Data Annotations and Validation

.NET

Data Annotations Attributes enable you to perform validation by adding attributes to a model class property.

HTTPS://DOCS.MICROSOFT.COM/EN-US/ASPNET/MVC/OVERVIEW/OLDER-VERSIONS-1/MODELS-DATA/VALIDATION-WITH-THE-DATA-ANNOTATION-VALIDATORS-CS

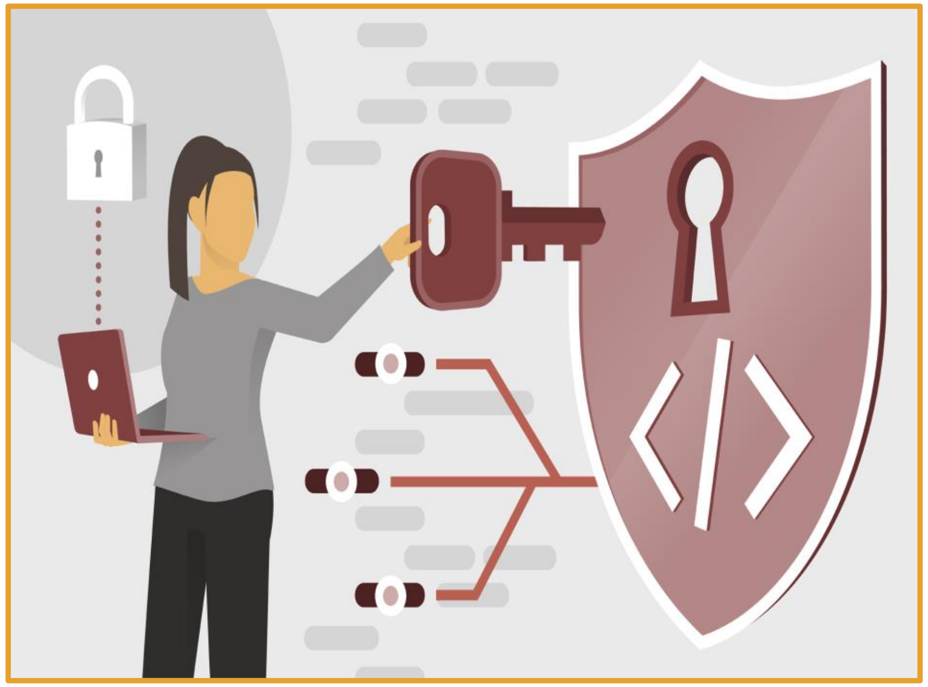
Why Validate User Input?

Client-Side validation gives the user faster error checking. They don't need to submit a form to see that their input was invalid.

For Client-Side validation, built-in HTML validation attributes can be used. .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).

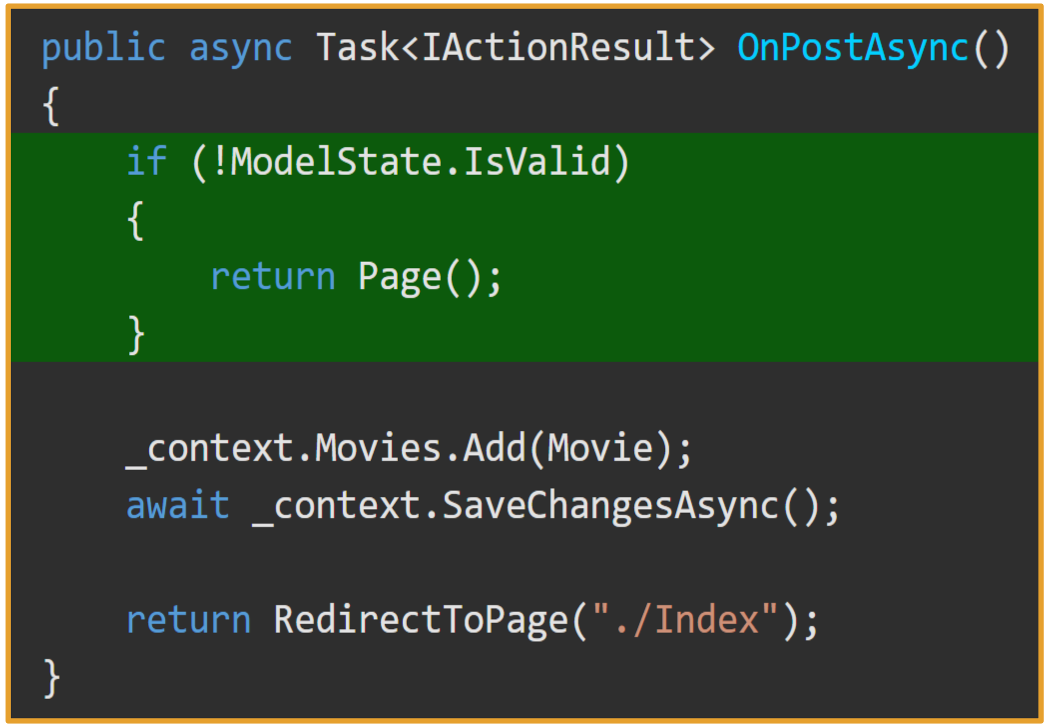
Custom Client-Side validation is also possible.



