Controller trong ASP.NET Core MVC

Controller trong ASP.NET Core MVC là thành phần được xây dựng tương đối tự do. Bất kỳ class nào thỏa mãn một số tiêu đơn giản đều có thể trở thành controller trong ASP.NET Core MVC. Mvc Middleware sử dụng cơ chế reflection để tự động hóa việc phát hiện – khởi tạo – sử dụng controller.

NỘI DUNG CỦA BÀI [Ẩn]

Controller trong ASP.NET Core MVC

Controller trong ASP.NET Core MVC là class thỏa mãn các điều kiện sau:

- (1) Là loại class **có thể được khởi tạo**, nghĩa là class có hàm tạo công khai (public constructor) + không static và không abstract.
- (2) Có **tên gọi** kết thúc bằng "Controller", ví dụ HomeController, DefaultController.
- (3) Là lớp **dẫn xuất** của `Controller` hoặc `ControllerBase`.

Trong đó, điều kiện (2) và (3) không cần đồng thời thỏa mãn, nghĩa là nếu đã lớp dẫn xuất của `Controller` thì tên gọi không cần chứa hậu tố "Controller".

Như vậy có thể thấy cách xây dựng lớp controller trong ASP.NET Core MVC tương đối tự do. Ở giai đoạn hoạt động (runtime), MVC Middleware có khả năng phát hiện và sử dụng class với vai trò controller nếu class đó thỏa mãn các điều kiện đã nêu.

> ASP.NET Core MVC Middleware sử dụng cơ chế reflection để tìm tất cả các class trong assembly thỏa mãn các điều kiện trên. Khi tìm được class phù hợp, cơ chế này có thể tự động khởi tạo object tương ứng và sử dụng các phương thức của nó để xử lý truy vấn. Phương thức public của controller được sử dụng khi xử lý truy vấn cũng được gọi là **action**.

Mặc dù không bắt buộc, bạn nên xây dựng các lớp controller kế thừa từ lớp Controller. Lớp `Controller` cung cấp nhiều tiện ích giúp đơn giản hóa việc xây dựng các lớp controller cho ứng dụng. Một trong số những tiện ích quan trọng là các phương thức hỗ trợ trả kết quả (action result) mà bạn sẽ làm quen ở phần sau của bài học này.

Lớp `ControllerBase` có vai trò tương tự như `Controller` nhưng không chứa các phương thức hỗ trợ view. Vì vậy `ControllerBase` thường dùng khi xây dựng controller trong Web API.

Các lớp controller thường được đặt tập trung trong *thư mục Controllers* trực thuộc dự án. Tuy nhiên đây không phải là yêu cầu bắt buộc. Bạn có thể đặt lớp controller ở bất kỳ đâu trong dự án. Cơ chế reflection của Mvc Middleware phát hiện ra lớp controller dựa trên 3 tiêu chí đã nói ở trên chứ không căn cứ vào vị trí đặt file.

Trong ASP.NET Core MVC, truy vấn được xử lý bởi một trong các phương thức public trong lớp controller. Người ta gọi những phương thức như vậy là **action**. Bạn sẽ học chi tiết về action trong ASP.NET Core MVC ở một bài học khác.

Tự xây dựng lớp controller

Để hiểu rõ hơn về controller trong ASP.NET Core MVC, chúng ta sẽ cùng xây dựng một lớp controller ***không*** kế thừa từ lớp Controller sẵn có của ASP.NET Core MVC.

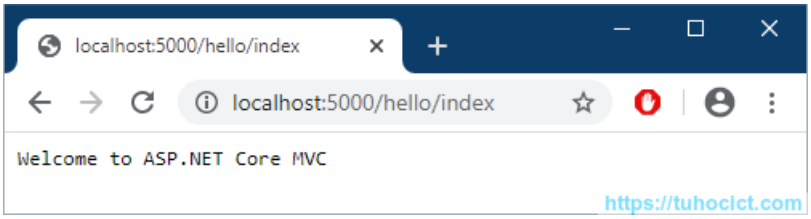
Bước 1. Cấu hình một dự án MVC mới từ một dự án rỗng sử dụng Startup class sau:

```
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. Xây dựng class HelloController trong file cùng tên trực thuộc dự án:

