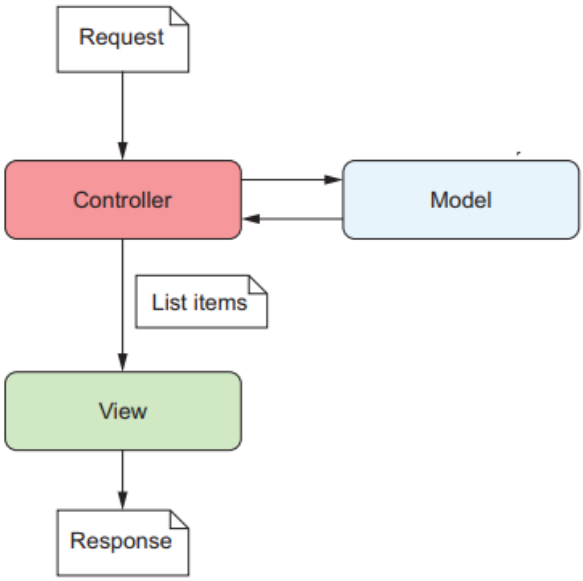
View trong ASP.NET Core MVC, layout, viewstart, viewimports

Trong mô tả chung của mẫu kiến trúc MVC, view là thành phần chịu trách nhiệm sinh ra giao diện cho ứng dụng. Đây là khâu cuối cùng trong chuỗi xử lý yêu cầu của người dùng trước khi trả lại kết quả.

Bạn có thể hình dung vị trí của view trong kiến trúc MVC qua sơ đồ minh họa dưới đây.



Tuy nhiên, mỗi framework khi thực thi MVC lại diễn đạt view và mối quan hệ giữa nó với các thành phần khác theo cách riêng. Chúng ta xem xét view trong quan hệ với các thành phần còn lại của MVC trong ASP.NET Core MVC.

NỘI DUNG CỦA BÀI [Ẩn]

Làm việc với view trong ASP.NET Core MVC

Để minh họa, trước hết chúng ta cùng thực hiện một ví dụ.

Bước 1. Tạo một project rỗng WebApplication1 và cấu hình để biến nó thành một project MVC sử dụng Startup class sau:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace WebApplication1 {
    public class Startup {
        public void ConfigureServices(IServiceCollection services) {
            services.AddControllersWithViews();
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {
            if (env.IsDevelopment()) {
                app.UseDeveloperExceptionPage();
            }

            app.UseRouting();

            app.UseEndpoints(endpoints => {
                endpoints.MapControllerRoute("default", "{controller}/{action}");
            });
        }
    }
}
```

Bước 2. Tạo thư mục Controllers. Trong thư mục này tạo class mới HomeController trong file HomeController.cs.

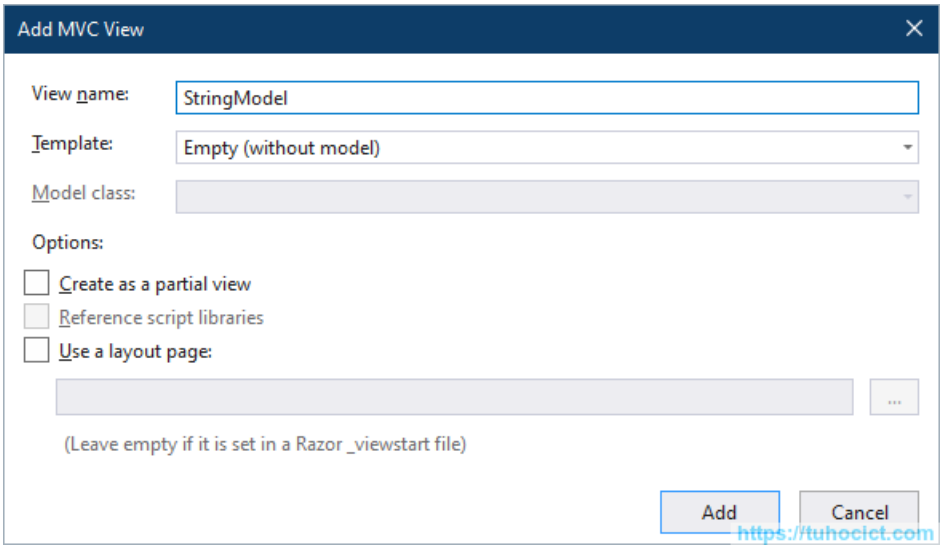
```
using System;
using Microsoft.AspNetCore.Mvc;

