Валидация в DDD

Райффайзен

Обо мне

Константин Густов

архитектор,
Райффайзенбанк

10+ лет опыта в разработке

konst.gustov@gmail.com

Темы

I. Проблема валидации

Темы

I. Проблема валидации

II. Валидация в DDD

Темы

I. Проблема валидации

II. Валидация в DDD

III. Фреймворки валидации

Темы

I. Проблема валидации

II. Валидация в DDD

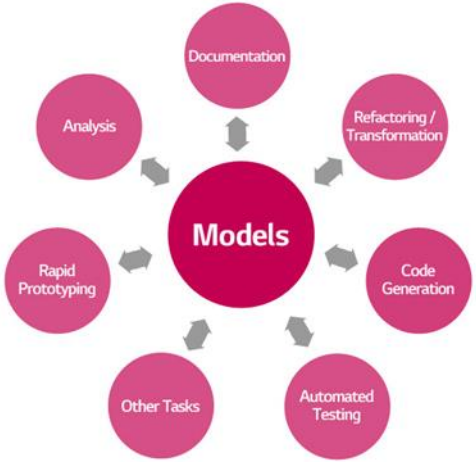
III. Фреймворки валидации

IV. Собственный фреймворк

Валидация данных

Процесс обеспечения чистоты и качества данных, чтобы в результате они были правильными и полезными

Domain-driven design



Виды валидации

- Простые проверки
  - Типы данных
  - Диапазоны
  - Целостность данных группы полей
  - И т.д.

Виды валидации

- Простые проверки
  - Типы данных
  - Диапазоны
  - Целостность данных группы полей
  - И т.д.
- Структурные проверки

Виды валидации

- Простые проверки
  - Типы данных
  - Диапазоны
  - Целостность данных группы полей
  - И т.д.
- Структурные проверки
- Бизнес-правила

Проблема валидации

```
public void BlockCard(int cardId, BlockDto blockDto)
{
  var repository = _entityRepositoryFactory.Create<Card>();
  var card = repository.Get(cardId);
  if (card == null)
    throw new InvalidOperationException("Card does not exist");

  _accountsService.LockAccount(card);
  card.BlockCard(blockDto.ToDetails());
  ...
}
```

Проблема валидации

```
public void BlockCard(int cardId, BlockDto blockDto)
{
  if (cardId <= 0)
    throw new ArgumentException("Card id must be greater than zero");
  if (blockDto == null);
    throw new ArgumentNullException("Block details must not be null");

  var repository = _entityRepositoryFactory.Create<Card>();
  var card = repository.Get(cardId);
  if (card == null)
    throw new InvalidOperationException("Card does not exist");

  _accountsService.LockAccount(card);
  card.BlockCard(blockDto.ToDetails());
  ...
}
```

Проблема валидации

```
public void BlockCard(int cardId, BlockDto blockDto)
{
  if (cardId <= 0)
    throw new ArgumentException("Card id must be greater than zero");
  if (blockDto == null);
    throw new ArgumentNullException("Block details must not be null");

  var repository = _entityRepositoryFactory.Create<Card>();
  var card = repository.Get(cardId);
  if (card == null)
    throw new InvalidOperationException("Card does not exist");

  if (card.IsInstant && blockDto.HasActiveAccounts)
    throw new InvalidOperationException("Card cannot be blocked");

  _accountsService.LockAccount(card);
  card.BlockCard(blockDto.ToDetails());
  ...
}
```

Проблема валидации

Проверки переплетаются с исполняемым кодом

```
if (!card.CardNumber.EndsWith(request.CardNumber))
  return new Result("Incorrect card number");

if (request.AppId != null)
{
  switch (card.Status)
  {
    case CardStatus.Opened:
      _cardIssueService.IssueCard(request, card);
    default:
      return new Result("Cannot issue card from {0} status", card.Status);
  }
}
```

Недостатки

Невозможно повторное использование проверок

Недостатки

Результат выполнения проверок не стандартизирован

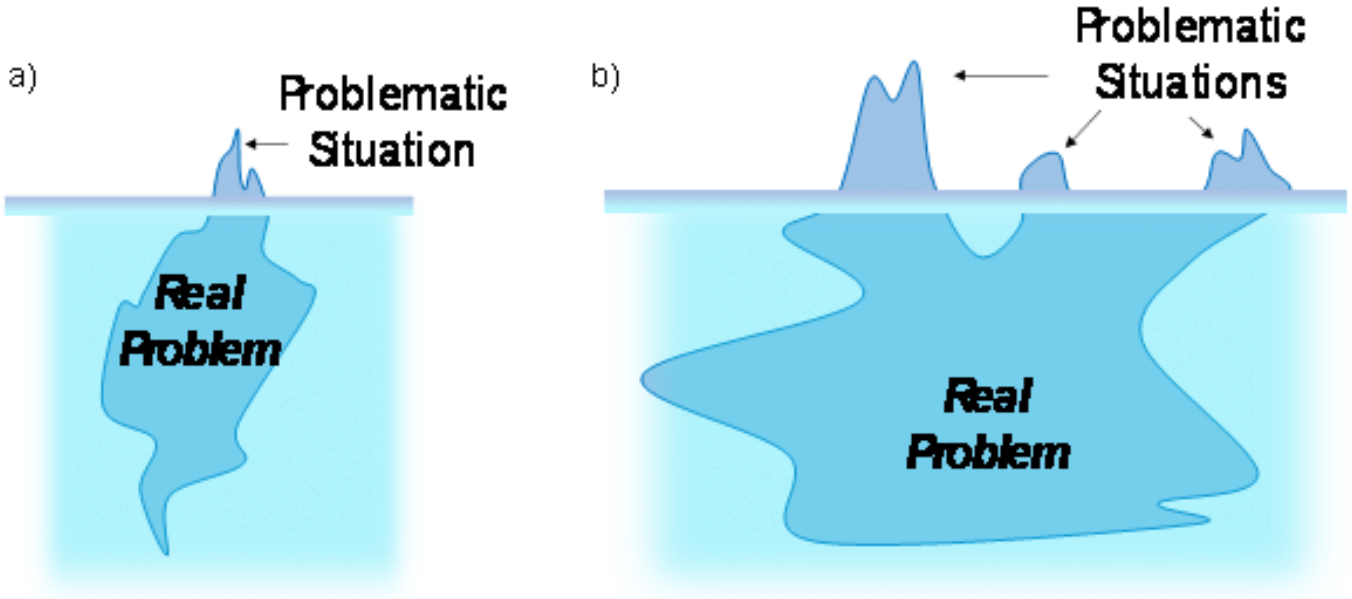
```
if (cardId <= 0)
  throw new ArgumentException("Card id must be greater than zero");
if (blockDto == null);
  throw new ArgumentNullException("Block details must not be null");

...

if (card.IsInstant && blockDto.HasActiveAccounts)
  throw new InvalidOperationException("Card cannot be blocked");

_accountsService.LockAccount(card);
card.BlockCard(blockDto.ToDetails());
...
```