Model State Validation

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#model-state
https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-5.0#automatic-http-400-responses

Both ***Model Binding*** and ***Model Validation*** occur before the execution of a ***Controller Action Method***. Web apps must manually inspect ModelState.IsValid. If false, redisplay the webpage with an error message. Web API ***Controllers*** using the [ApiController] attribute automatically respond with an ***HTTP 400*** response containing error details.



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

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

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

Model State

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#model-state

Model state comes from the ***filter*** pipeline and represents errors that come from two subsystems: ***Model Binding*** and ***Model Validation***.

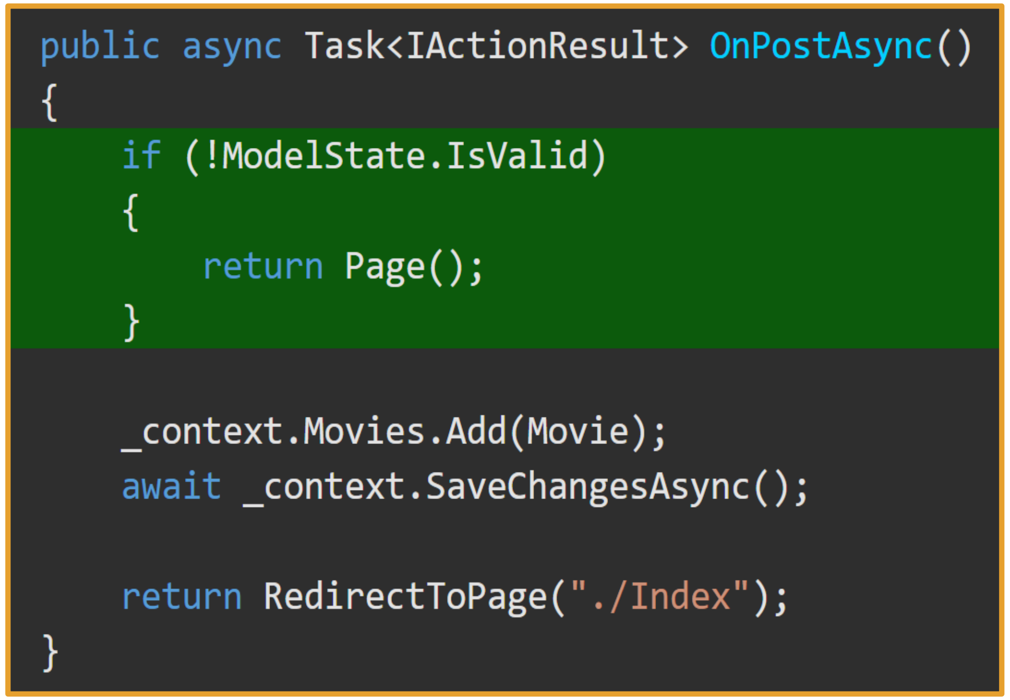
Model Binding errors are generally data conversion errors.

- Ex. A string is entered in an integer field.

Model validation occurs after ***Model Binding*** and reports errors when data doesn't conform to business rules.

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

A good way to prevent ***Model Binding*** errors is to use data annotations on the ***Model***.