namespace WebApplication1.Controllers {
    public class HomeController : Controller {
        public IActionResult StringModel() {
            ViewData.Model = "Hello world from Home-Index action";
            var view = new ViewResult();
            view.ViewName = "/Views/Home/StringModel.cshtml";
            view.ViewData = ViewData;
            return view;
        }

        public IActionResult TupleModel() {
            ViewData.Model = ("Donald", "Trump", new DateTime(1900, 12, 31));
            var view = new ViewResult();
            view.ViewData = ViewData;
            return view;
        }
    }
}
```

Bước 3. Tạo thư mục Views trực thuộc dự án. Trong thư mục Views tạo thư mục Home.

Tạo file view bằng cách click phải chuột vào thư mục Home, chọn Add => View. Đặt tên cho view là StringModel.



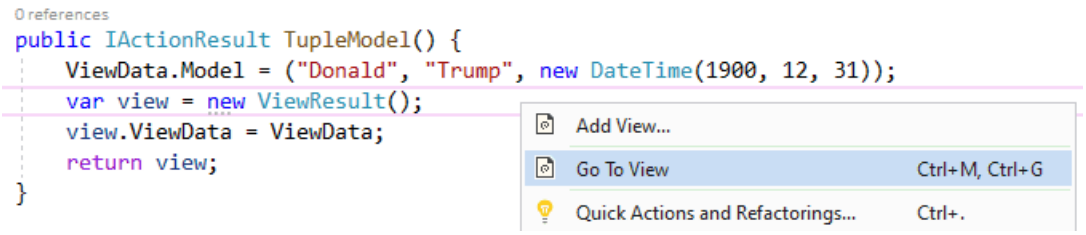
Bạn sẽ thu được file view đầu tien StringModel.cshtml. Viết code cho StringModel.cshtml như sau:

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

<!DOCTYPE html>

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

Bước 4. click phải vào một dòng bất kỳ bên trong phương thức TupleModel và chọn Add View.



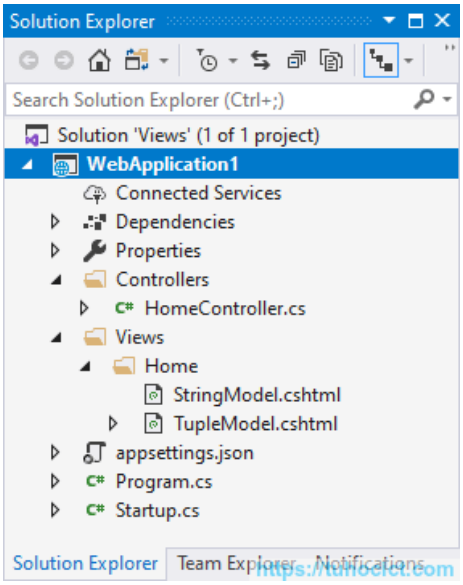
Từ hộp thoại Add View tạo một view với có tên TupleModel.cshtml. Viết code cho TupleModel.cshtml như sau:

```
@model (string FirstName, string LastName, DateTime BirthDay)
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>TupleModel</title>
</head>
<body>
    <h1>Hello, @Model.FirstName <strong>@Model.LastName</strong></h1>
```

