Client-Side and Server-Side Validation

.NET CORE

Client-Side validation gives users instant feedback on the information they submitted to a web page. It is an expected feature in today's applications. Server-Side validation is necessary because information arriving from the network should never be trusted .

HTTPS://WWW.C-SHARPCORNER.COM/ARTICLE/CUSTOM-DATA-ANNOTATION-VALIDATION-IN-MVC/

Why Validate User Input?

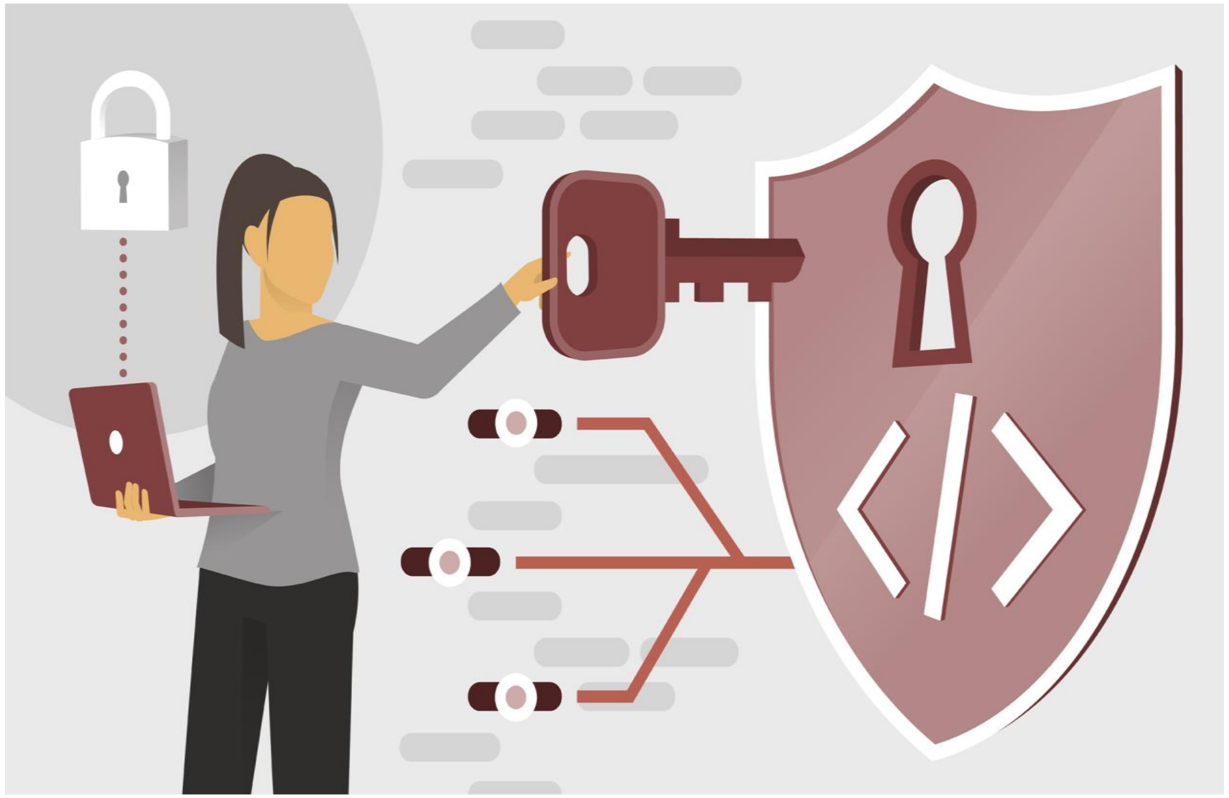
Client-Side validation is implemented mostly to enhance the users experience. The user gets faster error checking and doesn't need to submit a form to see that their input was invalid.

For Client-Side validation, you can use the built-in HTML validation attributes, but they are limited. .NET Tag Helpers are designed to work with the ***jQuery Unobtrusive Validation*** script. Microsoft ***jQuery Validation Library***, uses ***jQuery's Validate Plugin***.

