Handler và Action Results trong Razor Pages

Khi mới bắt đầu tiếp xúc với web nhiều bạn gặp khó khăn khi thực hiện những việc tưởng chừng như đơn giản: viết code để phản ứng phù hợp với (nhiều) yêu cầu của người dùng trên cùng một page.

Đây là điều rất băn khoăn của những bạn chuyển sang từ những nền tảng lập trình desktop UI như windows forms, vốn rất quen thuộc với mô hình sự kiện: viết code để xử lý riêng rẽ từng loại yêu cầu của người dùng qua các điều khiển trên giao diện.

Handler – phương thức xử lý truy vấn – trong Razor Pages là những phương thức được tự động chạy khi có truy vấn (tới từ trình duyệt hoặc chương trình client khác). Handler là thành phần không thể thiếu nếu bạn cần xử lý các yêu cầu từ người dùng. Đây cũng là cách Razor Pages tạo ra cái tương tự như mô hình sự kiện giúp đơn giản hóa xử lý tương tác với người dùng.

Bài học này sẽ bước đầu giới thiệu những vấn đề cơ bản về handler trong Razor Pages. Một số vấn đề nâng cao hơn sẽ lần lượt được xem xét trong những bài học riêng.

NỘI DUNG CỦA BÀI [Ẩn]

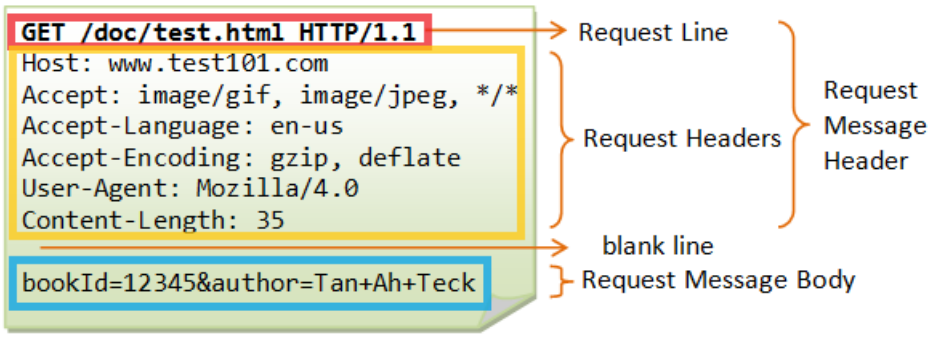
Truy vấn HTTP

Để hiểu handler là gì và tại sao lại cần handler, chúng ta phải hiểu sơ qua về giao thức HTTP – giao thức dùng để trao đổi dữ liệu giữa trình duyệt và chương trình máy chủ web.

HTTP là một giao thức dạng văn bản, nghĩa là dữ liệu của nó thuần túy là chuỗi văn bản (rất dài) được định dạng theo quy định thống nhất. Chuỗi trình duyệt gửi server gọi là HTTP *Request* (truy vấn); Chuỗi server gửi lại trình duyệt gọi là HTTP *Response* (phản hồi).

Chuỗi truy vấn được trình duyệt tạo ra mỗi khi bạn nhập URL vào thanh địa chỉ của trình duyệt, hoặc khi bạn click vào nút *Submit* trên form. Ngoài ra chuỗi truy vấn cũng có thể được tạo ra tự động bởi chương trình.

Dưới đây là minh hoạ một truy vấn GET từ trình duyệt FireFox.



Dòng trên cùng (đánh dấu đỏ) gọi là *dòng truy vấn* (request line). Trong dòng này có hai thông tin quan trọng là *phương thức* (verb) và *URL* (Uniform Resource Locator).

Phương thức thường gặp bao gồm **Get**, **Post**, **Put**, **Delete**, **Patch** (và một số khác nữa) chỉ định loại hành động. Url chỉ định đối tượng chịu tác động của phương thức. Verb và Url là hai thông tin giúp server hiểu client muốn cái gì.

