View component trong ASP.NET Core

View component là cơ chế tái sử dụng trong Razor Pages (cùng với Partial Pages). View component hướng tới xây dựng các thành phần tương đối độc lập, phức tạp với các logic xử lý dữ liệu riêng.

Hãy tưởng tượng một giao diện web thường gặp với các thành phần header, menu, sidebar, footer. Menu thường được tạo ra dựa trên dữ liệu từ cơ sở dữ liệu thay vì được thiết kế tĩnh. Sidebar thường chứa các thành phần như tagcloud, danh mục bài viết mới – các thành phần được tạo ra từ dữ liệu.

View component chính là giải pháp trong Razor Pages cho việc phân chia giao diện ra các thành phần tương đối độc lập như vậy để có thể tái sử dụng qua các trang.

> Các vấn đề trình bày trong bài học này có thể áp dụng chung cho cả Razor Pages và MVC. Để đơn giản, các ví dụ được thể hiện với Razor Pages. Bạn có thể áp dụng nguyên vẹn với MVC.

NỘI DUNG CỦA BÀI [Ẩn]

Tại sao bạn cần đến View Component

Hãy để ý chính website bạn đang đọc. Cấu trúc của nó đại khái chia làm phần header, sidebar và footer. Header có logo và hai menu (main menu và top menu). Sidebar có tag cloud, danh sách bài mới, mục lục bài học. Footer có danh sách chuyên mục, danh sách bài đặc biệt và một số thông tin khác.

Các phần này có mặt trên hầu hết các page. Trong Razor Pages, bạn dễ dàng hình dung ra rằng các thành phần như header, footer và sidebar cần phải đặt trong layout.

Tuy nhiên bạn cũng cần để ý rằng, main menu của site này không được thiết kế tĩnh. Các mục của menu được tải từ cơ sở dữ liệu. Tương tự như vậy, tag cloud, danh sách bài mới hay mục lục bài học (gọi chung là các widget – theo cách gọi của WordPress) đều cần làm việc với cơ sở dữ liệu. Chúng có những tính năng độc lập, có logic xử lý của riêng mình.

Như bạn đã biết, layout trong Razor Pages dường như không thực hiện được những logic phức tạp như vậy vì nó không có model class hỗ trợ như các page. Thực tế, layout không

phải là một trang Razor thông thường. Nó chỉ là nơi ghép nối các thành phần vào một thể thống nhất.

Như vậy, chúng ta cần một cơ chế cho phép xây dựng ra các thành phần giao diện độc lập với khả năng xử lý dữ liệu và logic phức tạp, đồng thời có thể ghép nối (tái sử dụng) nó vào các trang Razor.

Cơ chế này trong Razor Pages (và Asp.net Core) được gọi là View Component.

View Component là một bộ phận giao diện độc lập có thể tái sử dụng. Mỗi View Component được tạo thành từ một model class (xử lý logic) và một trang Razor (hiển thị) tương tự như một page thông thường. Tuy nhiên View Component bắt buộc phải được nhúng vào page hoặc layout.

Theo logic này, ngay từ đầu bạn có thể phân chia giao diện ra các View Component (header, menu, tag cloud, footer, v.v.) và phát triển độc lập. Sau đó, chúng được ghép nối vào layout (nếu cần sử dụng trên tất cả các trang) hoặc nhúng vào một số trang cụ thể.

> Một cơ chế tái sử dụng UI khác trong Razor Pages (và Asp.net Core) là Partial Page. Tuy nhiên Partial Page hướng tới tái sử dụng các khối code Razor, thay vì là một thành phần độc lập với khả năng xử lý dữ liệu và logic riêng như View Component.

Thực hành 1 – Xây dựng View Component đơn giản

Tạo một project thuộc loại Web Application và thực hiện các bước sau đây.

Bước 1. Tạo thư mục ViewComponents trực thuộc project.

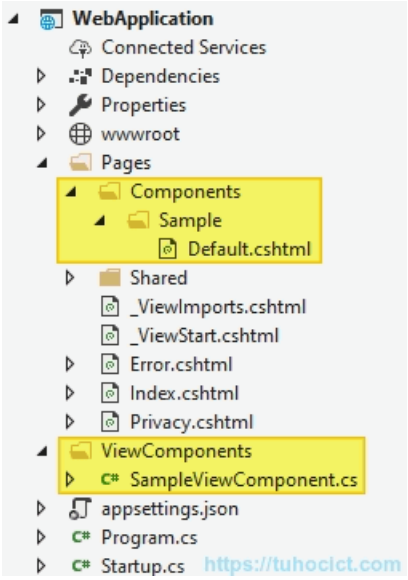
Bước 2. Tạo thư mục Components nằm trong thư mục Pages.