Недостатки

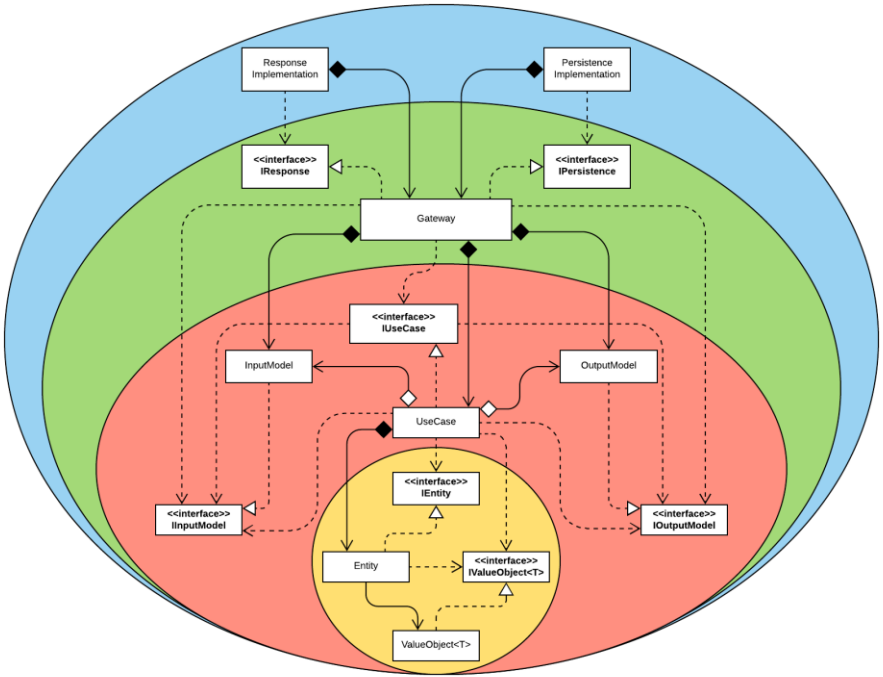
Валидация недостаточно надежна

Недостатки

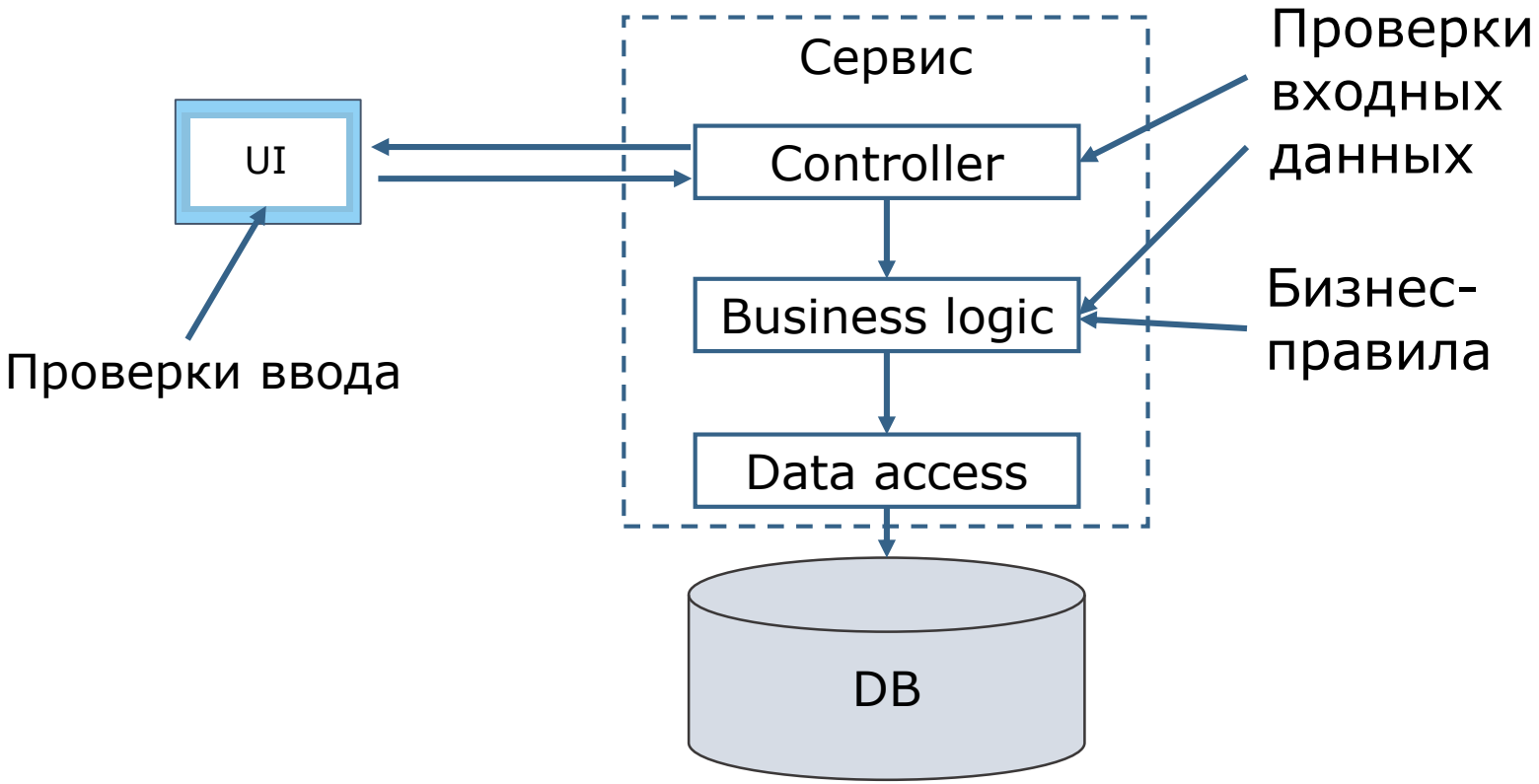
Валидация показывает первую возникшую проблему



