Project Coding Language: CSHTML

Information retrieved from:

* https://www.w3schools.com/asp/razor\_syntax.asp
* https://docs.microsoft.com/en-us/aspnet/core/mvc/views/partial?view=aspnetcore-3.0

Table of Contents

[Used Programming Language 3](#_Toc22029632)

[What is CSHTML 3](#_Toc22029633)

[ASP.NET Razor - C# and VB Code Syntax 3](#_Toc22029634)

[Main Razor Syntax Rules for C# 3](#_Toc22029635)

[C# Example 3](#_Toc22029636)

[How Does it Work? 4](#_Toc22029637)

[Working with Objects 4](#_Toc22029638)

[Example 4](#_Toc22029639)

[If and Else Conditions 5](#_Toc22029640)

[Example 5](#_Toc22029641)

[Reading User Input 5](#_Toc22029642)

[Example 5](#_Toc22029643)

[Partial Views 6](#_Toc22029644)

[When to use partial views 6](#_Toc22029645)

[Declare partial views 6](#_Toc22029646)

[Reference a partial view 6](#_Toc22029647)

[Use a partial view in a Razor Pages PageModel 6](#_Toc22029648)

[Use a partial view in a markup file 7](#_Toc22029649)

[Partial Tag Helper 7](#_Toc22029650)

[Razor Pages 7](#_Toc22029651)

[MVC 7](#_Toc22029652)

[Asynchronous HTML Helper 8](#_Toc22029653)

[Razor Pages 8](#_Toc22029654)

[MVC 8](#_Toc22029655)

[Synchronous HTML Helper 9](#_Toc22029656)

[Important 9](#_Toc22029657)

[Partial view discovery 9](#_Toc22029658)

[Razor Pages 9](#_Toc22029659)

[MVC 9](#_Toc22029660)

[Note 10](#_Toc22029661)

[Access data from partial views 10](#_Toc22029662)

[Razor Pages 10](#_Toc22029663)

[MVC 11](#_Toc22029664)

[Useful Links 13](#_Toc22029665)

# 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>

# Partial Views

A partial view is a Razor markup file (.cshtml) that renders HTML output within another markup file's rendered output.

The term partial view is used when developing either an MVC app, where markup files are called views, or a Razor Pages app, where markup files are called pages. This topic generically refers to MVC views and Razor Pages pages as markup files.

View or download sample code (how to download)

## When to use partial views

Partial views are an effective way to:

Break up large markup files into smaller components.

In a large, complex markup file composed of several logical pieces, there's an advantage to working with each piece isolated into a partial view. The code in the markup file is manageable because the markup only contains the overall page structure and references to partial views.

Reduce the duplication of common markup content across markup files.

When the same markup elements are used across markup files, a partial view removes the duplication of markup content into one partial view file. When the markup is changed in the partial view, it updates the rendered output of the markup files that use the partial view.

Partial views shouldn't be used to maintain common layout elements. Common layout elements should be specified in \_Layout.cshtml files.

Don't use a partial view where complex rendering logic or code execution is required to render the markup. Instead of a partial view, use a view component.

## Declare partial views

A partial view is a .cshtml markup file maintained within the Views folder (MVC) or Pages folder (Razor Pages).

In ASP.NET Core MVC, a controller's ViewResult is capable of returning either a view or a partial view. In Razor Pages, a PageModel can return a partial view represented as a PartialViewResult object. Referencing and rendering partial views is described in the Reference a partial view section.

Unlike MVC view or page rendering, a partial view doesn't run \_ViewStart.cshtml. For more information on \_ViewStart.cshtml, see Layout in ASP.NET Core.

Partial view file names often begin with an underscore (\_). This naming convention isn't required, but it helps to visually differentiate partial views from views and pages.

## Reference a partial view

### Use a partial view in a Razor Pages PageModel

In ASP.NET Core 2.0 or 2.1, the following handler method renders the \_AuthorPartialRP.cshtml partial view to the response:

C#

public IActionResult OnGetPartial() =>

new PartialViewResult

{

ViewName = "\_AuthorPartialRP",

ViewData = ViewData,

};

In ASP.NET Core 2.2 or later, a handler method can alternatively call the Partial method to produce a PartialViewResult object:

C#

public IActionResult OnGetPartial() =>

Partial("\_AuthorPartialRP");

### Use a partial view in a markup file

Within a markup file, there are several ways to reference a partial view. We recommend that apps use one of the following asynchronous rendering approaches:

* Partial Tag Helper
* Asynchronous HTML Helper

## Partial Tag Helper

The Partial Tag Helper requires ASP.NET Core 2.1 or later.

