Workflow-архитектура сервисов на .NET

Михаил
Дошевский

m.doshevsky@gmail.com

DOTNEXT

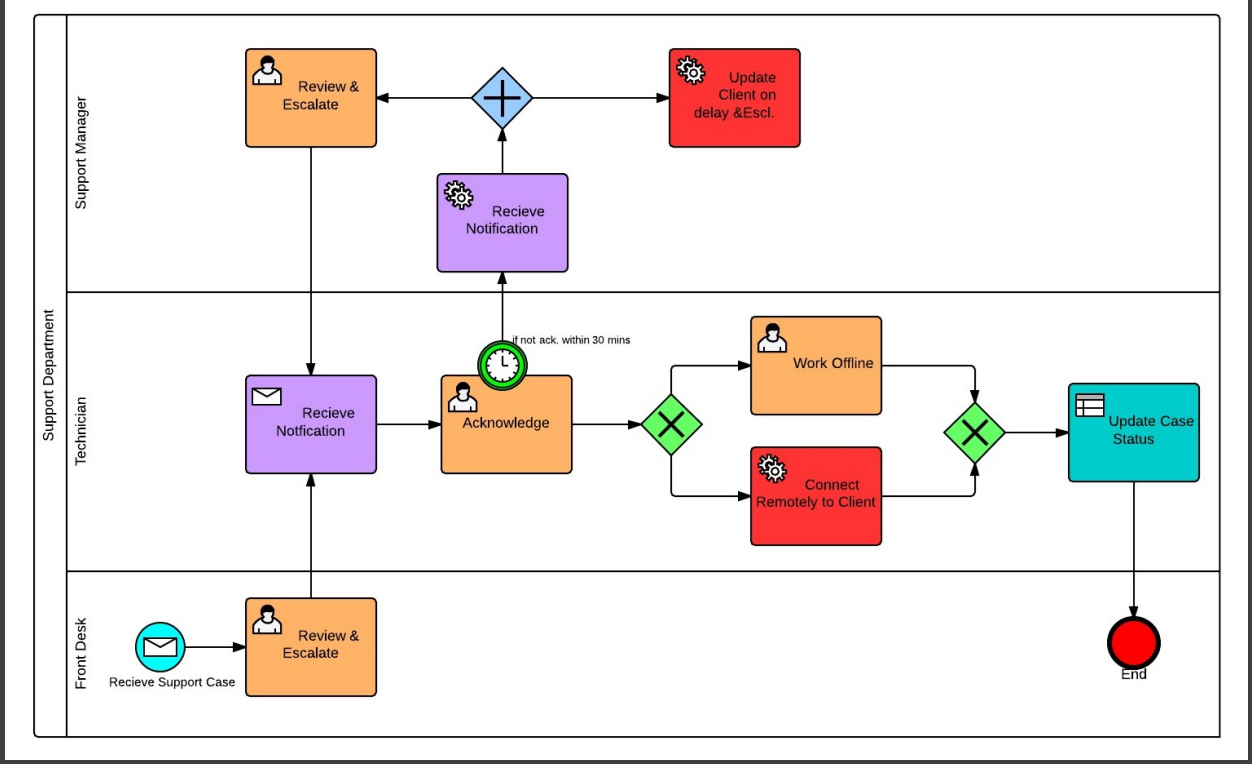
Про что доклад

- Workflow-ориентированный архитектурный подход
- Проблемы, которые у нас возникли при разработки софта для банкоматов
- Обзор инструментов
- Какие задачи можно решить с помощью workflow
- Создание telegram-бота на основе workflow

Для каких задач подходит workflow-архитектура

Для каких задач подходит workflow-архитектура

Автоматизация рабочих процессов (документооборот, CRM)

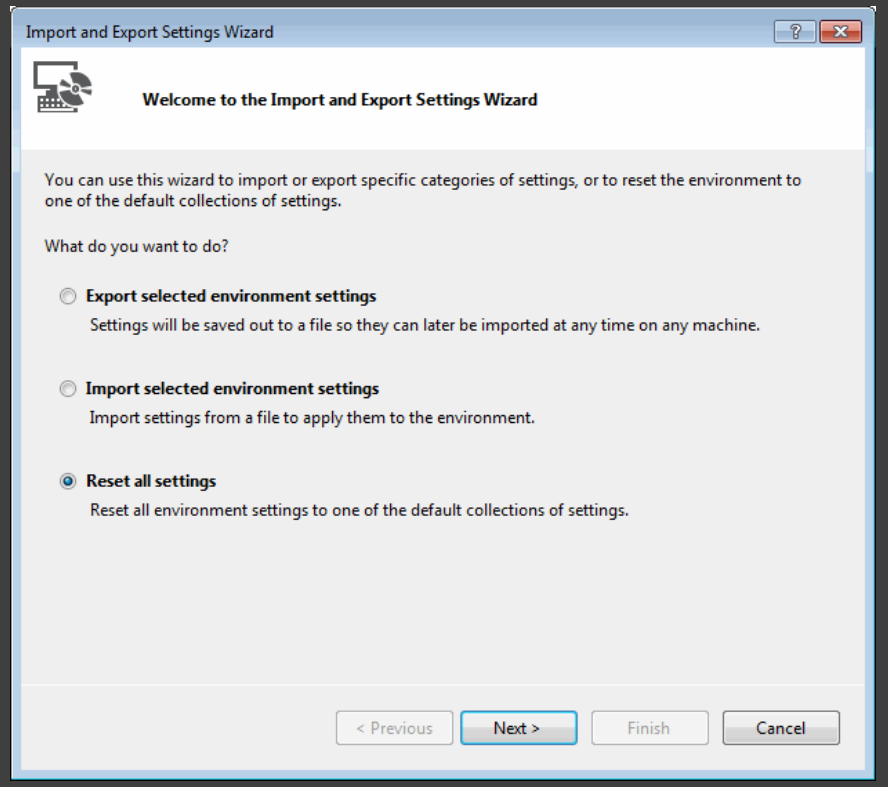


Для каких задач подходит workflow-архитектура

Чат-боты

Для каких задач подходит workflow-архитектура

Визарды



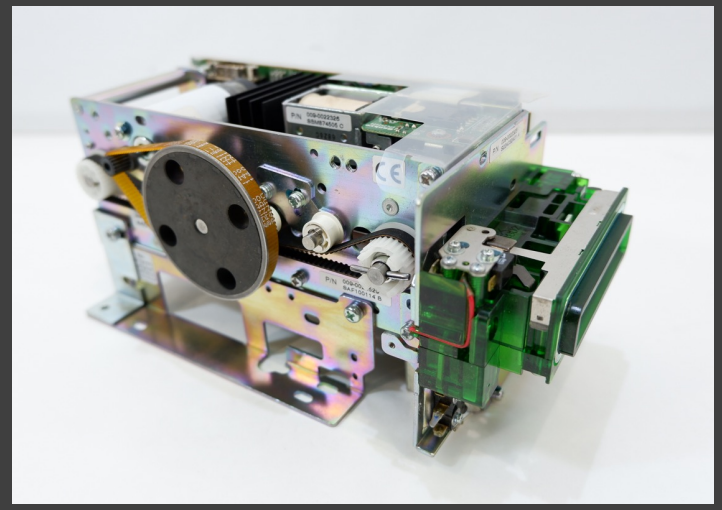
Для каких задач подходит workflow-архитектура

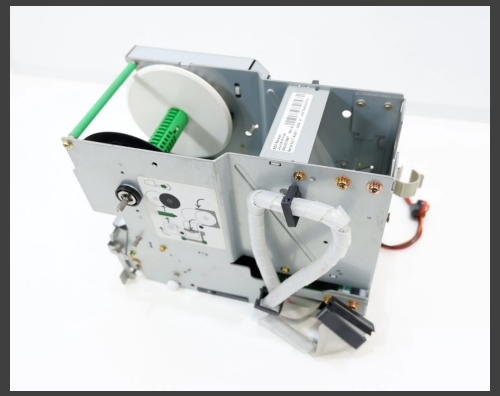
Оркестрация микросервисов

```
builder
    .StartWith(context => Console.WriteLine("Begin"))
    .Saga(saga => saga
        .StartWith<Task1>()
            .CompensateWith<UndoTask1>()
        .Then<Task2>()
            .CompensateWith<UndoTask2>()
        .Then<Task3>()
            .CompensateWith<UndoTask3>())
        .CompensateWith<CleanUp>()
    .Then(context => Console.WriteLine("End"));
```