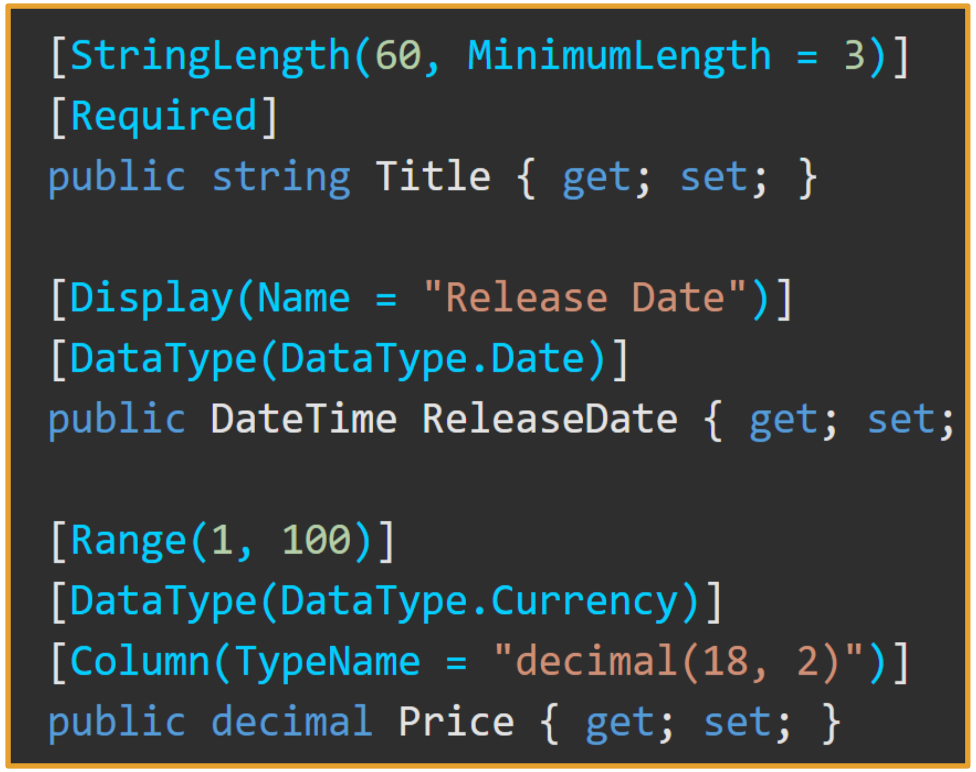
```
public async Task<IActionResult> OnPostAsync()
{
    if (!ModelState.IsValid)
    {
        return Page();
    }

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

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

Data Annotations – Overview

https://docs.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/models-data/validation-with-the-data-annotation-validators-cs

The *DataAnnotations* namespace provides a set of built-in ***validation attributes*** that are applied declaratively to a ***class*** or ***property***.

DataAnnotations also contains formatting ***attributes*** like *DataType* that help with formatting but don't provide ***validation***.



```
[StringLength(60, MinimumLength = 3)]
[Required]
public string Title { get; set; }