```
15.         <h2>Your birthday is @Model.BirthDay</h2>
16.     </body>
17.     </html>
```

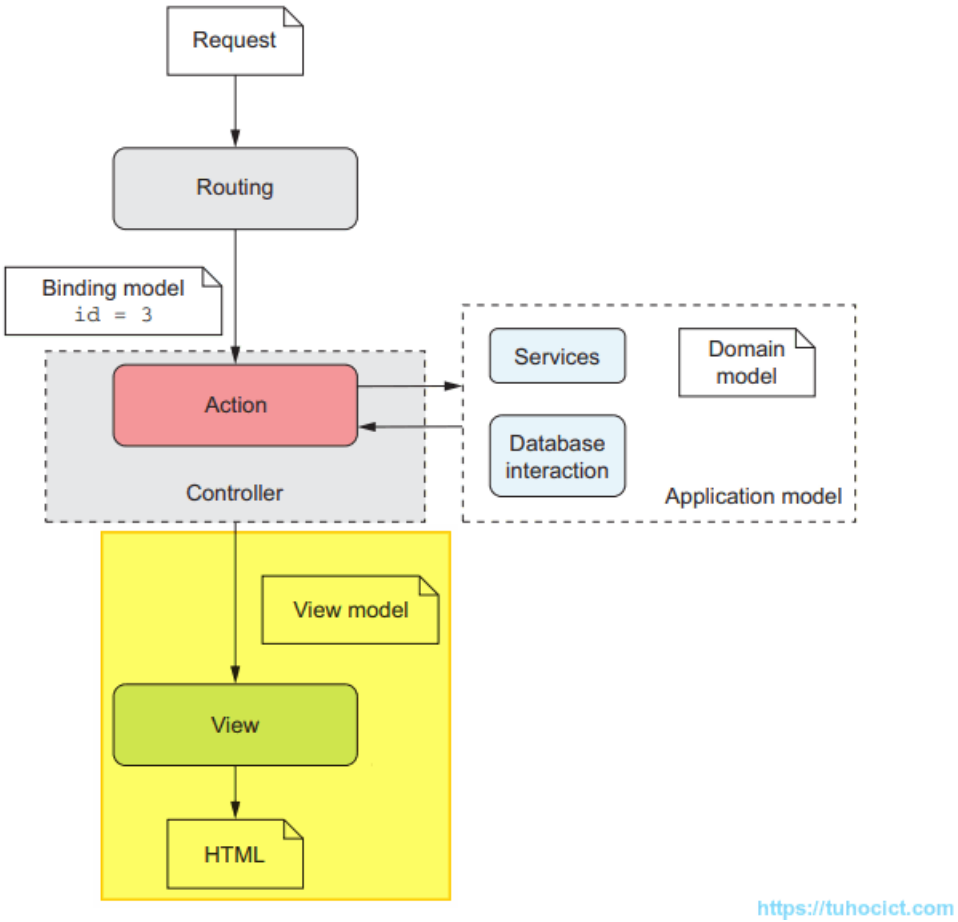


Dịch và chạy thử ứng dụng với hai URL:
(1) /home/stringmodel
(2) /home/tuplemodel

Chúng ta tạm dừng thực hành để nói về lý thuyết.

Quan hệ action – view

Trong một ứng ụng web, giao diện ứng dụng được tạo ra bởi tổ hợp HTML (cho nội dung) + CSS (cho định dạng) + JavaScript (cho tương tác). Các phần của giao diện được đóng gói trong gói tin phản hồi HTTP để trả về cho trình duyệt.

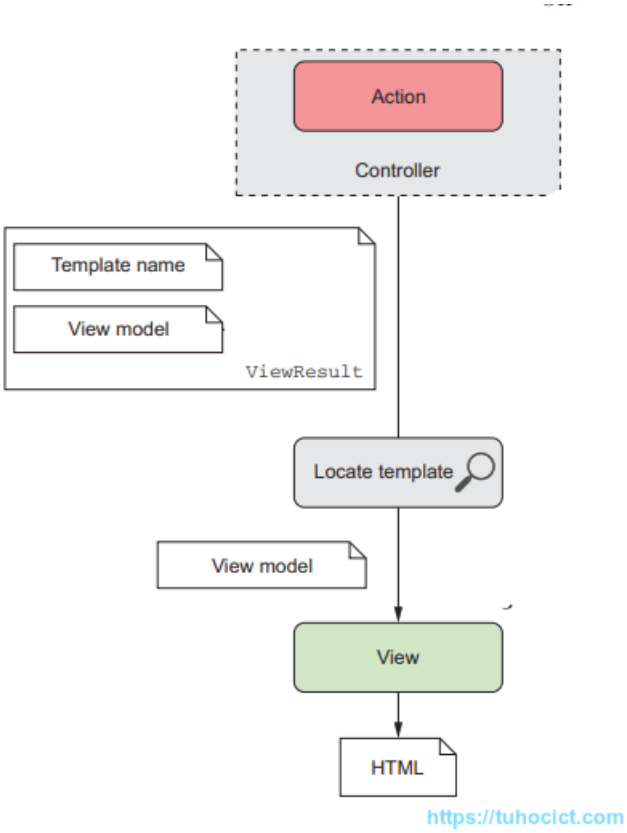


Trong bài học về controller và action bạn đã biết rằng, action chỉ định view để thể hiện dữ liệu, đồng thời cung cấp dữ liệu cần hiển thị.

Trong ví dụ trên chúng ta có HomeController với hai action StringModel và TupleModel. StringModel chỉ định rõ ràng là phải sử dụng file `/Views/Home/StringModel.cshtml`. Đồng thời cung cấp dữ liệu là một chuỗi ký tự `"Hello world ... "`.

Tự bản thân view trong MVC không phải là giao diện. View là khuôn mẫu để tạo ra giao diện từ dữ liệu.

Để sinh ra giao diện, view cần biết dữ liệu sẽ được hiển thị. Dữ liệu này do controller action cung cấp. Action lấy dữ liệu từ việc tương tác với model (chính xác hơn là application model). Dữ liệu này có thể là một object, một danh sách object, v.v.. Để phân biệt, trong ASP.NET Core MVC, loại dữ liệu này được gọi là **view model**.



Trong hai action trên, view model lần lượt là chuỗi ký tự `"Hello world from Home-StringModel action"` và tuple `("Donald", "Trump", new DateTime(1900, 12, 31))`.

