HTML Form trong Razor Pages – xử lý truy vấn POST

Trong các ứng dụng web, form được sử dụng để thu thập dữ liệu (từ người dùng) và gửi trở lại server (để xử lý). Tạo và xử lý form cũng là yêu cầu chính của mọi ứng dụng web.

Trong bài học này chúng ta sẽ học cách sử dụng form cơ bản trong ứng dụng Razor Pages.

NỘI DUNG CỦA BÀI [Ẩn]

Thực hành

Để hiểu cách làm việc cơ bản với form, hãy cùng thực hiện một ví dụ.

Bước 1. Tạo project mới theo mẫu Web Application.

Bước 2. Viết code cho model class trong file Index.cshtml.cs như sau:

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

namespace WebApplication1.Pages {
    public class IndexModel : PageModel {
        public string Title { get; private set; }
        public string Text { get; private set; }
        public void OnPost(string title, string text) {
            Title = title;
            Text = text;
        }
    }
```

```
}
```

Bước 3. Viết code razor cho page Index.cshtml như sau:

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

@if (Request.Method.ToUpper() == "GET") {
    <h1 class="display-4">Welcome</h1>
    <form method="post">
        <div class="form-group">
            <label for="title">Title</label>
            <input type="text" class="form-control" id="title" name="title" />
        </div>
        <div class="form-group">
            <label for="text">Content</label>
            <textarea class="form-control" id="text" name="text"></textarea>
        </div>
        <div>
            <button class="btn btn-success" type="submit">Submit</button>
            <button class="btn btn-secondary" type="reset">Clear</button>
        </div>
    </form>
}
else if (Request.Method.ToUpper() == "POST") {
    <h1 class="display-4">@Model.Title</h1>
    <p>@Model.Text</p>
}
```

> <input> hay <textarea> được gọi là các điều khiển. Các điều khiển cần nằm trong cặp thẻ <form></form> để có thể hoạt động theo đúng mục đích. Lưu ý mỗi điều khiển đều phải gán giá trị cho thuộc tính name.
>
> Trong ví dụ trên chúng ta cũng sử dụng một số class của Bootstrap để định dạng cho đẹp. Đọc thêm về cách sử dụng Bootstrap với form.

Tùy thuộc vào truy vấn page sẽ hiển thị các nội dung khác nhau. Trong lần tải page đầu tiên (luôn là truy vấn GET) sẽ hiển thị form. Khi bấm nút Submit để gửi truy vấn POST về server, page sẽ tải lại nhưng giờ sẽ chỉ hiển thị lại những nội dung người dùng đã nhập. Khi tải lại trang sau truy vấn POST sẽ không hiển thị form.

Trong ví dụ trên bạn đã thấy, dữ liệu của điều khiển Input với thuộc tính `name = "title"` sẽ được truyền vào tham số `title` của `OnPost` – phương thức xử lý truy vấn POST. Tương như như vậy với TextArea `name="text"`.

Razor Pages cũng cho phép bạn truy xuất dữ liệu trong truy vấn POST sử dụng đặc tính `Request.Form` như sau:

```
public void OnPost() {
    Title = Request.Form["title"];
    Text = Request.Form["text"];
}
```

Qua đây bạn cũng thấy việc xử lý dữ liệu tới từ form (và truy vấn POST) không có gì khác biệt với xử lý dữ liệu đến từ URL (qua truy vấn GET).

> Trong bài học sau bạn sẽ làm quen với model-binding – cơ chế giúp đơn giản hóa hơn nữa cách truy xuất dữ liệu từ form.

Bạn đã thấy Razor Pages cho phép dễ dàng xử lý nhiều truy vấn đến cùng một page theo những cách riêng rẽ.

Form và các điều khiển

Trong ví dụ trên bạn đã sử dụng thẻ <form> và một số điều khiển (control) cơ bản. Chúng ta sẽ giới thiệu chi tiết hơn về <form> và điều khiển có thể sử dụng trên form.

Form

Ngôn ngữ HTML sử dụng thẻ `<form></form>` để định nghĩa một form – nơi chứa các điều khiển. Điều khiển, cũng được gọi là phần tử của form, là những công cụ chuyên dụng cho mục đích nhập dữ liệu, như hộp văn bản, nút đài, hộp chọn, v.v..

Tự bản thân thẻ <form> không hiển thị gì trên trang. Nó chỉ đóng vai trò nhóm các phần tử nhập dữ liệu lại và chỉ định cách truyền dữ liệu từ các điều khiển về server.

