Layout in ASP.NET Core

Article • 06/03/2022

By Steve Smith and Dave Brock

Pages and views frequently share visual and programmatic elements. This article demonstrates how to:

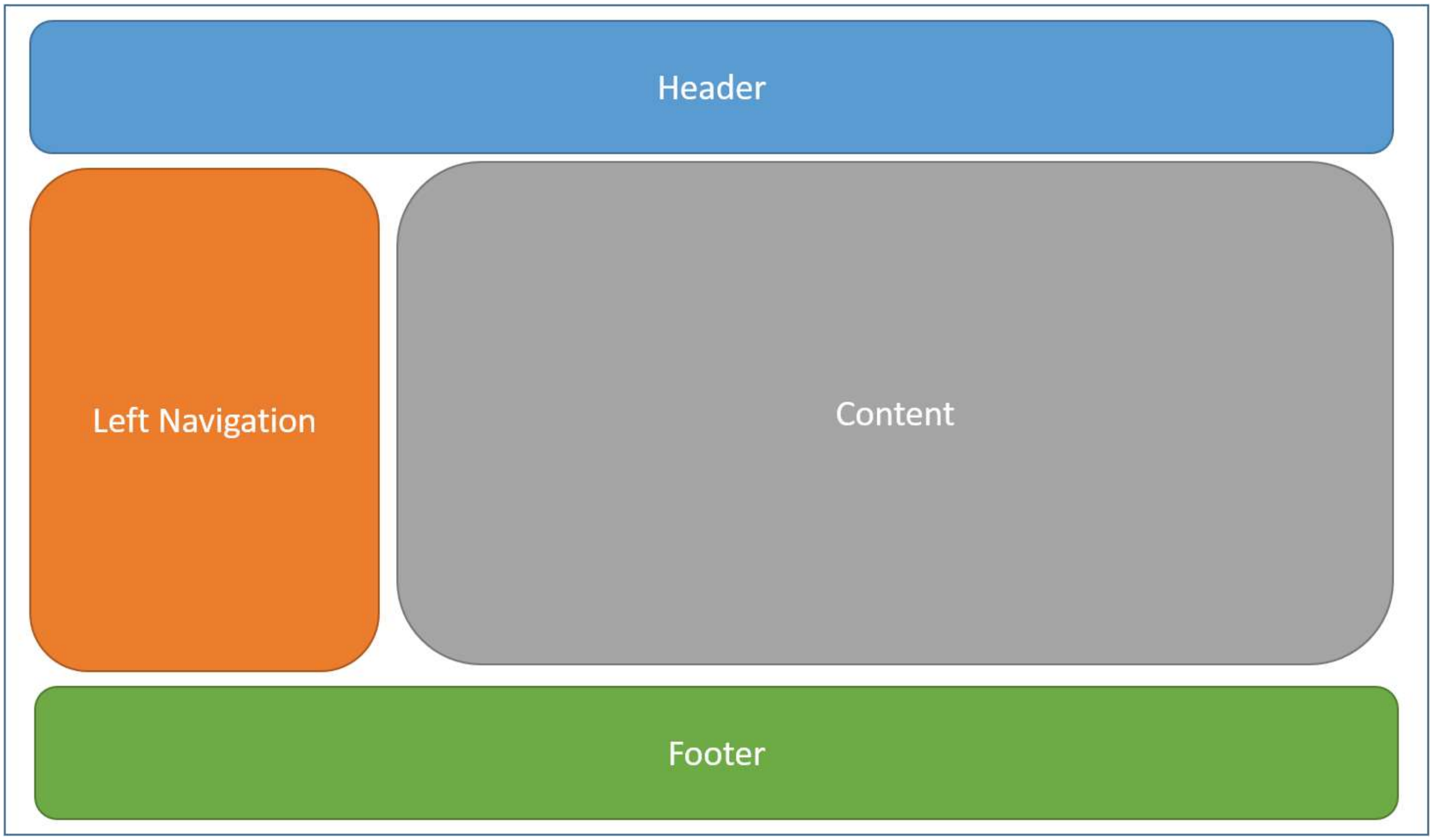
- Use common layouts.
- Share directives.
- Run common code before rendering pages or views.