Tag Helpers put *HTML5 data attributes* into form controls, which the Validation Library uses to configure validation logic and display validation messages on the Client-Side. This enables ***data annotations*** to drive consistent validation on both the Server-Side and the Client-Side (before sending to server).

Custom Client-Side validation is also possible.

Server-Side validation is still necessary. The user could have their JavaScript disabled or have malicious intent.



jQuery Unobtrusive Validation

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#client-side-validation

The ***jQuery Unobtrusive Validation*** script is a custom Microsoft front-end library that builds on the ***jQuery Validate*** plugin. Without ***jQuery Unobtrusive Validation***, Tag Helpers and HTML helpers use the validation attributes and type metadata from ***model*** properties to render HTML 5 data-attributes. ***jQuery Unobtrusive Validation*** parses the data-attributes and passes the logic to ***jQuery Validate***, effectively "copying" the server-side validation logic to the client. This way you can display validation errors to the client using ***Tag Helpers***.

The below scripts import the ***jQuery Unobtrusive Validation*** scripts.

In ***_Layout.cshtml***

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
```

In ***_ValidationScriptsPartial.cshtml***

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.19.1/jquery.validate.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-unobtrusive/3.2.11/jquery.validate.unobtrusive.min.js"></script>
```

Model State

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#model-state

Model state represents errors that come from two subsystems: ***model binding*** and ***model validation***.

Model Binding errors are generally data conversion errors.

- Ex. an "x" is entered in an integer field.

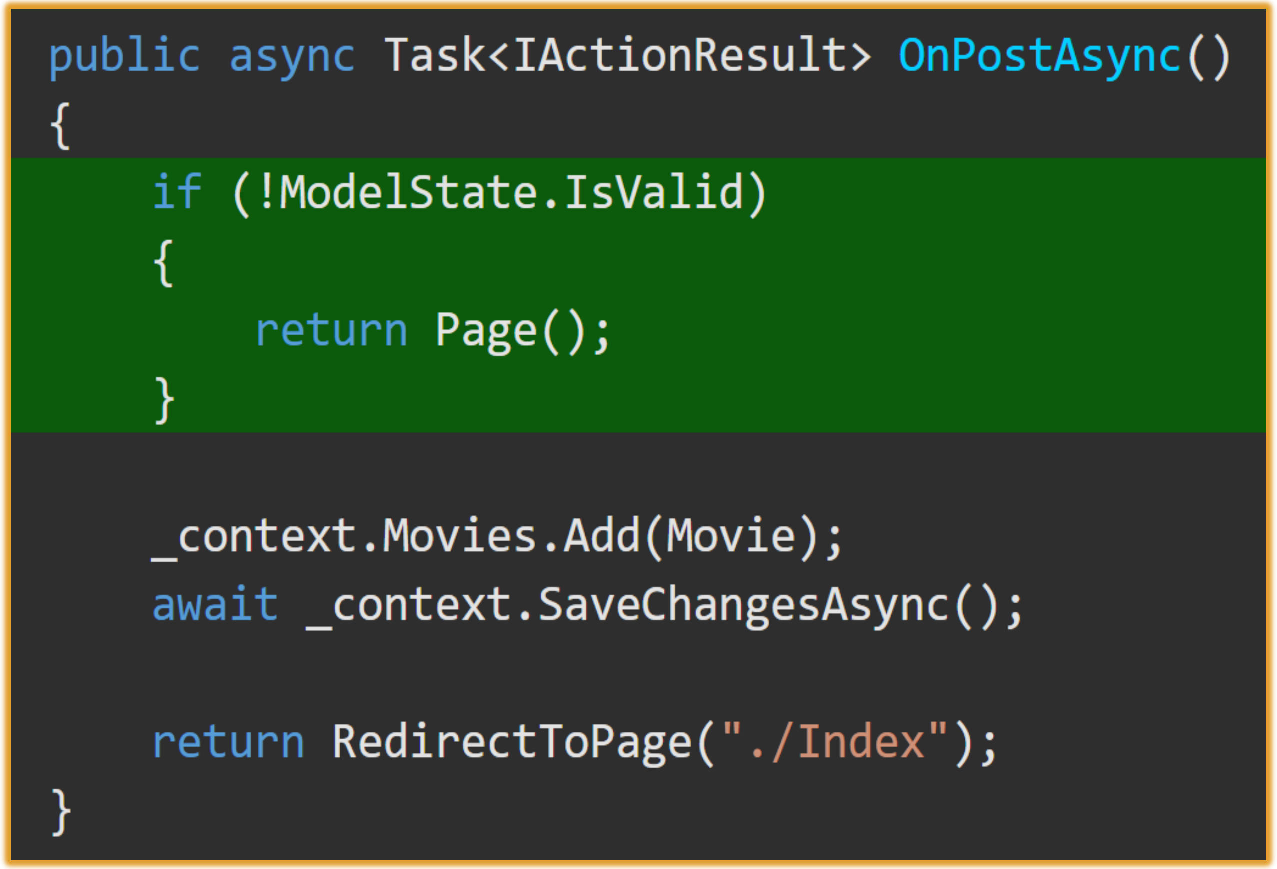
Model validation occurs after ***model binding*** and reports errors where data doesn't conform to business rules.