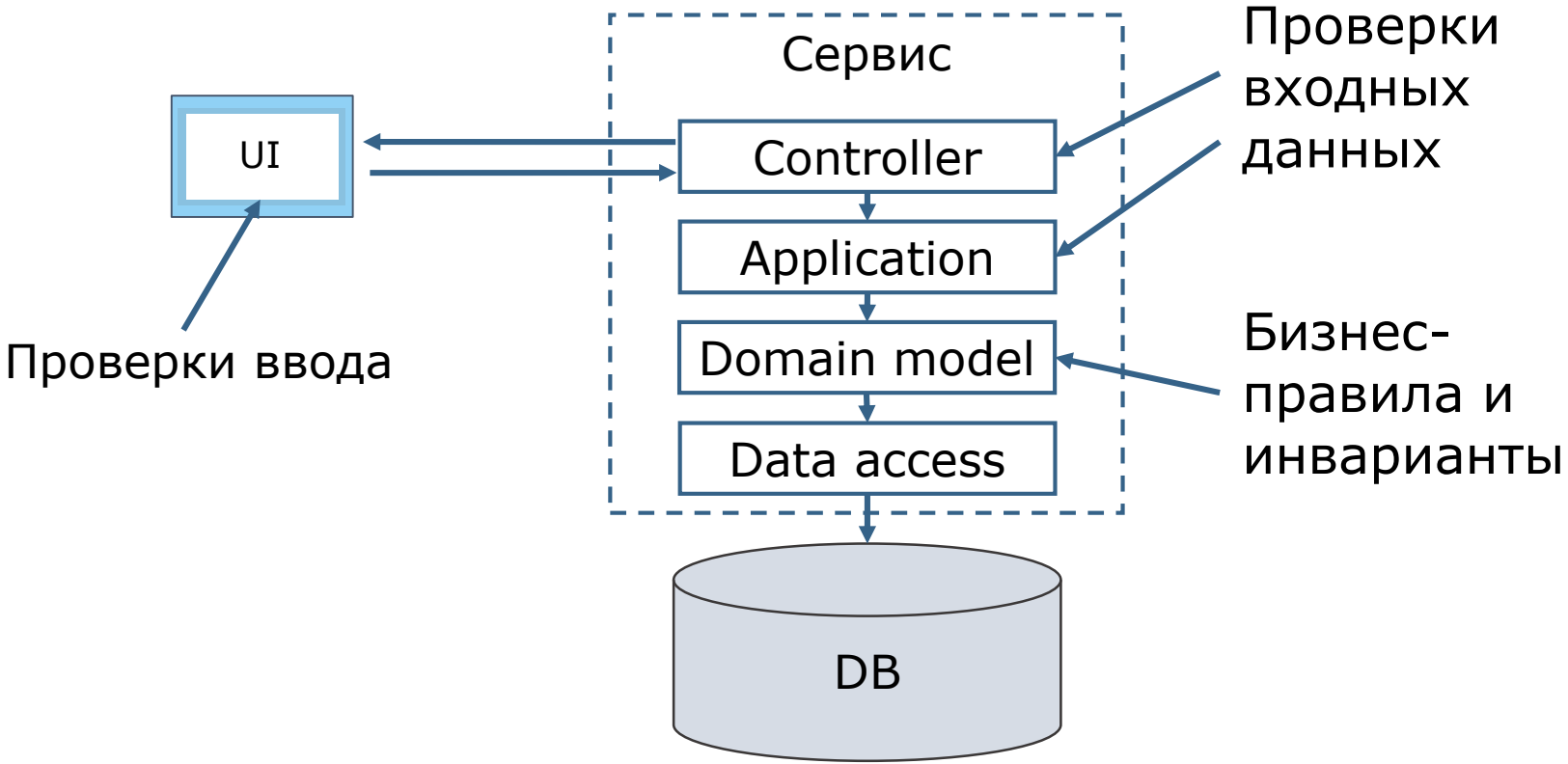
Валидация в DDD



Валидация в клиент-серверном приложении



Валидация в domain-driven design



Always-valid entity

```
public class Card : IEntity<int>, IEventProvider
{
  private IEventCollector _eventCollector;

  BlockType BlockType { get; protected virtual set; }

  ...
  public virtual void BlockCard([NotNull] BlockDetails blockDetails)
  {
    if (blockDetails.BlockType.NotIn(BlockType.Partial, BlockType.Full))
    {
      throw new ArgumentException("Block type is not correct");
    }
    BlockType = blockDetails.BlockType;
  }
}
```

Always-valid entity

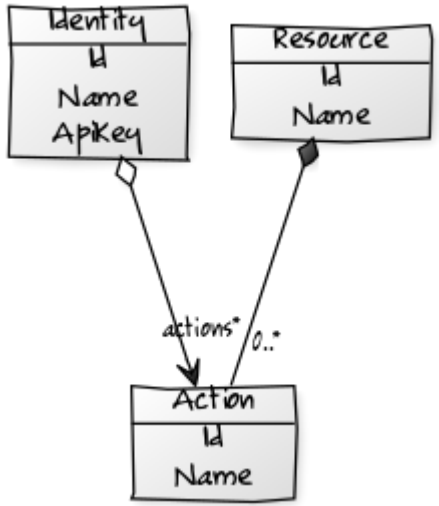
```
public class Card : IEntity<int>, IEventProvider
{
  BlockType BlockType { get; protected virtual set; }

  public IReadOnlyCollection<string> BlockCard(BlockDetails blockDetails)
  {
    var errors = new List<string>();
    if (BlockType.NotIn(BlockType.Partial, BlockType.Full))
    {
      errors.Add("Block type is not correct");
      return errors;
    }
    BlockType = blockDetails.BlockType;
    ...
  }
}
```

Always-valid entity

```
public class Card : IEntity<int>, IEventProvider
{
  BlockType BlockType { get; protected virtual set; }

  public IReadOnlyCollection<string> CanAddBlock()
  {
    var errors = new List<string>();
    if (BlockType.NotIn(BlockType.Partial, BlockType.Full))
      errors.Add("Block type is not correct");
    ...
    return errors;
  }
  public virtual void BlockCard([NotNull] BlockDetails blockDetails)
  {
    if (CanAddBlock().Any())
      throw new InvalidOperationException();
    BlockType = blockDetails.BlockType;
    ...
  }
}
```

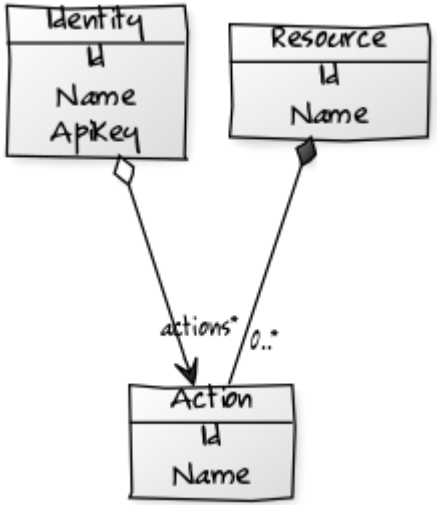
Преимущества

- Сущность сохраняет инвариант



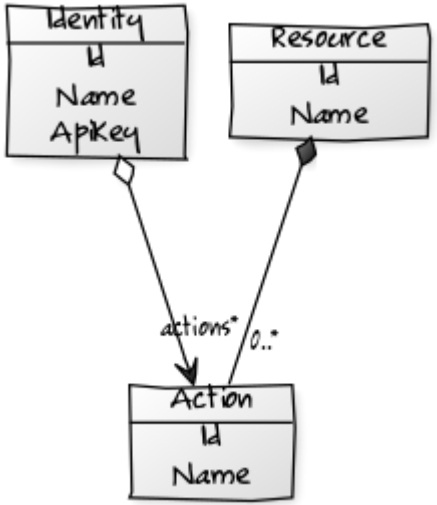
Преимущества

- Сущность сохраняет инвариант
- Невозможно забыть вызвать правило в сценариях использования



