Query string, route data, route template – xử lý truy vấn GET

Đọc dữ liệu người dùng là một yêu cầu bắt buộc của mọi ứng dụng web. Người dùng có hai cơ chế cơ bản để gửi dữ liệu cho server: qua Url hoặc qua thân truy vấn.

Dữ liệu gửi theo Url có thể ở hai dạng: chuỗi truy vấn hoặc dữ liệu route. Do đó phương pháp đọc dữ liệu của chúng có chút khác nhau.

Bài học này sẽ hướng dẫn bạn cách truy xuất dữ liệu người dùng gửi qua Url. Cách truy xuất dữ liệu qua thân truy vấn (từ form) sẽ học trong một bài riêng.

NỘI DUNG CỦA BÀI [Ẩn]

Truyền dữ liệu về server qua HTTP

Một yêu cầu phổ biến là gửi dữ liệu từ trình duyệt cho chương trên máy chủ qua HTTP. Lấy ví dụ, khi bạn nhập dữ liệu vào một form và ấn nút Submit. Dữ liệu này sẽ được đóng gói vào truy vấn HTTP và gửi về cho server.

Trình duyệt có ba cách để gửi dữ liệu về server qua truy vấn HTTP:

(1) Tạo **chuỗi truy vấn** (query string) và ghép vào Url.

Ví dụ: https://tuhocict.com?s=razor+pages là một Url chứa chuỗi truy vấn `s=razor+pages`. Chuỗi truy vấn và Url phân tách bởi ký tự `?` (dấu chấm hỏi). Chuỗi truy vấn tạo ra từ các cặp `<tham số>=<giá trị>`. Các cặp này phân tách bởi ký tự `&`.

Phương pháp này thường dùng với phương thức GET.

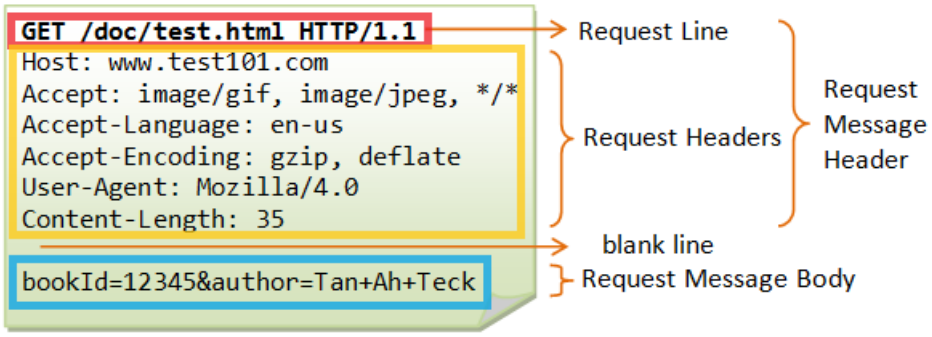
(2) Sử dụng **route data**: ghép trực tiếp tham số vào Url để trở thành một segment của Url.

Ví dụ, trong url https://tuhocict.com/topic/razor/, segment `/razor/` thực tế là một tham số cung cấp cho page chuyên hiển thị danh sách bài viết theo chủ đề.

Phương pháp này được sử dụng nếu không có nhiều tham số phức tạp. Ngoài ra, phương pháp này cũng tạo ra các Url "thân thiện" với máy tìm kiếm (như Google, Bing, Yandex).

(3) Gửi dữ liệu qua thân truy vấn HTTP.

Dữ liệu được tạo ra giống hệt như chuỗi truy vấn trong phương pháp 1) nhưng được ghép vào phần thân (body) của truy vấn HTTP (phần đánh dấu màu xanh).



Phương pháp này thường dùng với phương thức POST để truyền dữ liệu lớn (như file).