Банкомат

Банкомат. Управление железом



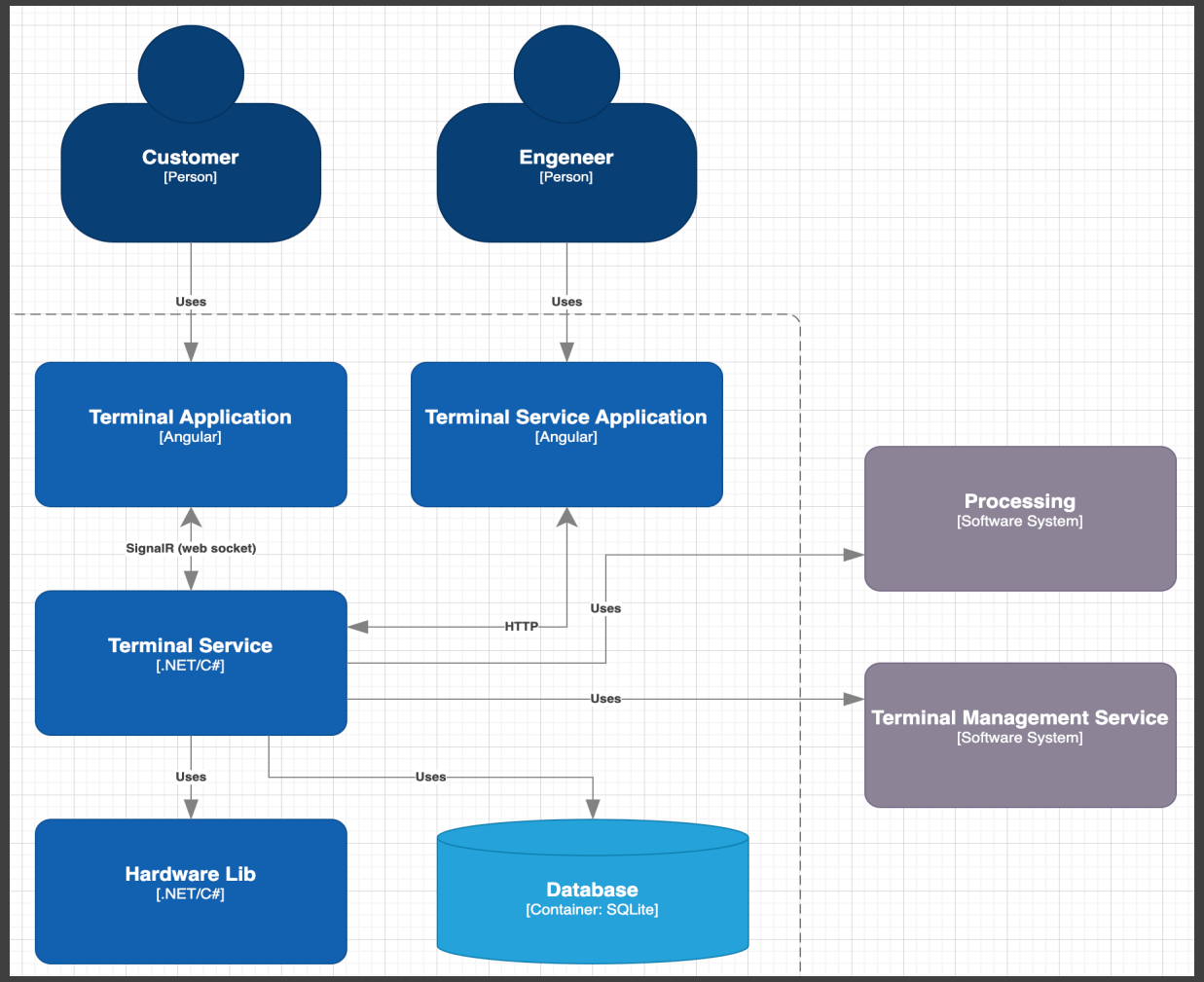


Банкомат. Взаимодействие с пользователями

Банкомат. Software

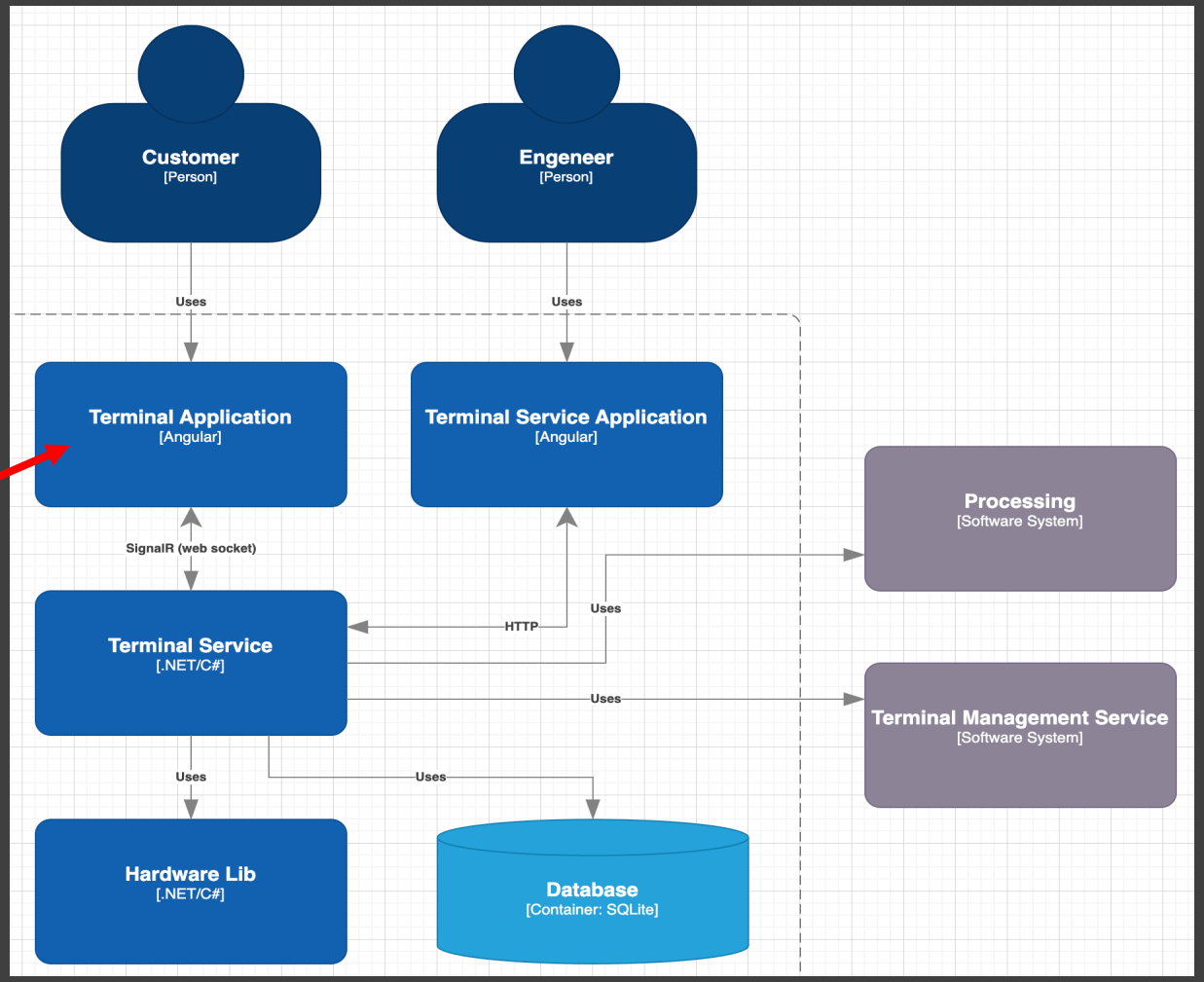
- Пользовательское приложение (разные для каждого банка)
- Сервисное приложение
- Сервис управления сетью банкоматов
- Процессинг

Архитектура решения



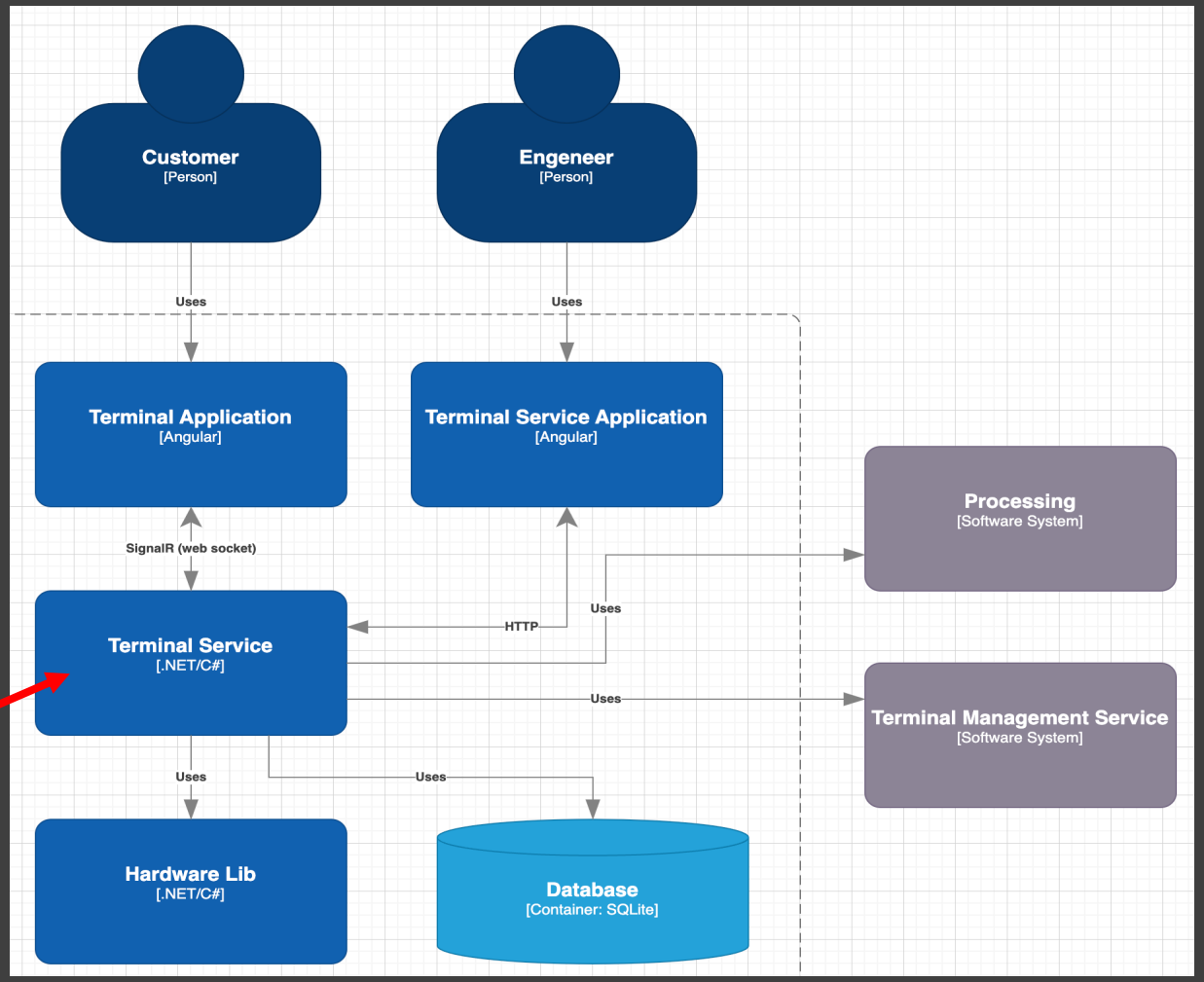
- .NET-сервис предоставляет API
- SPA использует API для формирования операций
- .NET-сервис взаимодействует с устройствами и процессингом

Архитектура решения



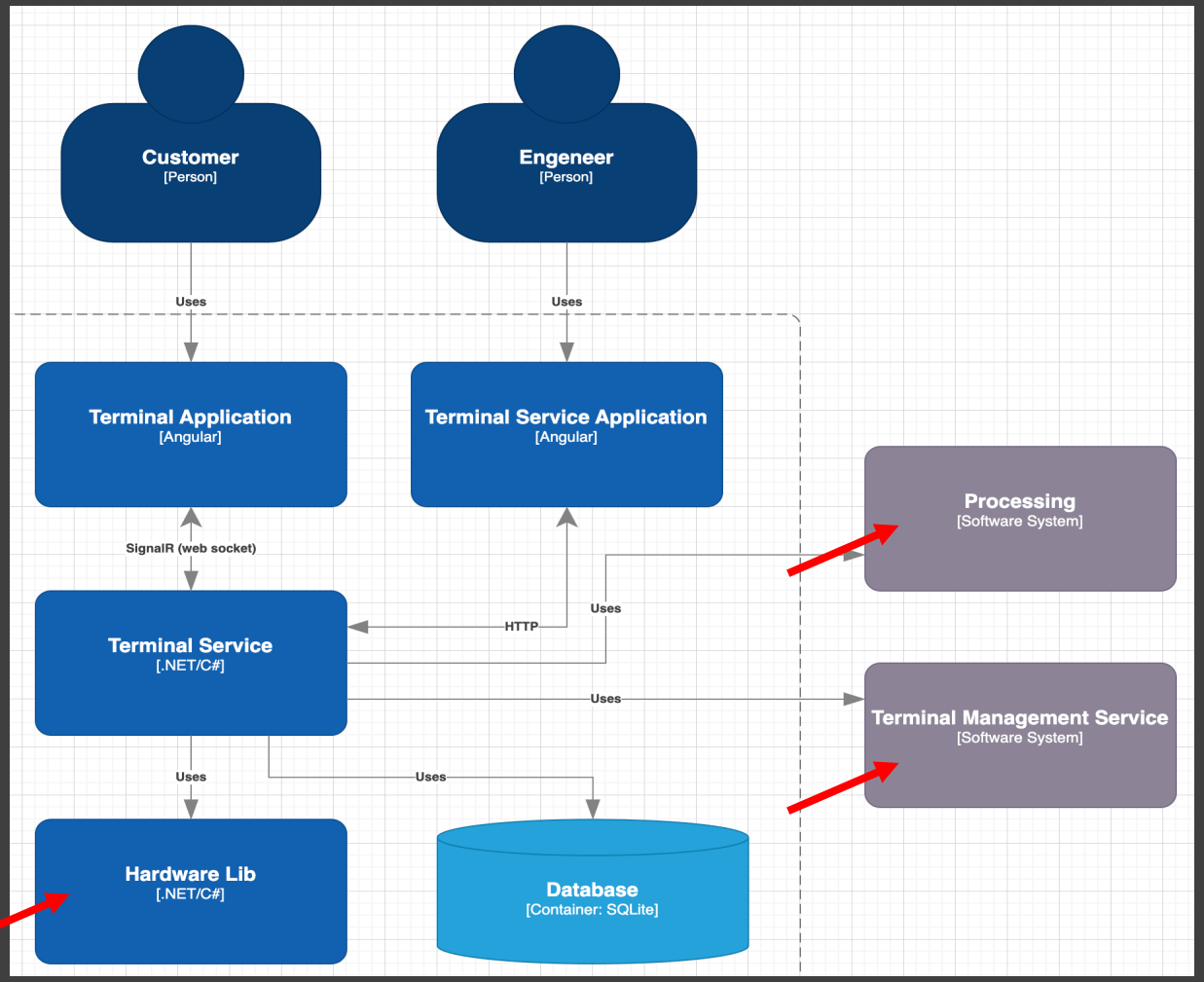
- .NET-сервис предоставляет API
- SPA использует API для формирования операций
- .NET-сервис взаимодействует с устройствами и процессингом

