

Компонентные тесты как способ написания приемочных тестов микросервисов

Аникин Дмитрий

Ведущий бэк-енд разработчик
компании БКС



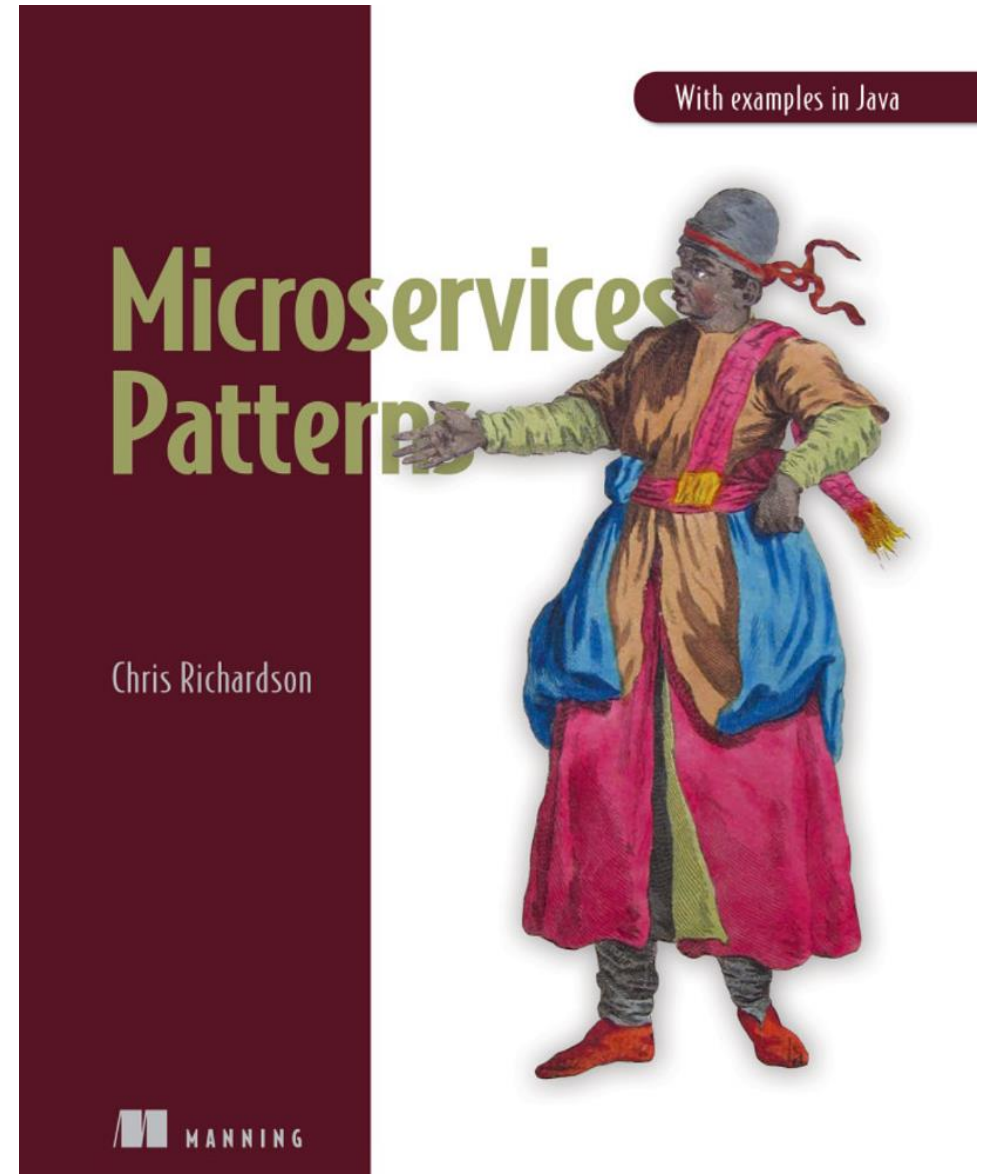
План моего доклада

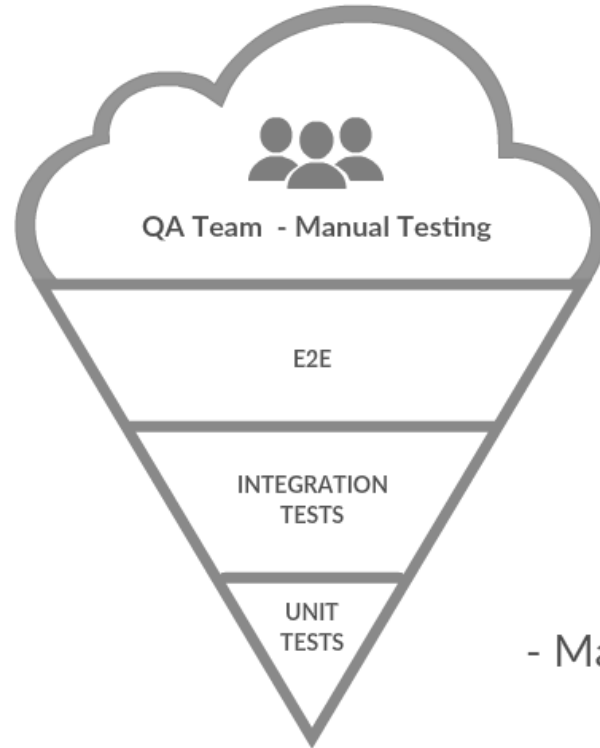
- Что такое компонентные тесты в контексте тестирования микросервисов?
- Зачем нужны компонентные тесты?
- Какое место компонентных тестов в пирамиде тестирования?
- В чем сложность компонентных тестов?
- Как реализовать компонентные тесты в .net core?
- Как превратить компонентные тесты в живые спецификации и зачем это нужно?
- Чего получилось достичь?



Терминология

Компонентные тесты - это приемочные тесты, которые тестируют поведение микросервиса, как черный ящик, в терминах его апи, в изоляции от внешнего мира.