Để thể hiện thông tin, view cung cấp một khuôn mẫu (template) chứa những thông tin tĩnh (từ quá trình thiết kế) ghép nối với thông tin động (từ view model). Khuôn mẫu này trong ASP.NET Core MVC là các file **cshtml** được xây dựng bằng **Razor** – loại cú pháp đặc biệt kết hợp giữa HTML (cho thông tin tĩnh) và C# (cho thông tin động).

Hãy để ý biến `Model` và directive `@model` trong các file cshtml ở trên. Đây là cách View trong MVC sử dụng view model do controller action cung cấp.

Directive @model chỉ định kiểu của Model. Model là property chứa view model mà action truyền sang cho view.

Qua ví dụ trên bạn đã thấy, `Model` sử dung trên Razor view chính là property `ViewData.Model` nhận từ `Controller`. Mọi view đều tham chiếu tới `ViewData` của `Controller`. "Model" chỉ là một cách sử dụng tắt trên view của ViewData.Model cho tiện lợi.

> Để thực sự biến view (với vai trò một template) thành giao diện (tổ hợp HTML + CSS + JavaScript), ASP.NET Core cần sử dụng một **view engine**. View engine mặc định trong ASP.NET Core là **Razor**. Do vậy, loại cú pháp view engine này sử dụng khi xây dựng template cũng được gọi là **cú pháp Razor**.

Cú pháp Razor được sử dụng chung cho tất cả các framework cần đến giao diện người dùng trong ASP.NET Core, bao gồm Razor Pages, MVC, Blazor (sẽ học sau). Bạn đã học cú pháp

này khi học Razor Pages. Bạn có thể tiếp tục sử dụng nó cho view trong MVC mà không cần học lại nữa.

