Thẩm định dữ liệu (model validation) trong ASP.NET Core

Thẩm định dữ liệu (data validation) là việc kiểm tra dữ liệu trước khi xử lý nhằm đảm bảo dữ liệu đáp ứng các yêu cầu đặt ra.

Đây là một chủ đề rất quan trọng trong phát triển ứng dụng nói chung. Đối với ứng dụng web, do tính chất phân tán và nhiều người dùng, data validation càng có vai trò đặc biệt.

Trong ứng dụng ASP.NET Core, thẩm định dữ liệu thực hiện trên trình duyệt (client-side) và trước khi xử lý dữ liệu trên server (server-side). Data validation được xây dựng tích hợp trong ASP.NET Core giúp lập trình viên dễ dàng kết hợp data validation trong ứng dụng, cả client-side và server-side.

NỘI DUNG CỦA BÀI [Ẩn]

Data validation là gì?

Trước khi sử dụng một dữ liệu (ví dụ, để tạo object), bạn cần đảm bảo rằng:

- Dữ liệu cần định dạng chính xác. Ví dụ một địa chỉ email phải đảm bảo có dạng chính xác của một email.
- Số cần nằm trong một giới hạn nhất định. Ví dụ, giá trị năm không thể nhận giá trị âm, số lượng không thể nhận giá trị âm.
- Một số giá trị là bắt buộc, một số dữ liệu khác là tùy chọn. Ví dụ bạn có thể bắt buộc người dùng phải cung cấp email cho proifile nhưng số điện thoại lại không bắt buộc.
- Giá trị phải tuân thủ yêu cầu của nghiệp vụ. Ví dụ hệ số lương phải đi theo ngạch-bậc chứ không thể nhận giá trị tự do.

Đây chỉ là một số yêu cầu cơ bản nhất liên quan đến dữ liệu mà bất kỳ chương trình nào cũng đặt ra.

Thẩm định dữ liệu (data validation) là quá trình kiểm tra dữ liệu để đảm bảo rằng dữ liệu đó đáp ứng các yêu cầu đặt ra.

Trong một ứng dụng web, dữ liệu có thể đến từ nhiều nguồn khác nhau. Ví dụ, bạn có thể tải dữ liệu từ file, đọc dữ liệu từ cơ sở dữ liệu, nhận dữ liệu từ người dùng qua form/Url.

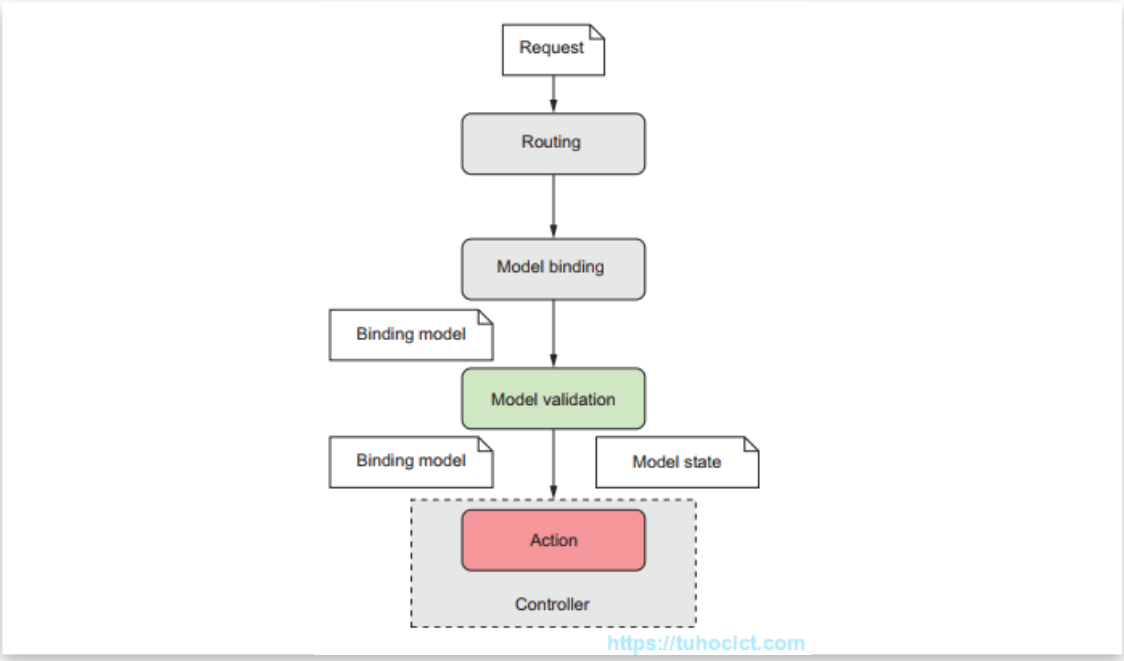
Nhìn chung bạn có thể tạm tin tưởng dữ liệu ở server (dữ liệu từ file trên server hoặc cơ sở dữ liệu).