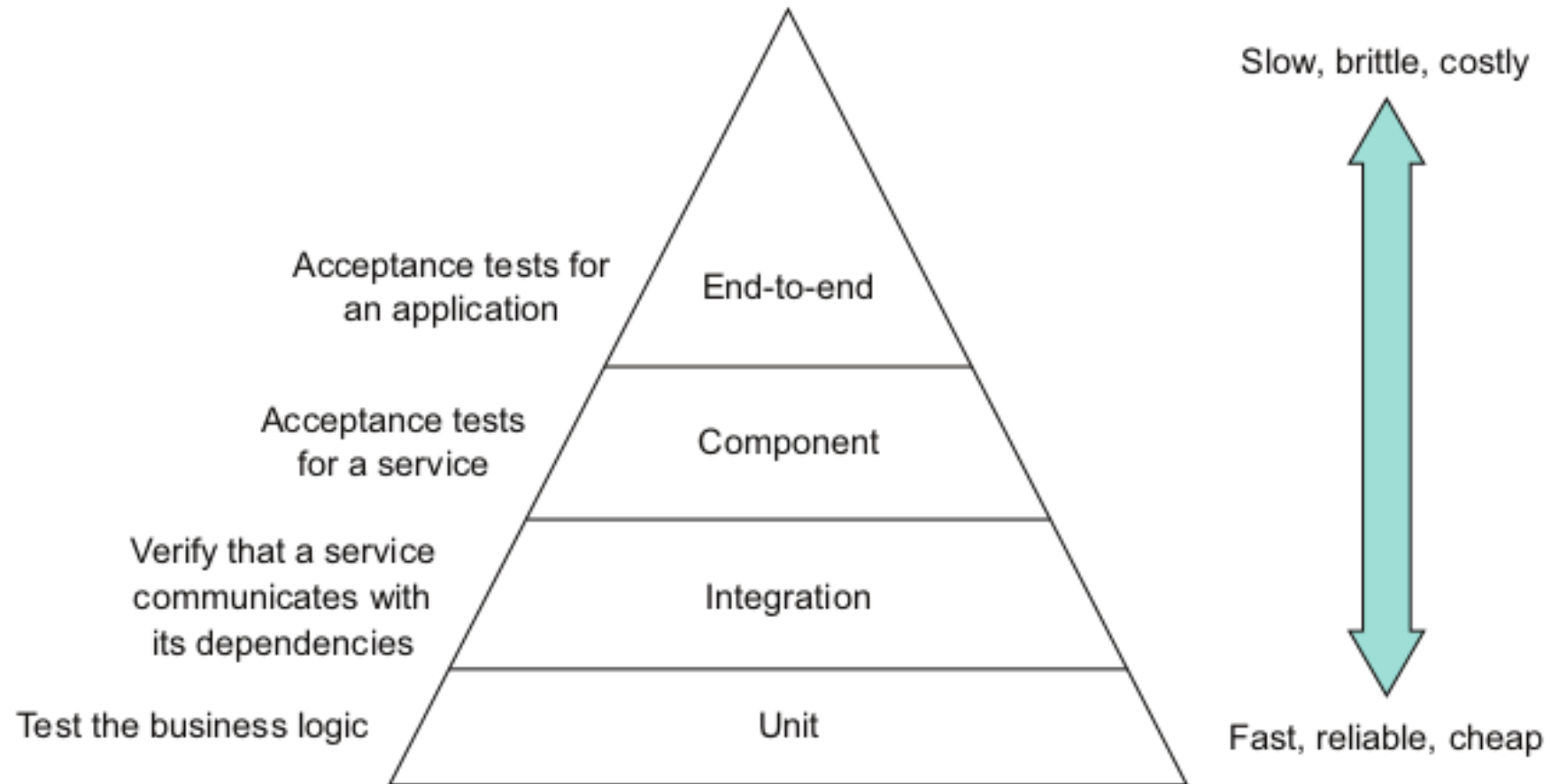




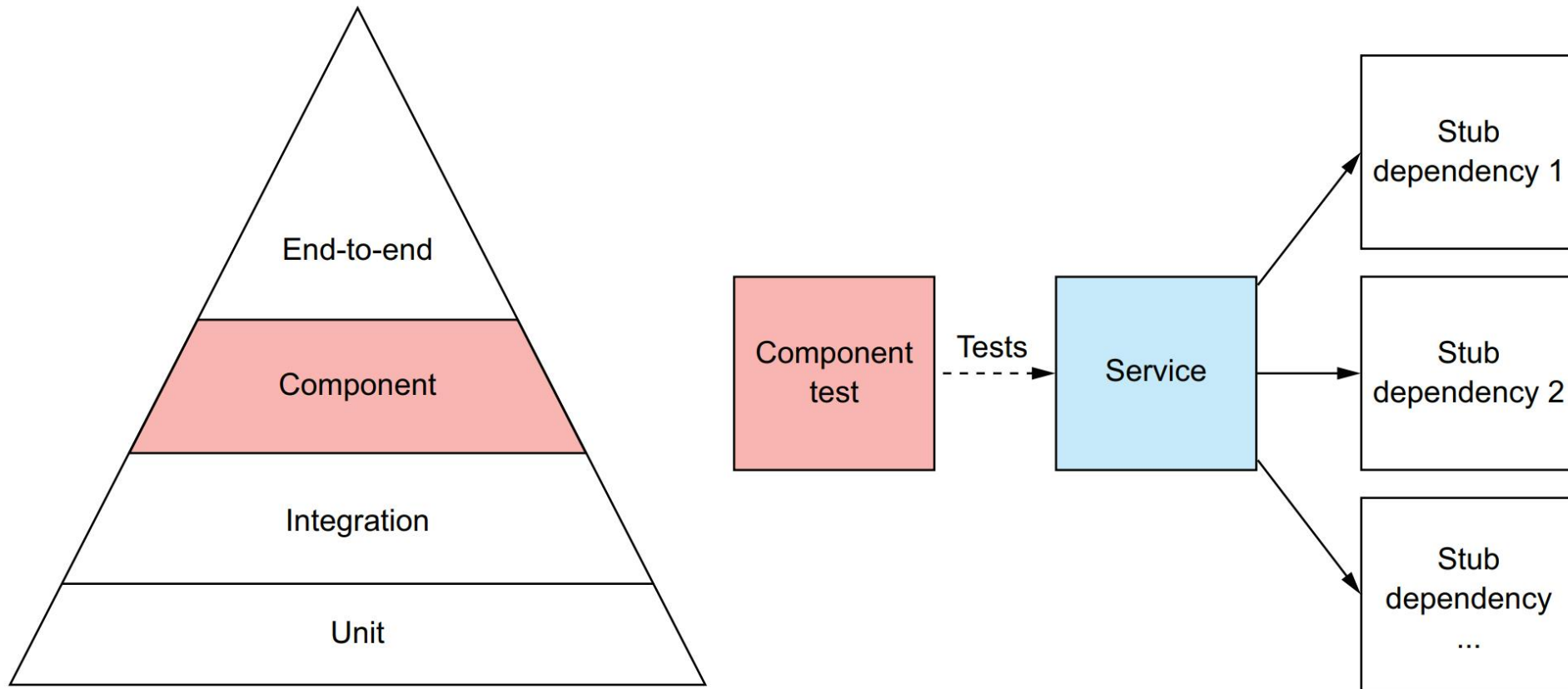
ICE CREAM CONE

- Very slow to execute
- Bugs are hard to find
- Manual Testers are swamped

Правильная пропорция тестов



Место компонентных тестов в пирамиде тестирования



Способы тестирования микросервиса в ИЗОЛЯЦИИ

In-Process

Использование In-Memory зависимостей

Плюсы

- Проще в реализации
- Быстрее работает

Минусы

- Тестирует НЕ то, что будет развернуто на проде

Out-Of-Process

Использование Docker, Mountebank

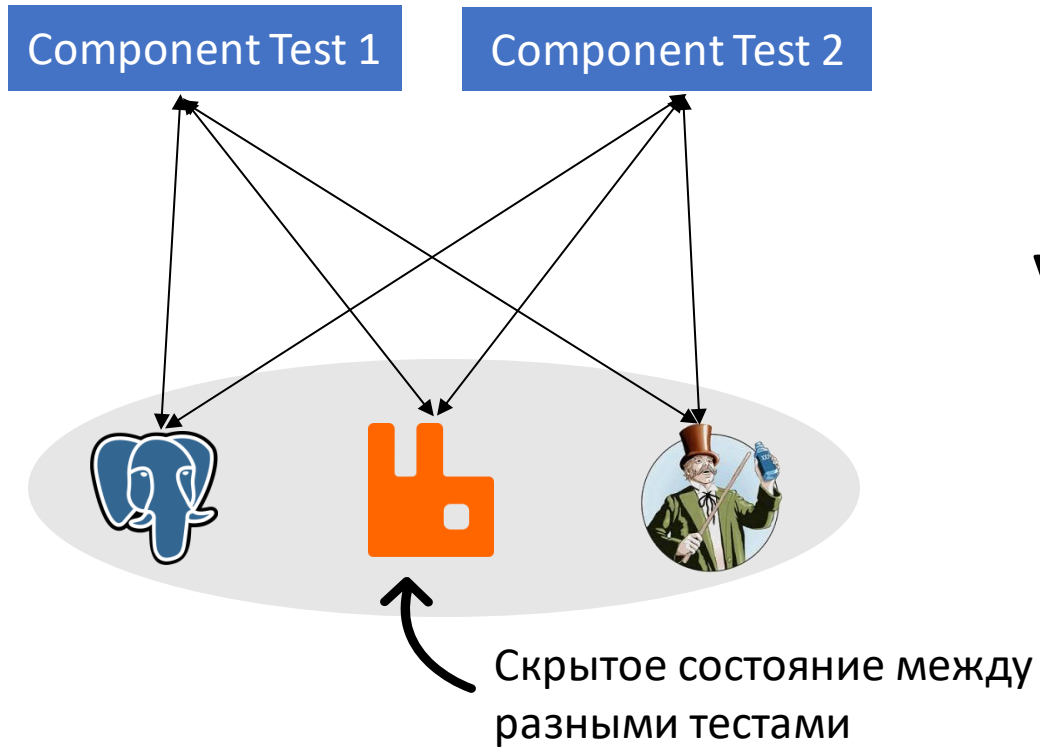
Плюсы

- Тестирует ТО, что будет развернуто на проде

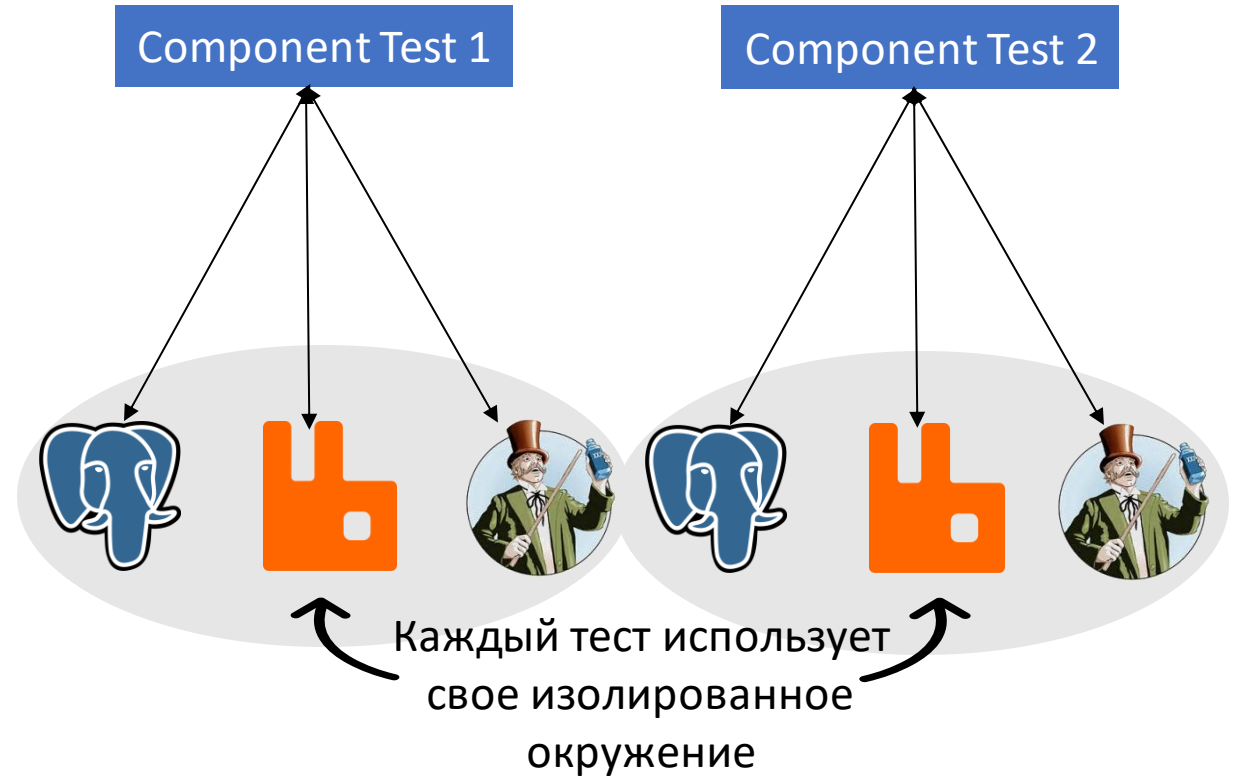
Минусы

- Сложнее в реализации
- Медленнее работает

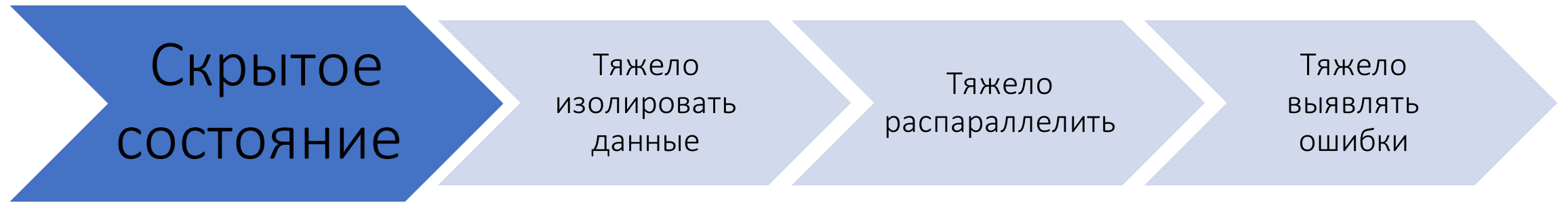
Изолированность компонентных тестов



VS



Проблемы скрытого состояния



Библиотеки для работы с docker контейнерами из кода в .net core