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

[Range(1, 100)]
[DataType(DataType.Currency)]
[Column(TypeName = "decimal(18, 2)")]
public decimal Price { get; set; }
```

Validation – Client-Side

https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-validation

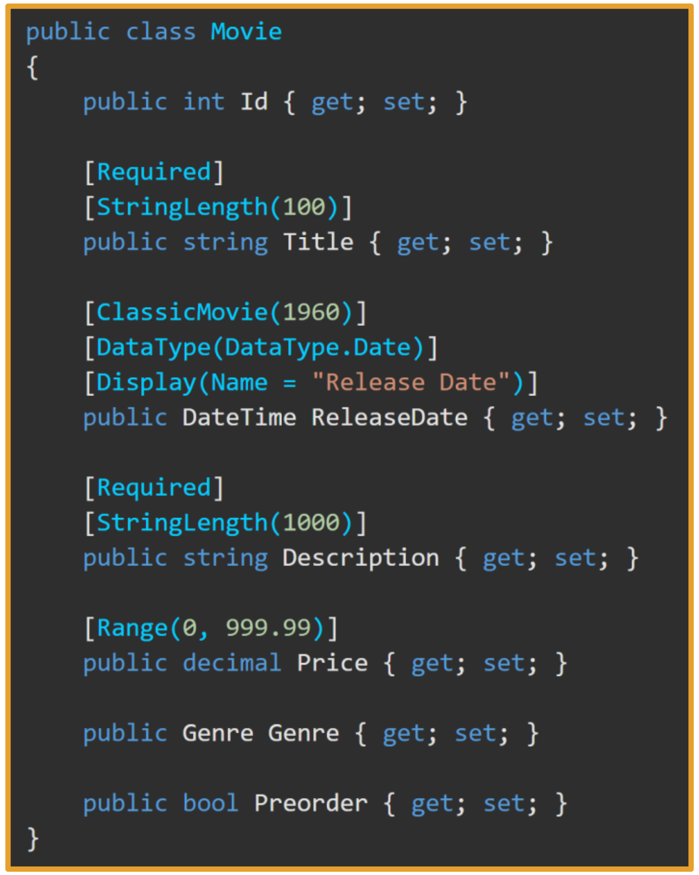
Annotation	Explanation
[StringLength]	Validates a string property value doesn't exceed a specified length limit.
[Url]	Validates the property has a URL format.
[Remote]	Validates input on the client by calling an action method on the server.
[MinLength(x)]	Minimum length is x. Also sets DB column size min.
[MaxLength(y, ErrorMessage="This is required")]	Maximum length is y and this error message is displayed. Also sets DB column size max.

*Custom createdvalidation is also

Validation – Client-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#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.

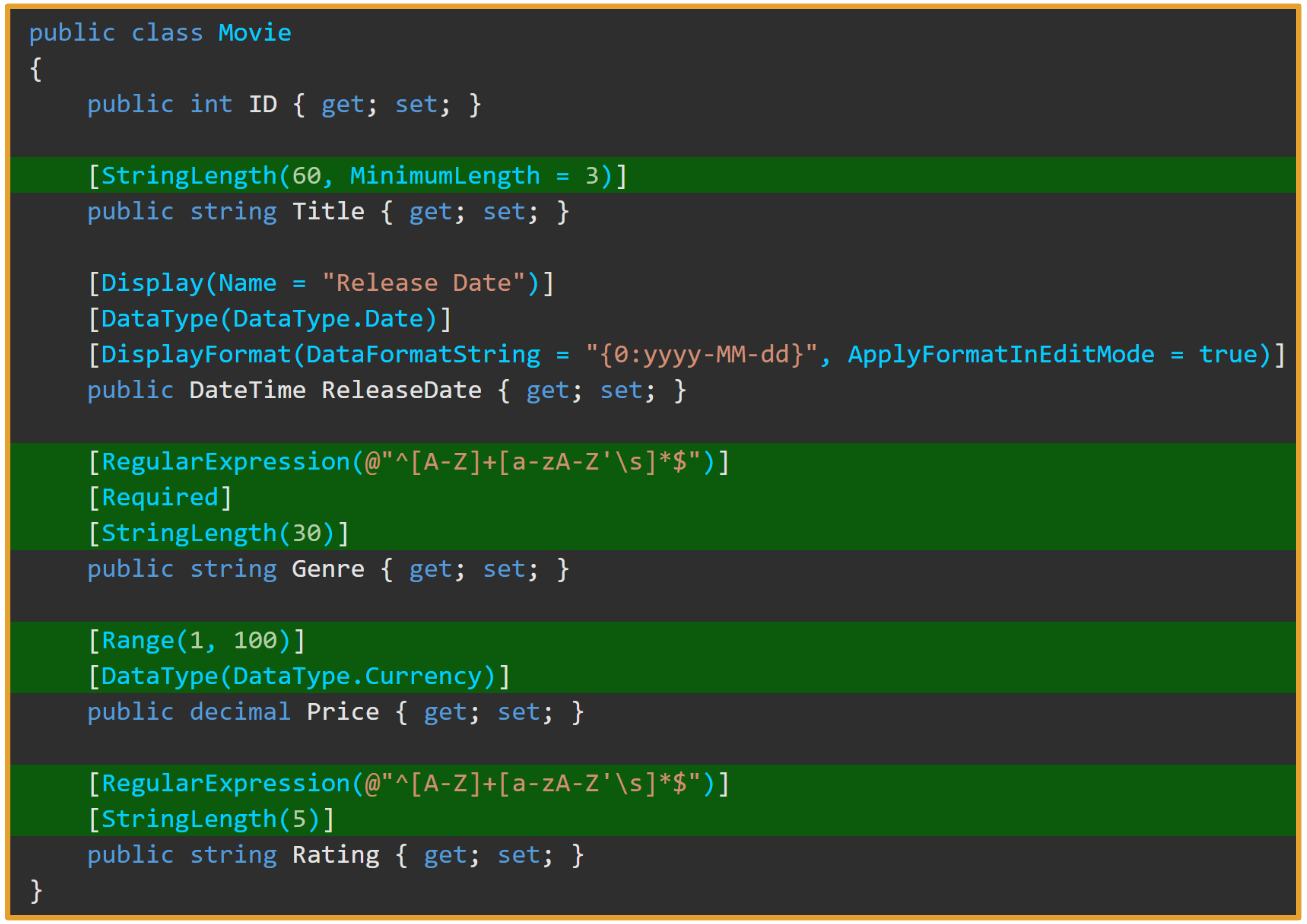


```
public class Movie
{
    public int Id { get; set; }

    [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.