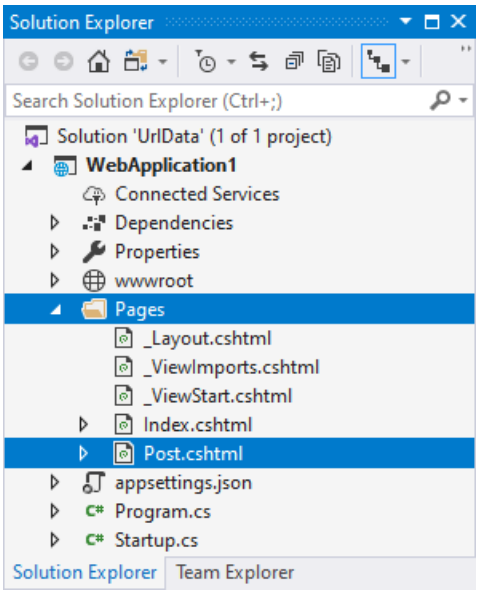
Khi dữ liệu tới chương trình, bạn cần lấy dữ liệu ra để xử lý và phản ứng lại cho phù hợp. Thao tác này gọi chung là xử lý truy vấn.

Mô hình xử lý truy vấn bạn đã học trong bài trước. Bài học này tập trung vào cách lấy dữ liệu từ truy vấn.

Đọc dữ liệu từ query string

Sử dụng tham số của handler

Trước hết hãy chuẩn bị một dự án mới theo mẫu Web Application project và tạo thêm trang Post trong thư mục Pages:



Viết code cho Post.cshtml.cs như sau:

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

namespace WebApplication1.Pages {
    public class PostModel : PageModel {
        [ViewData]
        public string Title { get; set; }

        public string Text { get; set; }

        public void OnGet(string title) {
            //var title = RouteData.Values["title"].ToString();
            Title = title
                .Replace('-', ' ')
                .ToUpper();
            Text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eius
        }
    }
}
```

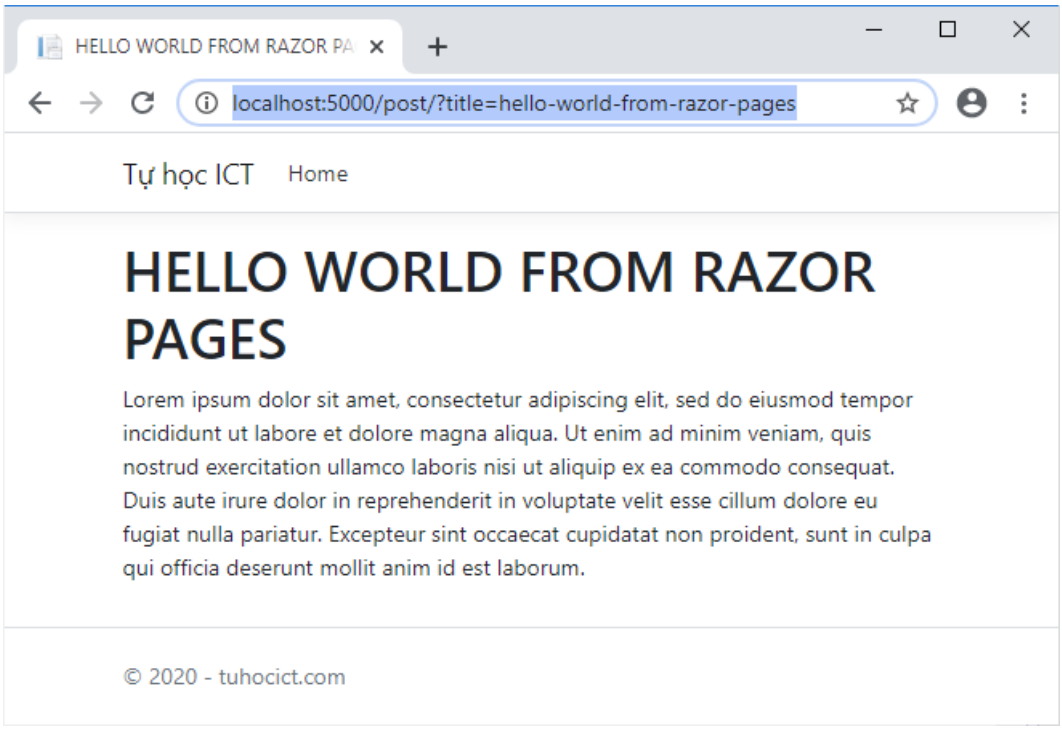
Viết code cho Post.cshtml như sau:

```
@page

@model WebApplication1.Pages.PostModel

<h1>@Model.Title</h1>
<p>@Model.Text</p>
```

Bạn sẽ thu được kết quả như sau:



Qua ví dụ trên bạn có thể thấy, cặp khóa/giá trị trong chuỗi truy vấn (trong trường hợp này là title=hello-world-from-razor-pages) sẽ được truyền vào làm tham số tương ứng trong phương thức handler (tham số title của OnGet).

Như vậy, nếu cần tiếp nhận giá trị từ chuỗi truy vấn, bạn chỉ cần tạo tham số có tên tương ứng trong phương thức handler.

Sử dụng object Request

Phương pháp thứ hai giúp truy xuất tham số từ chuỗi truy vấn GET là sử dụng object Request như sau:

```
var title = Request.Query["title"].ToString();
```

Request là một property của lớp PageModel, do đó bất kỳ model class nào (do kế thừa từ PageModel) đều có thể truy xuất. Object này chứa các thông tin về truy vấn. Query là một property của object này chứa các cặp khóa/giá trị tương ứng tách ra từ chuỗi truy vấn.

Khi sử dụng object Request bạn có thể viết lại phương thức OnGet như sau:

```
public void OnGet() {
    var title = Request.Query["title"].ToString();
    Title = title
        .Replace('-', ' ')
        .ToUpper();
    Text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod temp
}
```

Route template

Trước khi học cách đọc dữ liệu từ route data, chúng ta cần hiểu khái niệm **route template**.

> Route template là khuôn mẫu để định nghĩa và kiểm tra các Url hợp lệ của page.

Điều này có nghĩa là, khi bạn đưa ra một route template, bạn đang định nghĩa một tập Url hợp lệ cho page. Khi một Url tới từ trình duyệt, route template lại được sử dụng để kiểm tra xem Url này có hợp lệ hay không.

Trước đây khi học về cơ chế routing trong Razor Pages bạn đã biết rằng cơ chế này sử dụng route template mặc định là đường dẫn tới file cshtml. Nghĩa là mặc định Url hợp lệ tương tự như đường dẫn tương đối tới file cshtml tính từ thư mục Pages.

Bạn cũng đã biết rằng Razor Pages cho phép ghi đè khuôn mẫu mặc định. Một trong những phương pháp ghi đè là sử dụng directive `@page`.

Hãy cùng thực hiện một số ví dụ cho dễ hiểu. Vẫn tiếp tục với trang Post.cshtml ở trên.

Lần lượt thay đổi `@page` directive và thử dùng Url tương ứng:

STT	Directive	Url
1	@page "/article"	http://localhost:5000/article
2	@page "article"	http://localhost:5000/post/article
3	@page "{title}"	http://localhost:5000/post/anything-can-be-passed-here

STT	Directive	Url
4	@page "{title?}"	http://localhost:5000/post/anything-can-be-passed-here http://localhost:5000/post/
5	@page "/article/{title}"	http://localhost:5000/article/anything-can-be-passed-here
6	@page "{year}/{month}/{day}/{title}"	http://localhost:5000/post/post/2020/04/01/hello world
7	@page "{title=first post}"	Giống trường hợp 4
8	@page "{id:int}"	http://localhost:5000/post/10
9	@page "{title:minlength(2)}"	http://localhost:5000/post/anything-can-be-passed-here

Trường hợp 1 tạo một url cố định `/article` thay thế cho url mặc định `/post`. Bạn có thể thoải mái tạo ra cấu trúc này với ***nhiều segment*** như `/article/post`, `/post/article/id`. Không có giới hạn gì khi tạo url tĩnh.

Trường hợp 2, `article` trở thành segment chèn vào url mặc định `/post` (để thu được url hợp lệ cố định `/post/article`).