Trong các truy vấn, Get và Post là hai loại verb được sử dụng rộng rãi nhất. Put, Delete và Patch thường gặp trong các Restful API. Truy vấn GET tạo ra khi bạn nhập URL vào thanh địa chỉ trình duyệt. Truy vấn POST tạo ra khi bạn submit form. Các truy vấn Put, Delete, Patch thường tạo ra bằng chương trình.

Mỗi loại truy vấn sẽ đi kèm với dữ liệu và yêu cầu xử lý riêng.

Ví dụ, truy vấn Post thông thường luôn gửi nhiều dữ liệu, thậm chí là những dữ liệu cỡ lớn như file. Khi này ứng dụng trên server phải thực hiện các thao tác lưu dữ liệu vào cơ sở dữ liệu.

Truy vấn Get thường liên quan tới lấy thông tin. Để lấy thông tin phù hợp, Get thường gửi kèm một vài tham số qua Url (dưới dạng chuỗi truy vấn hoặc route data).

> Bạn sẽ học chi tiết về cách xử lý dữ liệu từ form, chuỗi truy vấn (query string) và route data trong một bài học riêng.

Do đó, bạn cần xử lý riêng biệt cho từng loại truy vấn. Razor page hỗ trợ bạn xử lý riêng rẽ từng loại truy vấn thông qua **handler** – phương thức xử lý truy vấn.

Handler mặc định trong Razor Pages

Khái niệm Handler

Trong Razor Pages, mặc định mỗi khi truy vấn tới, Razor Pages căn cứ vào loại truy vấn (verb) để tìm kiếm phương thức có tên tương ứng và tự thực thi:

- `OnGet` () – đối với truy vấn GET;
- `OnPost` () – đối với truy vấn POST;
- `OnPut` () – đối với truy vấn PUT;
- v.v.

Bạn đã hình dung ra mô hình đặt tên này: On<verb>(). Các phương thức này được gọi chung là *phương thức xử lý truy vấn*, hoặc *handler* cho ngắn gọn.

Dựa trên cơ chế này bạn có thể viết code để xử lý riêng cho từng trường hợp truy vấn. Dĩ nhiên, nếu bạn không viết handler nào, Razor vẫn xử lý truy vấn theo cách mặc định.

Bạn cũng có thể hình dung handler là công cụ giúp bạn can thiệp vào việc xử lý page của Razor Pages.

> Bạn cũng có thể sử dụng phương thức tương ứng tận cùng bằng Async: `OnGetAsync` (), `OnPostAsync` (), `OnPutAsync` (), v.v.. Trong Razor Pages, ví dụ, OnGet và OnGetAsync thực tế là cùng một handler. Do đó bạn không thể dùng cả hai phương thức này trên cùng một page.

Hãy cùng thực hiện một ví dụ nhỏ cho dễ hiểu.

Thực hành 1

Bước 1. Tạo một project Web Application.

Bước 2. Viết code cho Index.cshtml như sau:

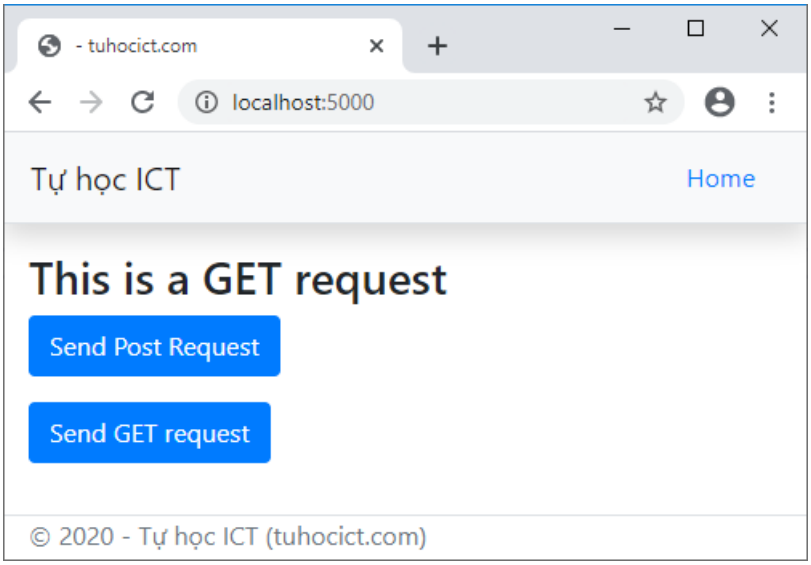
```
@page
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<h3>@Message</h3>
<form method="post">
    <p><button class="btn btn-primary" type="submit">Send Post Request</button></p>
</form>
<p><a href="/" class="btn btn-primary">Send GET request</a></p>

@functions{
    public string Message { get; set; }
    public void OnGet() {
        Message = "This is a GET request";
    }
    public void OnPost() {
        Message = "This is a POST request";
    }
}
```