The Partial Tag Helper renders content asynchronously and uses an HTML-like syntax:

CSHTML

<partial name="\_PartialName" />

When a file extension is present, the Tag Helper references a partial view that must be in the same folder as the markup file calling the partial view:

CSHTML

<partial name="\_PartialName.cshtml" />

The following example references a partial view from the app root. Paths that start with a tilde-slash (~/) or a slash (/) refer to the app root:

### Razor Pages

CSHTML

<partial name="~/Pages/Folder/\_PartialName.cshtml" />

<partial name="/Pages/Folder/\_PartialName.cshtml" />

### MVC

CSHTML

<partial name="~/Views/Folder/\_PartialName.cshtml" />

<partial name="/Views/Folder/\_PartialName.cshtml" />

The following example references a partial view with a relative path:

CSHTML

<partial name="../Account/\_PartialName.cshtml" />

For more information, see Partial Tag Helper in ASP.NET Core.

## Asynchronous HTML Helper

When using an HTML Helper, the best practice is to use PartialAsync. PartialAsync returns an IHtmlContent type wrapped in a Task<TResult>. The method is referenced by prefixing the awaited call with an @ character:

CSHTML

@await Html.PartialAsync("\_PartialName")

When the file extension is present, the HTML Helper references a partial view that must be in the same folder as the markup file calling the partial view:

CSHTML

@await Html.PartialAsync("\_PartialName.cshtml")

The following example references a partial view from the app root. Paths that start with a tilde-slash (~/) or a slash (/) refer to the app root:

### Razor Pages

CSHTML

@await Html.PartialAsync("~/Pages/Folder/\_PartialName.cshtml")

@await Html.PartialAsync("/Pages/Folder/\_PartialName.cshtml")

### MVC

CSHTML

@await Html.PartialAsync("~/Views/Folder/\_PartialName.cshtml")

@await Html.PartialAsync("/Views/Folder/\_PartialName.cshtml")

The following example references a partial view with a relative path:

CSHTML

@await Html.PartialAsync("../Account/\_LoginPartial.cshtml")

Alternatively, you can render a partial view with RenderPartialAsync. This method doesn't return an IHtmlContent. It streams the rendered output directly to the response. Because the method doesn't return a result, it must be called within a Razor code block:

CSHTML

@{

await Html.RenderPartialAsync("\_AuthorPartial");

}

Since RenderPartialAsync streams rendered content, it provides better performance in some scenarios. In performance-critical situations, benchmark the page using both approaches and use the approach that generates a faster response.

## Synchronous HTML Helper

Partial and RenderPartial are the synchronous equivalents of PartialAsync and RenderPartialAsync, respectively. The synchronous equivalents aren't recommended because there are scenarios in which they deadlock. The synchronous methods are targeted for removal in a future release.

### Important

If you need to execute code, use a view component instead of a partial view.

Calling Partial or RenderPartial results in a Visual Studio analyzer warning. For example, the presence of Partial yields the following warning message:

Use of IHtmlHelper.Partial may result in application deadlocks. Consider using <partial> Tag Helper or IHtmlHelper.PartialAsync.

Replace calls to @Html.Partial with @await Html.PartialAsync or the Partial Tag Helper. For more information on Partial Tag Helper migration, see Migrate from an HTML Helper.

## Partial view discovery

When a partial view is referenced by name without a file extension, the following locations are searched in the stated order:

### Razor Pages

Currently executing page's folder

Directory graph above the page's folder

/Shared

/Pages/Shared

/Views/Shared

### MVC

/Areas/<Area-Name>/Views/<Controller-Name>

/Areas/<Area-Name>/Views/Shared

/Views/Shared

/Pages/Shared

The following conventions apply to partial view discovery:

Different partial views with the same file name are allowed when the partial views are in different folders.

When referencing a partial view by name without a file extension and the partial view is present in both the caller's folder and the Shared folder, the partial view in the caller's folder supplies the partial view. If the partial view isn't present in the caller's folder, the partial view is provided from the Shared folder. Partial views in the Shared folder are called shared partial views or default partial views.

Partial views can be chained—a partial view can call another partial view if a circular reference isn't formed by the calls. Relative paths are always relative to the current file, not to the root or parent of the file.

### Note

A Razor section defined in a partial view is invisible to parent markup files. The section is only visible to the partial view in which it's defined.

## Access data from partial views

When a partial view is instantiated, it receives a copy of the parent's ViewData dictionary. Updates made to the data within the partial view aren't persisted to the parent view. ViewData changes in a partial view are lost when the partial view returns.