- testcontainers-dotnet
<https://github.com/testcontainers/testcontainers-dotnet>
- Docker.DotNet
<https://github.com/microsoft/Docker.DotNet>
- FluentDocker
<https://github.com/mariotoffia/FluentDocker>

Класс DockerContainer

```
14  ↓0  {
15
16      ...
21      public string Name => _containerService?.Name;
22      public bool Started => _containerService != null;
23      public int ReadinessTimeout { get; set; } = 30000;
24
25      protected DockerContainer(
26          string dockerImageName,
27          int containerPort,
28          Action<ContainerBuilder> configureBuilder){...}
38
39      public void Start(){...}
57
58      public void WaitForEndpoint(string endpoint){...}
75
76      public string GetExposedEndpointFor(int containerPort){...}
82      public int GetExposedPortFor(int containerPort){...}
95
96      protected void AddPortToBeExposed(int port){...}
100
101      protected void CheckContainerNotRunning(){...}
105      protected void CheckContainerRunning(bool startIfNotStarted){...}
116
117      protected virtual void BeforeStarted(){...}
120      protected virtual void OnStarted(){...}
123      ...
140  }
```

Создание docker контейнера из кода

```
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
  
protected DockerContainer(  
    string dockerImageName,  
    int containerPort,  
    Action<ContainerBuilder> configureBuilder)  
{  
    _portsToExpose = new HashSet<int> {containerPort};  
  
    ContainerBuilder = Fd  
        .UseContainer()  
        .UseImage(dockerImageName);  
  
    configureBuilder?.Invoke(ContainerBuilder);  
}
```

Запуск docker контейнера из кода

```
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
95
96
97
98
99

public void Start()
{
    if (Started) return;

    BeforeStarted();

    foreach (var portToExpose in _portsToExpose)
        ContainerBuilder.ExposePort(portToExpose);

    _containerService = ContainerBuilder.Build().Start();

    foreach (var portToExpose in _portsToExpose)
    {
        _containerService
            .ToHostExposedEndpoint($"{portToExpose}/tcp")?
            .WaitForPort();
    }

    OnStarted();
} ...

protected void AddPortToBeExposed(int port)
{
    _portsToExpose.Add(port);
}
```

Получение порта, назначенного docker'ом

```
83 public int GetExposedPortFor(int containerPort)
84 {
85     CheckContainerRunning( startIfNotStarted: false);
86
87     var ipEndpoint = _containerService.ToHostExposedEndpoint($"{containerPort}/tcp");
88
89     if (ipEndpoint == null)
90     {
91         throw new InvalidOperationException(
92             message: $"{containerPort} port of {_containerService.Name} container was not exposed");
93     }
94     return ipEndpoint.Port;
```