Trong đoạn mã trên, lần đầu tiên chúng ta tiếp xúc với form – loại thẻ html cho phép gửi dữ liệu về server. Bạn sẽ học chi tiết về form trong một bài học riêng. Form trong ví dụ trên chỉ chứa duy nhất một nút bấm (button) với nhiệm vụ gửi truy vấn post về server.

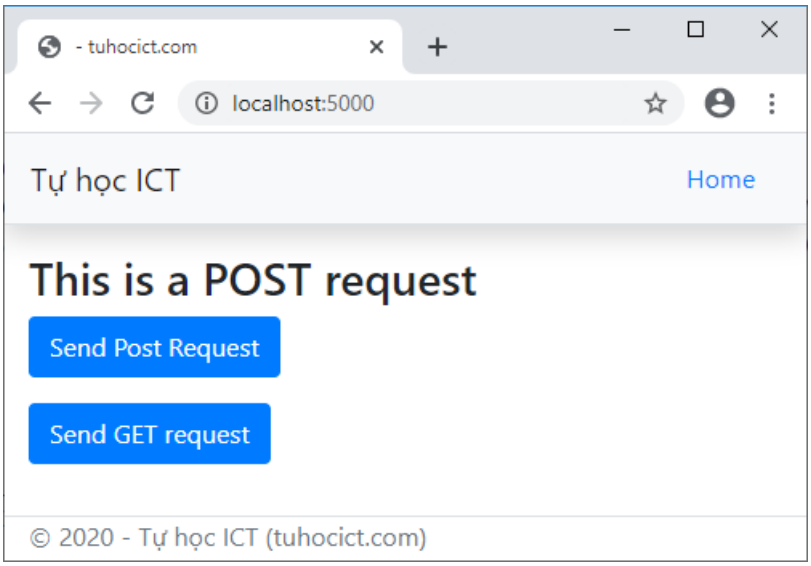
> Lưu ý directive `@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers` . Nếu bạn sử dụng project từ template Web Application thì không cần directive này. Nhưng nếu bạn tạo empty project rồi tự cài đặt Razor Pages thì **phải dùng** directive này. Nếu không truy vấn post sẽ không thực hiện. Vấn đề này liên quan tới cơ chế bảo vệ form Cross-Site Request Forgery.

Khi chạy ứng dụng kết quả thu được như sau:

Khi site được tải lần thứ nhất, phương thức `OnGet()` được thực thi nên dòng thông báo (từ property Message) có giá trị "This is a GET request":



Nếu bạn bấm nút "Send Post Request", phương thức `OnPost` () được thực thi và gán giá trị mới cho property Message:



Bấm nút Send GET request sẽ lại gửi truy vấn GET.

Thực hành 2

Chúng ta làm lại ví dụ trên nhưng giờ sử dụng model class:

Index.cshtml.cs | Index.cshtml

```
1.  using Microsoft.AspNetCore.Mvc.RazorPages;
2.
3.  namespace WebApplication1.Pages {
4.      public class IndexModel : PageModel {
5.          public string Message { get; set; }
```

```
        public void OnGet() {
            Message = "This is a GET request";
        }
        public void OnPost() {
            Message = "This is a POST request";
        }
    }
}
```

Bạn thu được cùng kết quả như khi sử dụng functions block.