Архитектура решения



- .NET-сервис предоставляет API
- SPA использует API для формирования операций
- .NET-сервис взаимодействует с устройствами и процессингом

Архитектура решения



- .NET-сервис предоставляет API
- SPA использует API для формирования операций
- .NET-сервис взаимодействует с устройствами и процессингом

Проблемы

- Сложности реализации логики операций - нужно выполнять много отдельных шагов в правильной последовательности, обрабатывать все нестандартные ситуации
- Логика расползлась между .NET-сервисом и SPA, часто дублировалась

Решение

- Workflow-архитектура
- Маленькие отдельные шаги формируют разные workflow под конкретную операцию
- Логика операций полностью переехала в .NET-сервис
- .NET-сервис стал управлять SPA, решал, какую View показывать, передавал нужные данные
- SPA стал просто получать ивенты от сервиса и показывать нужные экраны
- Добавили Flowchart-диаграммы операций
- Можно сделать описание workflow в JSON\YAML

Выбор библиотеки. Требования

- Управление потоком workflow: ветвления, циклы, параллелизм
- Поддержка внешних событий
- Поддержка .NET 6+
- Легкая embedded-библиотека, без отдельного сервера и дизайнера

Выбор библиотеки. Варианты

- WWF (CoreWF)
- WorkflowEngine.NET
- Elsa
- Workflow Core (наш выбор)

Workflow. Шаги

```
public class CalculateSum : IStepBody
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int Result { get; set; }

    public Task<ExecutionResult> RunAsync(
        IStepExecutionContext context)
    {
        Result = NumberA + NumberB;
        return Task.FromResult(ExecutionResult.Next());
    }
}
```

- Каждый шаг – отдельный класс
- Входные и выходные параметры передаются через свойства

Workflow. Шаги

```
public class CalculateSum : IStepBody
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int Result { get; set; }

    public Task<ExecutionResult> RunAsync(
        IStepExecutionContext context)
    {
        Result = NumberA + NumberB;
        return Task.FromResult(ExecutionResult.Next());
    }
}
```

- Каждый шаг – отдельный класс
- Входные и выходные параметры передаются через свойства

Workflow. Шаги

```
public class SendEmail : IStepBody
{
    public SendEmail(IEmailService emailService)
    {
        _emailService = emailService;
    }

    public string? Message { get; set; }

    public async Task<ExecutionResult> RunAsync(
        IStepExecutionContext context)
    {
        bool result = await _emailService.SendMessage(Message);
        return result ?
            ExecutionResult.Next() :
            ExecutionResult.Outcome(false);
    }

    private readonly IEmailService _emailService;
}
```

- Зависимости можно передавать через Dependency Injection
- Можно управлять потоком выполнения workflow

Workflow. Шаги

```
public class SendEmail : IStepBody
{
    public SendEmail(IEmailService emailService)
    {
        _emailService = emailService;
    }

    public string? Message { get; set; }

    public async Task<ExecutionResult> RunAsync(
        IStepExecutionContext context)
    {
        bool result = await _emailService.SendMessage(Message);
        return result ?
            ExecutionResult.Next() :
            ExecutionResult.Outcome(false);
    }

    private readonly IEmailService _emailService;
}
```

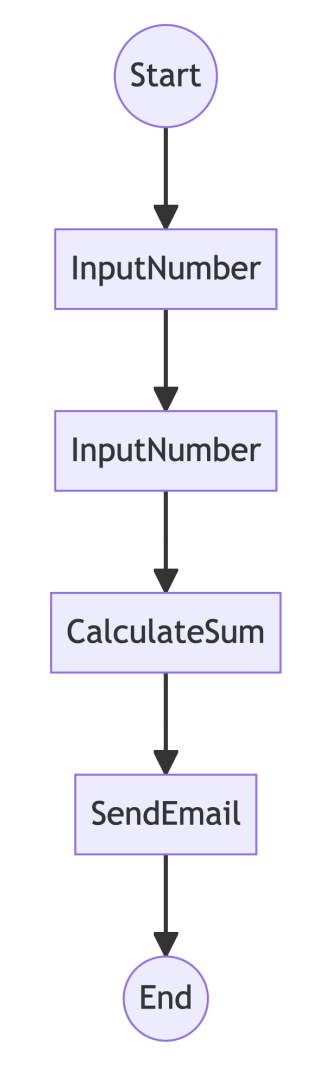
- Зависимости можно передавать через Dependency Injection
- Можно управлять потоком выполнения workflow

Workflow. Контекст данных

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

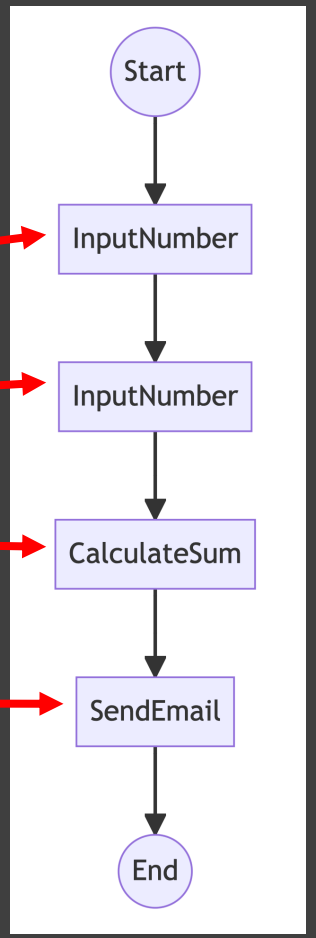
- Простой класс с набором свойств
- Контекст данных можно сохранять в базу. В WorkflowCore есть разные Persistence Providers

Workflow


Start
InputNumber
InputNumber
CalculateSum
SendEmail
End

Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```



Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

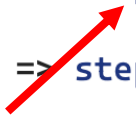
Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```



```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

Workflow

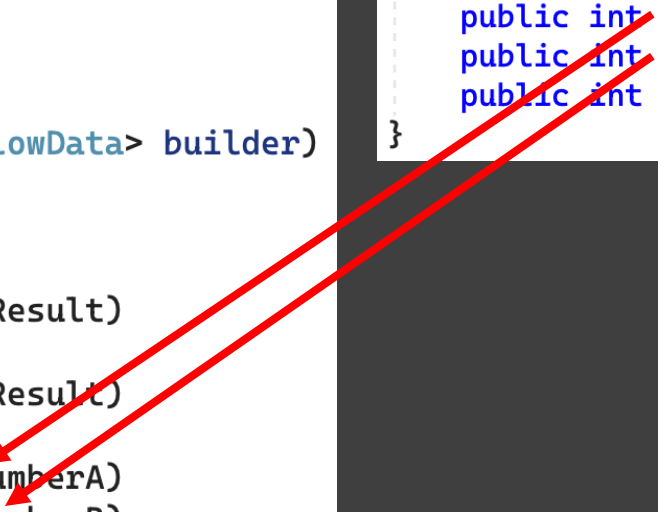
```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

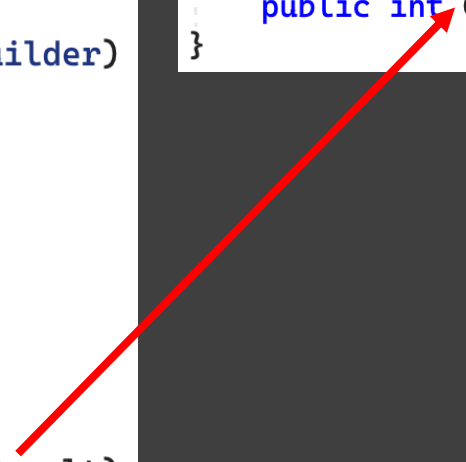
```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```



Workflow

```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```



Workflow

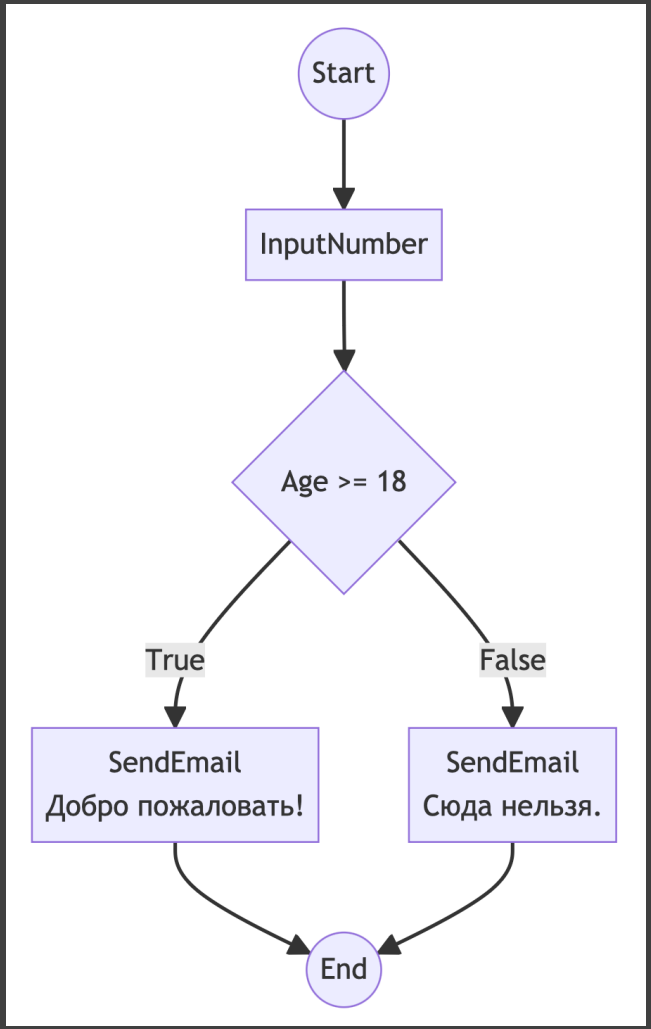
```
public class CalculationWorkflow : IWorkflow<CalculationWorkflowData>
{
    public string Id => "CalculationWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.NumberA, step => step.Result)
            .Then<InputNumber>()
            .Output(data => data.NumberB, step => step.Result)
            .Then<CalculateSum>()
            .Input(step => step.NumberA, data => data.NumberA)
            .Input(step => step.NumberB, data => data.NumberB)
            .Output(data => data.CalculationResult, step => step.Result)
            .Then<SendEmail>()
            .Input(step => step.Message,
                data => data.CalculationResult.ToString())
            .EndWorkflow();
    }
}
```

```
public class CalculationWorkflowData
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int CalculationResult { get; set; }
}
```