Một thẻ form thường có dạng như sau:

```
<form action="/index" method="POST">
    <!-- các phần tử (điều khiển) -->
</form
```

Thẻ form có một số đặc tính (attribute) quan trọng sau:

action: chỉ định URL đích của truy vấn HTTP. Ví dụ **`action = "index"`**, nghĩa là truy vấn HTTP sẽ gửi tới Url **`/index`**. Nếu không chỉ định gì, truy vấn mặc định sẽ gửi trở lại trang hiện tại.

enctype: chỉ định phương pháp đóng gói dữ liệu vào thân truy vấn HTTP (xem phần tiếp theo). Hai phương pháp chính thường dùng là url-encoded (mặc định) và multipart/form-data (dùng khi tải file từ trình duyệt lên server).

method: chỉ định phương thức truy vấn. Thường sử dụng nhất là POST, nhưng giá trị mặc định lại là GET.

Các phần tử của form

Trên form bạn có thể đặt các điều khiển để người dùng nhập dữ liệu.

Các phần tử của form, cũng gọi là các điều khiển, là những đối tượng đồ họa giúp người dùng nhập dữ liệu. Có nhiều loại phần tử khác nhau dành cho từng mục đích nhập liệu. Rất dễ hình dung các điều khiển trên HTML form cũng giống như các điều khiển trên giao diện đồ họa của ứng dụng desktop. Chúng cũng có cùng hình thức, cùng tên gọi và mục đích.

Khi học khóa học này chúng tôi giả định rằng bạn đã có những kiến thức nhất định về HTML. Do vậy chúng tôi sẽ không giới thiệu lại chúng nữa. Nếu không nhớ, bạn có thể xem lại danh sách các điều khiển HTML bạn có thể xem ở đây.

Khi làm việc với các điều khiển cần lưu ý:

(1) Để có thể thu thập được dữ liệu vào đóng gói vào truy vấn HTTP, trình duyệt yêu cầu mỗi điều khiển phải được đặt tên bằng cách thiết lập giá trị cho thuộc tính `name`.

(2) Giá trị của thuộc tính `name` cũng đồng thời là khóa để truy cập giá trị trong code C# qua object `Request.Form`, hoặc cũng chính là tên tham số của OnPost – phương thức xử lý truy vấn POST từ form.

(3) Mỗi form phải có một nút Submit (`<input type="submit">` hoặc `<button type="submit>`) để kết thúc quá trình nhập dữ liệu và gửi về server.

Để hiểu tại sao phải thiết lập giá trị cho thuộc tính name, chúng ta cùng xem xét phương pháp "đóng gói" dữ liệu vào thân truy vấn HTTP để gửi theo phương thức POST.

Đóng gói dữ liệu form vào thân truy vấn POST

Khi bạn ấn nút Submit, dữ liệu từ các điều khiển trên form được thu thập và đóng gói vào truy vấn HTTP để gửi lại server. Việc "đóng gói" dữ liệu phụ thuộc vào thiết lập enctype, và cũng phụ thuộc vào loại dữ liệu cần gửi đi.

Có hai phương pháp thường gặp để đóng gói dữ liệu vào truy vấn HTTP: **multipart/form-data** và **x-www-form-url-encoded**. Cả hai phương pháp đều có thể đóng gói và gửi về server dữ liệu ở dạng văn bản và nhị phân.

Phương pháp **x-www-form-url-encoded** sẽ lấy tên các điều khiển trên form và giá trị tương ứng ghép lại với nhau thành một chuỗi tương tự như chuỗi truy vấn của URL.

Phương pháp **multipart/form-data** sẽ chọn một cụm ký tự (không thể xuất hiện trong dữ liệu) làm *chuỗi đánh dấu* (boundary). Vai trò của chuỗi đánh dấu là phân tách dữ liệu ra từng *khối* (chunk).

Lấy ví dụ, nếu trên form bạn có hai điều khiển như sau:

```
<input type="text" name="title" />
<textarea name="text"></textarea>
```

Nếu người dùng nhập "Hello world" vào textbox, và "From Razor Pages" vào textarea. Phương pháp **x-www-form-url-encoded** sẽ đóng gói dữ liệu vào truy vấn HTTP như sau:

```
POST / HTTP/1.1
Host: localhost:5000
Content-Type: application/x-www-form-urlencoded

title=Hello world&text=From Razor Pages
```

Phương pháp **multipart/form-data** sẽ đóng gói dữ liệu như sau:

```
POST / HTTP/1.1
Host: localhost:5000
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="title"

Hello world
----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="text"

```

```
From Razor Pages
----WebKitFormBoundary7MA4YWxkTrZu0gW
```

Ở đây —-WebKitFormBoundary7MA4YWxkTrZu0gW được chọn làm chuỗi đánh dấu.