This document discusses layouts for the two different approaches to ASP.NET Core MVC: Razor Pages and controllers with views. For this topic, the differences are minimal:

- Razor Pages are in the *Pages* folder.
- Controllers with views uses a *Views* folder for views.

What is a Layout

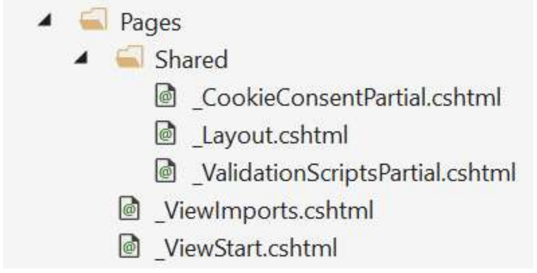
Most web apps have a common layout that provides the user with a consistent experience as they navigate from page to page. The layout typically includes common user interface elements such as the app header, navigation or menu elements, and footer.



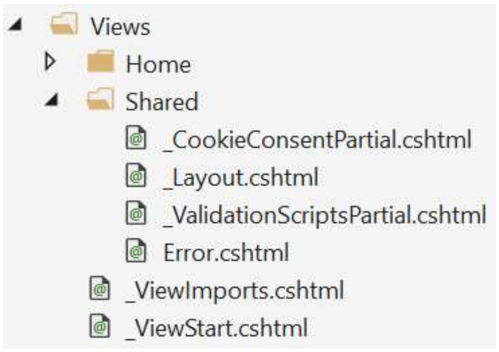
Common HTML structures such as scripts and stylesheets are also frequently used by many pages within an app. All of these shared elements may be defined in a *layout* file, which can then be referenced by any view used within the app. Layouts reduce duplicate code in views.

By convention, the default layout for an ASP.NET Core app is named `_Layout.cshtml`. The layout files for new ASP.NET Core projects created with the templates are:

- Razor Pages: `Pages/Shared/_Layout.cshtml`



- Controller with views: `Views/Shared/_Layout.cshtml`



The layout defines a top level template for views in the app. Apps don't require a layout. Apps can define more than one layout, with different views specifying different layouts.

The following code shows the layout file for a template created project with a controller and views:

CSHTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - WebApplication1</title>

    <environment include="Development">
        <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />
        <link rel="stylesheet" href="~/css/site.css" />
    </environment>
    <environment exclude="Development">
        <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />
        <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
    </environment>
</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a asp-page="/Index" class="navbar-brand">WebApplication1</a>
            </div>
            <div class="navbar-collapse collapse">
                <ul class="nav navbar-nav">
                    <li><a asp-page="/Index">Home</a></li>
                    <li><a asp-page="/About">About</a></li>
                    <li><a asp-page="/Contact">Contact</a></li>
                </ul>
            </div>
        </div>
    </nav>

    <partial name="_CookieConsentPartial" />

    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
```

```
            <p>&copy; 2018 - WebApplication1</p>
        </footer>
    </div>

    <environment include="Development">
        <script src="~/lib/jquery/dist/jquery.js"></script>
        <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
        <script src="~/js/site.js" asp-append-version="true"></script>
    </environment>
    <environment exclude="Development">
        <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
                asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
                asp-fallback-test="window.jQuery"
                crossorigin="anonymous"
                integrity="sha384-tsQFqpERu7ZLhBV2VZlAu7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT">
        </script>
        <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
                asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
                asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
                crossorigin="anonymous"
                integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa">
        </script>
        <script src="~/js/site.min.js" asp-append-version="true"></script>
    </environment>

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

Specifying a Layout

Razor views have a `Layout` property. Individual views specify a layout by setting this property:

CSHTML

```
@{
    Layout = "_Layout";
}
```

The layout specified can use a full path (for example, `/Pages/Shared/_Layout.cshtml` or `/Views/Shared/_Layout.cshtml`) or a partial name (example: `_Layout`). When a partial name is provided, the Razor view engine searches for the layout file using its standard discovery process. The folder where the handler method (or controller) exists is searched first, followed by the *Shared* folder. This discovery process is identical to the process used to discover partial views.

By default, every layout must call `RenderBody`. Wherever the call to `RenderBody` is placed, the contents of the view will be rendered.