Bước 3. Tạo class `SampleViewComponent` trong file *SampleViewComponent.cs* trong *thư mục ViewComponents*.

Bước 4. Tạo thư mục con Sample bên trong thư mục Components.

Bước 5. Tạo file Default.cshtml trong thư mục Sample.

Sau 5 bước này bạn thu được cấu trúc thư mục/file như sau:



Bước 6. Mở file SampleViewComponent.cs và viết code như sau:

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication.ViewComponents {
    public class SampleViewComponent : ViewComponent {
        public IViewComponentResult Invoke() {
            (string greeting, string name) data = ("Hello", "Donald Trump");
            return View(data);
        }
    }
}
```

Bước 7. Mở trang Default.cshtml và code như sau:

```
@model (string greeting, string name)

<div class="sample-component">
    <p class="h4">@Model.greeting, <strong>@Model.name</strong></p>
    <p>Welcome to the ASP.NET Core course!</p>
</div>
```

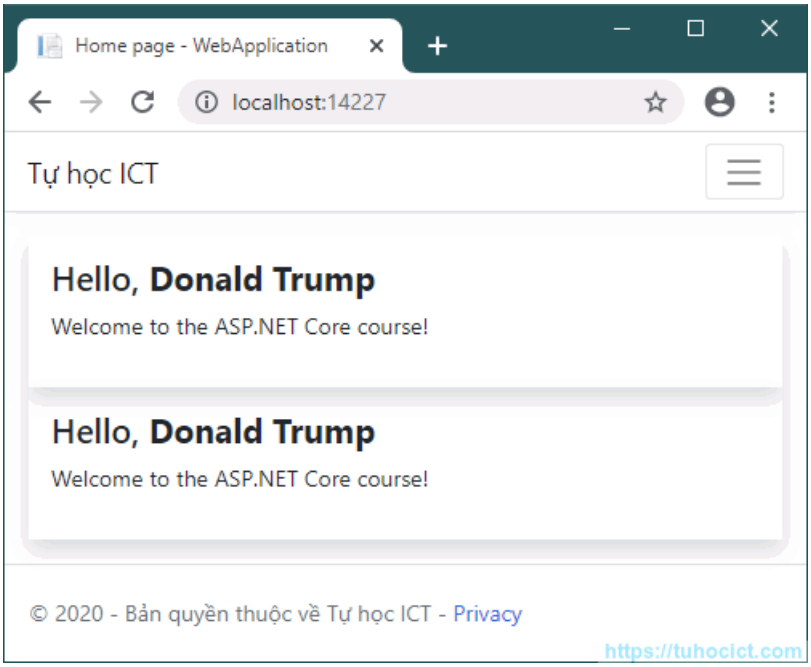
Bước 8. Mở trang Index.cshtml và code như sau:

```
@page
@model IndexModel
@addTagHelper *, WebApplication

<div class="shadow p-3">@await Component.InvokeAsync("Sample")</div>

<div class="shadow p-3"><vc:sample /></div>
```

Chạy chương trình bạn sẽ thu được kết quả như sau:



Qua phần thực hành này bạn đã xây dựng được một view component đơn giản có tên là Sample. Component này không thực hiện những gì phức tạp. Nó chỉ đơn giản là in ra vài dòng chữ từ dữ liệu có sẵn.

Bạn có thể thấy component này có hai thành phần: một class xử lý có tên là `SampleViewComponent`, một Razor view `Default.cshtml` chịu trách nhiệm hiển thị dữ liệu. SampleViewComponent còn được gọi là **component class**; Default.cshtml được gọi là **view**.

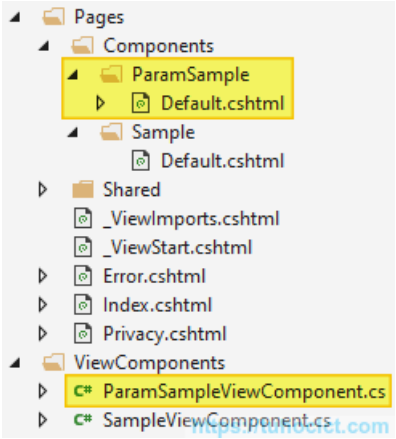
Bạn cũng để ý thấy chúng ta có thể sử dụng component này theo hai cách khác nhau, dưới dạng một biểu thức Razor, hoặc ở dạng thẻ markup.

Thực hành 2 – View Component với tham số

Ở phần thực hành 1 bạn đã xây dựng được một view component đơn giản. Tuy nhiên component này hoàn toàn cô lập. Bạn không truyền giá trị từ ngoài vào để điều khiển hoạt động của nó.

Trong phần thực hành này chúng ta sẽ tạo ra một view component mới có khả năng tiếp nhận dữ liệu từ page gọi nó.

Tiếp tục với project từ thực hành 1, lặp lại bước 3-4-5 để tạo component mới ParamSample.



Viết code cho class **`ParamSampleViewComponent`** như sau:

```
using Microsoft.AspNetCore.Mvc;

namespace WebApplication.ViewComponents {
    public class ParamSampleViewComponent : ViewComponent {
        public IViewComponentResult Invoke(string greeting, string name) {
            (string greeting, string name) data = (greeting, name);
            return View("Default", data);
        }
    }
}
```

Viết code cho **`ParamSample/Default.cshtml`** như sau:

```
@model (string greeting, string name)

<div class="sample-component">
    <p class="h4">@Model.greeting, <strong>@Model.name</strong></p>
    <p>Welcome to the ASP.NET Core course!</p>
</div>
```

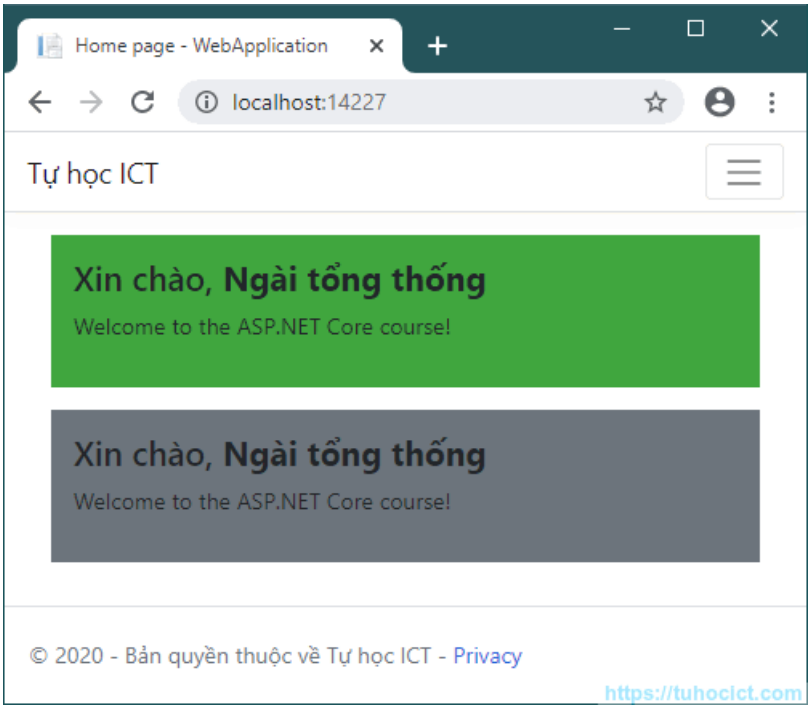
Điều chỉnh lại **`Index.cshtml`** như sau:

```
@page
@model IndexModel
@addTagHelper *, WebApplication

<div class="p-3 m-3 bg-success">@await Component.InvokeAsync("ParamSample", new { greeti

<div class="p-3 m-3 bg-secondary"><vc:param-sample greeting="Xin chào" name="Ngài tổng t
```

Chạy chương trình và bạn thu được kết quả như sau:



Ở phần thực hành này bạn tạo ra một component với cùng tính năng như của phần thực hành 1. Khác biệt duy nhất là bạn có thể cung cấp thông tin khi gọi component, thay vì để nó hoạt động cô lập.

Khi truyền tham số cho phương thức InvokeAsync, hãy để ý tham số thứ hai:

```
new { greeting = "Xin chào", name = "Ngài tổng thống" }
```

Đây là một object thuộc kiểu vô danh với hai property trùng tên và kiểu với tham số của phương thức Invoke (trong class ParamSampleViewComponent).

Đối với lối gọi markup,