The following example demonstrates how to pass an instance of ViewDataDictionary to a partial view:

CSHTML

@await Html.PartialAsync("\_PartialName", customViewData)

You can pass a model into a partial view. The model can be a custom object. You can pass a model with PartialAsync (renders a block of content to the caller) or RenderPartialAsync (streams the content to the output):

CSHTML

@await Html.PartialAsync("\_PartialName", model)

### Razor Pages

The following markup in the sample app is from the Pages/ArticlesRP/ReadRP.cshtml page. The page contains two partial views. The second partial view passes in a model and ViewData to the partial view. The ViewDataDictionary constructor overload is used to pass a new ViewData dictionary while retaining the existing ViewData dictionary.

CSHTML

@model ReadRPModel

<h2>@Model.Article.Title</h2>

@\* Pass the author's name to Pages\Shared\\_AuthorPartialRP.cshtml \*@

@await Html.PartialAsync("../Shared/\_AuthorPartialRP", Model.Article.AuthorName)

@Model.Article.PublicationDate

@\* Loop over the Sections and pass in a section and additional ViewData to

the strongly typed Pages\ArticlesRP\\_ArticleSectionRP.cshtml partial view. \*@

@{

var index = 0;

foreach (var section in Model.Article.Sections)

{

await Html.PartialAsync("\_ArticleSectionRP",

section,

new ViewDataDictionary(ViewData)

{

{ "index", index }

});

index++;

}

}

Pages/Shared/\_AuthorPartialRP.cshtml is the first partial view referenced by the ReadRP.cshtml markup file:

CSHTML

@model string

<div>

<h3>@Model</h3>

This partial view from /Pages/Shared/\_AuthorPartialRP.cshtml.

</div>

Pages/ArticlesRP/\_ArticleSectionRP.cshtml is the second partial view referenced by the ReadRP.cshtml markup file:

CSHTML

@using PartialViewsSample.ViewModels

@model ArticleSection

<h3>@Model.Title Index: @ViewData["index"]</h3>

<div>

@Model.Content

</div>

### MVC

The following markup in the sample app shows the Views/Articles/Read.cshtml view. The view contains two partial views. The second partial view passes in a model and ViewData to the partial view. The ViewDataDictionary constructor overload is used to pass a new ViewData dictionary while retaining the existing ViewData dictionary.

CSHTML

@model PartialViewsSample.ViewModels.Article

<h2>@Model.Title</h2>

@\* Pass the author's name to Views\Shared\\_AuthorPartial.cshtml \*@

@await Html.PartialAsync("\_AuthorPartial", Model.AuthorName)

@Model.PublicationDate

@\* Loop over the Sections and pass in a section and additional ViewData to

the strongly typed Views\Articles\\_ArticleSection.cshtml partial view. \*@

@{

var index = 0;

foreach (var section in Model.Sections)

{

await Html.PartialAsync("\_ArticleSection",

section,

new ViewDataDictionary(ViewData)

{

{ "index", index }

});

index++;

}

}

Views/Shared/\_AuthorPartial.cshtml is the first partial view referenced by the Read.cshtml markup file:

CSHTML

@model string

<div>

<h3>@Model</h3>

This partial view from /Views/Shared/\_AuthorPartial.cshtml.

</div>

Views/Articles/\_ArticleSection.cshtml is the second partial view referenced by the Read.cshtml markup file:

CSHTML

@using PartialViewsSample.ViewModels

@model ArticleSection

<h3>@Model.Title Index: @ViewData["index"]</h3>

<div>

@Model.Content

</div>

At runtime, the partials are rendered into the parent markup file's rendered output, which itself is rendered within the shared \_Layout.cshtml. The first partial view renders the article author's name and publication date:

Abraham Lincoln

This partial view from <shared partial view file path>. 11/19/1863 12:00:00 AM

The second partial view renders the article's sections:

Section One Index: 0

Four score and seven years ago ...

Section Two Index: 1

Now we are engaged in a great civil war, testing ...

Section Three Index: 2

But, in a larger sense, we cannot dedicate ...

# Useful Links

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/?view=aspnetcore-3.0>

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/?view=aspnetcore-3.0>

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-3.0>

<https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.controllerbase.redirecttoaction?view=aspnetcore-2.2>

<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/partial?view=aspnetcore-3.0>

<https://asp.mvc-tutorial.com/httpcontext/forms-post-data/>

<https://docs.microsoft.com/en-us/dotnet/api/system.configuration.configurationmanager.connectionstrings?view=netframework-4.8>

<https://www.learnentityframeworkcore.com/raw-sql>