Преимущества

- Сущность сохраняет инвариант
- Невозможно забыть вызвать правило в сценариях использования
- Правила валидации находятся максимально близко к основной логике



Недостатки от Джефа Палермо

- Валидация может быть актуальна не во всех сценариях

Недостатки от Джефа Палермо

- Валидация может быть актуальна не во всех сценариях
- Текст сообщения – ответственность слоя представления

Недостатки от Джефа Палермо

- Валидация может быть актуальна не во всех сценариях
- Текст сообщения – ответственность слоя представления
- Проблемы при загрузке исторических данных

Валидация по требованию

```
public class Card : IEntity<int>, IEventProvider
{
  BlockType BlockType { get; protected virtual set; }

  public IReadOnlyCollection<string> ValidateForStandardBlock()
  {
    var errors = new List<string>();
    if (BlockType.NotIn(BlockType.Partial, BlockType.Full))
    {
      errors.Add("Block type is not correct");
    }
    return errors;
  }
  public virtual void BlockCard([NotNull] BlockDetails blockDetails)
  {
    BlockType = blockDetails.BlockType;
    ...
  }
}
```

Спецификация

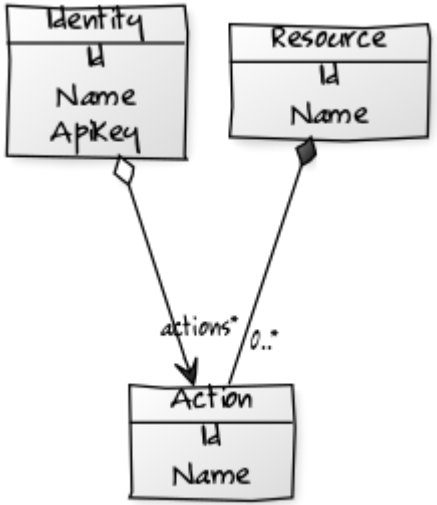
```
public abstract class Specification<T> : ISpecification<T>
{
  protected Expression<Func<T, bool>> expression = null;
  public bool IsSatisfiedBy(T entity)
  {
    var predicate = ToExpression().Compile();
    return predicate(entity);
  }

  public abstract Expression<Func<T, bool>> ToExpression();

  public static AndSpecification<T> operator &(Specification<T> spec1, Specification<T> spec2)
  {
    return new AndSpecification<T>(spec1, spec2);
  }
  public static OrSpecification<T> operator |(Specification<T> spec1, Specification<T> spec2)
  {
    return new OrSpecification<T>(spec1, spec2);
  }
  public static NotSpecification<T> operator !(Specification<T> spec)
  {
    return new NotSpecification<T>(spec);
  }
```

Спецификация

```
public sealed class AndSpecification<T> : Specification<T>
{
  public AndSpecification(Specification<T> firstSpec, Specification<T> secondSpec)
  {
    var firstExp = firstSpec.ToExpression();
    var secondExp = secondSpec.ToExpression();

    expression = Expression.Lambda<Func<T, bool>>(
                  Expression.And(firstExp.Body,
                    Expression.Invoke(secondExp, firstExp.Parameters)), firstExp.Parameters);

  }

  public override Expression<Func<T, bool>> ToExpression()
  {
    return expression;
  }
}
```

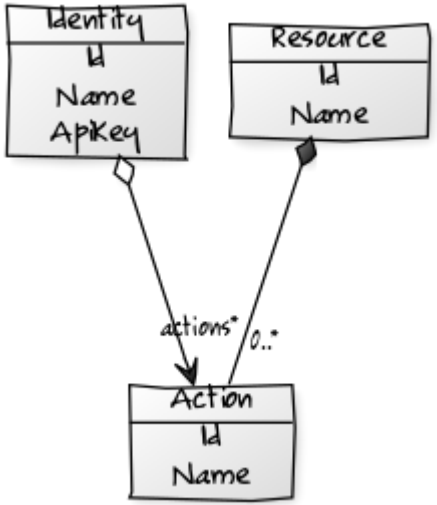
Когда сущности недостаточно

- Для валидации недостаточно данных сущности



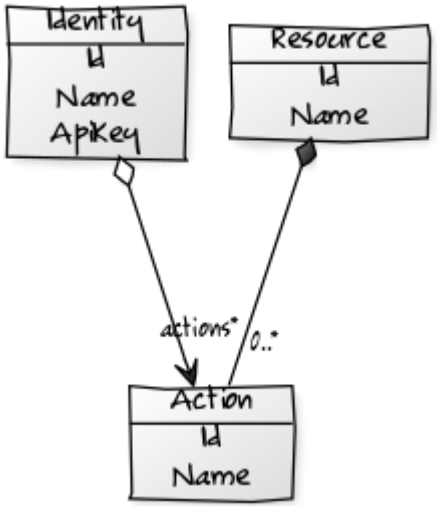
Когда сущности недостаточно

- Для валидации недостаточно данных сущности
- Валидация затрагивает несколько сущностей



Когда сущности недостаточно

- Для валидации недостаточно данных сущности
- Валидация затрагивает несколько сущностей
- Требуется подробный отчет о возникших ошибках