Nếu không muốn sử dụng Razor, ASP.NET Core còn cho phép bạn tự xây dựng và sử dụng view engine của riêng mình. Tuy nhiên đây không phải là lựa chọn của đa số lập trình viên ASP.NET Core.

> Lưu ý rằng, Razor view (ViewResult) chỉ là một trong số các loại view của ASP.NET Core MVC. View hoàn toàn có thể không liên quan gì đến Razor như JSON hay XML, cũng có thể chỉ là một file tĩnh (FileResult), là các mã lỗi (NotFoundResult), mã điều hướng (RedirectResult), v.v..

File và thư mục cho view

Mỗi view trong ASP.NET Core MVC thường tương ứng với một controller action, và mỗi controller action thường có một view tương ứng. Do vậy, giữa tên và đường dẫn của view với controller action có một quy ước xác định: tên gọi và đường dẫn của view cần tương ứng với cấu trúc `/Views/{controller}/{action}.cshtml`.

Ví dụ, phương thức action `Index` của `HomeController` sẽ có view tương ứng là `/Views/Home/Index.cshtml`. Và khi nhìn đến đường dẫn của file cshtml này ta cũng hình dung ra ngay nó là view tương ứng của action nào.

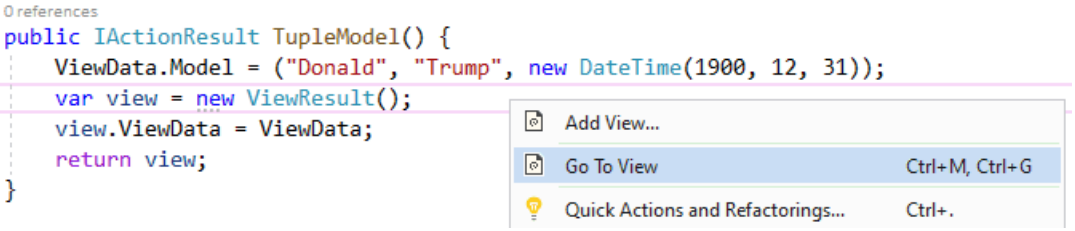
Như vậy, theo quy ước này, các view của một controller được tập trung vào một thư mục cùng tên với controller nằm trong thư mục cha `/Views`.

Trong ví dụ trên, cả hai file view đều nằm trong thư mục /Views/Home với tên tương ứng với action: StringModel.cshtml và TupleModel.cshtml.

Quy ước này không chỉ tiện lợi cho việc quản lý mà còn giúp bạn viết code ngắn gọn hơn thông qua cơ chế cấu hình dựa trên quy ước.