```
namespace WebApplication1 {
    public class HelloController {
        public string Index() {
            return "Welcome to ASP.NET Core MVC";
        }
    }
}
```

Bước 3. Chạy ứng dụng và nhập Url `/hello/index`

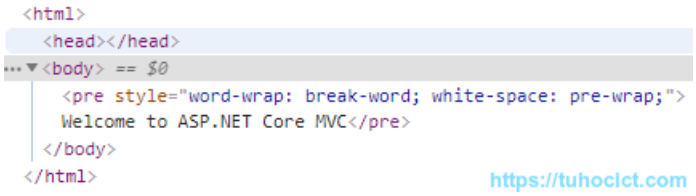


> Do chúng ta sử dụng mẫu routing {controller}/{action}, bạn gọi tới Index của HelloController thông qua Url `/hello/index` .

Trong ví dụ trên chúng ta đã xây dựng một class controller đơn giản nhất HelloController. Class này có thể được sử dụng như một controller trong ASP.NET Core MVC vì tên gọi của nó chứa hậu tố Controller. Trong HelloController chỉ có một action Index trả về một chuỗi ký tự.

Sử dụng Razor view với controller trong ASP.NET Core MVC

Mặc dù controller HelloController trên hoàn toàn hợp lệ, nó không làm được gì nhiều. Hạn chế lớn nhất của nó là chỉ có thể trả về một chuỗi ký tự, vốn sẽ được Mvc Middleware đóng gói vào một chuỗi HTML đơn giản:



Nếu muốn hiển thị kết quả phức tạp hơn, bạn cần sử dụng đến Razor view.

Tiếp tục hoàn thiện HelloController như sau.

Bước 1. Bổ sung phương thức sau vào HelloController:

```
public ViewResult WithView() {
        var view = new ViewResult() { ViewName = "/HelloWithView.cshtml" };
        return view;
 }
```

Bước 2. Tạo file Razor HelloWithView.cshtml trực thuộc dự án và viết code như sau:

```
<!doctype html>

<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Tự học ICT</title>
</head>

<body>
    <h1>Hello world from Razor view!</h1>
    <p>It's @DateTime.Now.ToLongTimeString() now.</p>
</body>
</html>
```

Bước 3. Chạy thử với Url /hello/withview

Như bạn có thể thấy, để sử dụng Razor view cùng với controller, phương thức phải trả về kết quả có kiểu là ViewResult. Object của ViewResult chứa các thông tin cần thiết để Mvc Middleware tìm được file Razor. Khi tìm được file, Mvc Middleware sẽ gọi đến cơ chế Razor để chuyển view thành HTML.

> ViewResult là một trong các class thực thi giao diện IActionResult – giao diện chung mà mọi kết quả trả về của ASP.NET Core MVC đều thực thi. Bạn sẽ học kỹ hơn về action và ActionResult trong một bài học khác.

Truyền view model từ controller sang view

Ở trên chúng ta đã sử dụng được Razor view cùng với controller. Tuy nhiên còn một vấn đề nữa cần giải quyết: làm thế nào để controller truyền dữ liệu sang view. Dữ liệu truyền từ controller sang view để hiển thị được gọi là **view model**.

Tiếp tục hoàn thiện lớp HelloController ở trên.

Bước 1. Bổ sung các lệnh sau vào HelloController:

```
1.  private IModelMetadataProvider _provider;
2.  public ViewDataDictionary ViewData { get; set; }
3.  public ModelStateDictionary ModelState = new ModelStateDictionary();
4.  public HelloController(IModelMetadataProvider provider) {
5.      _provider = provider;
6.      ViewData = new ViewDataDictionary(_provider, ModelState);
7.  }
```

Đoạn code này khai báo thêm hai property cho HelloController: ViewData và ModelState. Để khởi tạo ViewData, bạn cần đến một object của IModelMetadataProvider. Object này được ASP.NET Core MVC tự sinh ra và truyền vào hàm tạo của controller thông qua cơ chế Dependency Injection.

Bạn đã từng làm quen với ViewData trong Razor Pages. Object ViewData chúng ta tạo ra ở đây có vai trò tương tự. Nhiệm vụ của nó là chứa dữ liệu có thể truy xuất được từ view cũng như controller.

Bước 2. Xây dựng phương thức WithViewData trong HelloController:

```
1.  public ViewResult WithViewData() {
```

```
    var model = new List<(string, string)> {
        ("Donald", "Trump"),
        ("Barack", "Obama"),
        ("George W.", "Bush")
    };

    var view = new ViewResult() { ViewName = "/HelloWithViewData.cshtml" };
    view.ViewData = ViewData;
    view.ViewData.Model = model;

    return view;
}
```

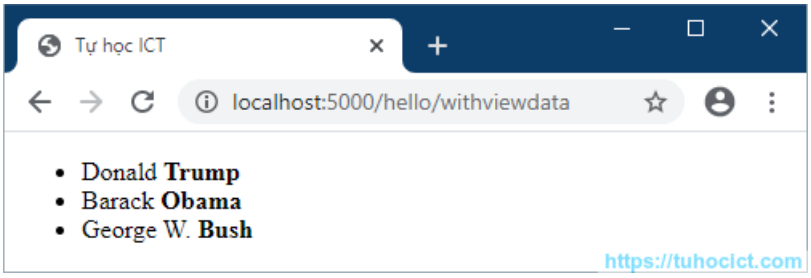
Bạn có thể để ý thấy rằng, trong object của ViewResult đã có sẵn khai báo property ViewData. ViewData của view được gán giá trị ViewData của HelleController. Do đó ở trong view bạn có thể dễ dàng truy xuất các giá trị gán từ controller.

Property Model của ViewData có vai trò đặc biệt. Hãy nhớ lại khi học Razor Pages, bạn thường xuyên gọi @Model trong Razor view. @Model trong view chính là property Model của ViewData.

Bước 3. Xây dựng Razor view HelloWithViewData.cshtml như sau:

```
@model List<(string firstName, string lastName)>

<!doctype html>

<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Tự học ICT</title>
</head>

<body>
    <div>
        <ul>
            @foreach (var i in Model) {
                <li>@i.firstName <b>@i.lastName</b></li>
            }
        </ul>
    </div>
</body>
</html>
```

Chạy ứng dụng với url /hello/withviewdata bạn thu được kết quả như sau:



Lớp `Controller` trong ASP.NET Core MVC

Trong ví dụ trên bạn đã thấy việc truyền dữ liệu từ controller sang view trong ASP.NET Core MVC mặc dù không phức tạp lắm nhưng cũng khá rắc rối. Mỗi lần xây dựng controller mới bạn sẽ phải lặp lại các thao tác tương tự để tạo ra ViewData, ModelState, provider. Khi cần chọn view và truyền model, bạn sẽ phải lặp lại các thao tác khởi tạo và gán giá trị.

ASP.NET Core MVC tạo sẵn class Controller giúp bạn giải quyết hết các vấn đề trên.

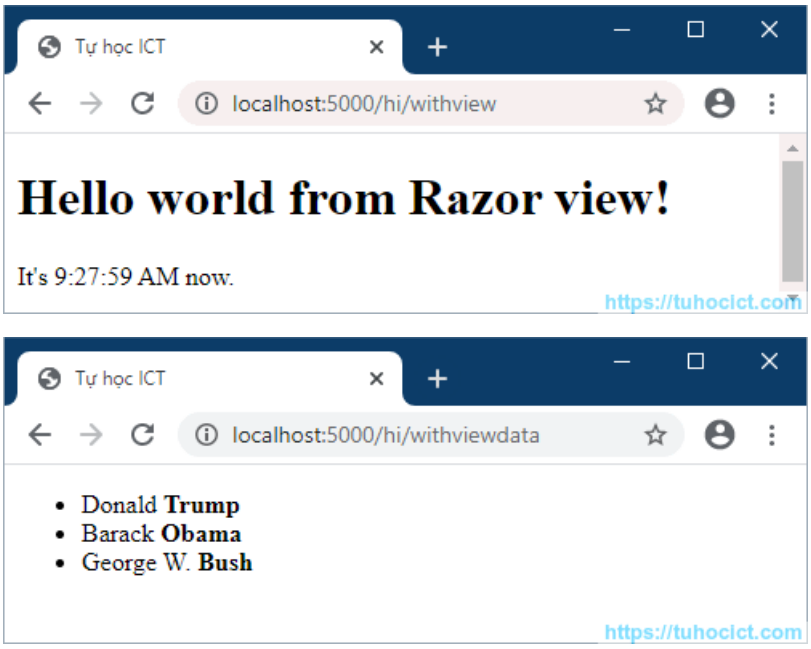
Xây dựng controller mới HiController trong file tương ứng và viết code như sau:

```
using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc;

namespace WebApplication1 {
    public class HiController : Controller {
        public ViewResult WithView() {
            return View("/HelloWithView.cshtml");
        }

        public ViewResult WithViewData() {
            var model = new List<(string, string)> {
                ("Donald", "Trump"),
                ("Barack", "Obama"),
                ("George W.", "Bush")
            };

            return View("/HelloWithViewData.cshtml", model);
        }
    }
}
```

Hãy để ý rằng, HiController giờ đây kế thừa từ Controller.

Trong HiController bạn xây dựng lại hai action quen thuộc WithView và WithViewData. Bạn đồng thời tái sử dụng hai view tương ứng sẵn có.

Chạy ứng dụng với các Url tương ứng /hi/withview và /hi/withviewdata:



Qua ví dụ trên bạn có thể thấy:

(1) Chúng ta không cần tự mình xây dựng các property ViewData và ModelState. Hai property này đã được khai báo và khởi tạo sẵn trong lớp cha Controller.

(2) Bạn không cần tự mình khởi tạo object của ViewResult. Lớp cha Controller đã cung cấp sẵn phương thức trở giúp View để đơn giản hóa các công việc liên quan đến tạo ViewResult,

chọn view và truyền model.

Kết luận

Trong bài học này bạn đã tìm hiểu chi tiết về controller trong ASP.NET Core MVC.

ASP.NET Core MVC cho phép bạn tương đối tự do khi xây dựng lớp controller. Bạn có thể tùy ý xây dựng class riêng, hoặc kế thừa từ lớp Controller sẵn có.

Trong đa số các trường hợp bạn nên xây dựng controller kế thừa từ Controller thay vì xây dựng mọi thứ từ đầu.