Workflow. Управление потоком выполнения

- Ветвления
- Циклы
- Параллельное исполнение
- Ожидание событий

Workflow. Ветвления



Workflow. Ветвления

```
public class UserAgeWorkflow : IWorkflow<UserAgeWorkflowData>
{
    public string Id => "UserAgeWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<UserAgeWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.Age, step => step.Result)
            .If(data => data.Age >= 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Добро пожаловать!"))
            .If(data => data.Age < 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Сюда нельзя."))
            .EndWorkflow();
    }
}
```

```
public class UserAgeWorkflowData
{
    public int Age { get; set; }
}
```

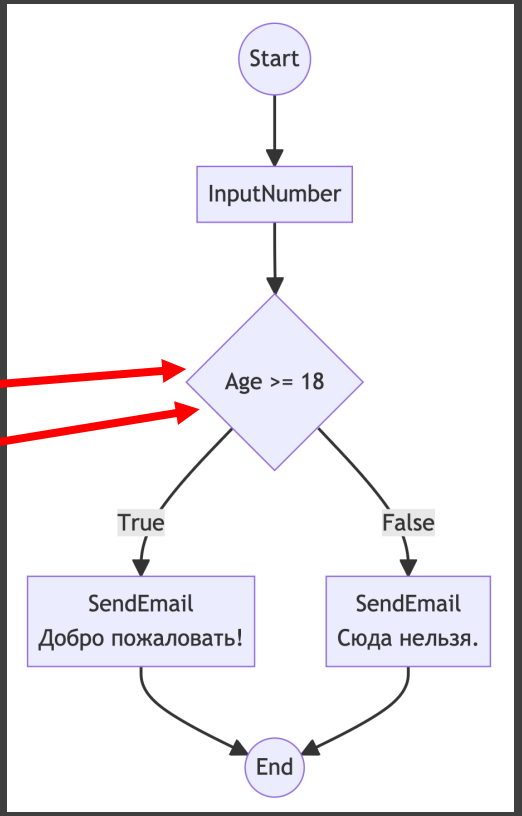
Workflow. Ветвления

```
public class UserAgeWorkflow : IWorkflow<UserAgeWorkflowData>
{
    public string Id => "UserAgeWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<UserAgeWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.Age, step => step.Result)
            .If(data => data.Age >= 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Добро пожаловать!"))
            .If(data => data.Age < 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Сюда нельзя."))
            .EndWorkflow();
    }
}
```

```
public class UserAgeWorkflowData
{
    public int Age { get; set; }
}
```

Workflow. Ветвления

```
public class UserAgeWorkflow : IWorkflow<UserAgeWorkflowData>
{
    public string Id => "UserAgeWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<UserAgeWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.Age, step => step.Result)
            .If(data => data.Age >= 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Добро пожаловать!"))
            .If(data => data.Age < 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Сюда нельзя."))
            .EndWorkflow();
    }
}
```



Workflow. Ветвления

```
public class UserAgeWorkflow : IWorkflow<UserAgeWorkflowData>
{
    public string Id => "UserAgeWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<UserAgeWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.Age, step => step.Result)
            .If(data => data.Age >= 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Добро пожаловать!"))
            .If(data => data.Age < 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Сюда нельзя."))
            .EndWorkflow();
    }
}
```

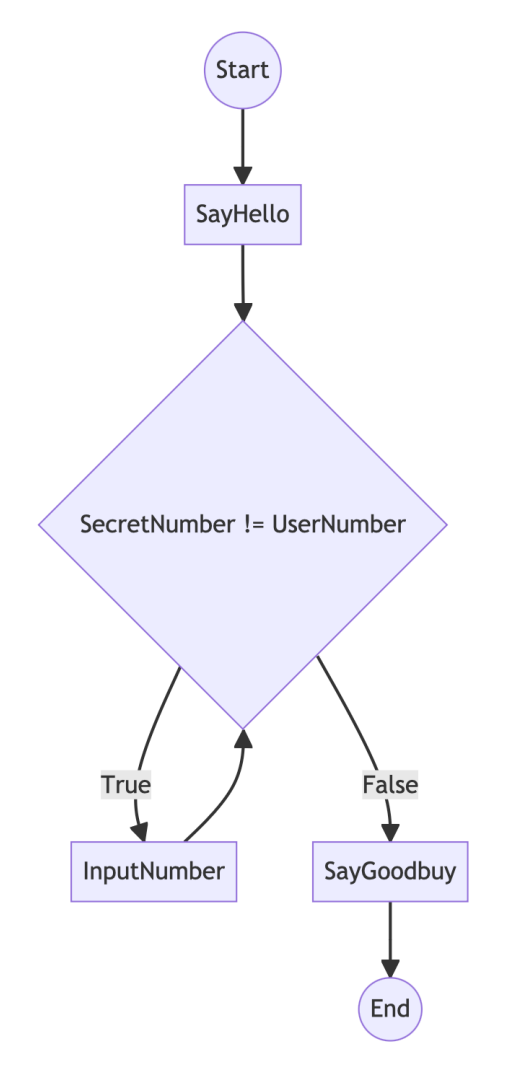
```
public class UserAgeWorkflowData
{
    public int Age { get; set; }
}
```

Workflow. Ветвления

```
public class UserAgeWorkflow : IWorkflow<UserAgeWorkflowData>
{
    public string Id => "UserAgeWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<UserAgeWorkflowData> builder)
    {
        builder
            .StartWith<InputNumber>()
            .Output(data => data.Age, step => step.Result)
            .If(data => data.Age >= 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Добро пожаловать!"))
            .If(data => data.Age < 18).Do(_ => _
                .StartWith<SendEmail>()
                .Input(step => step.Message, _ => "Сюда нельзя."))
            .EndWorkflow();
    }
}
```

```
public class UserAgeWorkflowData
{
    public int Age { get; set; }
}
```

Workflow. Циклы



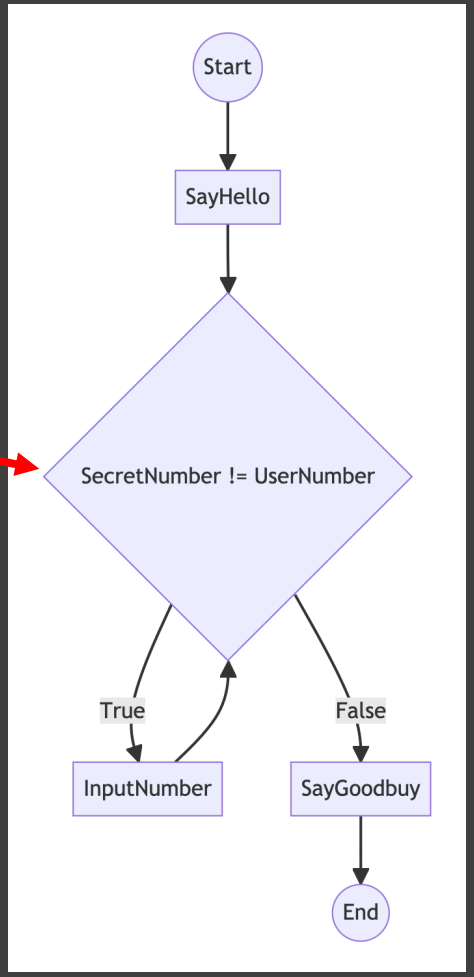
Workflow. Циклы

```
public class GuessNumberWorkflow : IWorkflow<GuessNumberWorkflowData>
{
    public string Id => "GuessNumberWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<GuessNumberWorkflowData> builder)
    {
        builder
            .StartWith<SayHello>()
            .While(data => data.SecretNumber != data.UserNumber).Do(_ => _
                .StartWith<InputNumber>()
                .Output(data => data.UserNumber, step => step.Result))
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```

```
public class GuessNumberWorkflowData
{
    public int SecretNumber { get; } = 42;
    public int UserNumber { get; set; }
}
```

Workflow. Циклы

```
public class GuessNumberWorkflow : IWorkflow<GuessNumberWorkflowData>
{
    public string Id => "GuessNumberWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<GuessNumberWorkflowData> builder)
    {
        builder
            .StartWith<SayHello>()
            .While(data => data.SecretNumber != data.UserNumber).Do( _ => _
                .StartWith<InputNumber>()
                .Output(data => data.UserNumber, step => step.Result))
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```