Trường hợp 3 và 4 chấp nhận bất kỳ chuỗi nào để tạo thành *segment cuối của url mặc định* `/post/{title}`. Trong các trường hợp này, route template tạo ra một ***tập hợp url*** hợp lệ (thay cho 1 url như trong trường hợp 1 và 2). Đây là phương pháp cơ bản để tạo ra các URL chứa ***route data*** – loại dữ liệu/tham số nhưng đặt trực tiếp làm thành phần (segment) của url.

Bản thân `{title}` chỉ được xem như ***cụm giữ chỗ*** (placeholder). Bạn có thể dùng bất kỳ cụm ký tự nào làm placeholder, như `{id}`, `{year}`, `{author}`, v.v., miễn nó được đặt trong cặp ngoặc nhọn để báo rằng đây là cụm giữ chỗ.

Ký tự `?` đi sau placeholder báo rằng segment này là không bắt buộc, nghĩa là nó chấp nhận cả url `/post/` (không có gì đằng sau).

> Bạn có thể dễ dàng hình dung {title} như là một biến của ngôn ngữ lập trình. Bạn có thể thay thế các giá trị khác nhau vào vị trí của {title}. Bản thân cơ chế xử lý sự kiện (handler) của Razor Pages cũng coi {title} là một tham số đầu vào cho handler.

Số lượng cụm giữ chỗ không bị giới hạn trong một template. Bạn thậm chí có thể tạo ra template như `@page "{year}/{month}/{day}/{title}"`. Template này tương ứng với bất kỳ url nào có 4 segment đứng sau /post. Ví dụ `/post/2020/04/01/hello-world` (trường hợp 6).

Trường hợp 5 là phương án ghép 1 và 3. Bạn có thể thoải mái ghép nối các phương án cơ bản (từ 1 đến 4)

Trường hợp 7 tương tự như trường hợp 4. Sự khác biệt chỉ xuất hiện khi lập trình (chút sẽ xem xét). Đây là tình huống cung cấp giá trị mặc định cho route data nếu segment cuối cùng bị để trống.

Trường hợp 8 và 9 tương tự như (3) nhưng đặt ra giới hạn: (8) chỉ nhận chữ số, (9) chỉ nhận các chuỗi dài hơn 2 ký tự. Đây là cách thức ***đặt giới hạn*** trong route template.

Có khá nhiều loại giới hạn khác nhau về kiểu và khoảng giá trị. Sau đây là một số giới hạn thường gặp: **alpha** (chữ cái a-z, A-Z), **int** (số nguyên 32bit), **bool** (boolean), **double** (số thực 64bit), **float** (số thực 32bit), **long** (số nguyên 64bit), **datetime** (thời gian), **decimal** (số thực lớn), **min** (giá trị nguyên nhỏ nhất), **max** (giá trị nguyên lớn nhất), **range** (dải giá trị nguyên), **minlength** (độ dài xâu nhỏ nhất), **maxlength** (độ dài xâu lớn nhất), **length** (dải độ dài xâu).

Như vậy bạn có thể để ý rằng, loại template với placeholder cho phép tạo ra một loại url "động" với khả năng mang giá trị tham số. Loại tham số đi với url trong trường hợp này được gọi là **route data**. Chú ý phân biệt route data với **query string**.

> Những quy tắc viết route template trên có thể sử dụng cả với phương thức `AddPageRoute` (). Ví dụ:

```
services
    .AddRazorPages()
    .AddRazorPagesOptions(options =>
    {
        options.Conventions.AddPageRoute("/Archive/Post", "Post/{year}/{month}/{day}/{ti
    });
```

Đọc dữ liệu từ route data

Truy xuất qua tham số của handler

Khi bạn đã hiểu cách thức ghi đè route template, bạn cần biết thêm rằng:

> Trong các template có sử dụng placeholder, tên của placeholder được xem là một tên biến và giá trị của segment tương ứng trong url chính là giá trị của biến. Loại template này cho phép tạo ra các url với khả năng mang tham số.

Cùng thực hiện ví dụ sau cho dễ hiểu.