Tuy nhiên bạn tuyệt đối không được tin tưởng dữ liệu đến từ truy vấn. Dữ liệu này có thể nằm trong form, trong route data, hoặc query string. Bạn không thể đảm bảo rằng người dùng luôn nhập đúng dữ liệu mình yêu cầu. Bạn cũng không thể chắc chắn là dữ liệu được truyền đi nguyên vẹn từ trình duyệt tới server.

Do vậy, bạn luôn cần kiểm tra dữ liệu người dùng trước khi sử dụng trong action.

Trong ứng dụng web, thẩm định dữ liệu có thể diễn ra ở hai nơi:

1. Trên trình duyệt: cơ chế này được gọi là **client-side validation** và sử dụng Javascript. Mặc dù rất tiện lợi cho người dùng, cơ chế này rất không đáng tin.
2. Trên server: còn được gọi là **server-side validation**. Cơ chế thẩm định dữ liệu trên server của ASP.NET Core hoạt động ngay sau model binding nhưng trước khi thực thi action. Đây là cơ chế thẩm định đáng tin cậy và bạn nên (thậm chí là phải) sử dụng nó.



Trong ASP.NET Core, cơ chế model binding tạo ra cho bạn một model (binding model). Cơ chế thẩm định dữ liệu hoạt động trên binding model. Do vậy server-side validation trong ASP.NET Core cũng được gọi là **model validation**.

Model validation đảm bảo rằng các trường dữ liệu của binding model đảm bảo các yêu cầu bạn đã đặt ra trước khi sử dụng.

Nhìn chung, bạn nên kết hợp cả hai loại validation: client-side validation để tạo trải nghiệm người dùng tốt cũng như hỗ trợ một phần cho server-side; server-side validation để đảm bảo sự chính xác và an toàn của dữ liệu trước khi xử lý.

Model validation trong ASP.NET Core MVC

Để hiểu rõ hơn các vấn đề vừa trình bày, hãy cùng thực hiện một bài thực hành nhỏ. Trong bài thực hành này chúng ta sẽ xây dựng tính năng tạo mới dữ liệu trong một ứng dụng MVC quản lý profile người dùng.

1. Tạo dự án ASP.NET Core MVC mới

Tạo dự án ASP.NET Core mới sử dụng template Web Application (Model-View-Controller). Bạn cũng có thể tạo project rỗng rồi tự tự cấu hình. Đặt tên project là WebApplication1.

2. Tạo domain model class

Tạo file Profile.cs trong thư mục Models và viết code như sau:

```
using System.ComponentModel.DataAnnotations;

namespace WebApplication1.Models {
    public class Profile {
        public int Id { get; set; }

        [Required]
        [StringLength(100)]
        public string FirstName { get; set; }

        [Required]
        [StringLength(100)]
        public string LastName { get; set; }

        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Phone]
        public string PhoneNumber { get; set; }
    }
}
```

Các attribute [Required], [StringLength], [EmailAddress], [Phone] được gọi là các **data annotation**. Chúng bổ sung thông tin cho các property. Các thông tin này được cơ chế model validation của ASP.NET Core đọc và xử lý cả ở client và server.