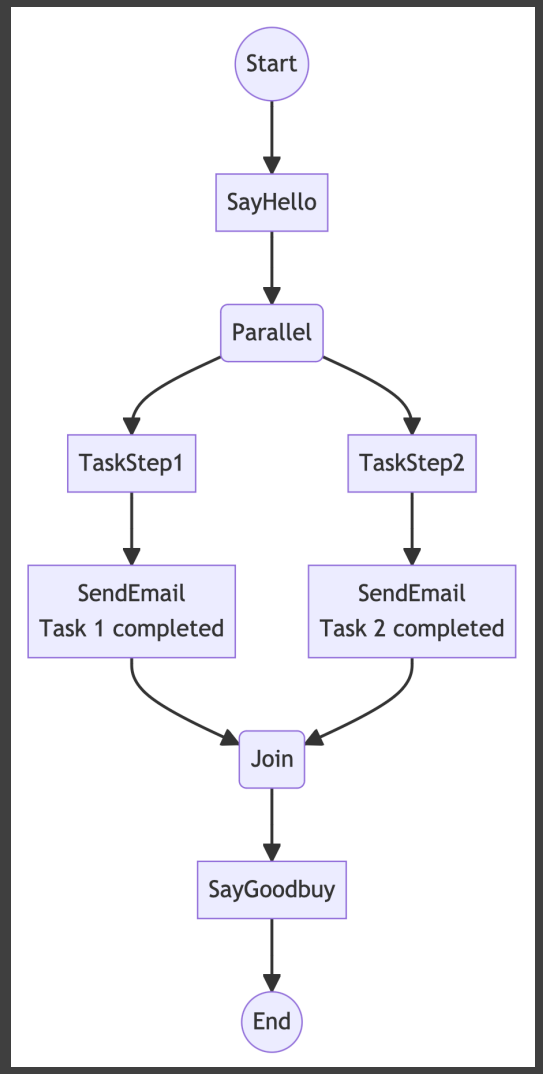


Workflow. Циклы

```
public class GuessNumberWorkflow : IWorkflow<GuessNumberWorkflowData>
{
    public string Id => "GuessNumberWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<GuessNumberWorkflowData> builder)
    {
        builder
            .StartWith<SayHello>()
            .While(data => data.SecretNumber != data.UserNumber).Do(_ => _
                .StartWith<InputNumber>()
                .Output(data => data.UserNumber, step => step.Result))
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```

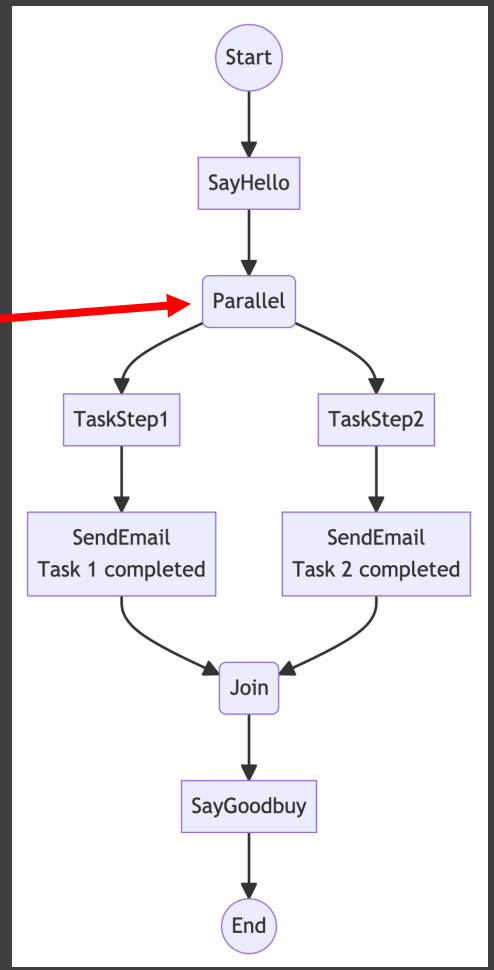
```
public class GuessNumberWorkflowData
{
    public int SecretNumber { get; } = 42;
    public int UserNumber { get; set; }
}
```

Workflow. Параллельное исполнение



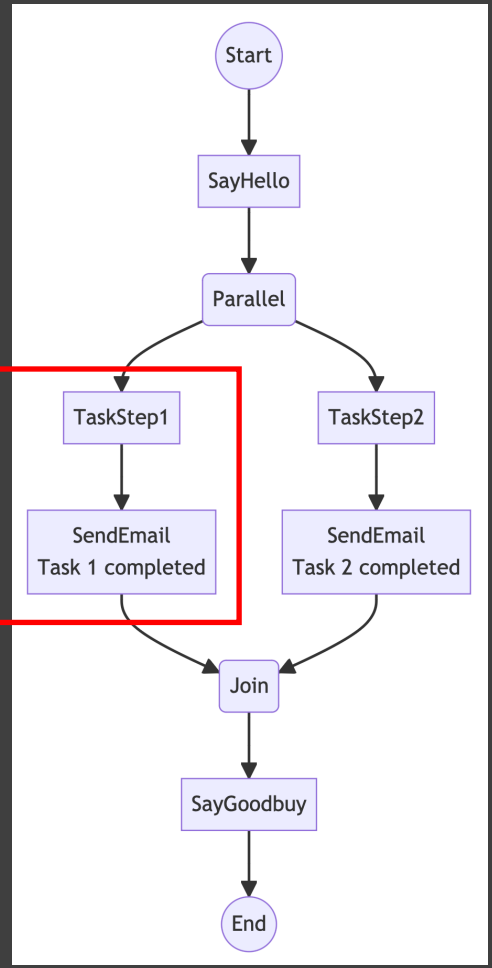
Workflow. Параллельное исполнение

```
public class ParallelWorkflow : IWorkflow
{
    public string Id => "ParallelWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<SayHello>()
            .Parallel()
            .Do(_ => _
                .StartWith<TaskStep1>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 1 completed"))
            .Do(_ => _
                .StartWith<TaskStep2>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 2 completed"))
            .Join()
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```



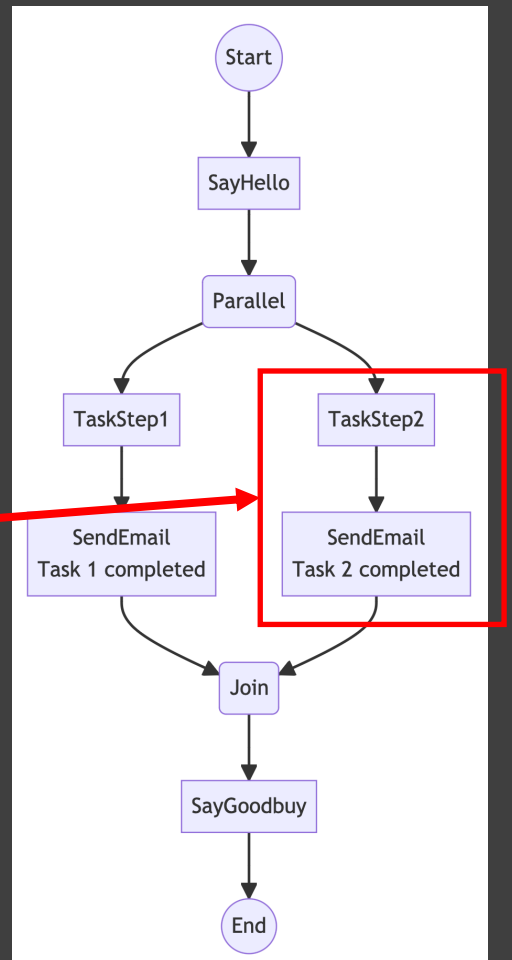
Workflow. Параллельное исполнение

```
public class ParallelWorkflow : IWorkflow
{
    public string Id => "ParallelWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<SayHello>()
            .Parallel()
            .Do(_ => _
                .StartWith<TaskStep1>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 1 completed"))
            .Do(_ => _
                .StartWith<TaskStep2>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 2 completed"))
            .Join()
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```



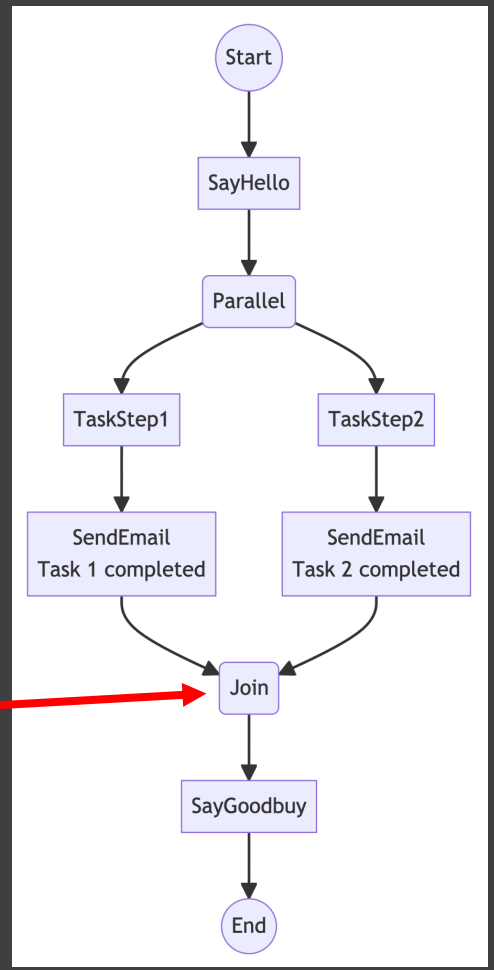
Workflow. Параллельное исполнение

```
public class ParallelWorkflow : IWorkflow
{
    public string Id => "ParallelWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<SayHello>()
            .Parallel()
            .Do(_ => _
                .StartWith<TaskStep1>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 1 completed"))
            .Do(_ => _
                .StartWith<TaskStep2>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 2 completed"))
            .Join()
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```

