Validation Attributes

Model state represents errors that come from two subsystems: model binding and model validation. Errors that originate from model binding are generally data conversion errors. For example, 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. For example, a 0 is entered in a field that expects a rating between 1 and 5.

Here are some of the built-in validation attributes:

[CreditCard]
Validates that the property has a credit card format. Requires jQuery Validation Additional Methods.

[Compare]
Validates that two properties in a model match.

[EmailAddress]
Validates that the property has an email format.

[Phone]
Validates that the property has a telephone number format.

[Range]
Validates that the property value falls within a specified range.

[RegularExpression]
Validates that the property value matches a specified regular expression.

[HiddenInput(DisplayValue = true)]
Instantiates a new instance of the HiddenInputAttribute class.
Gets or sets a value indicating whether to display the value as well as provide a hidden <input> element. The default value is true.

[Required]
Validates that the field is not null. See [Required] attribute for details about this attribute's behavior.

[StringLength]
Validates that a string property value doesn't exceed a specified length limit.

[Url]
Validates that the property has a URL format.

[Remote]
Validates input on the client by calling an action method on the server.
See [Remote] attribute for details about this attribute's behavior.

[ClassicMovie(1960)]
Custom attributes

[DataType(DataType.Password)]
Mask the filed with dots

[ClassicMovie(1960)]

```
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;
    }
}
```

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

To implement remote validation:

1. Create an action method for JavaScript to call. The jQuery Validation remote method expects a JSON response:
   - true means the input data is valid.
   - false, undefined, or null means the input is invalid. Display the default error message.
   - Any other string means the input is invalid. Display the string as a custom error message.

   Here's an example of an action method that returns a custom error message:

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

      return Json(true);
   }
   ```

2. In the model class, annotate the property with a [Remote] attribute that points to the validation action method, as shown in the following example:

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

Authorization

[Authorize]
is used just to authenticate the user.Authentication is the process of determining a user's identityschemes should be used to authenticate the user
`[Authorize(AuthenticationSchemes = AuthSchemes)]`

Database

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
applied to primary key

public int AuthorFK { get; set; }
[ForeignKey("AuthorFK")]
public Author Author { get; set; }
foe the navigation prop Author a foreign key will be AuthorFK

[Table("MyAccountsTable")]
applied to class

[Key]
applied to primary key

Binding
[Bind]
Can be applied to a class or a method parameter. Specifies which properties of a model should be included in model binding. `[Bind]`.in the following example, only the specified properties of the `Instructor` model are bound when any handler or action method is called:

```
 [Bind("LastName,FirstMidName,HireDate")]
public class Instructor
```

In the following example, only the specified properties of the `Instructor` model are bound when the `OnPost` method is called:

```
[HttpPost]
public IActionResult OnPost([Bind("LastName,FirstMidName,HireDate")] Instructor
instructor)
```

or

```
[Bind(nameof(UserModel.Name))] UserModel model
```

The `[Bind]` attribute can be used to protect against overposting in *create* scenarios. It doesn't work well in edit scenarios because excluded properties are set to null or a default value instead of being left unchanged. For defense against overposting, view models are recommended rather than the `[Bind]` attribute.

[BindRequired]

Can only be applied to model properties, not to method parameters. Causes model binding to add a model state error if binding cannot occur for a model's property.

[BindNever]

Can only be applied to model properties, not to method parameters. Prevents model binding from setting a model's property.

[FromBody] annotation "overrides" the BindBehaviourAttribute (BindNever is a simple specialization). The model is populated by all data available from the body (your JSON data in this case).