3. Tạo controller class

Tạo file ProfileController.cs trong thư mục Controllers và viết code như sau:

```
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using WebApplication1.Models;

namespace WebApplication1.Controllers {
    public class ProfileController : Controller {
        private static List<Profile> Profiles { get; } = new List<Profile> {
            new Profile { Id = 1, FirstName = "Donald", LastName = "Trump", Email = "tru
            new Profile { Id = 2, FirstName = "Barack", LastName = "Obama", Email = "oba
        };

        public IActionResult Create() {
            var profile = new Profile();
            return View(profile);
        }

        [HttpPost]
        public IActionResult Create(Profile profile) {
            if (!ModelState.IsValid) {
                return View(profile);
            }
            Profiles.Add(profile);
```

```
            return RedirectToAction("Index");
        }

        public IActionResult Index() {
            return View(Profiles);
        }
    }
}
```

3. Tạo các view

Tạo **Index** view (/Views/Profile/Index.cshtml) như sau:

```
@using WebApplication1.Models
@model List<Profile>

<a class="btn btn-primary" asp-action="Create">Create</a>
<table class="table table-striped">
    <thead>
        <tr>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
            <th>Phone</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var p in Model) {
            <tr>
                <td>@p.FirstName</td>
                <td>@p.LastName</td>
                <td>@p.Email</td>
                <td>@p.PhoneNumber</td>
            </tr>
        }
    </tbody>
</table>
```

Tạo **Create** view (/Views/Profile/Create.cshtml) như sau:

```
@model WebApplication1.Models.Profile

<div class="row">
    <div class="col-md-4">
        <form asp-action="Create" method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="FirstName" class="control-label"></label>
                <input asp-for="FirstName" class="form-control" />
                <span asp-validation-for="FirstName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="LastName" class="control-label"></label>
                <input asp-for="LastName" class="form-control" />
                <span asp-validation-for="LastName" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Email" class="control-label"></label>
                <input asp-for="Email" class="form-control" />
                <span asp-validation-for="Email" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="PhoneNumber" class="control-label"></label>
                <input asp-for="PhoneNumber" class="form-control" />
                <span asp-validation-for="PhoneNumber" class="text-danger"></span>
            </div>
            <div class="form-group">
                <input type="submit" value="Create" class="btn btn-primary" />
                <a asp-action="Index">Back</a>
            </div>
        </form>
    </div>
</div>

@section Scripts {
```

```
36.         @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
37.     }
```

Chạy thử ứng dụng với Url /profile. Khi thử nghiệm khả năng tạo dữ liệu mới hãy để ý:

1. Nếu bạn không nhập các trường dữ liệu bắt buộc (FirstName, LastName, Email), site sẽ tải lại với thông báo lỗi ở sau từng ô nhập.
2. Nếu bạn nhập giá trị không hợp lệ cho Email, site không tải lại mà đưa ra thông báo lỗi ngay lập tức.
3. Nếu bạn nhập giá trị Phone không hợp lệ, site tải lại và báo lỗi ở sau ô nhập Phone.
4. Nếu bạn nhập dữ liệu hợp lệ, site sẽ điều hướng trở lại trang /profile.

Trong trường hợp 1, server-side validation hoạt động kết hợp với client-side validation (khi tải lại trang). Trong trường hợp 2 chỉ có client-side validation hoạt động (dữ liệu chưa trả về server). Trường hợp 3 giống như trường hợp 1 nhưng với điều kiện khác (dữ liệu Phone không hợp lệ).

DataAnnotations

Data annotation là các attribute cung cấp thông tin bổ sung cho các property hoặc class của C#. Loại thông tin bổ sung này còn được gọi là **metadata** – dữ liệu về dữ liệu.

Trong ví dụ trên bạn đã mô tả property FirstName với hai attribute `[Required]` và `[StringLength(100)]`. `[Required]` bổ sung thông tin rằng FirstName là một trường bắt buộc – nghĩa là nó không được nhận giá trị null. `[StringLength(100)]` bổ sung thông tin rằng độ dài tối đa của trường này là 100 ký tự.

Tương tự như vậy, `[EmailAddress]` báo rằng dữ liệu của trường Email phải có định dạng của một email hợp lệ, `[Phone]` báo rằng dữ liệu của trường PhoneNumber phải có định dạng của một số điện thoại hợp lệ.