- Ex. a 0 is entered in a field that expects a rating between 1 and 5.

Model State Validation

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#model-state
https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-3.1#automatic-http-400-responses

Both ***model binding*** and ***model validation*** occur before the execution of a ***controller action***. For web apps, it's the app's responsibility to inspect ***ModelState.IsValid*** and react appropriately. When ***!ModelState.IsValid***, web apps typically redisplay the page with an error message. Web API ***controllers*** don't have to check ***ModelState.IsValid*** if they have the ***[ApiController]*** attribute. If they do, an automatic ***HTTP 400*** response containing error details is returned.



```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

    _context.Movies.Add(Movie);
    await _context.SaveChangesAsync();

    return RedirectToPage("./Index");
}
```

Validation – Client-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#validation-attributes

Attribute	Purpose
[CreditCard]	Validates the property has a credit card format.
[Compare]	Validates two properties in a model match.
[EmailAddress]	Validates the property has an email format.
[Phone]	Validates the property has a telephone number format.
[Range]	Validates the property value falls within a specified range.
[RegularExpression]	Validates the property value matches a specified regular expression.
[Required]	Validates the field is not null.
[StringLength]	Validates a string property value doesn't exceed a specified length limit.
[Url]	Validates the property has a URL format.
[Remote]	Validates input on the client by calling an action method on the server. (very cool!)

```
public class Movie
{
    public int Id { get; set; }

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

    [ClassicMovie(1960)]
    [DataType(DataType.Date)]
    [Display(Name = "Release Date")]
    public DateTime ReleaseDate { get; set; }

    [Required]
    [StringLength(1000)]
    public string Description { get; set; }

    [Range(0, 999.99)]
    public decimal Price { get; set; }

    public Genre Genre { get; set; }

    public bool Preorder { get; set; }
}
```

See a complete list of validation attributes.

Validation – Client-Side Error Messages

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#error-messages

Error messages get displayed on the web page for the user to see.

```
[StringLength(8, ErrorMessage = "Name length can't be more than 8.")]
```

```
[StringLength(8, ErrorMessage = "{0} length must be between {2} and {1}.", MinimumLength = 6)]
```

When applied to a Name property, the error message created by the preceding code would be "Name length must be between 6 and 8.".

Validation – Custom Data Annotations

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#custom-attributes

Create custom validation attributes.