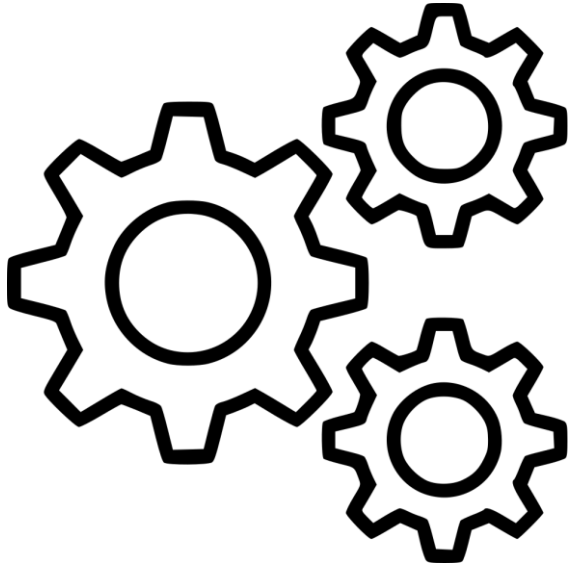


Фреймворки валидации

Business Rules
and Process Rules

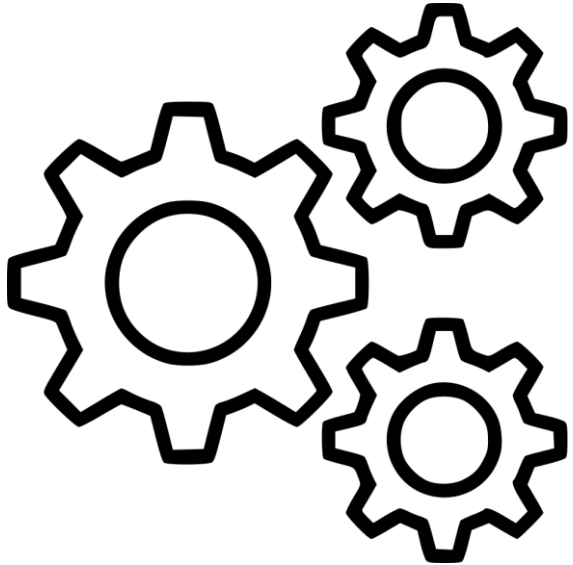
Преимущества

- Структурирует бизнес-правила



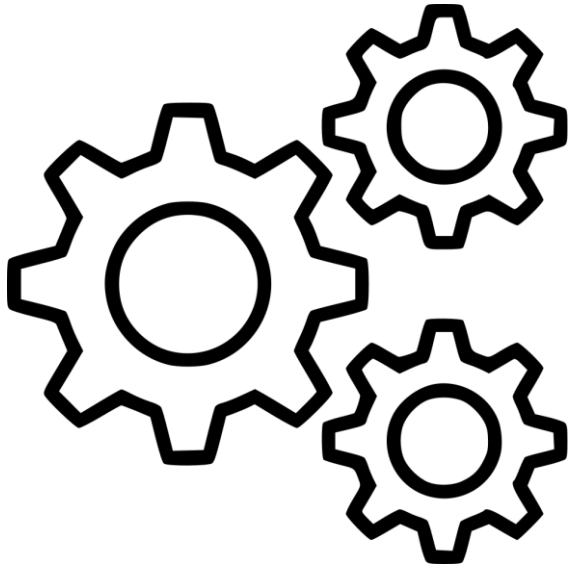
Преимущества

- Структурирует бизнес-правила
- Обеспечивает взаимосвязь правил



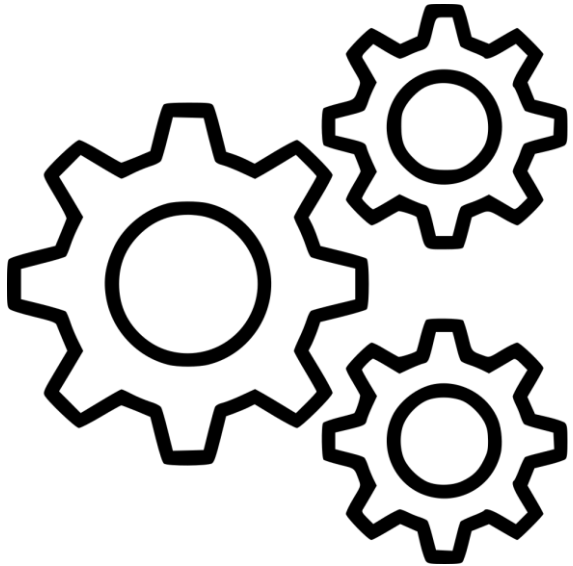
Преимущества

- Структурирует бизнес-правила
- Обеспечивает взаимосвязь правил
- Позволяет повторно использовать правила



Преимущества

- Структурирует бизнес-правила
- Обеспечивает взаимосвязь правил
- Позволяет повторно использовать правила
- Сочетает валидацию с нотификацией



FluentValidation

```
public class AddressValidator : AbstractValidator<Address>
{
  public AddressValidator()
  {
    RuleFor(address => address.PostalIndex).NotEmpty().WithName("Почтовый индекс");
    RuleFor(address => address.RegionCode).NotEmpty().WithName("Код региона");
    RuleFor(address => address.RegionName).NotEmpty().WithName("Навание региона");
    RuleFor(address => address.RegionType).NotEmpty().WithName("Тип региона");
    RuleFor(address => address.House).NotEmpty().WithName("Номер дома");
    RuleFor(address => address.FullAddress).NotEmpty().WithName("Полный адрес");
  }
}
```

Вызов валидатора

```
var result = _validator.Validate(address,
validationRequest.ValidationRuleSets.ToArray());

...

return new AddressValidationResult()
{
  IsValid = result.IsValid,
  FailureMessages = result.Errors.Select(x => new RuleValidationResult(null,
((CommonRuleIdentity)x.CustomState)
                                  .Return(s => s.Id), x.ErrorMessage))
                                  .ToList()
};
```

Недостатки

Большие классы валидаторов

```
public class CustomerValidator : BaseValidator<ICustomer>
{
  public CustomerValidator(ICustomerValidatorResources resources,
ICustomerValidatorDataAccessor accessor, IAccessor otherAccessor,
IComponentFactory<ISomeService> someServiceFactory)
: this(resources, accessor)
  {
    RuleSet(FirstRuleSet, CheckFirst);
    RuleSet(SecondRuleSet, CheckSecond);
    ...
    private void CheckNumber()
    {
      RuleFor(customer => customer.Number)
        .Length(1, 20)
        .WithMessage(_resources.NumberInvalidLengthError);
    }
}
```

177 строка

1470 строка

Недостатки

RuleSet задается через передачу строкового имени

```
RuleSet("CheckFirst", CheckFirst);
RuleSet("CheckSecond", CheckSecond);
RuleSet("RequiredFieldsRules", RequiredFieldsRules);
RuleSet("RequiredFieldsFioRules", RequiredFieldFioRules);
RuleSet("CheckThird", CheckThird);
RuleSet("CheckFourth", CheckFourth);
```

```
var validationResult =
isValid
? ((BaseValidator<ICustomer>)_validator).Validate(customer, new [] {"CheckFourth"})
: new ValidationResult();
```

Недостатки

Всего один способ прервать валидацию

```
public CustomerValidator(ICustomerValidatorResources resources,
ICustomerValidatorDataAccessor accessor, IAccessor otherAccessor,
IComponentFactory<ISomeService> someServiceFactory)
: this(resources, accessor)
  {
    CascadeMode = CascadeMode.StopOnFirstFailure;

    RuleSet(FirstRuleSet, CheckFirst);
    RuleSet(SecondRuleSet, CheckSecond);
```

Недостатки

Ограниченные возможности по передаче дополнительных данных

```
var instanceToValidate = new Address();
var context = new ValidationContext<Address>(address);
context.RootContextData["MyCustomData"] = "Test";
var validator = new AddressValidator();
validator.Validate(context);
```

```
RuleFor(x => x.Building).Custom((x, context) =>
{
  if (context.ParentContext.RootContextData.ContainsKey("MyCustomData"))
  {
    context.AddFailure("My error message");
  }
});
```

Требования

- Независимые правила с возможностью композиции

Требования

- Независимые правила с возможностью композиции
- Возможность независимого тестирования правил

Требования

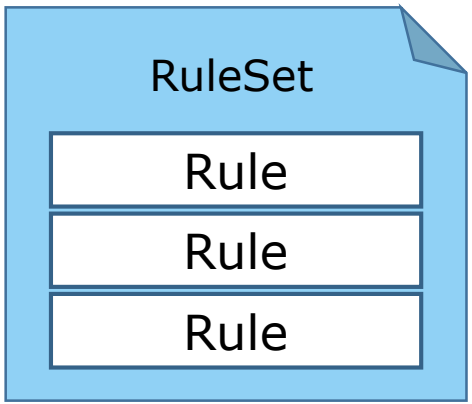
- Независимые правила с возможностью композиции
- Возможность независимого тестирования правил
- Широкая возможность передачи дополнительного контекста