Sections

A layout can optionally reference one or more *sections*, by calling `RenderSection`. Sections provide a way to organize where certain page elements should be placed. Each call to `RenderSection` can specify whether that section is required or optional:

HTML

```
<script type="text/javascript" src="~/scripts/global.js"></script>

@RenderSection("Scripts", required: false)
```

If a required section isn't found, an exception is thrown. Individual views specify the content to be rendered within a section using the `@section` Razor syntax. If a page or view defines a section, it must be rendered (or an error will occur).

An example `@section` definition in Razor Pages view:

HTML

```
@section Scripts {
     <script type="text/javascript" src="~/scripts/main.js"></script>
}
```

In the preceding code, `scripts/main.js` is added to the `scripts` section on a page or view. Other pages or views in the same app might not require this script and wouldn't define a scripts section.

The following markup uses the Partial Tag Helper to render `_ValidationScriptsPartial.cshtml`:

HTML

```html
@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
```

The preceding markup was generated by scaffolding Identity.

Sections defined in a page or view are available only in its immediate layout page. They cannot be referenced from partials, view components, or other parts of the view system.

Ignoring sections

By default, the body and all sections in a content page must all be rendered by the layout page. The Razor view engine enforces this by tracking whether the body and each section have been rendered.

To instruct the view engine to ignore the body or sections, call the `IgnoreBody` and `IgnoreSection` methods.

The body and every section in a Razor page must be either rendered or ignored.

Importing Shared Directives

Views and pages can use Razor directives to import namespaces and use dependency injection. Directives shared by many views may be specified in a common `_ViewImports.cshtml` file. The `_ViewImports` file supports the following directives:

- `@addTagHelper`
- `@removeTagHelper`
- `@tagHelperPrefix`
- `@using`
- `@model`
- `@inherits`
- `@inject`
- `@namespace`

The file doesn't support other Razor features, such as functions and section definitions.

A sample `_ViewImports.cshtml` file:

CSHTML

```cshtml
@using WebApplication1
@using WebApplication1.Models
@using WebApplication1.Models.AccountViewModels
@using WebApplication1.Models.ManageViewModels
@using Microsoft.AspNetCore.Identity
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

The `_ViewImports.cshtml` file for an ASP.NET Core MVC app is typically placed in the *Pages* (or *Views*) folder. A `_ViewImports.cshtml` file can be placed within any folder, in which case it will only be applied to pages or views within that folder and its subfolders. `_ViewImports` files are processed starting at the root level and then for each folder leading up to the location of the page or view itself. `_ViewImports` settings specified at the root level may be overridden at the folder level.

For example, suppose:

- The root level `_ViewImports.cshtml` file includes `@model MyModel1` and `@addTagHelper *, MyTagHelper1`.
- A subfolder `_ViewImports.cshtml` file includes `@model MyModel2` and `@addTagHelper *, MyTagHelper2`.

Pages and views in the subfolder will have access to both Tag Helpers and the `MyModel2` model.

If multiple `_ViewImports.cshtml` files are found in the file hierarchy, the combined behavior of the directives are:

- `@addTagHelper`, `@removeTagHelper`: all run, in order
- `@tagHelperPrefix`: the closest one to the view overrides any others

- `@model`: the closest one to the view overrides any others
- `@inherits`: the closest one to the view overrides any others
- `@using`: all are included; duplicates are ignored
- `@inject`: for each property, the closest one to the view overrides any others with the same property name

Running Code Before Each View

Code that needs to run before each view or page should be placed in the `_ViewStart.cshtml` file. By convention, the `_ViewStart.cshtml` file is located in the *Pages* (or *Views*) folder. The statements listed in `_ViewStart.cshtml` are run before every full view (not layouts, and not partial views). Like ViewImports.cshtml, `_ViewStart.cshtml` is hierarchical. If a `_ViewStart.cshtml` file is defined in the view or pages folder, it will be run after the one defined in the root of the *Pages* (or *Views*) folder (if any).

A sample `_ViewStart.cshtml` file:

CSHTML

```
@{
    Layout = "_Layout";
}
```

The file above specifies that all views will use the `_Layout.cshtml` layout.

`_ViewStart.cshtml` and `_ViewImports.cshtml` are **not** typically placed in the */Pages/Shared* (or */Views/Shared*) folder. The app-level versions of these files should be placed directly in the */Pages* (or */Views*) folder.