Data Annotations – Examples

https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-validation

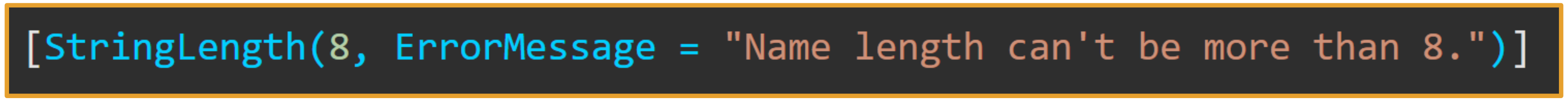


```
public class Movie
{
    public int ID { get; set; }

    [StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
    public DateTime ReleaseDate { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z'\s]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z'\s]*$")]
    [StringLength(5)]
    public string Rating { get; set; }
}
```

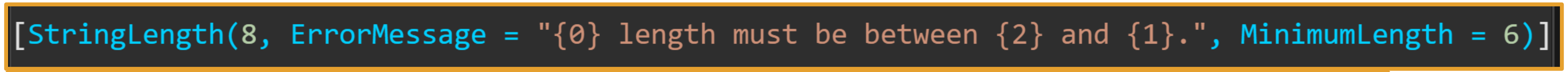
Validation – Client-Side Error Messages

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#error-messages

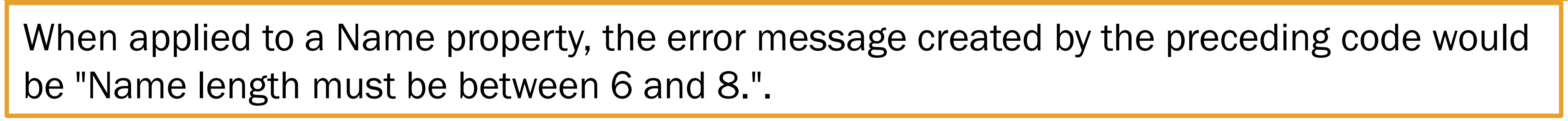
Error messages can be displayed on a web page for the user to see.



```
[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 – [Required] Server-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#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 the ***SuppressImplicitRequiredAttributeForNonNullableReferenceTypes*** property of the options object in Startup.ConfigureServices() to true***.***

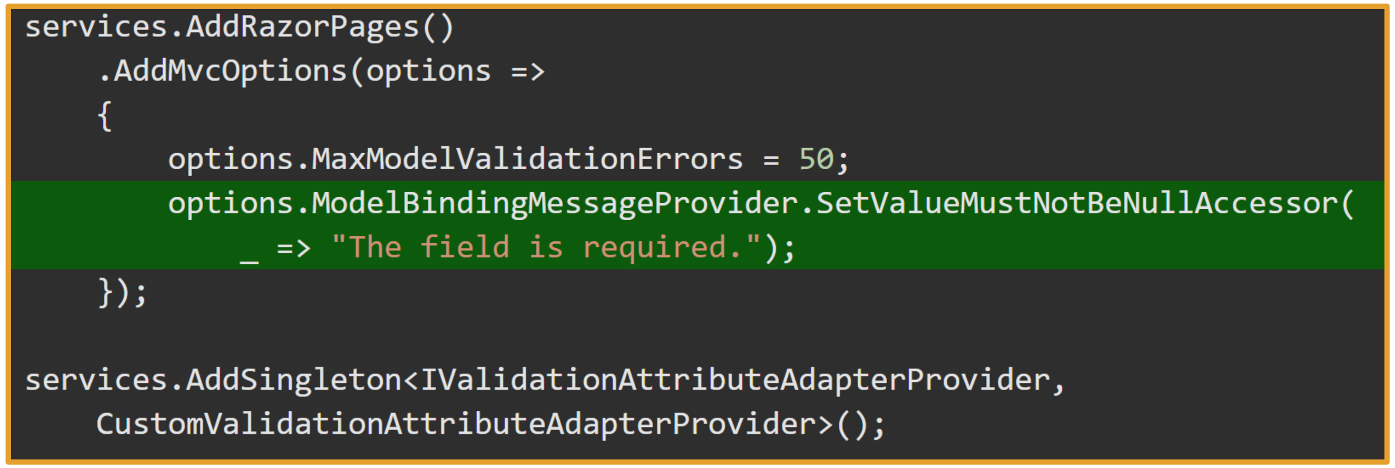
```
services.AddControllers(options =>
options.SuppressImplicitRequiredAttributeForNonNullableReferenceTypes = true);
```

Validation – [Required] Server-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#required-validation-on-the-server