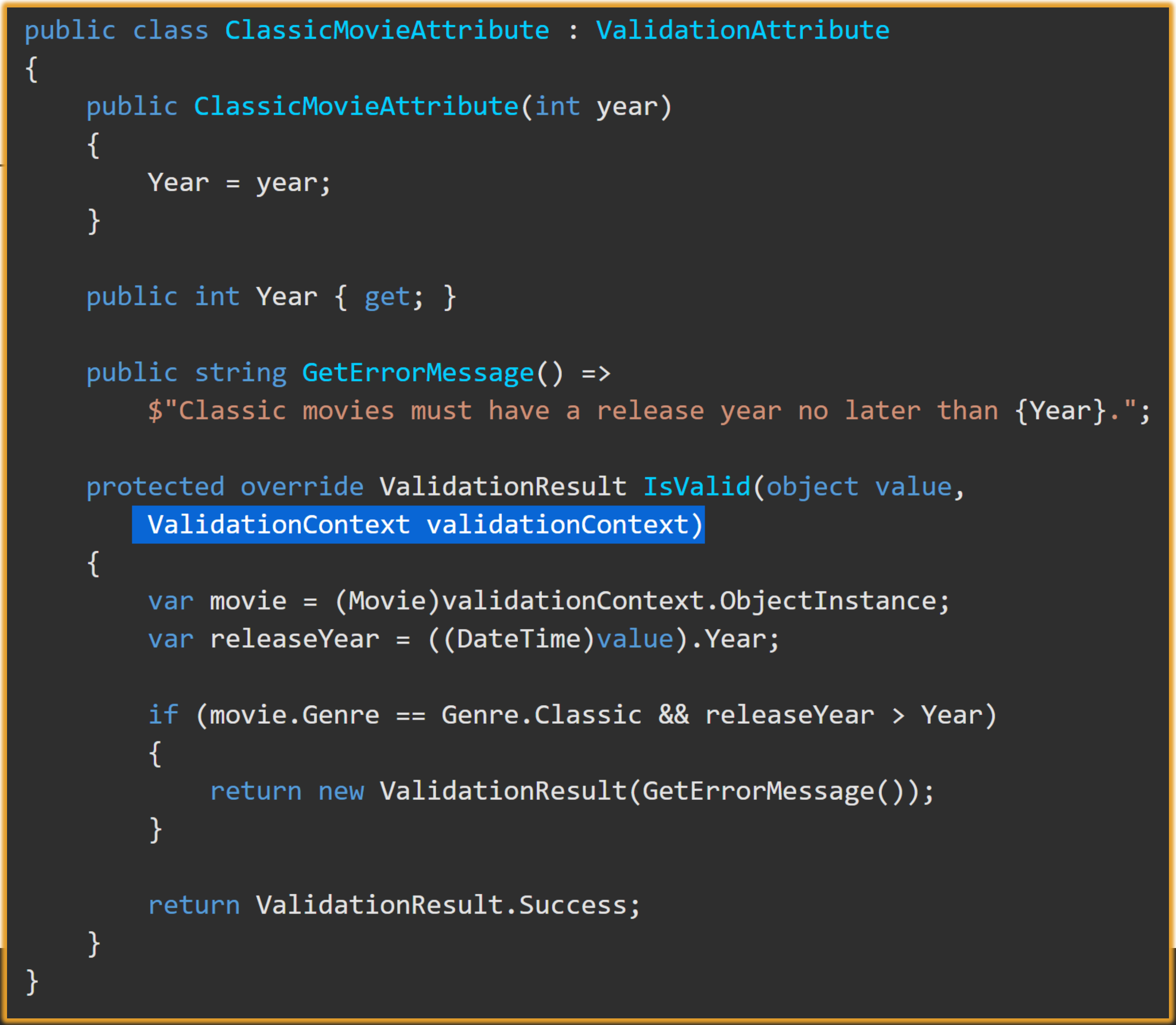
1) Create a class that inherits from *ValidationAttribute.*

2) Override the ***IsValid*** method.

- The ***IsValid*** method accepts an object named value, which is the input to be validated. An overload also accepts a ***ValidationContext*** object, which provides additional information, such as the model instance created by model binding.

This example validates that the release date for a movie in the Classic genre isn't after a specified year. The ***[ClassicMovie]*** attribute is only run on the server.

