IValidatableObject in C#

IValidatableObject is an interface in the System.ComponentModel.DataAnnotations namespace. It contains a single method, Validate, which allows us to implement custom validation logic in our models.

Usage Scenarios: IValidatableObject is particularly useful when we need to validate interdependent properties or when the validation process involves calling external services.

Let's dive in!

@lijotech

Example-1

Simple request model implementing IValidatableObject

```
public class Product : IValidatableObject
{
    public string Name { get; set; }
    public DateTime StartDate { get; set; }
    public DateTime EndDate { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        if (EndDate <= StartDate)
        {
            yield return new ValidationResult(
                "EndDate must be greater than StartDate",
                new[] { nameof(EndDate) });
        }
    }
}
```

In this example, the **Product** class implements **IValidatableObject** and provides a Validate method. Inside this method, we check if **EndDate** is later than **StartDate**.
This approach allows us to implement complex validation logic that can't be expressed with simple attribute-based validation.

@lijotech

Example-2

IValidatableObject inside a list

```
public class Product
{
    public string Name { get; set; }
    public decimal Price { get; set; }
}

public class Order : IValidatableObject
{
    public List<Product> Products { get; set; }
    public decimal TotalPriceLimit { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        if (Products.Sum(p => p.Price) > TotalPriceLimit)
        {
            yield return new ValidationResult(
                $"The total price of all products in the order cannot exceed {TotalPriceLimit}",
                new[] { nameof(Products) });
        }
    }
}
```

In this example, the **Order** class implements **IValidatableObject** and provides a **Validate** method. Inside this method, we calculate the total price of all products in the order and check if it exceeds the **TotalPriceLimit**. If it does, we yield a **ValidationResult** indicating the error.
This is a type of validation that cannot be achieved using simple data annotations.

Example-3

IValidatable using external service

We have a **User** model and we want to ensure that the email of a user is unique in the database. Inside this method, we check if the email already exists in the database using the **EmailExists** method of the **IUserService**. This approach allows us to implement complex validation logic that can't be expressed with simple attribute-based validation.

```
public class User : IValidatableObject
{
    public string Email { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        var userService = (IUserService)validationContext.GetService(typeof(IUserService));

        if (userService.EmailExists(Email))
        {
            yield return new ValidationResult("Email already exists in the database",
                new[] { nameof(Email) });
        }
    }
}
```

@lijotech

Evaluate IValidatableObject

Benefits of IValidatableObject:

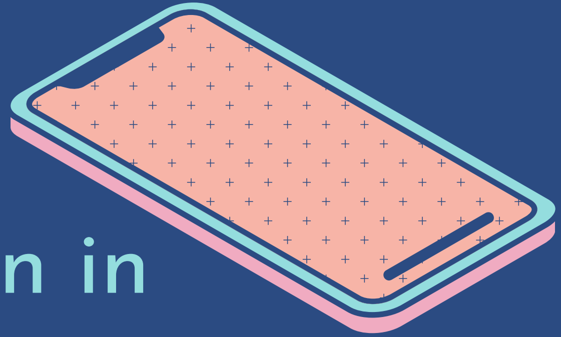
- Flexibility: It allows us to define complex validation rules that can't be expressed with simple attribute based validation.
- Control: We have full control over the validation process and can access all properties of the model.

Drawbacks of IValidatableObject:

- Complexity: It can make the model classes more complex, especially if the validation logic is complicated.
- Maintenance: If the validation rules change frequently, maintaining the code can be challenging.

@lijotech

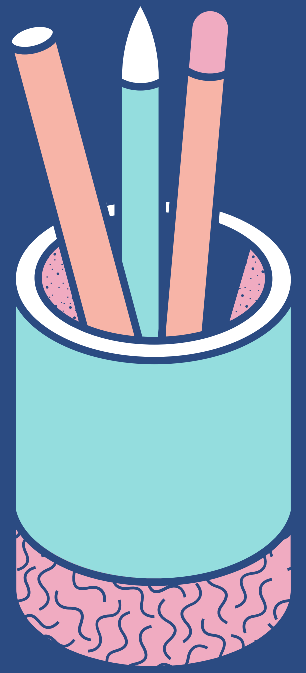
Compare with FluentValidation



Both IValidatableObject and FluentValidation are used for model validation in .NET, but they have some differences in terms of features, flexibility, and usage. Here's a simple comparison:

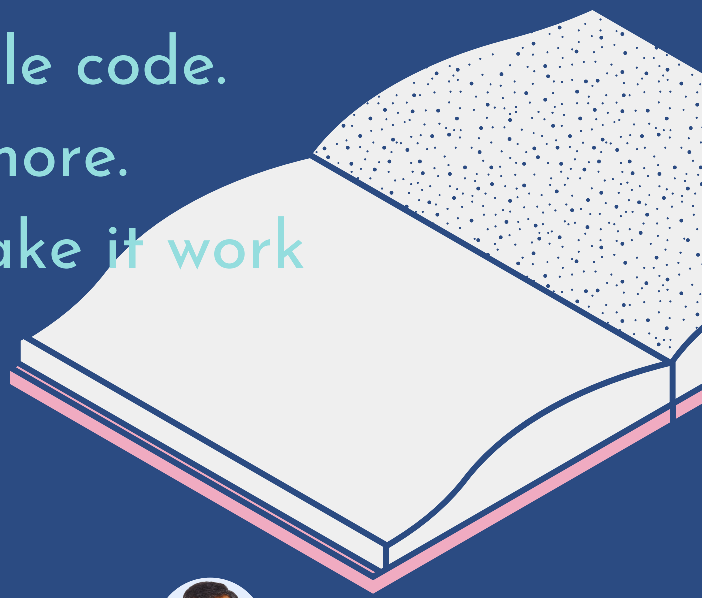
IValidatableObject:

- It's a built-in interface in .NET for model validation.
- It provides a Validate method that you can override to implement custom validation logic.
- It's directly integrated with the framework, so it works well with model binding and ModelState.
- It's great for simple scenarios, but for complex validation rules, the code can get messy and hard to manage.



FluentValidation:

- It's a third-party library that provides a fluent interface and lambda expressions for building validation rules.
- It allows you to create separate validator classes, which leads to cleaner and more maintainable code.
- It provides more advanced features, such as conditional validation, collection validation, and more.
- It's not directly integrated with the framework, so you need to set up some configuration to make it work with model binding and ModelState.



In conclusion, **IValidatableObject** is a good choice for simple validation scenarios, while FluentValidation is more powerful and flexible for complex validation scenarios.

@lijotech

Stay Focused & Happy Coding

Follow Lijo Sebastian for more coding tips

@lijotech