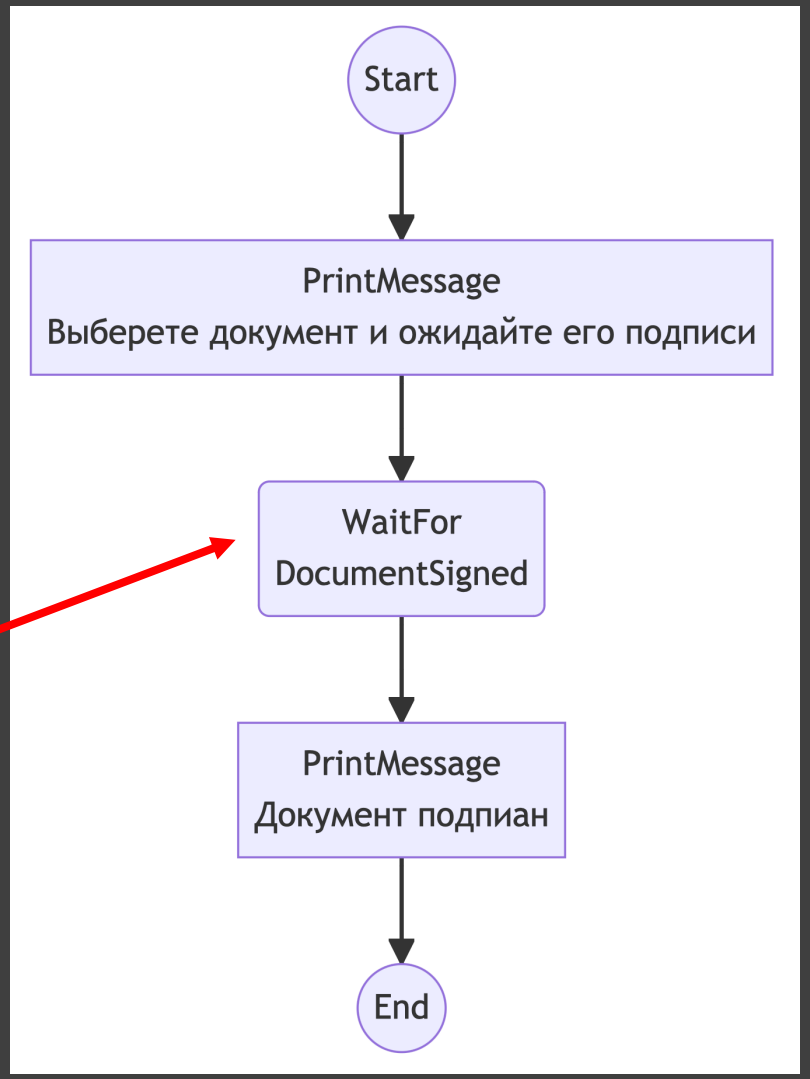


Workflow. Параллельное исполнение

```
public class ParallelWorkflow : IWorkflow
{
    public string Id => "ParallelWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<SayHello>()
            .Parallel()
            .Do(_ => _
                .StartWith<TaskStep1>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 1 completed"))
            .Do(_ => _
                .StartWith<TaskStep2>()
                .Then<SendEmail>()
                .Input(step => step.Message, _ => "Task 2 completed"))
            .Join()
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```

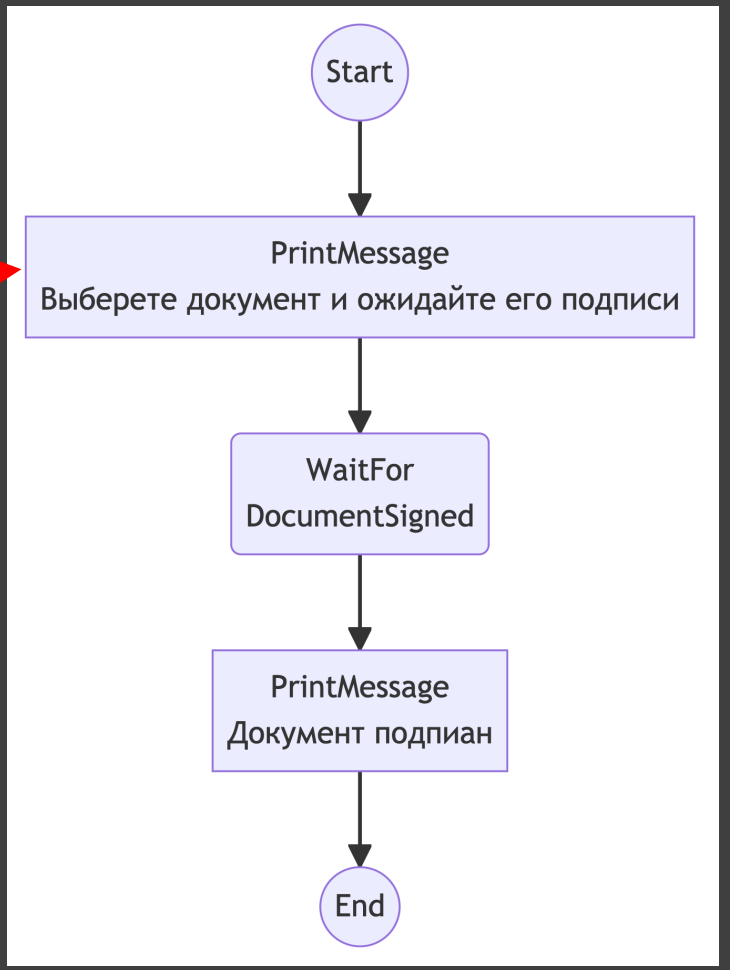


Workflow. События



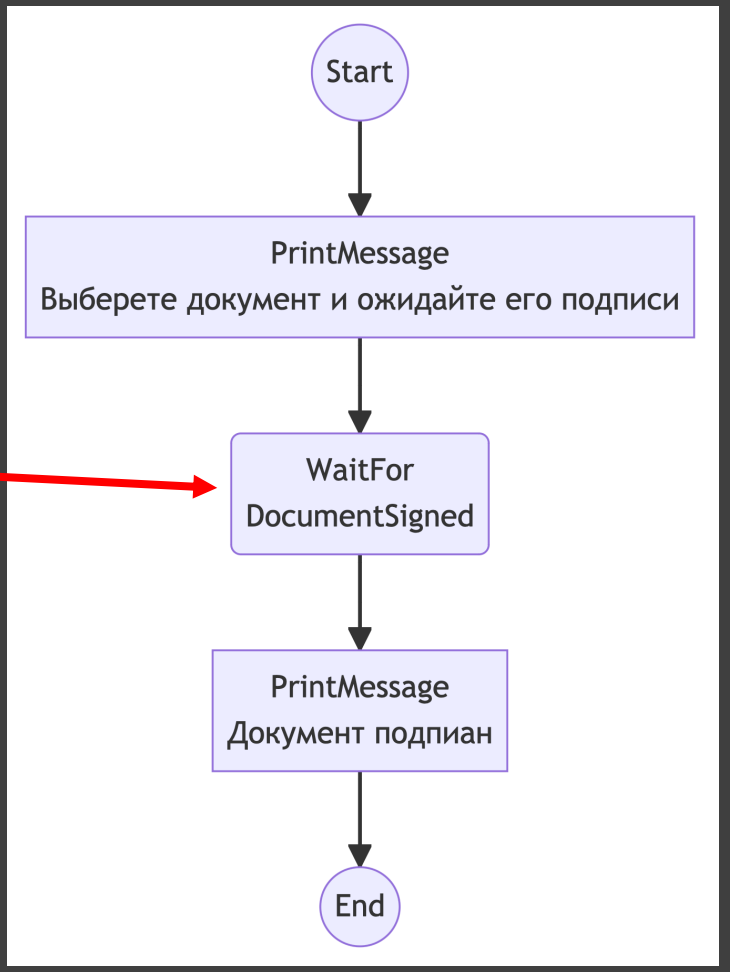
Workflow. События

```
public class SignDocumentWorkflow : IWorkflow
{
    public string Id => "SignDocumentWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<PrintMessage>()
            .Input(step => step.Message,
                _ => "Выберете документ и ожидайте его подписи")
            .WaitFor("DocumentSigned", _ => "134ан", _ => DateTime.UtcNow)
            .Then<PrintMessage>()
            .Input(step => step.Message, _ => "Документ подпиан")
            .EndWorkflow();
    }
}
```



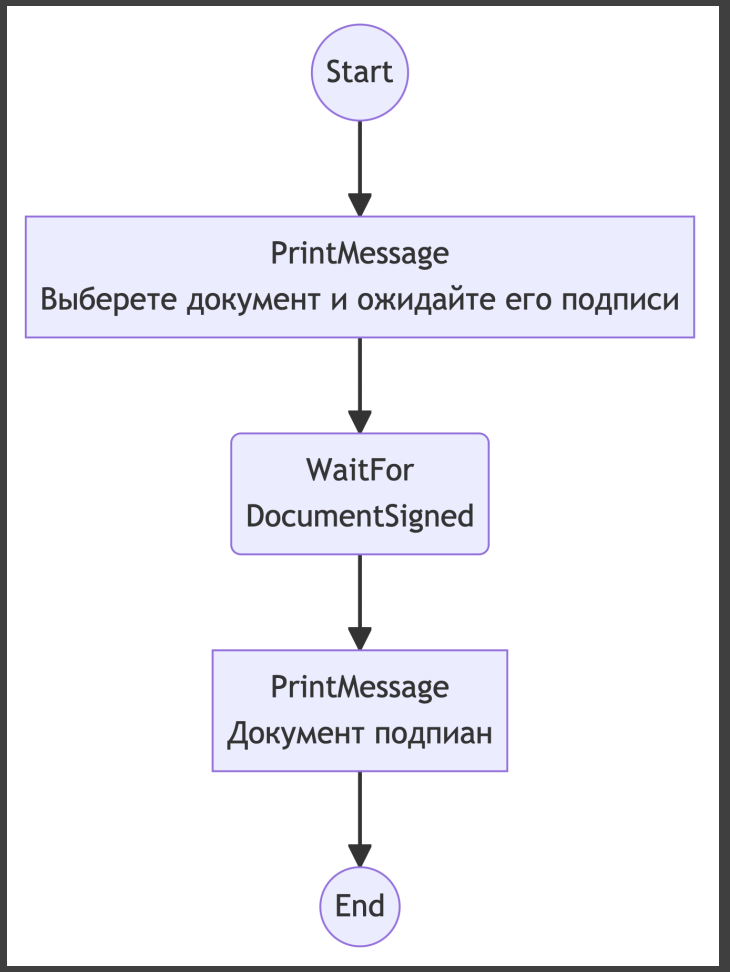
Workflow. События

```
public class SignDocumentWorkflow : IWorkflow
{
    public string Id => "SignDocumentWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<PrintMessage>()
            .Input(step => step.Message,
                _ => "Выберете документ и ожидайте его подписи")
            .WaitFor("DocumentSigned", _ => "134ап", _ => DateTime.UtcNow)
            .Then<PrintMessage>()
            .Input(step => step.Message, _ => "Документ подпиан")
            .EndWorkflow();
    }
}
```



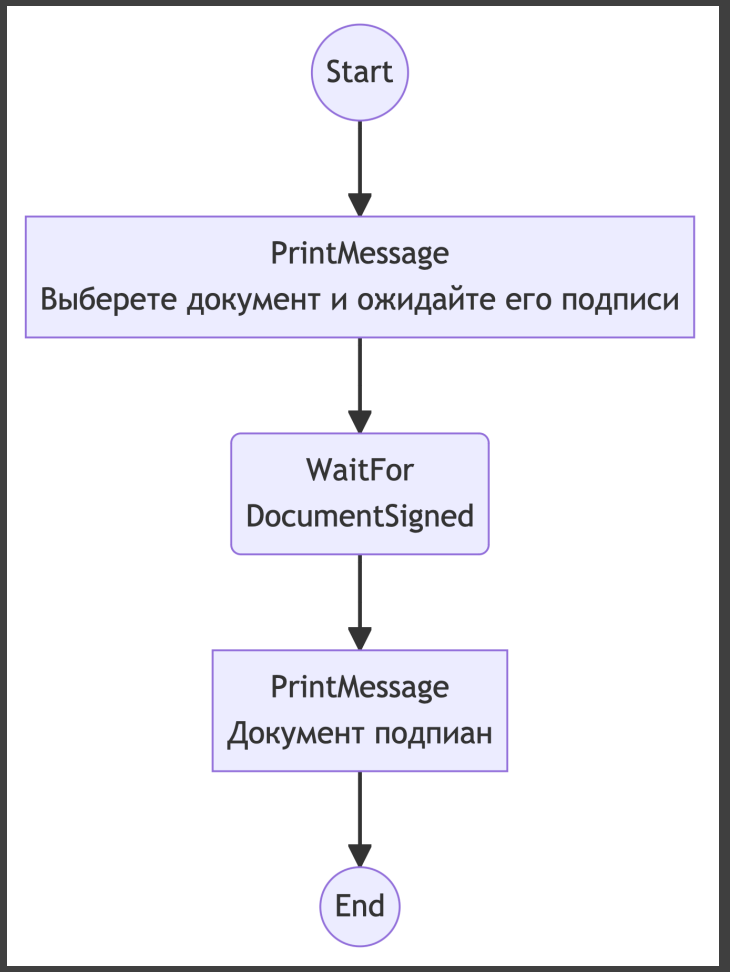
Workflow. События

```
public class SignDocumentWorkflow : IWorkflow
{
    public string Id => "SignDocumentWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<PrintMessage>()
            .Input(step => step.Message,
                _ => "Выберете документ и ожидайте его подписи")
            .WaitFor("DocumentSigned", _ => "134an", _ => DateTime.UtcNow)
            .Then<PrintMessage>()
            .Input(step => step.Message, _ => "Документ подпиан")
            .EndWorkflow();
    }
}
```



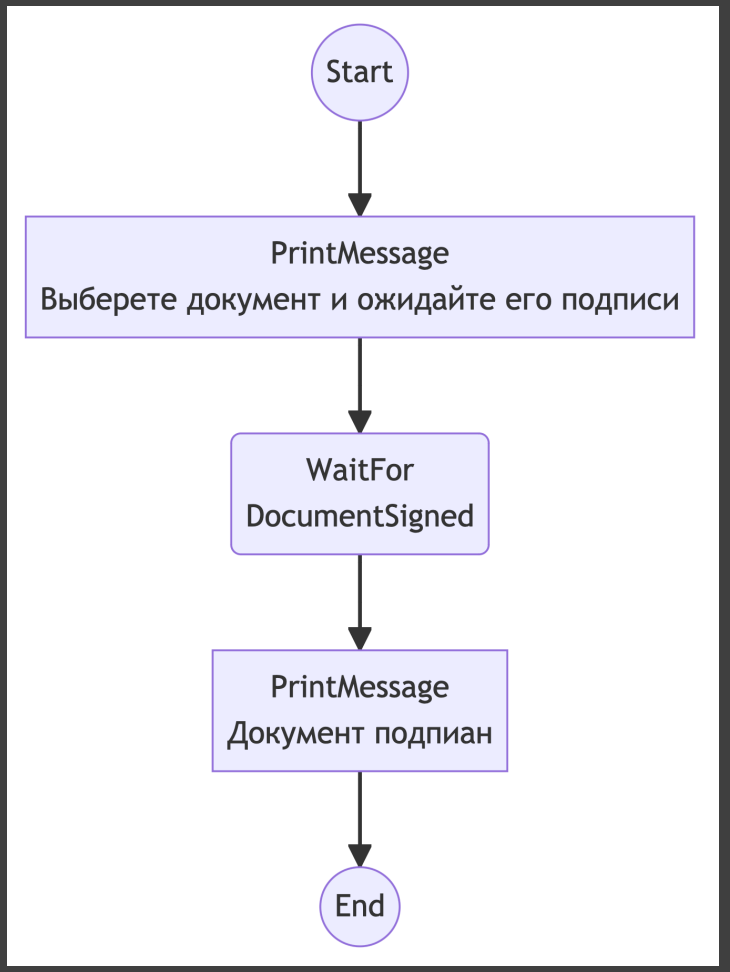
Workflow. События

