add it to the web.config. SO we will be able to access it.



```
@section Scripts{
    <script>
        alert("hi");
    </script>

}
@ewrwerewrwerwer
```



Derleme Hatası

Açıklama: Bu isteği yerine getirmek için gereken kaynak derlenirken bir hata oluştu. Lütfen aşağıdal

Derleyici Hata İletisi: CS0103: The name 'ewrwerewrwerwer' does not exist in the current conte

Kaynak Hatası:

```
Satır 17:
Satır 18: }
Satır 19: @ewrwerewrwerwer
```

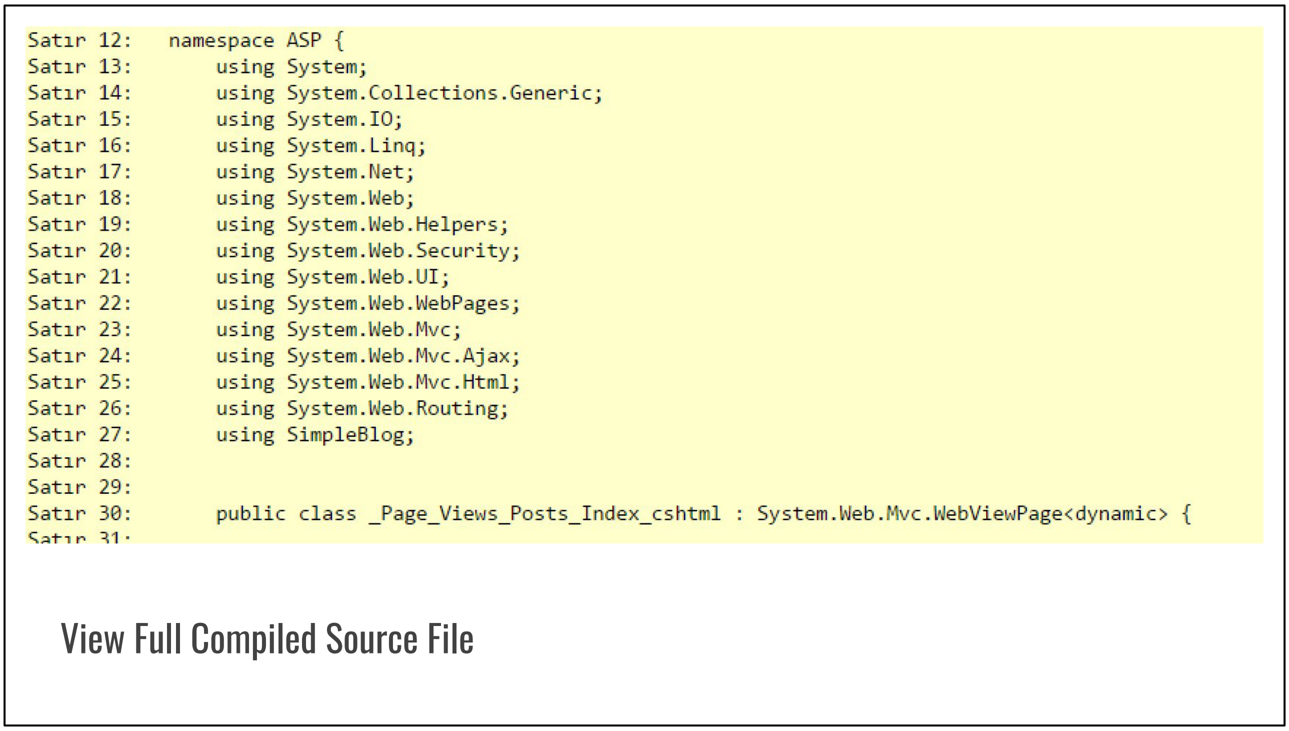
Kaynak Dosya: C:\SimpleBlog\SimpleBlog\SimpleBlog\Views\Posts\Index.cshtml **Satır:** 19

Ayrıntılı Derleyici Çıktısını Göster:

Derleme Kaynağının Tamamını Göster:

Syntax Error In My Razor View

I am goint to add a syntax error on Razor file Index.html. The reason I am adding an error is to show you Complete Complition Source.
If I open this up, this will blow your mind.



```
namespace ASP {
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Net;
    using System.Web;
    using System.Web.Helpers;
    using System.Web.Security;
    using System.Web.UI;
    using System.Web.WebPages;
    using System.Web.Mvc;
    using System.Web.Mvc.Ajax;
    using System.Web.Mvc.Html;
    using System.Web.Routing;
    using SimpleBlog;


    public class _Page_Views_Posts_Index_cshtml : System.Web.Mvc.WebViewPage<dynamic> {
```

View Full Compiled Source File

This is a class. _Page_Views_Posts_Index_cshtml. It effectively add all the code I added inside of it. What does it mean?

This means, Razor files are classes. They are taken by the Razor compiler and then turn into a class executed in ASP.NET. So just to demostrate very quickly you see there is a lot of using statements here.

What I will do is, I will add a new using statement by adding web.config a new line. I will add a new namespace <add namepace="SimpleBlog"> . After compiling I can see a new using statement at the begining of our class.

Remember our cshtml files are turned into c# classes.