```
// Post.cshtml.cs
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebApplication1.Pages {
    public class PostModel : PageModel {
        [ViewData]
        public string Title { get; set; }

        public string Text { get; set; }

        public void OnGet(string title) {
```

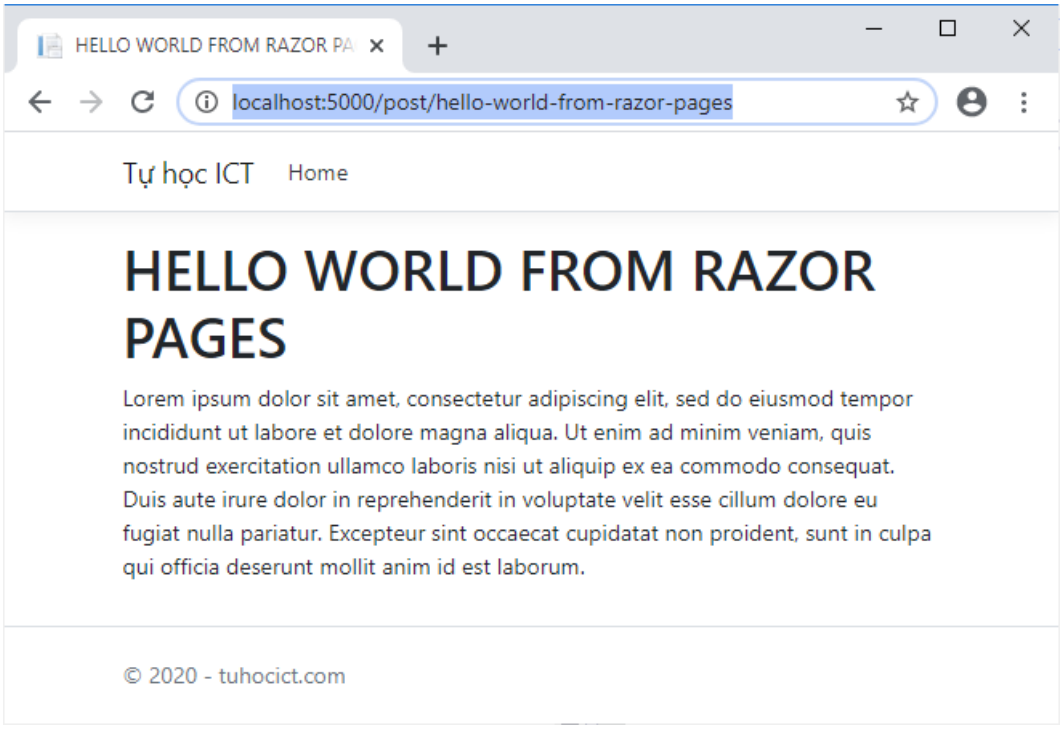
```
            Title = title
                .Replace('-', ' ')
                .ToUpper();
            Text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eius
        }
    }
}
```

```
@page "{title}"

@model WebApplication1.Pages.PostModel

<h1>@Model.Title</h1>
<p>@Model.Text</p>
```

Để ý sự trùng khớp giữa `OnGet(string title)` và `@page "{title}"`.



Trong ví dụ trên chúng ta giả lập một blog đơn giản. Để ý rằng route template "{title}" tương ứng với các Url `/post/{title}` (xem lại trường hợp số 3). Tên của placeholder {title} cũng trùng với tham số của phương thức OnGet – phương thức xử lý truy vấn Get mặc định.

> Nếu trong route template có placeholder, Razor Pages sẽ tự động truyền giá trị của nó cho tham số có tên tương ứng của handler.

Theo đó, tất cả những gì nằm ở vị trí của `{title}` trong `/post/{title}` sẽ được truyền sang biến `title` của phương thức xử lý `OnGet`.

Do Url là chuỗi ký tự, kiểu giá trị mặc định (nếu không chỉ định giới hạn, xem trường hợp 8) của tham số sẽ là string.

Truy xuất qua object RouteData

Một khi bạn thiết lập route template sử dụng placeholder, Razor Pages đều phải phân tích Url tới xem có phù hợp hay không. Nếu một Url hợp lệ, các thành phần của nó được phân tách và lưu trữ trong một object có tên là RouteData.

Bạn có thể truy xuất giá trị tham số sử dụng tên của placeholder như sau:

```
var title = RouteData.Values["title"].ToString();
```

Bạn có thể viết lại handler OnGet như sau:

```
public void OnGet() {
    var title = RouteData.Values["title"].ToString();
    Title = title
        .Replace('-', ' ')
        .ToUpper();
    Text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod temp
}
```

> Một phương án nữa để truy xuất dữ liệu qua Url từ route data là sử dụng cơ chế model binding. Đây cũng là cơ chế tốt nhất để truy xuất dữ liệu người dùng từ client. Chúng ta sẽ học model binding trong một bài riêng.

Thực hành

Để kết thúc bài học đã khá dài này, chúng ta cùng thực hiện một ví dụ "kinh điển": viết ứng dụng giải phương trình bậc 2 sử dụng Razor Pages.

Bạn vẫn tiếp tục sử dụng project sẵn có (hoặc tạo project mới, tùy ý).

Bước 1. Tạo page mới Equation.cshtml.

Bước 2. Viết code cho model class EquationModel trong Equation.cshtml.cs như sau:

```
using System;

using Microsoft.AspNetCore.Mvc.RazorPages;

namespace WebApplication1.Pages {
    public class EquationModel : PageModel {
        public (double x1, double x2) Solutions { get; private set; }
        public (double a, double b, double c) Coefficients { get; private set; }

        public void OnGet(double a, double b, double c) {
            Coefficients = (a, b, c);
            var d = b * b - 4 * a * c;
            Solutions =
                ((-b + Math.Sqrt(d)) / (2 * a),
                (-b - Math.Sqrt(d)) / (2 * a));
        }
    }
}
```

Bước 3. Viết code cho page Equation.cshtml như sau:

```
@page "/solve/{a:double}_{b:double}_{c:double}"
@model WebApplication1.Pages.EquationModel
@{
    ViewData["Title"] = "Solve";
```

```
    var a = Model.Coefficients.a;
    var b = Model.Coefficients.b;
    var c = Model.Coefficients.c;
}

<h1>Solve equation</h1>
<h3>@(a)x<sup>2</sup> @(b<0?"":"+") @(b)x @(c<0?"":"+") @c = 0</h3>
<p><strong>X1 = @Model.Solutions.x1</strong></p>
<p><strong>X2 = @Model.Solutions.x2</strong></p>
```

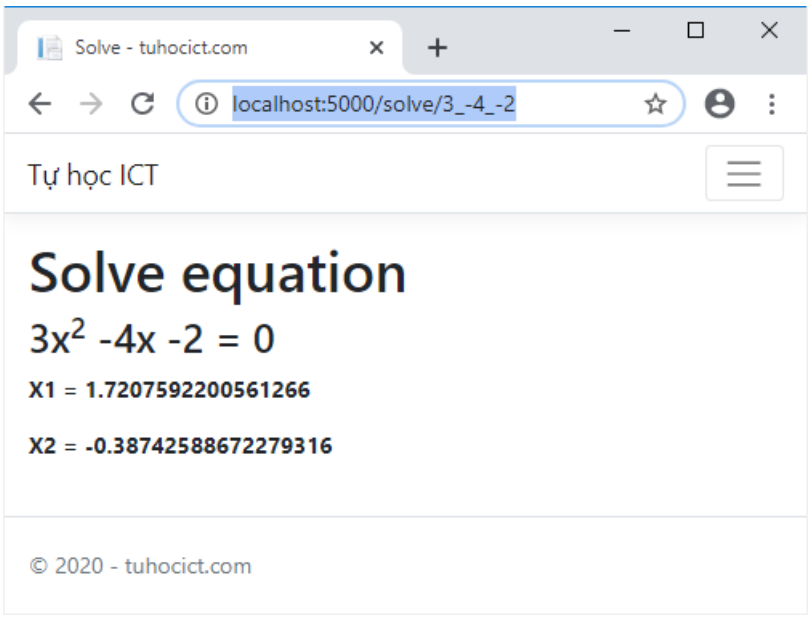
Hãy để ý cách chúng ta "chế tạo" mẫu url cho page Equation:

```
"/solve/{a:double}_{b:double}_{c:double}"
```

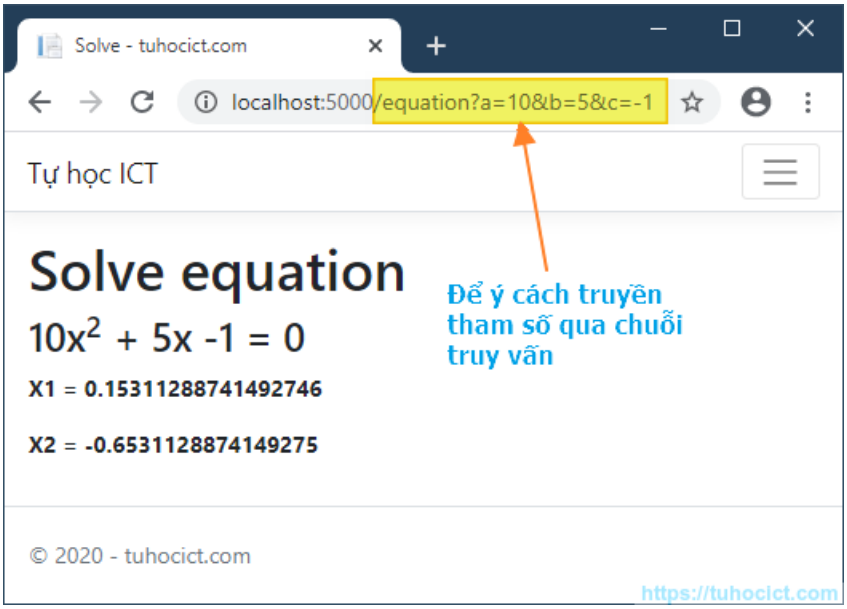
và danh sách tham số của OnGet: `public void OnGet(double a, double b, double c)`.

Mẫu này tương ứng với các Url như /solve/3_-4_-2, /solve/1_2_3, v.v., tức là segment cuối cùng chứa 3 con số phân tách bởi ký tự _. Các giá trị này sẽ được Razor Pages truyền cho các biến có tên tương ứng của OnGet.

Chạy thử với Url /solve/3_-4_-2 bạn thu được kết quả như sau:



Nếu bỏ route template sau @page directive, bạn có thể gọi phương thức OnGet như sau:



Khi này Equation.cshtml tương ứng với Url (mặc định) là /equation. Bạn truyền tham số cho OnGet thông qua chuỗi truy vấn `?a=10&b=5&c=-1`.

Qua bài thực hành này chúng ta thấy rằng, thông qua Url có thể "gọi" thẳng đến phương thức (handler) trên ứng dụng Razor Pages rất giống như khi làm việc với giao diện dòng lệnh. Route template của Razor Pages cho phép chúng ta chế tạo ra khuôn mẫu cho Url phù hợp yêu cầu.

Kết luận

Bài học đã hướng bạn cách thức truy xuất dữ liệu người dùng từ URL. Tùy thuộc vào cấu trúc URL, bạn có thể trích nó ra từ chuỗi truy vấn (query string) hoặc từ dữ liệu route (route data). Bạn cũng học thêm về route template – cách thức ghi đè chế độ routing mặc định.