```
public class SignDocumentWorkflow : IWorkflow
{
    public string Id => "SignDocumentWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<PrintMessage>()
            .Input(step => step.Message,
                _ => "Выберете документ и ожидайте его подписи")
            .WaitFor("DocumentSigned", _ => "134an", _ => DateTime.UtcNow)
            .Then<PrintMessage>()
            .Input(step => step.Message, _ => "Документ подпиан")
            .EndWorkflow();
    }
}
```



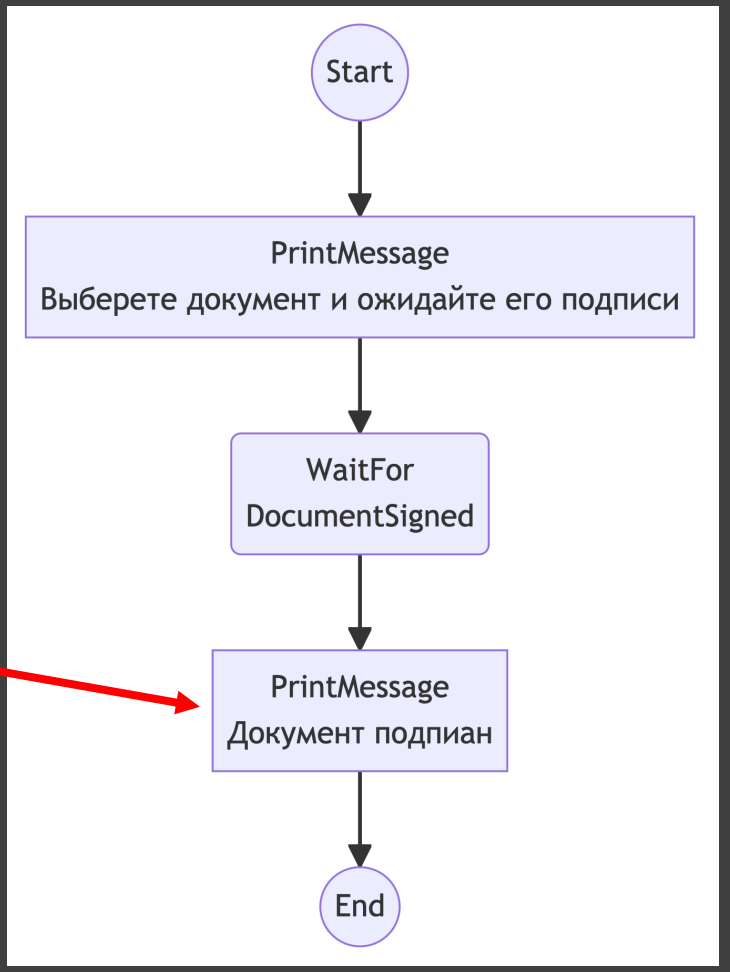
Workflow. События

```
public class SignDocumentWorkflow : IWorkflow
{
    public string Id => "SignDocumentWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<PrintMessage>()
            .Input(step => step.Message,
                _ => "Выберете документ и ожидайте его подписи")
            .WaitFor("DocumentSigned", _ => "134ан", _ => DateTime.UtcNow)
            .Then<PrintMessage>()
            .Input(step => step.Message, _ => "Документ подпиан")
            .EndWorkflow();
    }
}
```



Workflow. События

```
public class SignDocumentWorkflow : IWorkflow
{
    public string Id => "SignDocumentWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<object> builder)
    {
        builder
            .StartWith<PrintMessage>()
            .Input(step => step.Message,
                _ => "Выберете документ и ожидайте его подписи")
            .WaitFor("DocumentSigned", _ => "134ап", _ => DateTime.UtcNow)
            .Then<PrintMessage>()
            .Input(step => step.Message, _ => "Документ подпиан")
            .EndWorkflow();
    }
}
```



Workflow. Публикация события

```
workflowHost.PublishEvent("DocumentSigned", "134an", null, DateTime.UtcNow);
```

Workflow. JSON/YAML представления

```json
{
  "Id": "AddWorkflow",
  "Version": 1,
  "DataType": "MyApp.MyDataClass, MyApp",
  "Steps": [
    {
      "Id": "Hello",
      "StepType": "MyApp.HelloWorld, MyApp",
      "NextStepId": "Add"
    },
    {
      "Id": "Add",
      "StepType": "MyApp.AddNumbers, MyApp",
      "NextStepId": "Bye",
      "Inputs": {
          "Value1": "data.Value1",
          "Value2": "data.Value2"
       },
      "Outputs": {
          "Answer": "step.Result"
      }
    },
    {
      "Id": "Bye",
      "StepType": "MyApp.GoodbyeWorld, MyApp"
    }
  ]
}
```

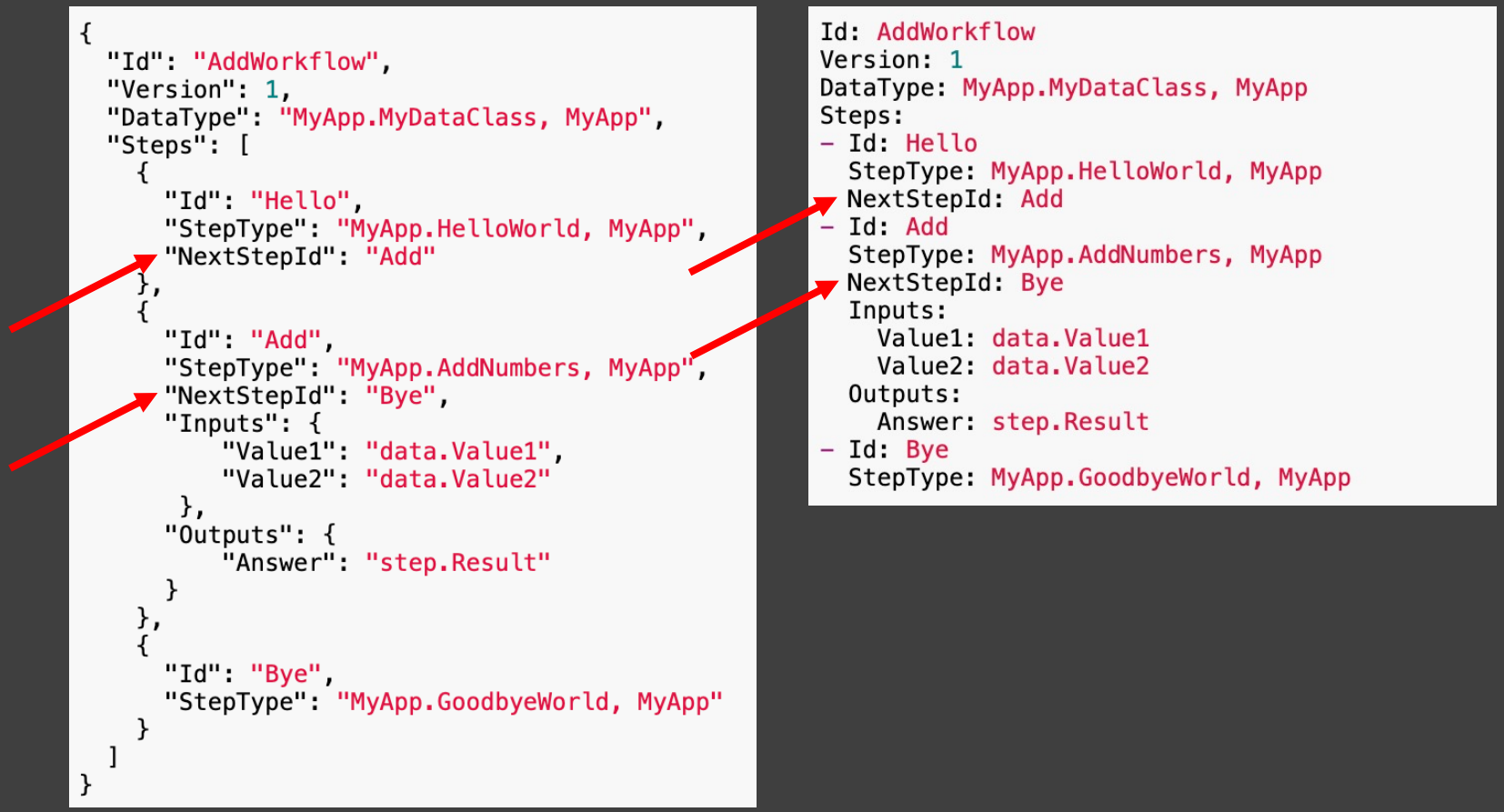
```yaml
Id: AddWorkflow
Version: 1
DataType: MyApp.MyDataClass, MyApp
Steps:
- Id: Hello
  StepType: MyApp.HelloWorld, MyApp
  NextStepId: Add
- Id: Add
  StepType: MyApp.AddNumbers, MyApp
  NextStepId: Bye
  Inputs:
    Value1: data.Value1
    Value2: data.Value2
  Outputs:
    Answer: step.Result
- Id: Bye
  StepType: MyApp.GoodbyeWorld, MyApp
```

Workflow. JSON/YAML представления

```
{
  "Id": "AddWorkflow",
  "Version": 1,
  "DataType": "MyApp.MyDataClass, MyApp",
  "Steps": [
    {
      "Id": "Hello",
      "StepType": "MyApp.HelloWorld, MyApp",
      "NextStepId": "Add"
    },
    {
      "Id": "Add",
      "StepType": "MyApp.AddNumbers, MyApp",
      "NextStepId": "Bye",
      "Inputs": {
          "Value1": "data.Value1",
          "Value2": "data.Value2"
       },
      "Outputs": {
          "Answer": "step.Result"
      }
    },
    {
      "Id": "Bye",
      "StepType": "MyApp.GoodbyeWorld, MyApp"
    }
  ]
}
```

```
Id: AddWorkflow
Version: 1
DataType: MyApp.MyDataClass, MyApp
Steps:
- Id: Hello
  StepType: MyApp.HelloWorld, MyApp
  NextStepId: Add
- Id: Add
  StepType: MyApp.AddNumbers, MyApp
  NextStepId: Bye
  Inputs:
    Value1: data.Value1
    Value2: data.Value2
  Outputs:
    Answer: step.Result
- Id: Bye
  StepType: MyApp.GoodbyeWorld, MyApp
```

Workflow. JSON/YAML представления

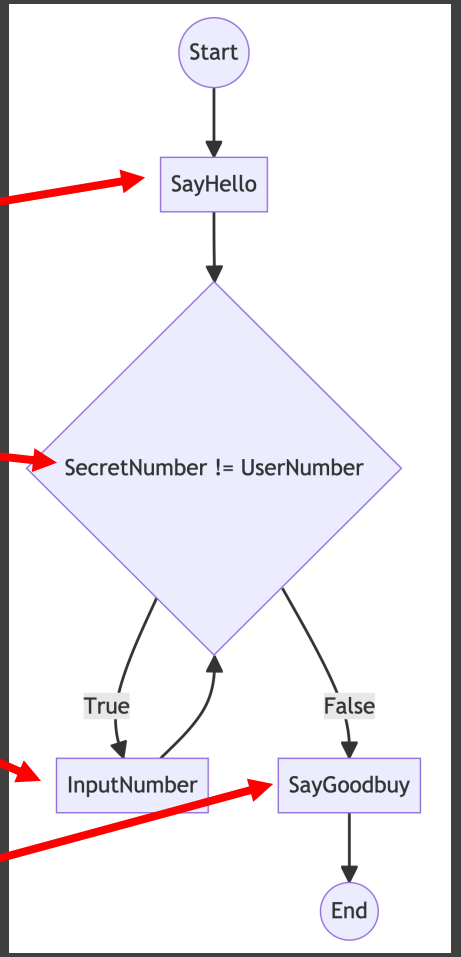