Trong ví dụ trên bạn để ý thấy rằng ở phương thức TupleModel bạn không hề chỉ định file view nào sẽ sử dụng. Tuy nhiên, ASP.NET Core MVC vẫn lựa chọn được chính xác file bạn muốn. Đây chính là tác dụng của việc đặt tên và đường dẫn view theo đúng quy ước: ASP.NET Core MVC cho phép không cần chỉ định tên filew view trong action nếu tên file và đường dẫn tuân thủ quy ước đặt tên.

Khi bạn sử dụng lệnh tạo view:



Visual Studio sẽ tự động tạo file theo đúng quy ước chúng ta đã nói. Đây là cách nhanh chóng và tiện lợi nhất để tạo view từ action.

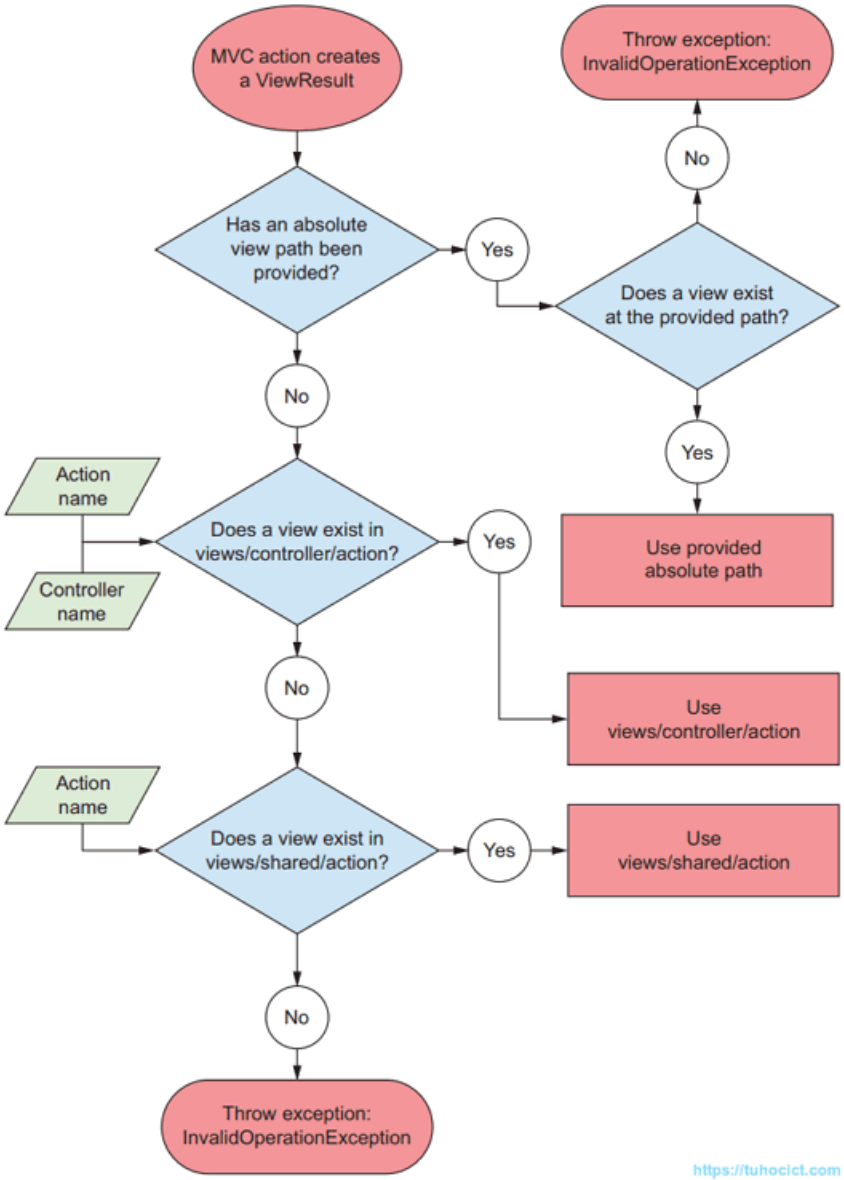
Trên thực tế, nếu bạn không chỉ định tên view, ASP.NET Core MVC sẽ làm như sau:

- Tìm xem có thư mục /Views/{controller} hay không.
- Nếu có thư mục này, tiếp tục tìm file {action}.cshtml trong đó. Nếu không thấy thư mục này thì sẽ tìm kiếm ngay {action}.cshtml trong /Views/Shared.
- Nếu không tìm thấy file trong /Views/{controller} sẽ tiếp tục tìm file {action}.cshtml trong thư mục /Views/Shared.

Trong bất kỳ trường hợp nào, nếu không tìm thấy file theo quy ước trên, ASP.NET Core MVC sẽ phát lỗi InvalidOperationException.

Tình hình cũng diễn ra tương tự nếu bạn chỉ định tên file view nhưng không chỉ định đường dẫn.

Dưới đây là sơ đồ chi tiết quá trình tìm kiếm view cho action:



Sử dụng phương thức hỗ trợ của Controller

Tiếp tục với bài thực hành đã làm ở trên.

Bước 1. Bổ sung ba phương thức sau vào HomeController:

```
public IActionResult ResultHelper() {
    return View();
}

public IActionResult SpecifyView() {
    return View("ResultHelper");
}

public IActionResult ReuseView() {
    var model = ("Putin", "Vladimir", new DateTime(1990, 5, 3));
    return View("TupleModel", model);
}
```

Bước 2. Sử dụng một trong những kỹ thuật đã biết để tạo view tương ứng với action này (ResultHelper.cshtml) và viết code như sau:

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ResultHelper</title>
</head>
<body>
    <h1>This view was located automatically</h1>
</body>
</html>
```

Bạn có thể chạy thử ứng dụng với URL `/home/resulthelper`, `/home/specifyview` và `/home/reuseview`.

Trong ví dụ trên, bạn gọi phương thức View sẵn có của lớp cha Controller.

Đây là phương thức tắt của Controller cung cấp. Nó kết hợp lệnh tạo ViewResult, lệnh gán ViewData.Model (nếu có) và lệnh chỉ định view (nếu có) mà bạn đã biết.

Kết quả trả về của View là ViewResult. Vì vậy bạn có thể gọi View() cho action có kiểu trả về là ActionResult, IActionResult hoặc ViewResult.

Khi sử dụng View() kết hợp với quy ước đặt tên view, bạn sẽ làm gọn code đi rất nhiều.

Ngoài View(), Controller cũng cung cấp một loạt phương thức hỗ trợ khác. Các phương thức này được đặt tên theo action result tương ứng bỏ đi hậu tố Result. Theo đó,

- ViewResult => phương thức View(),
- RedirectResult => phương thức Redirect(),
- FileResult => phương thức File(),
- NotFoundResult => phương thức NotFound(),
- v.v..

Đây cũng là một trong các lý do mà bạn nên xây dựng controller kế thừa từ lớp Controller, thay vì xây dựng mọi thứ từ số 0.

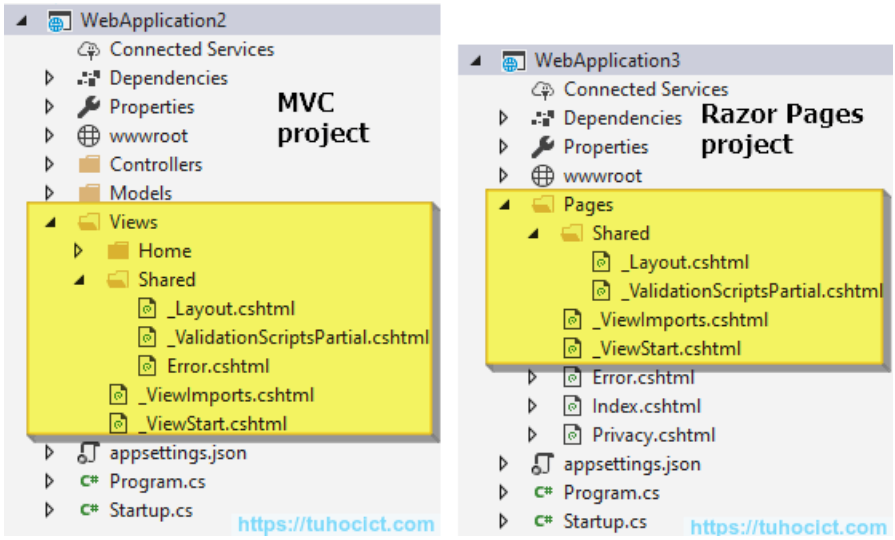
Mỗi phương thức trợ giúp có cách sử dụng riêng. Chúng ta sẽ học cách sử dụng một số phương thức trong các bài học học có liên quan.

Layout, view start, section, view imports trong MVC

Do cùng sử dụng Razor view engine, view trong MVC hoạt động hoàn toàn giống như trang nội dung (content page) của Razor Pages. Nghĩa là bạn cũng có thể sử dụng layout, view start, section hay view imports trong ứng dụng MVC.