The Data Annotation in the Model would look like this → ***[ClassicMovie(1957)]***



```
public class ClassicMovieAttribute : ValidationAttribute
{
    public ClassicMovieAttribute(int year)
    {
        Year = year;
    }

    public int Year { get; }

    public string GetErrorMessage() =>
        $"Classic movies must have a release year no later than {Year}.";

    protected override ValidationResult IsValid(object value,
        ValidationContext validationContext)
    {
        var movie = (Movie)validationContext.ObjectInstance;
        var releaseYear = ((DateTime)value).Year;

        if (movie.Genre == Genre.Classic && releaseYear > Year)
        {
            return new ValidationResult(GetErrorMessage());
        }

        return ValidationResult.Success;
    }
}
```

Validation – [Required] Server-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#required-validation-on-the-server

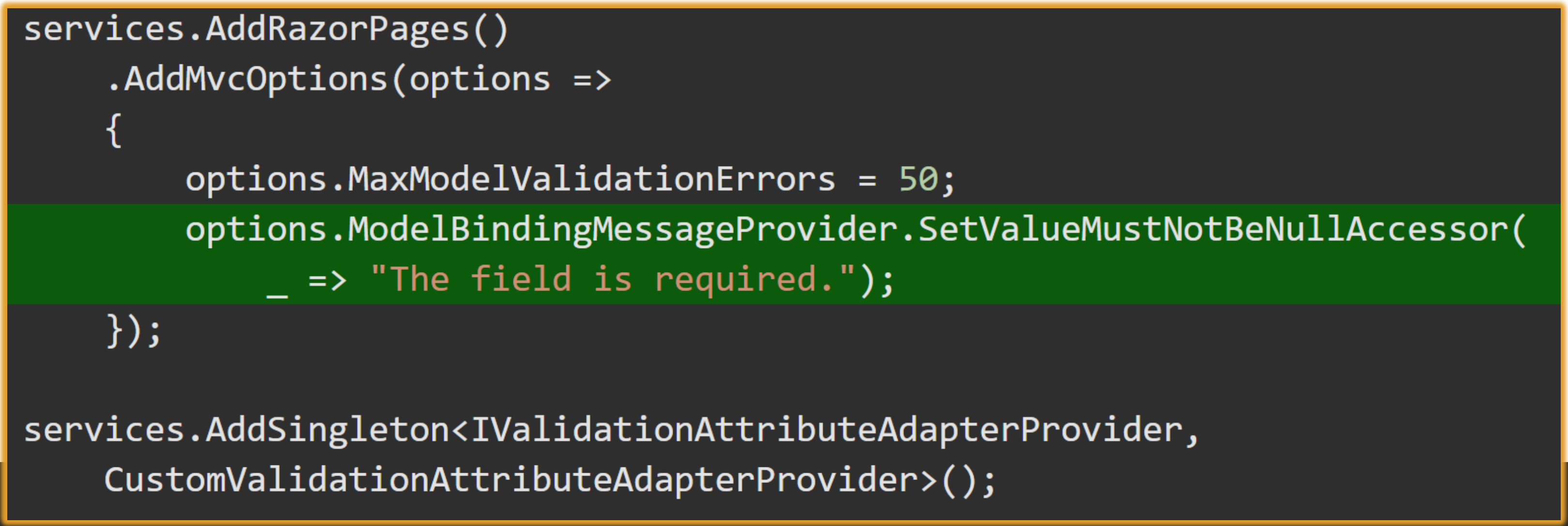
The validation system in .NET Core treats ***non-nullable*** parameters or ***bound*** properties as if they had a ***[Required]*** attribute. ***Value types*** such as ***decimal*** and ***int*** are ***non-nullable***. This behavior can be disabled by configuring *SuppressImplicitRequiredAttributeForNonNullableReferenceTypes* in *Startup.ConfigureServices* (in Startup.cs)**.**

```
services.AddControllers(options =>
options.SuppressImplicitRequiredAttributeForNonNullableReferenceTypes = true);
```

Validation – [Required] Server-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#required-validation-on-the-server

Model binding for a non-nullable property can FAIL which leaves the value ***null***. On the server, a ***[Required]*** value is considered missing if the property is null, but a ***non-nullable*** field (***int*** or ***decimal***) is always valid server-side. This means the ***[Required]*** attribute's error message is never displayed.



```
services.AddRazorPages()
    .AddMvcOptions(options =>
    {
        options.MaxModelValidationErrors = 50;
        options.ModelBindingMessageProvider.SetValueMustNotBeNullAccessor(
            _ => "The field is required.");
    });

services.AddSingleton<IValidationAttributeAdapterProvider,
    CustomValidationAttributeAdapterProvider>();
```

There are two options to specify a custom error message for server-side validation of non-nullable types.