Phương pháp multipart/form-data thường dùng khi cần tải file lên server, trong khi x-www-form-url-encoded được sử dụng trong những trường hợp khác.

Như bạn nhìn thấy, trong cả hai phương pháp đóng gói, HTTP đều sử dụng đến giá trị và tên của điều khiển. Do vậy, khi tạo form, bạn đừng quên thuộc tính **name**.

Bảo vệ form khỏi tấn công CSRF

CSRF (Cross-Site Request Forgery), tạm dịch là giả mạo yêu cầu liên trang, là một dạng tấn công rất phổ biến đối với các ứng dụng web.

Razor Pages cung cấp sẵn một cơ chế bảo vệ khỏi tấn công CSRF có tên gọi là **xác minh truy vấn** (Request Verification). Cơ chế này được sử dụng mặc định mỗi khi bạn xây dựng form.

Quay trở lại ứng dụng đơn giản chúng ta đã xây dựng ở trên. Nếu bạn mở Development Tool (F12), tab Elements để xem mã HTML nhận được, bạn sẽ nhìn thấy một điều khiển lạ thuộc loại hidden như sau:

```
<input name="__RequestVerificationToken" type="hidden" value="CfDJ8MtmB74bsk1JrKZpMoA
```

Dĩ nhiên, giá trị của bạn sẽ không giống với ví dụ này, và mỗi lần bạn tải lại page, giá trị cũng biến đổi. Giá trị này chỉ có ý nghĩa một lần duy nhất.

Khi bạn ấn nút submit, trường __RequestVerificationToken và giá trị của nó cũng được đóng gói vào truy vấn POST gửi ngược trở lại server.

Khi chuyển sang tab Application, mục Storage -> Cookies, bạn sẽ nhìn thấy site đã thả lên máy bạn một cookie có tên .AspNetCore.Antiforgery.UfgPPsOClls (giá trị dưới đây chỉ mang tính minh họa):

```
.AspNetCore.Antiforgery.UfgPPsOClls=CfDJ8MtmB74bsk1JrKZpMoA99MB8-cCl3wc83jpA3dMa3Me6A
```

Tổ hợp giá trị cookie và chuỗi xác minh truy vấn được ASP.NET Core kiểm tra mỗi khi nhận truy vấn POST để đảm bảo rằng truy vấn đến từ đúng form "xịn". Nếu không có giá trị hoặc xác minh không đạt, ASP.NET Core sẽ trả lại phản hồi với mã 400 – Bad Request, và truy vấn POST không thể thực hiện được.

> Trong một số tình huống bạn không thể làm việc với form hoặc truy vấn POST do liên quan đến cơ chế bảo vệ này.

Cơ chế này mặc dù được sử dụng tự động, nghĩa là ASP.NET Core luôn luôn kiểm tra cookie và chuỗi xác minh. Tuy nhiên chuỗi xác minh trong trường __RequestVerificationToken lại chỉ được sinh ra nếu bạn sử dụng tag helper với directive

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

Khi sử dụng template Web Application, directive này được bật sẵn trong file _ViewImports.cshtml. Nếu bạn sử dụng template Empty, bạn phải tự mình thêm directive trên vào page hoặc vào _ViewImports.

Mặc dù không khuyến khích nhưng trong một số tình huống bạn có thể **tắt** chế độ kiểm tra này từ phương thức ConfigureServices (toàn cục) như sau:

```
public void ConfigureServices(IServiceCollection services) {
    services
        .AddRazorPages()
        .AddRazorPagesOptions(o=> {
            o.Conventions.ConfigureFilter(new IgnoreAntiforgeryTokenAttribute());
    });
}
```

hoặc tắt riêng cho một page nào đó:

```
[IgnoreAntiforgeryToken(Order = 1001)]
public class IndexModel : PageModel
{
    public void OnPost()
    {
    }
}
```

Lưu ý giá trị Order phải lớn hơn 1000.

Kết luận

Trong bài học này bạn đã học cách làm việc cơ bản với form và xử lý truy vấn POST. Dễ dàng thấy rằng, cách làm việc với form trong Razor Pages cũng rất đơn giản. Bạn có thể đọc dữ liệu từ form thông qua tham số của phương thức handler hoặc qua thuộc tính Request.Form.

Tải mã nguồn Razor Form

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!