Требования

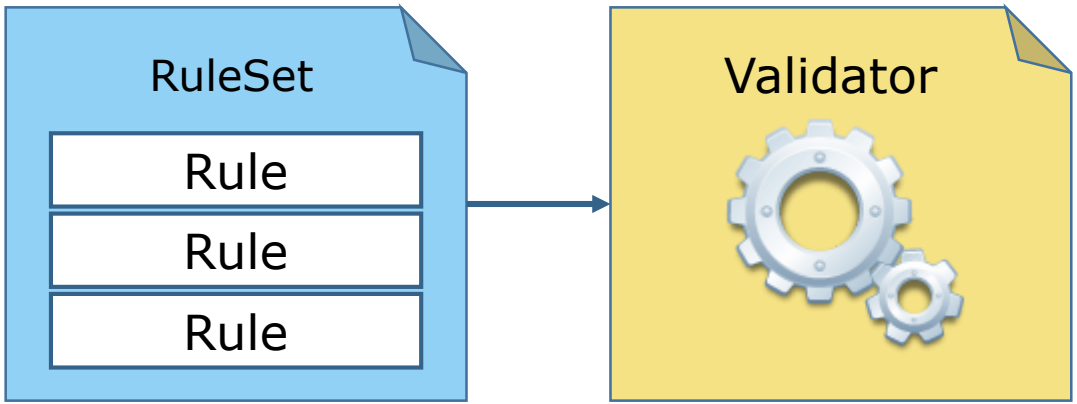
- Независимые правила с возможностью композиции
- Возможность независимого тестирования правил
- Широкая возможность передачи дополнительного контекста
- Гибкое управление потоком валидации

Собственный фреймворк

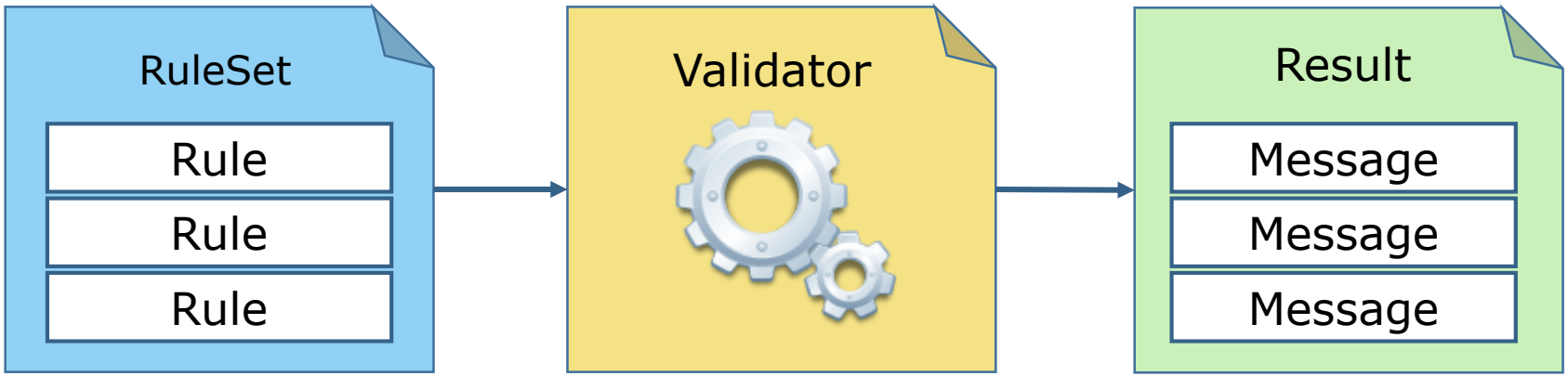
Архитектура



Архитектура



Архитектура



Правило

```
public class BlockCardDateRule : BaseRule<Card>
{
  public override RuleValidationResult Validate(Card card,
                                                ValidationContext context)
  {
    var result = new RuleValidationResult(Identity);
    if (card.BlockDate == null)
      result.Append(new ErrorRuleMessage(messageCode,
        new MessageIdentity(internalCode), "Block date must have a value"));

    var today = DateTime.Today;
    if (card.BlockDate < today)
      result.Append(new WarningRuleMessage(messageCode,
        new MessageIdentity(internalCode),
          $"Block date must not be earlier than {today}");

    return result;
  }
}
```

Fluent-правило

```
internal class BlockCardDateRule : BaseFluentRule<Card>
{
  public BlockCardDateRule() : base()
  {
    ForProperty(card => card.BlockDate)
      .NotNull()
      .WithErrorMessage("Block date must have a value");
      .Must((card, c) => card >= ((BlockContext)c.Context).Today)
      .WithWarningMessage("Block date must not be earlier than {0}",
                          ((BlockContext)c.Context).Today);
  }
}
```

Контекст валидации

```
public class ValidationContext
{
  /// <summary>
  /// Инициализирует контекст валидации объектом контекста
  /// </summary>
  /// <param name="context">Объект контекста</param>
  public ValidationContext(object context)
  {
    Context = context;
  }

  /// <summary>
  /// Объект контекста
  /// </summary>
  public object Context { get; }
}
```

Пример простого валидатора

```
internal class EqualValidatorAsync<T> : PropertyRuleValidatorAsync<T>
{
  private readonly IEqualityComparer<T> _comparer;
  private readonly Task<T> _valueToCompare;

  public EqualValidatorAsync(Task<T> valueToCompare, IEqualityComparer<T> comparer)
  {
    _valueToCompare = valueToCompare;
    _comparer = comparer;
  }

  protected override async Task<bool> IsValidAsync(T instance, ValidationContext context)
  {
    T compareValue = await _valueToCompare;
    return Compare(instance, compareValue);
  }

  protected bool Compare(T comparisonValue, T propertyValue) =>
    _comparer?.Equals(comparisonValue, propertyValue) ?? Equals(comparisonValue, propertyValue);
}
```

RuleSet

```
internal class BlockCardRuleSet : BaseValidationRuleSet<Card>
{
  public BlockCardRuleSet(BlockCardNumberRule blockCardNumberRule,
    BlockCardStatusRule blockCardStatusRule, ...)
  {
    SetRule(blockCardNumberRule);

    SetRule(blockCardStatusRule).StopOnFailure();

    When(blockCardTypeRule,
           () => SetRule(blockCardDateRule));

    SetRule(blockCardExternalDateRule)
      .DependsOn(blockCardExternalSystemRule)
      .DependsOn(blockCardDateRule);

    SetCollectionContext(card => card.Blocks, blockRuleSet);
  }
}
```