Một số lưu ý

Khi sử dụng handler mặc định bận cần để ý một số lưu ý quan trọng sau:

(1) Phương thức handler có thể viết trong functions block hoặc model class. Nhìn chung khi dùng handler bạn nên viết trong **model class**. Đây là mô hình được khuyến nghị trong Razor Pages.

(2) Phương thức handler phải có mức truy cập là `public`. Bạn nên đặc biệt lưu ý điều này nếu gặp tình huống phương thức không được kích hoạt như dự định.

(3) Phương thức handler có thể trả về kết quả thuộc bất kỳ kiểu dữ liệu nào. Thông thường kiểu trả về là `void` (hoặc `Task` nếu bạn sử dụng bản Async). `ActionResult` và `IActionResult` cũng là kiểu trả về thường gặp.

(4) Razor Pages phân biệt phương thức handler căn cứ vào **tên** phương thức chứ không quan tâm đến danh sách tham số.

Ví dụ, khi bạn viết hai phương thức `OnGet()` và `OnGet(string str)` trong cùng model class, compiler sẽ đồng ý (vẫn dịch thành công) nhưng khi chạy Razor sẽ báo lỗi.

Cơ chế handler mặc định cho phép bạn phản ứng với **từng loại truy vấn**. Nếu cần phản ứng với **từng truy vấn** cụ thể, bạn cần đến một cơ chế khác: **named handler**.

Named handler – định danh phương thức xử lý truy vấn

Nếu sử dụng handler mặc định, trên mỗi page bạn chỉ có thể viết một handler cho mỗi **loại truy vấn**. Nó có nghĩa là Razor Pages đối xử như nhau với tất cả truy vấn Get, không quan tâm đến các thông tin khác trong truy vấn. Tình hình xảy ra tương tự với các truy vấn Post.

Thông thường yêu cầu này đáp ứng tốt cho đa số site.

Tuy nhiên, trong một số trường hợp bạn muốn có ***nhiều handler cho cùng một verb***.

Ví dụ, mặc dù cùng là truy vấn GET, nhưng nếu GET đến từ nút bấm A sẽ được đối xử khác với GET đến từ nút bấm B.

Hoặc lấy ví dụ khác, nếu trên page bạn có 3 form cho 3 mục đích khác nhau, bạn sẽ cần đến 3 handler để tránh trộn lẫn chức năng. Khi đó bạn sẽ cần đến nhiều handler cho cùng một verb trên cùng một model class.

Razor Pages hỗ trợ khả năng này thông qua một tính năng có tên gọi là **named handler** – handler có định danh.

Thực hành 3

Hãy cùng thực hiện ví dụ sau (cùng trên project bạn đã dùng ở thực hành 1-2):

Bước 1. Tạo thêm page Multiform.cshtml với model class.

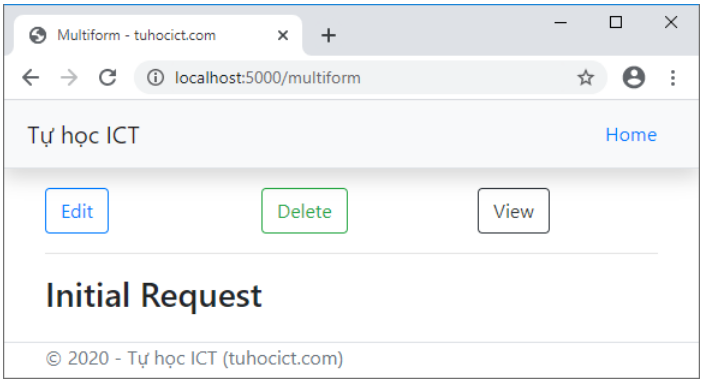
Bước 2. Viết code cho Multiform như sau:

Multiform.cshtml.cs | Multiform.cshtml

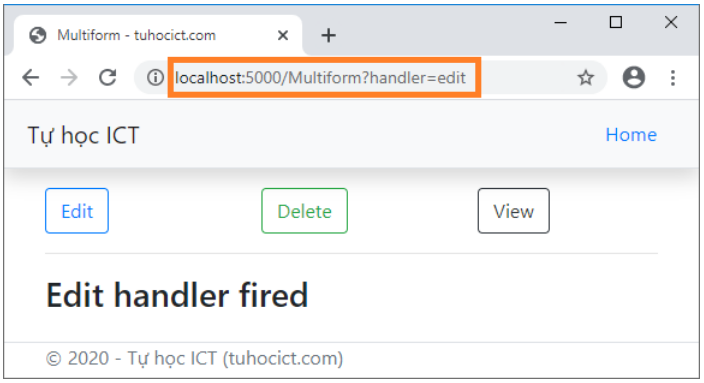
```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebApplication1.Pages {
    public class MultiformModel : PageModel {
        public string Message { get; set; } = "Initial Request";
        public void OnGet() {
        }
        public void OnPost() {
            Message = "Form Posted";
        }
        public void OnPostDelete() {
            Message = "Delete handler fired";
        }
        public void OnPostEdit(int id) {
            Message = "Edit handler fired";
        }
        public void OnPostView(int id) {
            Message = "View handler fired";
        }
    }
}
```

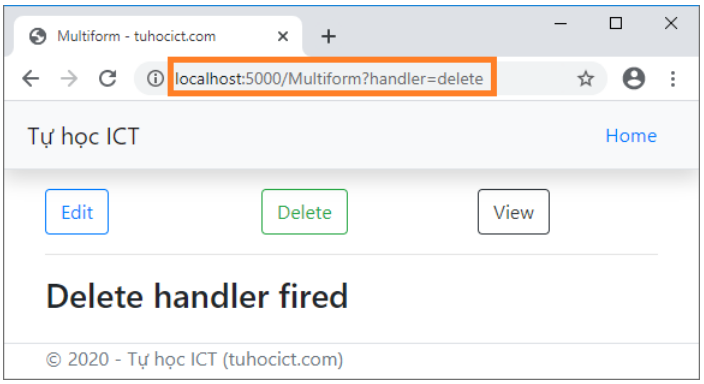
> Bạn sẽ học về **tag helper** trong một bài riêng.

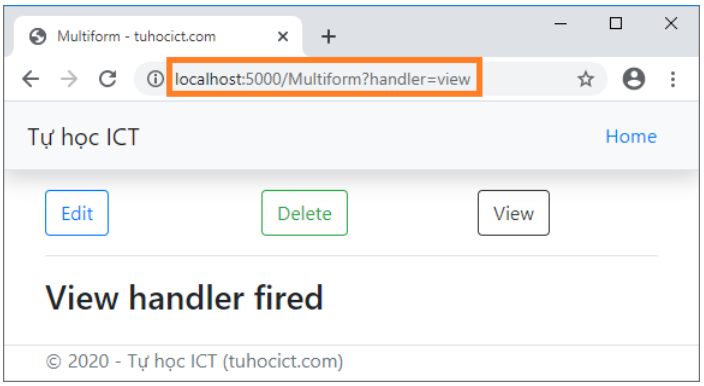
Bạn thu được kết quả từ lần chạy đầu tiên như sau:



Click vào các nút, từng form sẽ lần lượt gửi truy vấn post riêng về server. Đồng thời phương thức handler tương ứng được kích hoạt.







Cơ chế named handler

Trong file Multiform.cshtml để ý một attribute lạ trong thẻ form: `asp-page-handler`.

