Leveraging the Strategy Pattern to Encapsulate Business Logic

Benjamin Day
TRAINER | COACH | DEVELOPER

@benday www.benday.com

Overview

Strategy Pattern

Validation Logic

Calculation Logic

System.ComponentModel.DataAnnotations

First up:
Validation in ASP.NET Core

Validation

How do you figure out if your data is good?

Validation in ASP.NET Core

```
public class MyController
{
    [HttpPost]
    public IActionResult Save(PersonModel model)
    {
        if (ModelState.IsValid == true)
        {
            Save(model);

            return RedirectToAction("Index");
        }
        else
        {
            return new BadRequestResult();
        }
    }
}
```

ModelState.IsValid

```
public class MyController
{
    [HttpPost]
    public IActionResult Save(PersonModel model)
    {
        if (ModelState.IsValid == true)
        {
            Save(model);

            return RedirectToAction("Index");
        }
        else
        {
            return new BadRequestResult();
        }
    }
}
```

It's doing two things

Was the model binding successful?

Is the model data valid?

System.ComponentModel.DataAnnotations

Unit Testing ModelState.IsValid Is Hard

```
public class MyController
{
    [HttpPost]
    public IActionResult Save(PersonModel model)
    {
        if (ModelState.IsValid == true)
        {
            Save(model);

            return RedirectToAction("Index");
        }
        else
        {
            return new BadRequestResult();
        }
    }
}
```

Hard / impossible to unit test

- Integration test: do-able
- Unit test: nearly impossible

ModelState requires an ASP.NET Core Integration Test

- WebApplicationFactory

Unit test → test in isolation

Single Responsibility Principle violation

Design for Testability

Strategy Pattern

Strategy Pattern

Super simple

Encapsulates an algorithm

Validation Logic

Calculation Logic

ModelState.IsValid → Strategy

```
public class MyController
{
    [HttpPost]
    public IActionResult Save(PersonModel model)
    {
        if (ModelState.IsValid == true)
        {
            Save(model);

            return RedirectToAction("Index");
        }
        else
        {
            return new BadRequestResult();
        }
    }
}
```

Create a Strategy class

- PersonValidatorStrategy

Create a Strategy interface

- IPersonValidatorStrategy

Put the validation logic in the Strategy class

Use Dependency Injection to access validation logic

Unit test the Strategy class

Next up:
How does validation work in ASP.NET Core?

How does validation work in ASP.NET Core?

You've Got Options

Write it yourself

- 100% custom
- Re-invent the wheel

System.ComponentModel.DataAnnotations

- Nice extra stuff
- HTML View Helpers
- Client-side validations

Data Annotations

Describe how to validate your objects

- Attributes
- Custom Attributes
- IValidatableObject

Validator

- Validates the objects

Data Annotation Attributes

[Required]

[StringLength]

[MaxLength] / [MinLength]

[EmailAddress]

[Phone]

[CreditCard]

[Compare]

[Range]

[RegularExpression]

Custom Attribute

Create your own

Create a class that extends from ValidationAttribute

Implement IsValid()

IValidatableObject

Implement IValidatableObject on your class

Validate(ValidationContext) method

Your Validation Strategy Implementation

Wrap a call to Validator.Validate()

Next up:
Code Demos

Demo

Implement validation using Data Annotation Attributes

Unit tests

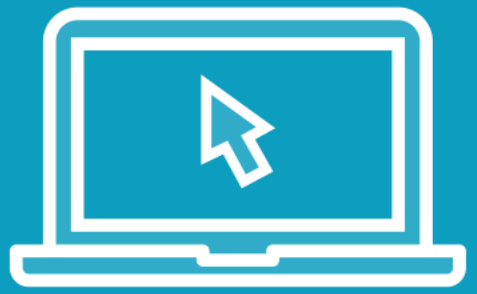
IValidatorStrategy<T>

DefaultValidatorStrategy<T>

Hook it in to a Controller using Dependency Injection

Next up:
Implement Validation using a Custom Attribute

Demo



Implement validation using a custom annotation attribute

Unit tests

IValidatorStrategy<T>

DefaultValidatorStrategy<T>

Next up:
Implement Validation using IValidatableObject

Demo

Use IValidatableObject to create validation logic

Unit tests

IValidatorStrategy<T>

DefaultValidatorStrategy<T>

Next up:
Implement a calculation using the Strategy Pattern

Demo

US President

Calculate the # of days in office

IDaysInOfficeCalculationStrategy

DaysInOfficeCalculationStrategy

Summary

Strategy Pattern

Validation Logic

Calculation Logic

System.ComponentModel.DataAnnotations

Next up:
Testing Routing Logic