Như vậy, kết hợp lại, khi định nghĩa lớp `Profile`, bạn đã đặt ra yêu cầu:
1. FirstName và LastName phải có giá trị và ngắn hơn 100 ký tự;
2. Email là bắt buộc và phải là một email hợp lệ;
3. Phone phải là số điện thoại hợp lệ nhưng không bắt buộc.

Các metadata được cơ chế server-side đọc và sử dụng khi thẩm định binding model. Đối với client-side validation, tag helper cũng sử dụng các thông tin này để sinh ra html và kết hợp hoạt động với jQuery.

> Nếu muốn sử dụng cơ chế validation trong ASP.NET Core, bạn bắt buộc phải mô tả model với data annotation.

C# cung cấp một loạt các attribute như vậy trong không gian tên `System.ComponentModel.DataAnnotations`.

Sau đây là một số attribute thường gặp:

- [CreditCard] – phải có định dạng của thẻ credit hợp lệ
- [MinLength(min)] – độ dài tối thiểu của collection
- [Range(min, max)] – xác định vùng giá trị
- [RegularExpression(regex)] – giá trị phải phù hợp với regex
- [Url] – giá trị phải có định dạng của Url hợp lệ
- [Compare] – so sánh giá trị của hai property (ví dụ email và confirm email, password và confirm password)

> Data annotation không chỉ được sử dụng trong ASP.NET Core validation. Entity Framework cũng sử dụng các attribute này.

Server-side validation

Trong ASP.NET Core, cơ chế thẩm định dữ liệu **luôn được thực hiện** trên binding model ngay trước khi thực thi action. Tức là dù bạn không làm gì hết, cơ chế thẩm định vẫn tự động thực hiện nếu có tạo ra binding model.

Trong ví dụ trên, trước khi gọi phương thức action `IActionResult Create(Profile profile) { ... }`, ASP.NET Core phải tạo ra binding model (kiểu Profile). Khi tạo ra object profile, cơ chế model validation được gọi tự động để kiểm tra tính hợp lệ của các trường FirstName, LastName, Email, Phone. Nếu tất cả các trường hợp lệ thì kết quả thẩm định là đạt. Chỉ cần 1 trường không đạt thì thẩm định không đạt.

Tuy nhiên, trong action bạn phải tự mình xác định cần làm gì với kết quả thẩm định. ASP.NET Core không tự động xử lý kết quả này. Nó chỉ báo cho bạn biết kết quả thẩm định.

ASP.NET Core lưu kết quả thẩm định trong property `ModelState`. Đây là một object của class `ModelStateDictionary`. Nó lưu thông tin chung về thẩm định (thành công/ thất bại) trong property `IsValid`.

Mặc dù bạn có toàn bộ tự do khi xử lý kết quả thẩm định, ứng dụng ASP.NET Core thường áp dụng mô hình xử lý như sau:

1. Nếu validation thất bại => trả lại đúng binding model và view đã dùng để hiển thị binding model đó. Khi server-side validation chạy nó đã xác định rõ tất cả các vị trí lỗi. Do vậy khi render lại view (với binding model bị lỗi), bạn có thể chỉ rõ các vị trí lỗi để người dùng điều chỉnh. Việc render lại view với thông báo lỗi được ASP.NET Core hỗ trợ với Input tag helper và jQuery. Bạn sẽ học kỹ hơn trong phần client-side validation.
2. Nếu validation thành công => xử lý và chuyển hướng sang một action khác.

Đây là cách chúng ta đã áp dụng trong ProfileController:

```
public IActionResult Create(Profile profile) {
    if (!ModelState.IsValid) {
        return View(profile); // render lại Create view với object profile lỗi
    }
    Profiles.Add(profile);
    return RedirectToAction("Index"); // chuyển hướng sang Index
}
```

Lưu ý rằng cơ chế thẩm định trên server không có bất kỳ quan hệ gì với giao diện. Server kiểm tra và đưa lại cho bạn kết quả đúng/sai.