Model Binding for a non-nullable ***Property*** can sometimes FAIL. This 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 counted as valid, server-side. This means the [Required] attribute's error message is never displayed on ***non-nullable*** fields when this error occurs.
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).
- Specify the default error message to be used by ***Model Binding***. (not recommended)



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

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

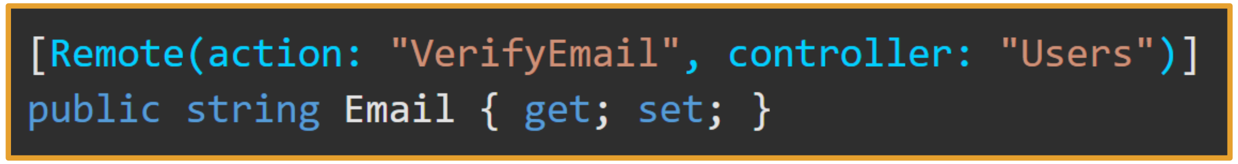
Validation – [Remote] Server-Side

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#remote-attribute

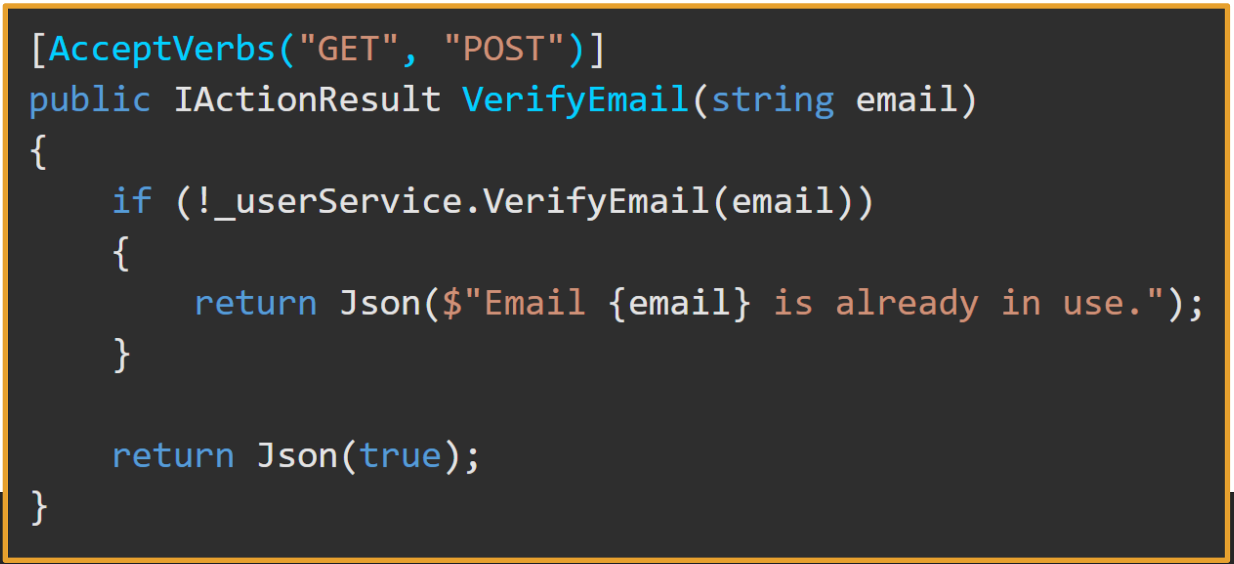
The [Remote] attribute implements client-side validation that requires calling an *Action* method on the server to determine whether field input is valid. For example, the app may need to verify whether a userName is already in use.

To do this, 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

Custom Data Annotations

https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-5.0#custom-attributes

Create custom validation attributes.

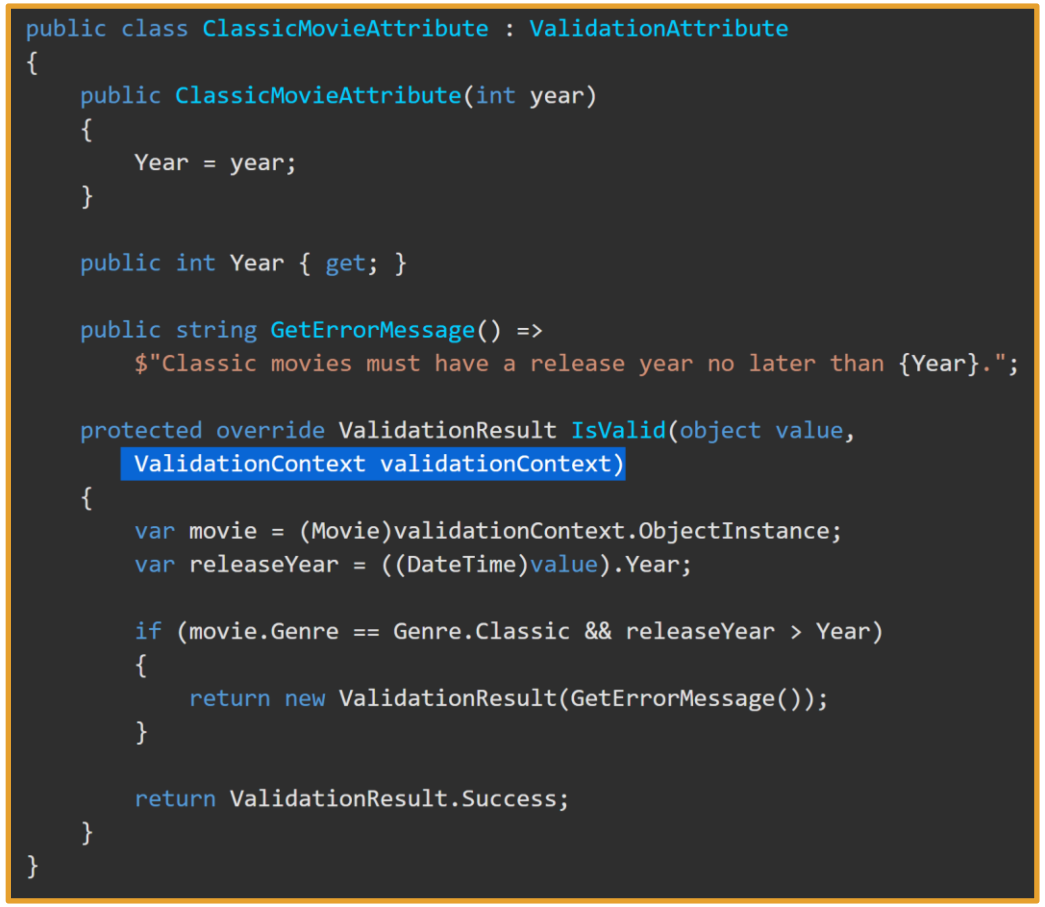
1) Create a class that inherits from ***ValidationAttribute*** and contains the data to be validated against as a property.

2) Override IsValid() of ***ValidationAttribute***.

- IsValid() accepts an object, which is the input to be validated.
- An overload of IsValid() also accepts a ***ValidationContext*** object, which provides additional information, like 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*** syntax on the ***Model*** is:
[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;
    }
}
```

EF Code-First Data Annotations

https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/data-annotations#the-model

Data Annotation	Explanation
[Key]	Annotates a property as the Key of the entity .
[Column(order=2)]	Used with [Key] to create a composite column. This will be the 2nd key.
[ForeignKey(“FK_ModelName")]	Marks a certain model as the FK for this model.
[Required]	This property will be required in the Db and client-side.
[NotMapped]	This property will not be mapped to the Db.
[ComplexType]	This annotation is placed on a subtype of a model to alert EF that the property on the model has properties of it own.
[ConcurrencyCheck]	Checks of changes between .SaveChanges() calls.
[Table(“TableName")]	Placed above the Model Class name. Allows you to change the name of the table in the Db.
[Column(“ColumnName”)]	Allows you to name a column other than the property name.

*There are many more available