RuleSet

```
internal class BlockCardRuleSet : BaseValidationRuleSet<Card>
{
  public BlockCardRuleSet(BlockCardNumberRule blockCardNumberRule,
    BlockCardStatusRule blockCardStatusRule, ...)
  {
    SetRule(blockCardNumberRule);

    SetRule(blockCardStatusRule).StopOnFailure();

    When(blockCardTypeRule,
           () => SetRule(blockCardDateRule));

    SetRule(blockCardExternalDateRule)
      .DependsOn(blockCardExternalSystemRule)
      .DependsOn(blockCardDateRule);

    SetCollectionContext(card => card.Blocks, blockRuleSet);
  }
}
```

RuleSet

```
internal class BlockCardRuleSet : BaseValidationRuleSet<Card>
{
  public BlockCardRuleSet(BlockCardNumberRule blockCardNumberRule,
    BlockCardStatusRule blockCardStatusRule, ...)
  {
    SetRule(blockCardNumberRule);

    SetRule(blockCardStatusRule).StopOnFailure();

    When(blockCardTypeRule,
           () => SetRule(blockCardDateRule));

    SetRule(blockCardExternalDateRule)
      .DependsOn(blockCardExternalSystemRule)
      .DependsOn(blockCardDateRule);

    SetCollectionContext(card => card.Blocks, blockRuleSet);
  }
}
```

RuleSet

```
internal class BlockCardRuleSet : BaseValidationRuleSet<Card>
{
  public BlockCardRuleSet(BlockCardNumberRule blockCardNumberRule,
    BlockCardStatusRule blockCardStatusRule, ...)
  {
    SetRule(blockCardNumberRule);

    SetRule(blockCardStatusRule).StopOnFailure();

    When(blockCardTypeRule,
           () => SetRule(blockCardDateRule));

    SetRule(blockCardExternalDateRule)
      .DependsOn(blockCardExternalSystemRule)
      .DependsOn(blockCardDateRule);

    SetCollectionContext(card => card.Blocks, blockRuleSet);
  }
}
```

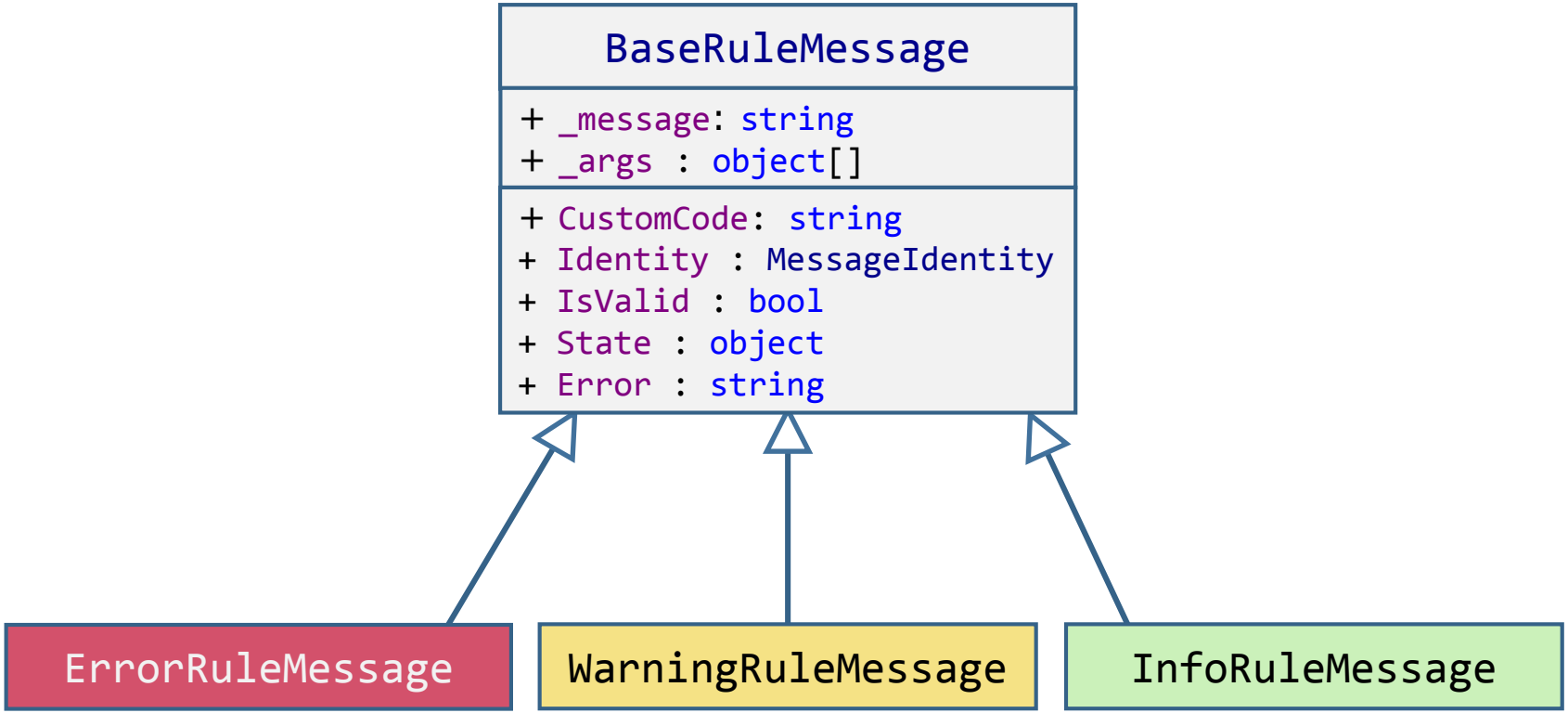
RuleSet

```
internal class BlockCardRuleSet : BaseValidationRuleSet<Card>
{
  public BlockCardRuleSet(BlockCardNumberRule blockCardNumberRule,
    BlockCardStatusRule blockCardStatusRule, ...)
  {
    SetRule(blockCardNumberRule);

    SetRule(blockCardStatusRule).StopOnFailure();

    When(blockCardTypeRule,
           () => SetRule(blockCardDateRule));

    SetRule(blockCardExternalDateRule)
      .DependsOn(blockCardExternalSystemRule)
      .DependsOn(blockCardDateRule);

    SetCollectionContext(card => card.Blocks, blockRuleSet);
  }
}
```

ValidationResult

```
public sealed class ValidationResult
{
  public ValidationResult()
  {
    Results = new List<RuleValidationResult>();
  }

  public bool IsValid
  {
    get { return Results.All(x => x.IsValid); }
  }

  public List<RuleValidationResult> Results { get; }

  public void MergeResult(ValidationResult result)
  {
    Results.AddRange(result.Results);
  }
}
```

Сообщения


BaseRuleMessage
+ _message: string
+ _args : object[]
+ CustomCode: string
+ Identity : MessageIdentity
+ IsValid : bool
+ State : object
+ Error : string
ErrorRuleMessage
WarningRuleMessage
InfoRuleMessage