> Nếu không nhớ, hãy đọc lại bài học về layout, view start và section trong ASP.NET Core Razor Pages. View imports đã học trong bài về cấu trúc điều khiển của Razor.

Sự khác biệt duy nhất bạn thấy là thư mục chứa view trong MVC là `/Views`, còn trong Razor Pages trang nội dung được đặt trong thư mục `/Pages`. Tức là /Views trong MVC hoàn toàn tương đương với /Pages trong Razor Pages.



Template cho dự án MVC cũng đặt tên cho file layout mặc định là **_Layout.cshtml** và đặt nó trong thư mục `/Views/Shared`.

Thư mục này cũng có cùng ý nghĩa với `/Pages/Shared` của Razor Pages. Đây là thư mục đặc biệt mà Razor view engine tự động tìm kiếm nếu không tìm thấy file cần thiết ở các thư mục theo quy ước.

Ví dụ, nếu không tìm thấy view cho action trong thư mục quy ước /Views/{controller}/{action}.cshtml, Razor view engine sẽ tìm kiếm file {action}.cshtml trong thư mục /Views/Shared.

Trong layout bạn cũng có thể sử dụng **section** để chỉ định vị trí xuất nội dung với lệnh @RenderSection. Trong layout mặc định cũng đặt lệnh @RenderSection("Scripts", false) ở ngay trước khi đóng thẻ </body> giúp bạn chèn các khối javascript từ các view khi cần thiết.

File **_ViewStart.cshtml** cho phép chạy một đoạn code Razor chung trước khi xử lý bất kỳ view nào. Do vậy _ViewStart được lợi dụng để chỉ định layout chung cho tất cả các view:

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

Lưu ý tên file bắt buộc phải là _ViewStart.cshtml. Trong dự án có thể sử dụng nhiều _ViewStart khác nhau. Mỗi _ViewStart có tác dụng trong thư mục chứa nó và tất cả thư mục con. _ViewStart trong thư mục con sẽ có tác dụng mạnh hơn _ViewStart từ thư mục cha. _ViewStart trong thư mục Views sẽ có tác dụng với tất cả các view.

Nếu bạn cần chỉ định namespace nào mà tất cả các view trong dự án cần truy xuất, bạn có thể đặt directive `@using` tương ứng trong _ViewImports.cshtml. Tương tự _ViewStart, _ViewImports cũng có tác dụng trong thư mục chứa nó và thư mục con. _ViewImports trong thư mục Views có tác dụng với tất cả các view.

Nói tóm lại, layout, section, view start, view imports trong MVC không có gì khác so với trong Razor Pages. Bạn không cần học thêm gì để sử dụng chúng trong dự án MVC. ASP.NET Core Razor Pages và ASP.NET Core MVC cùng sử dụng Razor view engine nên tất cả các khái niệm trên trong hai loại dự án thực tế là một.

Kết luận

Trong bài học này chúng ta đã xem xét chi tiết về view trong ASP.NET Core MVC, bao gồm quan hệ giữa view và action, quy ước đặt tên file và đường dẫn, cũng như sử dụng phương thức hỗ trợ cho tương tác action-view.

Do view trong ASP.NET Core MVC sử dụng cú pháp Razor và ViewData giống hệt như Razor Pages, chúng ta không cần học lại những vấn đề này nữa. Bạn có thể dễ dàng vận dụng những gi đã học về Razor view vào MVC.

View ASP.NET Core MVC cũng sử dụng các thành phần như layout, section, view imports, view start như trong dự án Razor Pages.

Lưu ý rằng, bài học này chỉ mang tính chất giới thiệu chung về view trong MVC. Để thực sự làm việc với giao diện người dùng bạn còn phải học nhiều vấn đề khác như sử dụng form, tag helper, validation, model binding, v.v.. Các nội dung này sẽ lần lượt được trình bày trong khóa học.