Project Coding Language: CSHTML

Information retrieved from: https://www.w3schools.com/asp/razor\_syntax.asp

Table of Contents

[Used Programming Language 2](#_Toc21433027)

[What is CSHTML 2](#_Toc21433028)

[ASP.NET Razor - C# and VB Code Syntax 2](#_Toc21433029)

[Main Razor Syntax Rules for C# 2](#_Toc21433030)

[C# Example 2](#_Toc21433031)

[How Does it Work? 3](#_Toc21433032)

[Working with Objects 3](#_Toc21433033)

[Example 3](#_Toc21433034)

[If and Else Conditions 4](#_Toc21433035)

[Example 4](#_Toc21433036)

[Reading User Input 4](#_Toc21433037)

[Example 4](#_Toc21433038)

# Used Programming Language

For this project, as it is a web application which must be developed in C# and ASP.NET, the coding language used is CSHTML. This regroups C# and ASP.Net languages which allows to create web application with a database integrated to it.

# What is CSHTML

C# ("C sharp") HTML webpage file used by Razor, an ASP.NET view engine used for generating Web pages for a user's Web browser; similar to a standard ASP.NET webpage (.ASP or .ASPX file), but uses a slightly different syntax; runs on a Web server, which generates the HTML for the client Web browser; can be programmed with syntax highlighting and Intellisense code suggestions using Microsoft Visual Studio.

CSHTML files are similar to .VBHTML (Visual Basic HTML) files, but they use syntax that is closer to the C# language rather than the Visual Basic language.

NOTE: Razor offers new templating syntax operations in addition to those supported by the standard ASP.NET MVC default view engine.

# ASP.NET Razor - C# and VB Code Syntax

Razor supports both C# (C sharp) and VB (Visual Basic).

## Main Razor Syntax Rules for C#

* Razor code blocks are enclosed in @{ ... }
* Inline expressions (variables and functions) start with @
* Code statements end with semicolon
* Variables are declared with the var keyword
* Strings are enclosed with quotation marks
* C# code is case sensitive
* C# files have the extension .cshtml

### C# Example

<!-- Single statement block -->  
@{ var myMessage = "Hello World"; }  
  
<!-- Inline expression or variable -->  
<p>The value of myMessage is: @myMessage</p>  
  
<!-- Multi-statement block -->  
@{  
var greeting = "Welcome to our site!";  
var weekDay = DateTime.Now.DayOfWeek;  
var greetingMessage = greeting + " Here in Huston it is: " + weekDay;  
}  
<p>The greeting is: @greetingMessage</p>

## How Does it Work?

Razor is a simple programming syntax for embedding server code in web pages.

Razor syntax is based on the ASP.NET framework, the part of the Microsoft.NET Framework that's specifically designed for creating web applications.

The Razor syntax gives you all the power of ASP.NET, but is using a simplified syntax that's easier to learn if you're a beginner, and makes you more productive if you're an expert.

Razor web pages can be described as HTML pages with two kinds of content: HTML content and Razor code.

When the server reads the page, it runs the Razor code first, before it sends the HTML page to the browser. The code that is executed on the server can perform tasks that cannot be done in the browser, for example accessing a server database. Server code can create dynamic HTML content on the fly, before it is sent to the browser. Seen from the browser, the HTML generated by server code is no different than static HTML content.

ASP.NET web pages with Razor syntax have the special file extension CSHTML (Razor using C#) or VBHTML(Razor using VB).

## Working with Objects

Server coding often involves objects.

The "DateTime" object is a typical built-in ASP.NET object, but objects can also be self-defined, a web page, a text box, a file, a database record, etc.

Objects may have methods they can perform. A database record might have a "Save" method, an image object might have a "Rotate" method, an email object might have a "Send" method, and so on.

Objects also have properties that describe their characteristics. A database record might have a FirstName and a LastName property (among others).

The ASP.NET DateTime object has a Now property (written as DateTime.Now), and the Now property has a Day property (written as DateTime.Now.Day). The example below shows how to access some properties of the DateTime object:

### Example

<table border="1">  
<tr>  
<th width="100px">Name</th>  
<td width="100px">Value</td>  
</tr>  
<tr>  
<td>Day</td><td>@DateTime.Now.Day</td>  
</tr>  
<tr>  
<td>Hour</td><td>@DateTime.Now.Hour</td>  
</tr>  
<tr>  
<td>Minute</td><td>@DateTime.Now.Minute</td>  
</tr>  
<tr>  
<td>Second</td><td>@DateTime.Now.Second</td>  
</tr>  
</td>  
</table>

## If and Else Conditions

An important feature of dynamic web pages is that you can determine what to do based on conditions. The common way to do this is with the if ... else statements:

### Example

@{  
var txt = "";  
if(DateTime.Now.Hour > 12)  
  {txt = "Good Evening";}  
else  
  {txt = "Good Morning";}  
}  
<html>  
<body>  
<p>The message is @txt</p>  
</body>  
</html>

## Reading User Input

Another important feature of dynamic web page is that you can read user input. Input is read by the Request[] function, and posting (input) is tested by the IsPost condition:

### Example

@{  
var totalMessage = "";  
if(IsPost)  
    {  
    var num1 = Request["text1"];  
    var num2 = Request["text2"];  
    var total = num1.AsInt() + num2.AsInt();  
    totalMessage = "Total = " + total;  
    }  
}  
<html>  
<body style="background-color: beige; font-family: Verdana, Arial;">  
<form action="" method="post">  
<p><label for="text1">First Number:</label><br>  
<input type="text" name="text1" /></p>  
<p><label for="text2">Second Number:</label><br>  
<input type="text" name="text2" /></p>  
<p><input type="submit" value=" Add " /></p>  
</form>  
<p>@totalMessage</p>  
</body>  
</html>