Результаты

Большие валидаторы

Отдельные классы правил

Результаты

Большие валидаторы → Отдельные классы правил

Проблемы с группировкой правил → Классы RuleSet

Результаты

Большие валидаторы	→	Отдельные классы правил
Проблемы с группировкой правил	→	Классы RuleSet
Всего один способ прервать валидацию	→	Классы RuleSet

Результаты

Большие валидаторы	→	Отдельные классы правил
Проблемы с группировкой правил	→	Классы RuleSet
Всего один способ прервать валидацию	→	Классы RuleSet
Проблема с дополнительными данными	→	ValidationContext

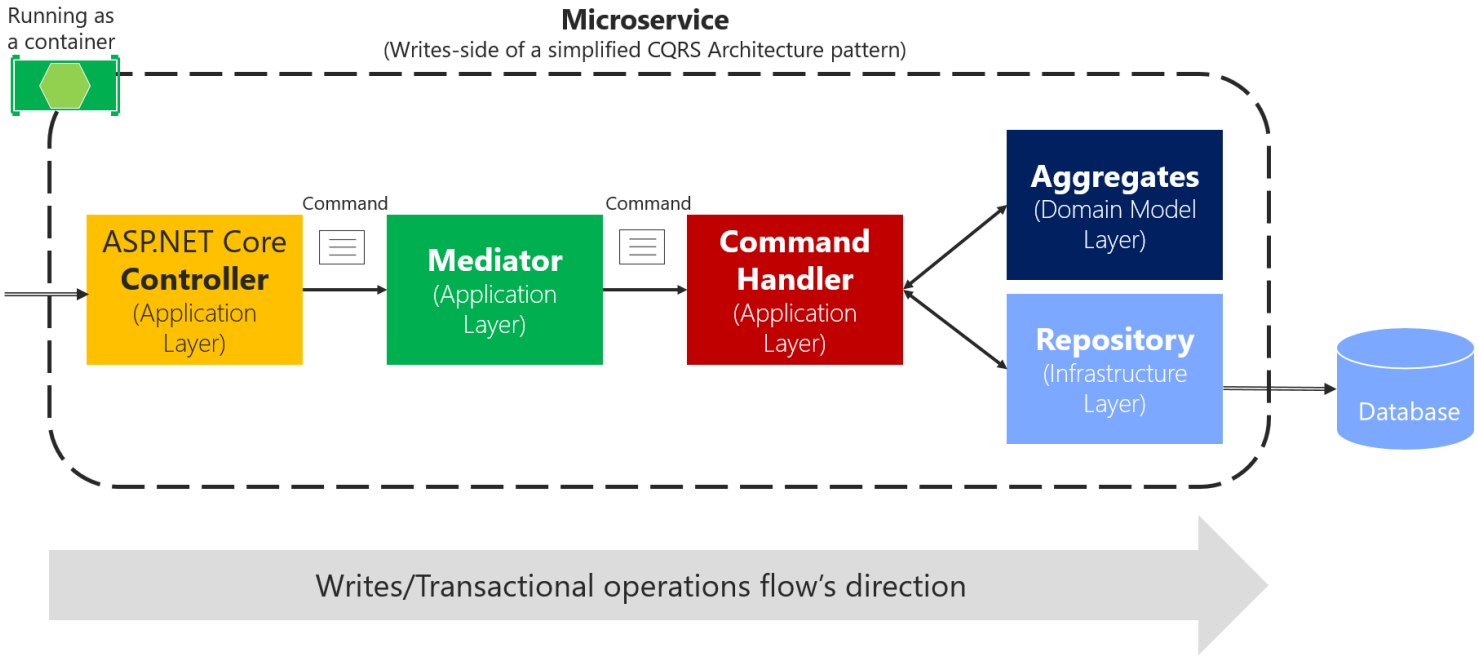
Always-valid entity

```
public class Card : IEntity<int>, IEventProvider
{
  private IEventCollector _eventCollector;

  BlockType BlockType { get; protected virtual set; }

  ...

  public virtual void BlockCard([NotNull] BlockDetails blockDetails)
  {
    BlockType = blockDetails.BlockType;
    RulesValidator.Validate(new BlockCardRuleSet(), this)
  }
}
```

Валидация в сервисах

```
internal class CardBlockingService : ICardValidationService
{
  private readonly IValidator _validator;
  private readonly BlockCardRuleSet _blockCardRuleSet;

  public CardBlockingService(IValidator validator, BlockCardRuleSet blockRuleSet)
  {
    _validator = validator;
    _blockCardRuleSet = blockCardRuleSet;
  }

  public CardValidationResult BlockCard(BlockDetails blockDetails)
  {
    var result = _validator.Validate(_blockCardRuleSet, blockDetails);
    if (result.IsValid)
      card.BlockCard(blockDetails);
  }
}
```

Pipeline


Running as a container
Microservice
(Writes-side of a simplified CQRS Architecture pattern)
ASP.NET Core Controller (Application Layer)
Command
Mediator (Application Layer)
Command
Command Handler (Application Layer)
Aggregates (Domain Model Layer)
Repository (Infrastructure Layer)
Database
Writes/Transactional operations flow's direction

Декоратор

```
public class MessageValidationBehavior : IBroadcastPreProcessor
{
  private readonly Dictionary<Type, IMessageValidation> _rules = new Dictionary<Type,
                                                                 IMessageValidation>();
  ...
  public Task ProcessAsync(IMessage message, CancellationToken cancellationToken)
  {
    var messageType = message.GetType();
    if (!_rules.TryGetValue(messageType, out var rule))
    {
      return Task.CompletedTask;
    }

    var result = rule.Validate(message);
    if (!result.IsValid)
    {
      throw new MessageValidationException(result.Results.ToErrorsString());
    }

    return Task.CompletedTask;
  }
}
```

Регистрация

```
public class ApiPackage : IPackage
{
  public void RegisterServices(Container container)
  {
    ...
    container.Collection.Register(typeof(IMessageValidation), GetType().Assembly);
    container.RegisterBroadcastPreProcessor<MessageValidationBehavior>(-1);
  }
}
```

Резюме

- Наивная валидация – верный способ усложнить себе жизнь

Резюме

- Наивная валидация – верный способ усложнить себе жизнь
- Наиболее популярный подход в DDD – always-valid entity

Резюме

- Наивная валидация – верный способ усложнить себе жизнь
- Наиболее популярный подход в DDD – always-valid entity
- Данный подход имеет ряд недостатков при усложнении проекта

Резюме

- Наивная валидация – верный способ усложнить себе жизнь
- Наиболее популярный подход в DDD – always-valid entity
- Данный подход имеет ряд недостатков при усложнении проекта
- Фреймворки валидации созданы для структуризации и стандартизации процесса

Резюме

- Наивная валидация – верный способ усложнить себе жизнь
- Наиболее популярный подход в DDD – always-valid entity
- Данный подход имеет ряд недостатков при усложнении проекта
- Фреймворки валидации созданы для структуризации и стандартизации процесса
- Не всегда готовое решение полностью подходит, иногда требуются доработки

Спасибо!

konst.gustov@gmail.com