Việc render lại view (với thông báo lỗi) được ASP.NET Core hỗ trợ thông qua sử dụng Input tag helper và jQuery. Đây lại thuộc về client-side validation.

Client-side validation

Đến đây hẳn bạn vẫn băn khoăn về cơ chế validation trong ASP.NET Core. Mặc dù chúng ta đã giải thích server-side validation và vai trò của data annotation.

Vấn đề rắc rối nằm ở việc kết hợp với client-side validation: Tại sao cùng một view khi tải lần đầu không hiển thị lỗi nhưng khi tải lần thứ hai (với model bị lỗi) lại có thể hiển thị lỗi; Tại sao có lỗi được báo ngay (email không hợp lệ); Tại sao có lỗi phải chờ gửi dữ liệu về server kiểm tra mới xuất hiện.

Quay lại với form trong Create view:

```
<div class="form-group">
    <label asp-for="FirstName" class="control-label"></label>
    <input asp-for="FirstName" class="form-control" />
    <span asp-validation-for="FirstName" class="text-danger"></span>
</div>
```

Đoạn mã Razor trên khi chạy lần đầu sẽ sinh ra HTML như sau:

```
<div class="form-group">
    <label class="control-label" for="FirstName">FirstName</label>
    <input class="form-control" type="text" data-val="true" data-val-length="The field F
    <span class="text-danger field-validation-valid" data-valmsg-for="FirstName" data-va
</div>
```

Hãy để ý thẻ input có hàng loạt attribute lạ `data-val-*`. Đây là những attribute do Input tag helper sinh ra dựa trên data annotation của model. Những attribute này dành cho sử dụng cùng thư viện jQuery.

Trong lượt tải đầu tiên, validation không hoạt động và thẻ span giờ đang trống.

Nếu bạn submit form với trường FirstName để trống, form sẽ tải lại. Nội dung của nhóm này giờ biến thành như sau:

```
<div class="form-group">
    <label class="control-label" for="FirstName">FirstName</label>
    <input class="form-control input-validation-error" type="text" data-val="true" data-
    <span class="text-danger field-validation-error" data-valmsg-for="FirstName" data-va
</div>
```

Ở đây thẻ span đã được điền thông tin lỗi, đồng thời giá trị class cũng được điều chỉnh từ field-validation-valid sang field-validation-error.

Như vậy, tag helper `<span asp-validation-for="FirstName"></span>` đã căn cứ vào kết quả thẩm định đối với FirstName để tự động sinh ra nội dung phù hợp. Lần đầu tải form, validation không hoạt động, tag helper không hiển thị lỗi. Lần tải thứ hai với model lỗi, tag helper sinh ra thông báo lỗi cho trường tương ứng.

Một điều rất may mắn là tất cả những gì chúng ta vừa trình bày ở trên đều được ASP.NET Core thực hiện tự động. Cái duy nhất bạn phải làm là sử dụng **validation mesage tag helper** `asp-validation-for` với thẻ `span` và ở vị trí mình cần.

Một sự hỗ trợ khác là **validation summary tag helper**:

```
1.  <div asp-validation-summary="All" class="text-danger"></div>
```

Tag helper này giúp hiển thị một danh sách lỗi tập trung. Bạn có thể sử dụng một trong ba giá trị: All, ModelOnly, None. Trong đó,

- "All" hiển thị tất cả các thông bao lỗi.
- "ModelOnly" chỉ hiển thị lỗi liên quan trực tiếp đến binding model chứ không hiển thị lỗi liên quan đến property (đã hiển thị sẵn ở các ô nhập liệu).
- "None" không hiển thị bất kỳ lỗi nào.

Kết luận

Trong bài học này chúng ta đã trao đổi chi tiết về cơ chế thẩm định dữ liệu trong ASP.NET Core, bao gồm data annotations attribute, server-side validation, client-side validation.

Mặc dù đây là một chủ đề phức tạp trong phát triển ứng dụng web, những hỗ trợ từ ASP.NET Core giúp model validation trở nên đơn giản và tiện lợi hơn rất nhiều.

Bạn cần lưu ý luôn luôn sử dụng validation trong ứng dụng của mình và sử dụng kết hợp cả client-side và server-side validation.