Đây không phải là attribute tiêu chuẩn của html form mà là một ***tag helper*** của Razor Pages. `asp-page-handler` giúp chỉ định phần tên của handler sẽ được kích hoạt để xử lý truy vấn post tương ứng:

- OnPostDelete xử lý truy vấn post đến từ form có `asp-page-handler` =”delete”;
- OnPostEdit xử lý truy vấn post đến từ form có `asp-page-handler` =”edit”;
- OnPostView xử lý truy vấn post đến từ form có `asp-page-handler` =”view”.

Nếu bạn mở cửa sổ developer của trình duyệt (F12) thì sẽ thấy kết quả như sau:

```
<div class="row">
  <div class="col-sm">
    <form method="post" action="/Multiform?handler=edit">…</form>
  </div>
  <div class="col-sm">
    <form method="post" action="/Multiform?handler=delete">…</form>
  </div>
  <div class="col-sm">
    <form method="post" action="/Multiform?handler=view">…</form>
  </div>
</div>
```

Đây là cách `asp-page-handler` chuyển thành HTML.

Hãy để ý đến chuỗi truy vấn trong Url (`?handler=...`). Đây là cách Razor Pages phân biệt giữa nhiều handler có tên khác nhau.

Nếu không muốn sử dụng chuỗi truy vấn trong Url, bạn có thể sử dụng directive `@page "{handler?}"` trong file Multiform.cshtml. Khi này Url sẽ chuyển thành:

> Đây là một cách ghi đè route template chúng ta sẽ học trong một bài riêng.

Dĩ nhiên, khi hiểu cơ chế của named handler bạn cũng có thể tự mình viết html mà không cần dùng tới tag helper.

Tham số của handler

Các truy vấn tới thông thường đều chứa tham số. Các tham số này có thể diễn đạt bằng những cách khác nhau tùy thuộc vào loại truy vấn.

Đối với truy vấn GET thường có hai cách cung cấp tham số: thông qua chuỗi truy vấn (query string) của URL; thông qua route data (là các segment của chính URL).

Đối với truy vấn POST, tham số thường được chứa trong phần thân của truy vấn HTTP.

Khi xử lý truy vấn, handler cũng có thể (và cần phải) truy xuất được các tham số này.

Razor Pages cho phép các handler có tham số tùy ý, cả kiểu phức tạp (class) lẫn các kiểu cơ sở (string, int, bool, v.v.).

Khi truy vấn tới, Razor Pages sẽ tự động truyền tham số của truy vấn sang tham số của handler nếu chúng trùng tên nhau.

Đối với tham số handler có kiểu phức tạp, Razor Pages cũng hỗ trợ tự động hóa việc chuyển đổi này, giúp người lập trình tiện lợi hơn khi đọc dữ liệu người dùng.

Cơ chế tự động chuyển đổi tham số của truy vấn sang tham số của handler được gọi là model binding hoặc parameter binding. Bạn sẽ học chi tiết về model binding trong một bài riêng.

Trong các bài học tiếp theo bạn sẽ lần lượt học cách truy xuất tham số từ URL hoặc qua body của truy vấn HTTP.

Một điều cần lưu ý rằng danh sách tham số không được dùng để phân biệt các handler với nhau. Các handler chỉ được phân biệt với nhau qua tên gọi. Mặc dù C# compiler cho phép có các handler trùng tên khác tham số, cơ chế Razor Pages lại không chấp nhận tình trạng này và sẽ báo lỗi ở giai đoạn runtime.

Action Result và handler

Mặc dù Razor Pages không giới hạn kiểu trả về của các phương thức handler, đôi khi bạn cần một số loại kết quả trả về đặc biệt.

Ví dụ, bình thường các handler có kiểu trả về là `void`. Khi đó trang nội dung tương ứng vẫn tiếp tục được xử lý khi kết thúc thực hiện handler.

Tuy vậy, đôi khi bạn muốn sau khi thực hiện xong handler thì sẽ chuyển hướng sang một trang khác chứ không tiếp tục tải trang hiện tại nữa. Bạn cũng có thể muốn trả lại mã lỗi cho trình duyệt, giả sử, khi người dùng không có quyền truy xuất tài nguyên. Bạn cũng có thể cần phải trả lại kết quả là một chuỗi html sinh ra từ partial page cho truy vấn AJAX.