```json
{
  "Id": "AddWorkflow",
  "Version": 1,
  "DataType": "MyApp.MyDataClass, MyApp",
  "Steps": [
    {
      "Id": "Hello",
      "StepType": "MyApp.HelloWorld, MyApp",
      "NextStepId": "Add"
    },
    {
      "Id": "Add",
      "StepType": "MyApp.AddNumbers, MyApp",
      "NextStepId": "Bye",
      "Inputs": {
          "Value1": "data.Value1",
          "Value2": "data.Value2"
       },
      "Outputs": {
          "Answer": "step.Result"
      }
    },
    {
      "Id": "Bye",
      "StepType": "MyApp.GoodbyeWorld, MyApp"
    }
  ]
}
```

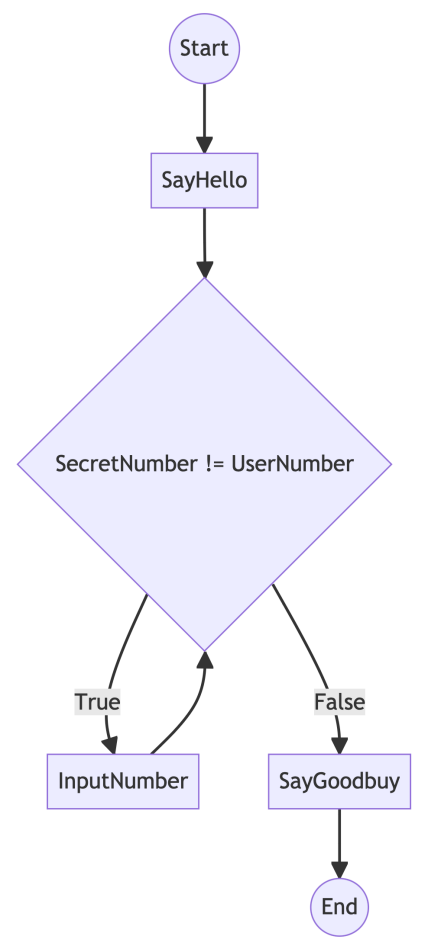
```yaml
Id: AddWorkflow
Version: 1
DataType: MyApp.MyDataClass, MyApp
Steps:
- Id: Hello
  StepType: MyApp.HelloWorld, MyApp
  NextStepId: Add
- Id: Add
  StepType: MyApp.AddNumbers, MyApp
  NextStepId: Bye
  Inputs:
    Value1: data.Value1
    Value2: data.Value2
  Outputs:
    Answer: step.Result
- Id: Bye
  StepType: MyApp.GoodbyeWorld, MyApp
```

Flowchart диаграммы

```
public class GuessNumberWorkflow : IWorkflow<GuessNumberWorkflowData>
{
    public string Id => "GuessNumberWorkflow";
    public int Version => 1;

    public void Build(IWorkflowBuilder<GuessNumberWorkflowData> builder)
    {
        builder
            .StartWith<SayHello>()
            .While(data => data.SecretNumber != data.UserNumber).Do(_ => _
                .StartWith<InputNumber>()
                .Output(data => data.UserNumber, step => step.Result))
            .Then<SayGoodbuy>()
            .EndWorkflow();
    }
}
```



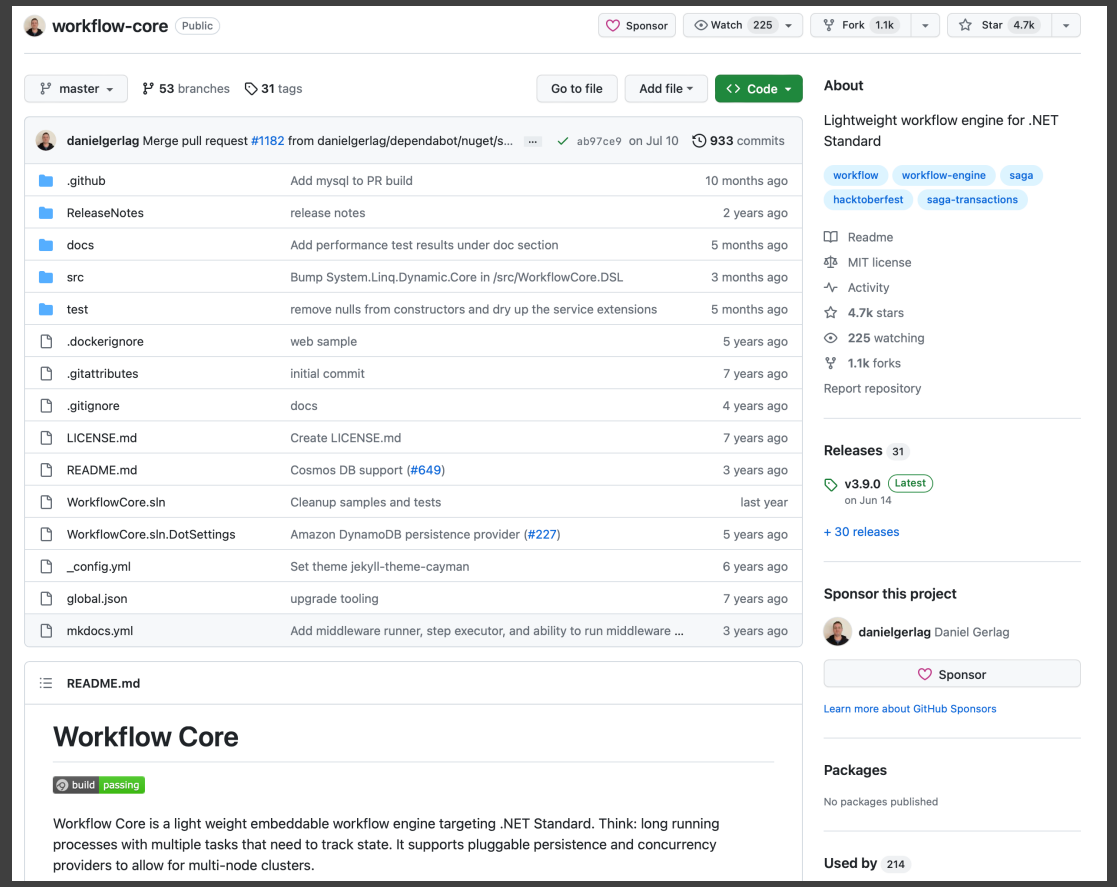
Flowchart диаграммы




```
flowchart TB
startNode((Start))
0[SayHello]
1{SecretNumber != UserNumber}
2[InputNumber]
3[SayGoodbuy]
4((End))

startNode --> 0
0 --> 1
1 -->|True| 2
1 -->|False| 3
2 --> 1
3 --> 4
```

Сложности

Сложности. Документация



Зачем, если есть

код

Сложности.
Передача параметров в шаги

```
builder
    .StartWith<InputNumber>()
    .Output(data => data.NumberA, step => step.Result)
    .Then<InputNumber>()
    .Output(data => data.NumberB, step => step.Result)
    .Then<CalculateSum>()
    .Input(step => step.NumberA, data => data.NumberA)
    .Input(step => step.NumberB, data => data.NumberB)
    .Output(data => data.CalculationResult, step => step.Result)
    .Then<SendEmail>()
    .Input(step => step.Message,
        data => data.CalculationResult.ToString())
    .EndWorkflow();
```

```
public class CalculateSum : IStepBody
{
    public int NumberA { get; set; }
    public int NumberB { get; set; }
    public int Result { get; set; }

    public Task<ExecutionResult> RunAsync(
        IStepExecutionContext context)
    {
        Result = NumberA + NumberB;
        return Task.FromResult(ExecutionResult.Next());
    }
}
```

Сложности.
Ожидание нескольких событий

Сложности.
Ожидание нескольких событий

```
builder
    .Parallel()
    .Do(branch => branch
        .StartWith<WaitFor>()
        .Input(step => step.EventName, data => data.EventName)
        .Input(step => step.EventKey, _ => "0")
        .Input(step => step.EffectiveDate, _ => DateTime.Now)
        .CancelCondition(data => !data.IsWaiting)
        .Output(data => data.Result, step => new WaitEventResult(WaitEventStatus.Received, step.EventData))
        .Output(data => data.IsWaiting, _ => false))
    .Do(branch => branch
        .Delay(data => data.WaitingTime)
        .CancelCondition(data => !data.IsWaiting)
        .Output(data => data.Result, step => new WaitEventResult(WaitEventStatus.TimedOut, null))
        .Output(data => data.IsWaiting, _ => false))
    .Join()
    .Then<PublishEvent>()
    .Input(step => step.EventName, _ => CompletedEventName)
    .Input(step => step.EventKey, data => data.EventName)
    .Input(step => step.EventData, data => data.Result)
    .EndWorkflow();
```

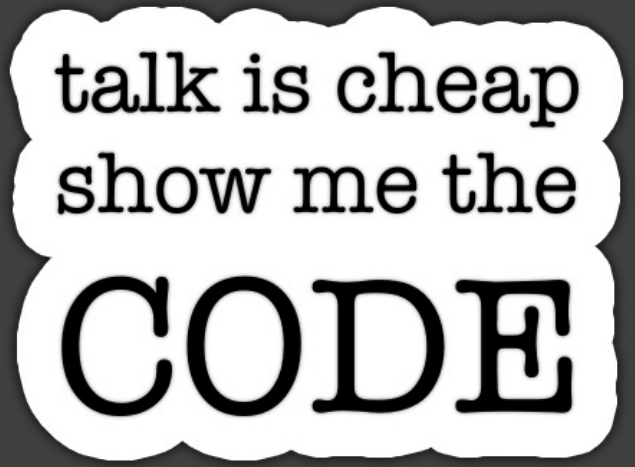
Сложности.
Ожидание нескольких событий

```
builder
    .Parallel()
    .Do(branch => branch
        .StartWith<WaitFor>()
        .Input(step => step.EventName, data => data.EventName)
        .Input(step => step.EventKey, _ => "0")
        .Input(step => step.EffectiveDate, _ => DateTime.Now)
        .CancelCondition(data => !data.IsWaiting)
        .Output(data => data.Result, step => new WaitEventResult(WaitEventStatus.Received, step.EventData))
        .Output(data => data.IsWaiting, _ => false))
    .Do(branch => branch
        .Delay(data => data.WaitingTime)
        .CancelCondition(data => !data.IsWaiting)
        .Output(data => data.Result, step => new WaitEventResult(WaitEventStatus.TimedOut, null))
        .Output(data => data.IsWaiting, _ => false))
    .Join()
    .Then<PublishEvent>()
    .Input(step => step.EventName, _ => CompletedEventName)
    .Input(step => step.EventKey, data => data.EventName)
    .Input(step => step.EventData, data => data.Result)
    .EndWorkflow();
```

Сложности.
Неуклюжий синтаксис

```
public void Build(IWorkflowBuilder<CalculationWorkflowData> builder)
{
    builder
        .StartWith<InputNumber>()
        .While(data => data.NumberA > 10).Do(_ => _
            .StartWith<InputNumber>()
            .If(data => data.NumberB > 10).Do(__ => __
                .StartWith<StepB>()
                .If(data => data.NumberA > 5).Do(___ => ___
                    .StartWith<InputNumber>())))
        .EndWorkflow();
}
```

Лайвкодинг



Что мы узнали

- Что такое Workflow
- Для решения каких задач подходит
- Какие могут возникнуть сложности, как их решать
- Как работать с библиотекой Workflow Core
- Сделали простого бота на workflow

Полезные ссылки

- https://github.com/danielgerlag/workflow-core
- https://workflow-core.readthedocs.io
- https://mermaid.js.org/syntax/flowchart.html
- https://github.com/MikD1/DotNext2023
- https://github.com/step-flow-platform/mermaid-charting-net
- https://github.com/step-flow-platform/workflow-core-flowchart

Спасибо!

Приходите еще =)