Diving Deeper into Fluent Validation



Vladimir Khorikov

@vkhorikov www.enterprisecraftsmanship.com



Introduction

Conditional validation

Cascade validation modes

Overriding error messages

Integrating with the ASP.NET pipeline



Conditional Validation



Validating multiple properties



Conditional Validation

Contact method

Phone number

Email address



Can indicate just one method



Recap: Conditional Validation



Conditions within the rule chain

```
RuleFor(x => x.Email)
    .NotEmpty()
    .Length(0, 150)
    .EmailAddress()
    .When(x => x.Email != null);

    Applies to all
    preceding checks
```

previous check

Recap: Conditional Validation



Conditions that group multiple rule chains

```
When(x => x.Email == null, () =>
{
         RuleFor(x => x.Phone).NotEmpty();
});
When(x => x.Phone == null, () =>
{
         RuleFor(x => x.Email).NotEmpty();
});

RuleFor(x => x.Phone).NotEmpty().When(x => x.Email == null);
RuleFor(x => x.Email).NotEmpty().When(x => x.Phone == null);
```





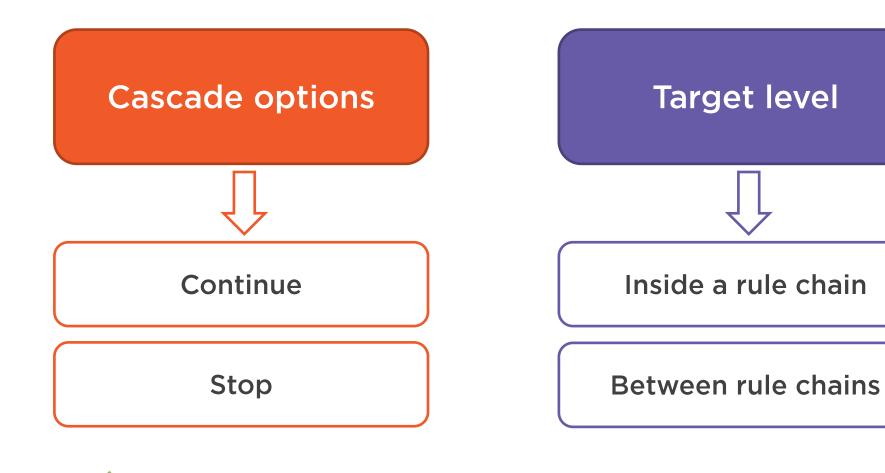
Cascade modes





Cascade mode controls validation flow





Continue is the default



```
RuleFor(x => x.Email).Cascade(CascadeMode.Stop).NotEmpty().Length(1, 150);
Stops validation inside the rule chain
```

Stops validation **both** inside and between rule chains

```
CascadeMode = CascadeMode.Stop;
RuleFor(x => x.Email).NotEmpty().Length(0, 150).EmailAddress();
RuleFor(x => x.Phone).NotEmpty().Matches("^[2-9][0-9]{9}$");
```



ValidatorOptions.Global.CascadeMode = CascadeMode.Stop;



Configures the setting for all validators





integrated FluentValidation into ASP.NET



```
Step 1: look at the
[HttpPost]
public IActionResult Register(RegisterRequest request)
                                                           data contract
                                                           Step 2: find the
              public class RegisterRequestValidator :
                  AbstractValidator<RegisterRequest>
                                                           validator
                              if (!ModelState.IsValid)
                                                           Step 3: populate
                                                           the model state
```





You can only have one validator per data contract

public class RegisterRequestValidator : AbstractValidator<RegisterRequest>

public class RegisterRequestValidator2 : AbstractValidator<RegisterRequest>



```
[ApiController]
public class ApplicationController : ControllerBase
{
}
```



The model state is checked automatically



No need for the FromBody attribute





Custom validation rules



Recap: Custom Validation Rules

```
public static IRuleBuilderOptionsConditions<T, IList<TElement>> ListMustContainNumberOfItems(
    this IRuleBuilder<T, IList<TElement>> ruleBuilder, int? min = null, int? max = null)
    return ruleBuilder.Custom((list, context) => {
        if (min.HasValue && list.Count < min.Value) {</pre>
            context.AddFailure(
                $"The list must contain {min.Value} items or more. It contains {list.Count} items.");
        if (max.HasValue && list.Count > max.Value) {
            context.AddFailure(
                $"The list must contain {max.Value} items or fewer. It contains {list.Count} items.");
    });
```



Custom method allows for more granular control



Recap: Custom Validation Rules

Custom validation rules



ListMustContain NumberOfItems()

Reusing a single rule in a chain

Custom validator classes

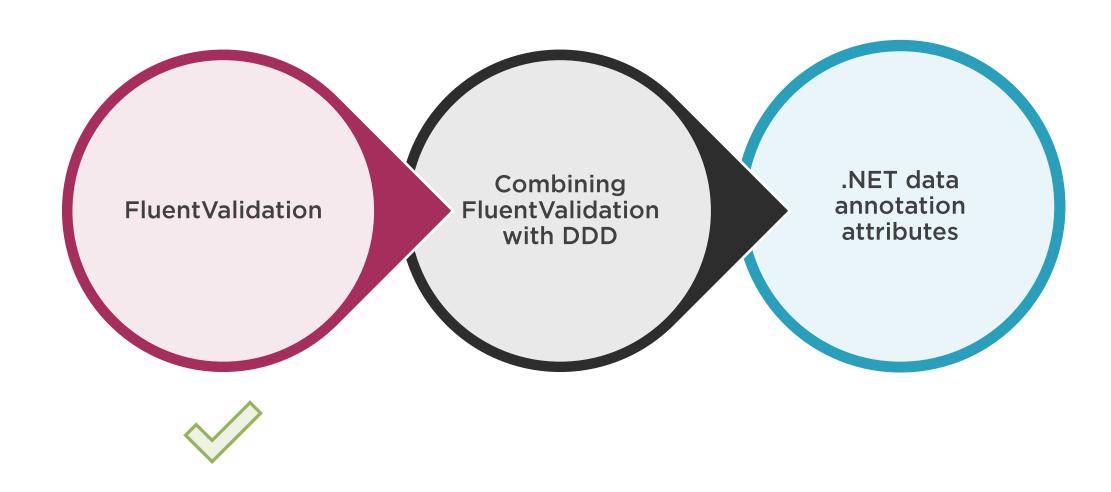


AddressesValidator

Reusing rules for the whole chain



Validation





Summary



Advanced features of the FluentValidation library

- Conditional validation
- Cascade modes

Integrating FluentValidation with ASP.NET

- Used the standard ModelState property
- Automated model state checks

Implemented custom validation rules



In the Next Module

Validating Input the DDD Way