Những tình huống tương tự dẫn tới nhu cầu tạo ra các kiểu trả về đặc biệt cho handler. Trong Asp.net Core, các kiểu trả về đặc biệt như thế được gọi chung là **action result**.

Mọi action result trong Asp.net Core đều được xây dựng từ lớp abstract **`ActionResult`** hoặc thực thi giao diện **`IActionResult`**.

Ví dụ, nếu muốn điều hướng sang một trang khác khi kết thúc thực thi handler, bạn thực hiện như sau:

```
public IActionResult OnGet() {
    // code xử lý khác

    return new RedirectToPageResult("Index");
}
```

Bạn thấy: (1) kiểu trả về của handler OnGet giờ là `IActionResult`; (2) `OnGet` trả về một object của `RedirectToPageResult`. `RedirectToPageResult` là một class thực thi interface `IActionResult`. Nó tạo ra mã phản hồi có tác dụng điều hướng sang một trang khác.

Để đơn giản hóa sử dụng action result, lớp `PageModel` – lớp cha của tất cả model class – xây dựng sẵn một số phương thức hỗ trợ.

Ví dụ, thay vì gọi

```
return new RedirectToPageResult("Index");
```

Bạn có thể gọi

```
return RedirectToPage("Index");
```

`RedirectToPage` là phương thức hỗ trợ cho việc sử dụng class `RedirectToPageResult`.

Trong quá trình học bạn sẽ gặp các class khác thực thi `IActionResult` hoặc `ActionResult`.

Mô hình sự kiện trên Razor Pages

Từ cơ chế named handler bạn có thể rút ra một ứng dụng rất mạnh của nó: tạo nhiều nút bấm/link, mỗi nút bấm/link sẽ được xử lý bằng một phương thức riêng. Điều này giúp viết ứng dụng Razor Pages đơn giản hơn và gần gũi với mô hình xử lý sự kiện trong Winforms hay WPF.

Để hiểu rõ hơn, hãy cùng thực hiện một ví dụ (sử dụng tiếp project từ phần thực hành 3):

Bước 1. Tạo thêm page Multiaction.

Bước 2. Viết code cho model class trong file Multiaction.cshtml.cs:

```
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebApplication1.Pages {
    public class MultiactionModel : PageModel {
        public string Message { get; set; } = "Default GET method";
        public void OnPostRegister() => Message = "Post Register fired!";
        public void OnPostRequestinfo() => Message = "Post Request Info fired!";
        public void OnGetClear() => Message = "Get Clear fired!";
        public void OnGetReset() => Message = "Get Reset fired!";
    }
}
```

Bước 3. Viết code cho trang Multiaction.cshtml như sau:

```
@page "{handler?}"
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@model WebApplication1.Pages.MultiactionModel
@{
    ViewData["Title"] = "Multiaction";
}

<form method="post">
    <button asp-page-handler="Register">Register Now</button>
    <button asp-page-handler="Requestinfo">Request Info</button>
</form>
<a class="btn btn-danger" asp-page-handler="clear">Clear All</a>
<a class="btn btn-info" asp-page-handler="reset">Reset All</a>
<h3>@Model.Message</h3>
```