- Make the field ***nullable*** (Ex, ***decimal?*** instead of ***decimal***).
- (not recommended) Specify the default error message to be used by model binding.

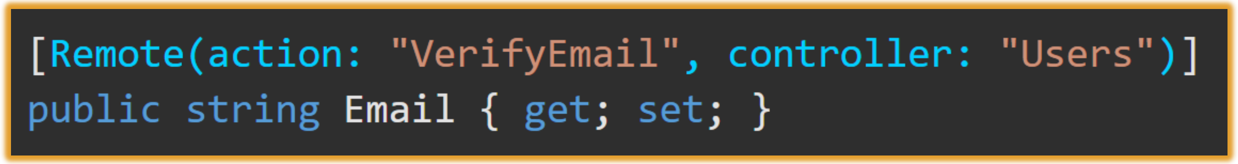
Validation – [Remote] Server-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#remote-attribute

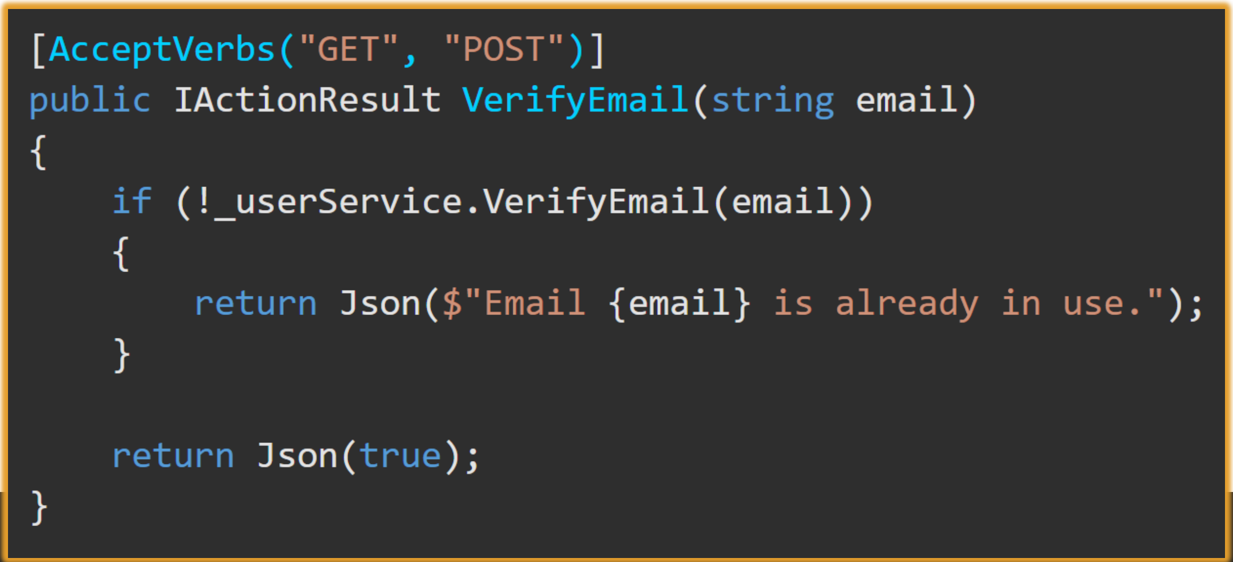
The *[Remote]* attribute implements client-side validation that requires calling an ***action method*** on the server to determine whether field input is valid. For example, the app may need to verify whether a userName is already in use.