```
<vc:param-sample greeting="Xin chào" name="Ngài tổng thống" />
```

cũng để ý tên gọi của component ParamSample trở thành param-sample. Đây là cách chuyển sang *lối viết kebab* cho phù hợp với yêu cầu của taghelper.

Các thành phần của View Component

Mỗi View Component bao gồm hai thành phần: một class C# làm nhiệm vụ xử lý dữ liệu và logic, gọi là **component class**; một trang Razor cshtml có nhiệm vụ hiển thị dữ liệu, gọi là **view**.

Cơ chế này rất gần gũi với một trang Razor thông thường. Tuy nhiên về mặt kỹ thuật có những điểm cần lưu ý.

Component class bắt buộc phải tuân thủ các yêu cầu sau:

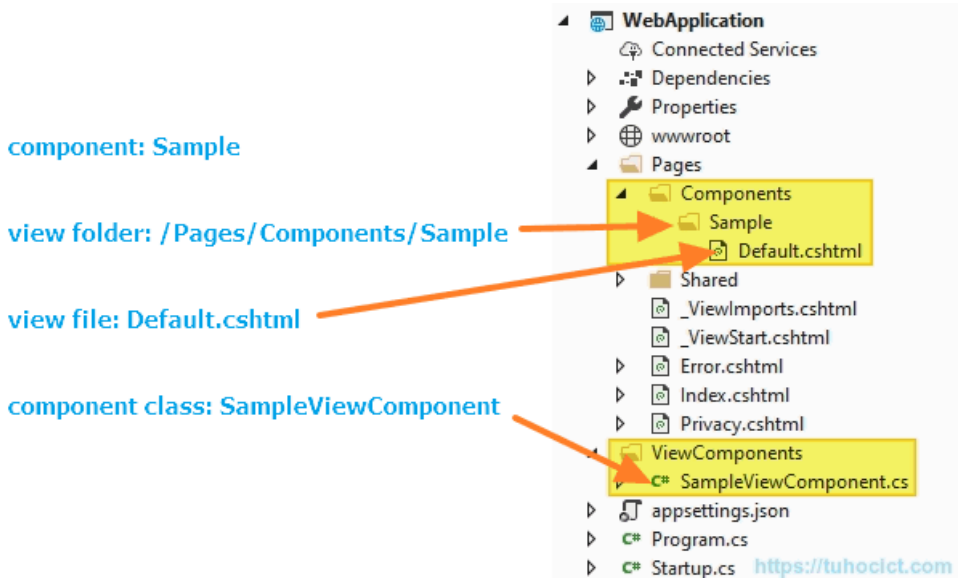
1. Phải kế thừa từ lớp cha ViewComponent;
2. Tên gọi phải kết thúc bằng ViewComponent, hoặc được đánh dấu với attribute `[ViewComponent]`;
3. Phải có phương thức `public IViewComponentResult Invoke()`, hoặc `public Task<IViewComponentResult> InvokeAsync()` (nếu bạn muốn View Component thực thi bất đồng bộ).

Bạn có thể thỏa mãn điều kiện 1 và 2 nếu class bạn xây dựng *kế thừa từ một View Component class khác*.

Điều kiện 3 thường được thực hiện bằng cách gọi `return View(...)` trong phương thức Invoke hoặc InvokeAsync.

File cho component class có thể đặt ở bất kỳ đâu. Tuy nhiên người ta thường đặt nó trong *thư mục ViewComponents trực thuộc dự án*.

Phần tên class không chứa cụm "ViewComponent" cũng là **tên component**. Ví dụ, với class SampleViewComponent thì Sample là tên của component. *Tên component được sử dụng làm tên thư mục chứa view* cho component.



Thành phần **view** là một file Razor (cshtml) tuân thủ các yêu cầu:

1. *Không chứa directive @page*, do đây không phải là một trang thông thường.
2. Để chỉ định kiểu dữ liệu tiếp nhận từ View Component class, view có thể chứa directive `@model` tương tự như một page thông thường.
3. Phải nằm trong thư mục `/Pages/Components/<tên component>/`.
4. Tên của view mặc định là `Default.cshtml`, tuy nhiên bạn có thể đặt tên bất kỳ.

Ví dụ, với Sample component, bạn phải tạo view là `/Pages/Components/Sample/Default.cshtml`.

Đối với điều kiện 4, nếu bạn đặt tên view là Default.cshtml thì khi gọi return View(…) bạn không cần chỉ định tên view. Razor Pages sẽ tự động sử dụng file Default.cshmlt trong thư mục tương ứng. Nếu đặt tên khác, bạn cần chỉ định cụ thể khi khởi tạo View.

Cũng do không chứa directive @page, mặc dù nằm trong thư mục Pages, bạn *không thể* truy xuất được file Default.cshtml. Tức là bạn không thể truy xuất nó theo Url (mặc định) ~~`/components/sample/default`~~. Nói cách khác, file này không tham gia vào cơ chế routing của Razor.

Sử dụng View Component trong trang Razor

View component có thể được sử dụng trong bất kỳ trang cshtml nào. Có hai cách gọi view component.

Ở cách thứ nhất bạn gọi nó như một **biểu thức Razor**:

```
@await Component.InvokeAsync("tên_component", {kiểu vô danh chứa tham số})
```

Tên component chính là tên của component class bỏ đi phần ViewComponent.

Tham số thứ hai của InvokeAsync là một object thuộc kiểu vô danh có nhiệm vụ truyền tham số vào phương thức Invoke (hoặc InvokeAsync) của component class.

Ví dụ với ParamSample component với phương thức Invoke:

```
public IViewComponentResult Invoke(string greeting, string name) { ... }
```

Bạn cần gọi component này từ view như sau:

```
@await Component.InvokeAsync("ParamSample", new { greeting = "Hello", name =
"Donald" })
```

`new { greeting = "Hello", name = "Donald" }` tạo ra một object kiểu vô danh chứa hai giá trị `greeting` và `name` để truyền vào Invoke.

Ở cách thứ hai bạn sử dụng lối viết như một **thẻ đánh dấu**:

```
<vc:[view-component-name]
  parameter1="parameter1 value"
  parameter2="parameter2 value">
</vc:[view-component-name]>
```

Ở đây `vc:` là namespace mặc định dành cho các view component.

Với hai component Sample và ParamSample bạn xây dựng ở trên, lời gọi theo kiểu markup như sau:

```
<vc:sample />
<vc:param-sample greeting="Hello" name="Donald" />
```

Lưu ý, khi viết ở dạng markup, tên component và tên các tham số của Invoke phải chuyển về dạng **kebab case**.

> Lối viết kebab case:
> (1) tất cả chữ cái hoa được chuyển thành chữ cái thường.
> (2) Nếu chữ hoa không phải là chữ cái đầu tiên thì thêm dấu gạch ngang phía trước.
> Ví dụ, ProductId sẽ chuyển thành product-id, MainNavigation chuyển thành main-navigation.
> Lối viết này được sử dụng khi viết attribute cho tag helper.

Để sử dụng lối viết này, ở đầu page hoặc trong file _ViewImports bạn cần đăng ký view component với directive @addTaghelper với cấu trúc như sau:

```
@addTagHelper *, <tên_assembly>
```

Một chú ý khác khi dùng lối viết này là bạn có thể truyền trực tiếp giá trị cho tham số cũng như cung cấp biểu thức hoặc tên biến cho tham số.

Ví dụ, bạn có thể viết

```
<vc:user id="new Random().Next(1, 10)"></vc:user>
```

tức là với tham số id, bạn không truyền trực tiếp giá trị mà truyền một biểu thức.

<tên_assembly> cần thay bằng tên project hoặc thư viện dll chứa component.

Kết luận

Bài học đã trình bày chi tiết về view component trong Razor Pages. Đây là một trong số các cơ chế tái sử dụng của Razor Pages và Asp.net Core. Nhờ view component, một giao diện phức tạp có thể được phân chia thành các phần độc lập đơn giản hơn để phát triển song song.

Lưu ý rằng, view component không phải là page mà là một phần của page. Do vậy, view component không trực tiếp xử lý được truy vấn. Truy vấn luôn đi tới page chứa component này.

Do đó, nếu bạn tạo một form trên view component, dữ liệu từ form này sẽ đi đến page nơi chứa component. Component class của nó không thể nhận được dữ liệu đến từ form.

Do đặc điểm này, view component thường được sử dụng để xây dựng các thành phần không tương tác với người dùng. Dữ liệu cho view component đến từ file cấu hình, cơ sở dữ liệu hoặc dịch vụ web khác.

Nếu muốn xây dựng view component cho mục đích tương tác (tiếp nhận dữ liệu) với người dùng, bạn cần xây dựng thành phần tiếp nhận dữ liệu trên page nơi sử dụng component.

Tải mã nguồn solution ViewComponent

1 file(s) 0.00 KB

TẢI MÃ NGUỒN

+ Nếu bạn thấy site hữu ích, trước khi rời đi hãy **giúp đỡ** site bằng một hành động nhỏ để site có thể phát triển và phục vụ bạn tốt hơn.
+ Nếu bạn thấy bài viết hữu ích, hãy giúp **chia sẻ** tới mọi người.
+ Nếu có thắc mắc hoặc cần trao đổi thêm, mời bạn viết trong phần **thảo luận** cuối trang.
Cảm ơn bạn!