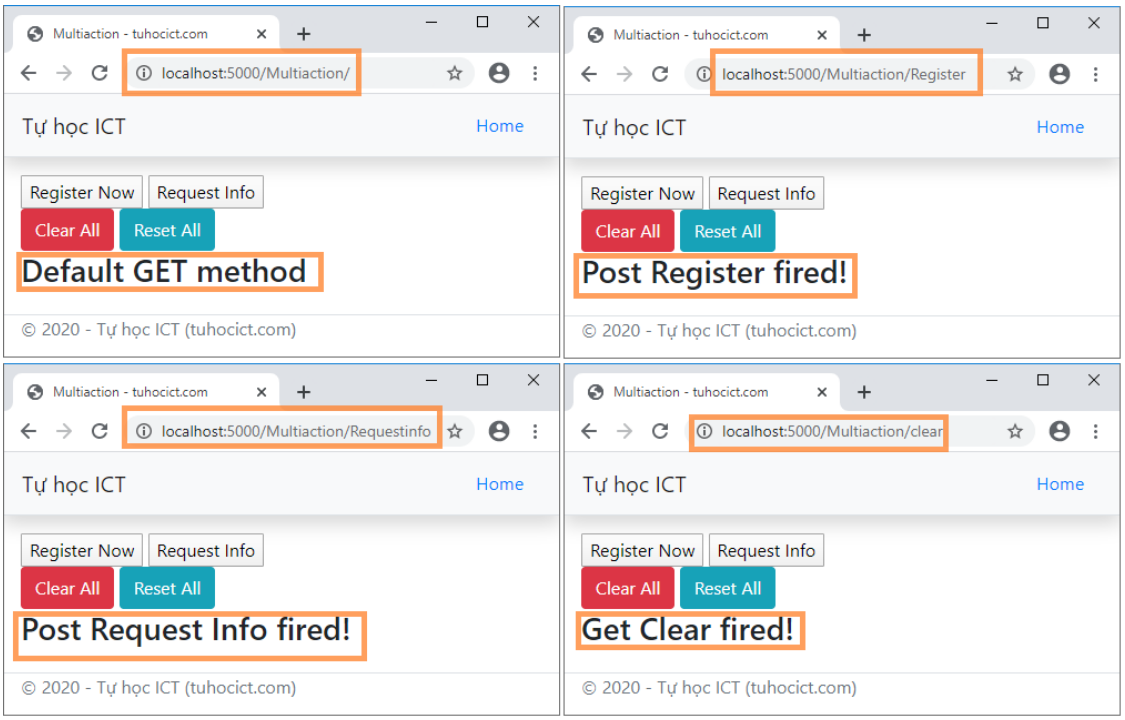
Hãy để ý đến cách chúng ta sử dụng tag helper **`asp-page-handler`** trong thẻ **`button`** hoặc **`a`**, thay vì dùng trong thẻ **`form`** như ở bài thực hành 3. Tuy vậy nó cũng có cùng một tác dụng chỉ định:

- `OnPostRegister` () xử lý phương thức *Post* khi click nút có `asp-page-handler = "Register"`
- `OnPostRequestinfo` () xử lý phương thức *Post* khi click nút có `asp-page-handler = "Requestinfo"`
- `OnGetClear` () xử lý phương thức *Get* khi click link có `asp-page-handler = "Clear"`
- `OnGetReset` () xử lý phương thức *Get* khi click link có `asp-page-handler = "Reset"`

Giá trị `asp-page-handler` không phân biệt hoa/thường.

Directive `@page "{handler?}"` cũng giúp chúng ta có url "đẹp" (không có chuỗi truy vấn).

Khi chạy chương trình bạn sẽ thu được kết quả như sau (chú ý đến url) khi lần lượt click vào các nút:



Qua ví dụ này bạn thấy rằng mỗi button hoặc link giờ đây được xử lý bằng một phương thức riêng biệt, rất giống mô hình xử lý sự kiện trong các UI framework như windows forms. Điều này khiến việc xử lý tương tác với người dùng trong Razor Pages dễ dàng hơn rất nhiều.

Kết luận

Trong bài học này chúng ta đã làm quen bước đầu với handler – phương thức xử lý truy vấn trong Razor Pages, bao gồm handler mặc định và handler định danh.

Bạn có thể thấy hai cơ chế handler này cho phép thực hiện mô hình sự kiện trên Razor Pages một cách dễ dàng.

Tải mã nguồn solution Handlers

1 file(s) 0.00 KB

TẢI MÃ NGUỒN SOLUTION

+ 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!