Create an ***action method*** for JavaScript to call. The *jQuery* Validate remote method expects a *JSON* response:

- ***true*** means the input data is valid.
- ***false***, ***undefined***, ***null*** or any other string means the input is invalid. Display the default error message.
- Display the string as a custom error message.



```
[Remote(action: "VerifyEmail", controller: "Users")]
public string Email { get; set; }
```



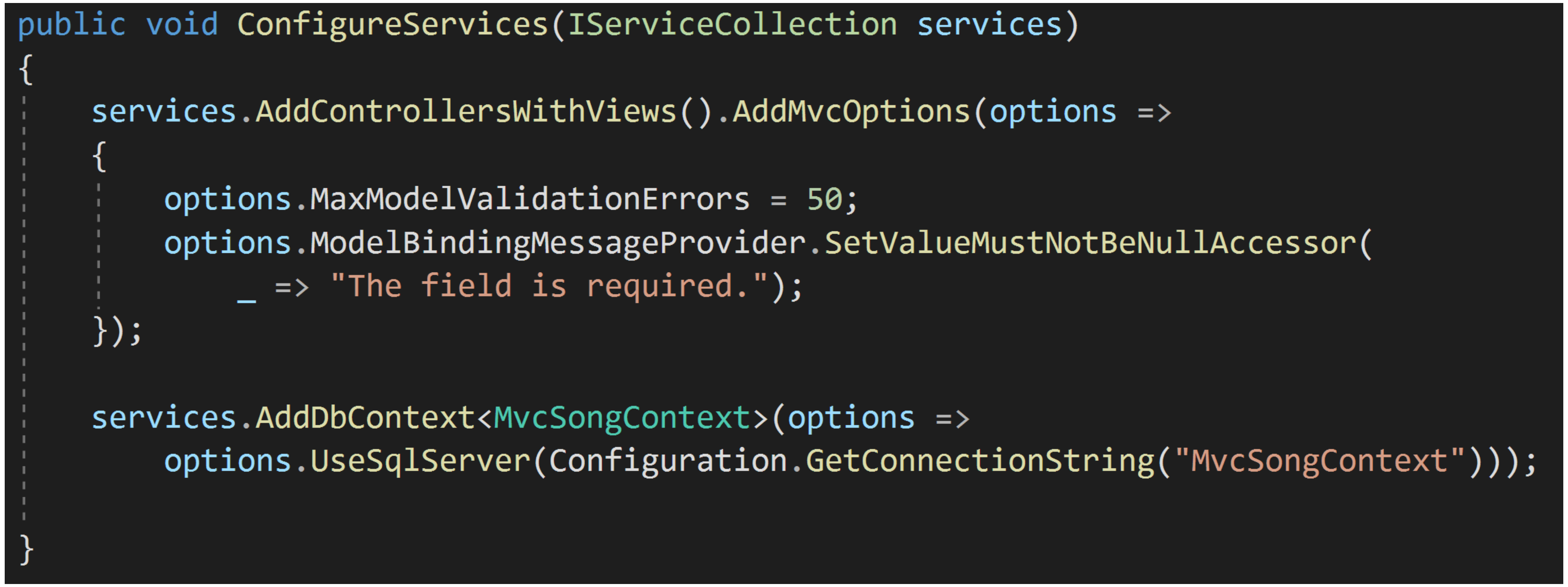
```
[AcceptVerbs("GET", "POST")]
public IActionResult VerifyEmail(string email)
{
    if (!_userService.VerifyEmail(email))
    {
        return Json($"Email {email} is already in use.");
    }

    return Json(true);
}
```

*You can also check multiple fields in combination

Validation – Maximum Errors

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-3.1#maximum-errors

Validation stops when the maximum number of errors is reached (200 by default). You can configure this number with the following code in Startup.ConfigureServices:



```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews().AddMvcOptions(options =>
    {
        options.MaxModelValidationErrors = 50;
        options.ModelBindingMessageProvider.SetValueMustNotBeNullAccessor(
            _ => "The field is required.");
    });

    services.AddDbContext<MvcSongContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("MvcSongContext")));

}
```

*You can also set Maximum Recursion.