Пример Postgres docker контейнера

```
10 public class PostgreSqlContainer : DockerContainer
11 {
12     private const string DockerImage = "postgres:11-alpine";
13
14     public const int ContainerPort = 5432;
15     public const string Host = "127.0.0.1";
16     public const string User = "someUser";
17     public const string Password = "somePassword";
18     public const string DbName = "someDbName";
19
20     public PostgreSqlContainer(Action<ContainerBuilder> configureBuilder = null)
21         : base(DockerImage, ContainerPort, configureBuilder) {}
22
23     public string GetConnectionString(bool startIfNotStarted = false){...}
24
25     protected override void BeforeStarted()
26     {
27         ContainerBuilder.WithEnvironment(
28             $"POSTGRES_PASSWORD={Password}",
29             $"POSTGRES_USER={User}",
30             $"POSTGRES_DB={DbName}");
31     }
32
33     protected override void OnStarted()
34     {
35         WaitForDatabaseReady();
36     }
37
38     private void WaitForDatabaseReady(){...}...
39
40
41
42
43
44
45
46
47
48
49
50
51
52 }
79
```


Ожидание, когда база станет доступной

```
52 private void WaitForDatabaseReady()  
53 {  
54     const int timeoutInSeconds = 10;  
55     const int sleepDurationInSeconds = 2;  
56  
57     Policy  
58         .Timeout(TimeSpan.FromSeconds(timeoutInSeconds))  
59         .Wrap(Policy  
60             .Handle<NpgsqlException>()  
61             .Or<SocketException>()  
62             .WaitAndRetryForever(_ => TimeSpan.FromSeconds(sleepDurationInSeconds)))  
63         .ExecuteAndCapture(CheckDatabaseReady);  
64 }  
65  
66 private void CheckDatabaseReady(){...}
```

Получение строки подключения к базе

```
23 public string GetConnectionString(bool startIfNotStarted = false)
24 {
25     CheckContainerRunning(startIfNotStarted);
26
27     return new NpgsqlConnectionStringBuilder
28     {
29         Host = Host,
30         Port = GetExposedPortFor(ContainerPort),
31         Database = DbName,
32         Username = User,
33         Password = Password,
34         Pooling = false
35     }
36     .ConnectionString;
37 }
```

Mountebank



Возможности mountebank'a:

- мокирование API на протоколах tcp, http, https, smtp;
- мокирование неограниченного количества API одновременно;
- гибкое переопределение логики mock-API прямо во время тестов используя конфигурационный API mountebank'a;
- проксирование запросов в API внешнего сервиса, сохранение ответов и возможность их последующего использования в mock-API;
- клиентские библиотеки для большинства языков программирования (C#, Java, JS, Python, Ruby, и многие другие).

Конфигурация импостера

```
11 public class SlackImposter : HttpImposter
12 {
13     public const int ContainerPort = 8091;
14     public const string ServiceName = "Slack";
15
16     public const string ApiKey = "mytoken";
17     public const string FeedbackChannel = "channelId";
18
19     public SlackImposter() : base(ContainerPort, ServiceName, recordRequests: true){...}
22
23     public SlackImposter WithPostMessageFake(string text, SendMessageResult response){...}
42
43     public SlackImposter WithUploadFileFake(UploadFileResult result){...}
62 }
```

Создание заглушек

```
23 public SlackImposter WithPostMessageFake(string text, SendMessageResult response)
24 {
25     var httpPredicateFields = new HttpPredicateFields
26     {
27         Method = Method.Post,
28         Path = "api/chat.postMessage",
29         Headers = new Dictionary<string, string>
30         {
31             {"Authorization", $"Bearer {ApiKey}"}
32         },
33         RequestBody = new SendMessageRequest(FeedbackChannel, message: text)
34     };
35     var matchesPredicate = new MatchesPredicate<HttpPredicateFields>(httpPredicateFields);
36
37     AddStub().On(matchesPredicate)
38         .ReturnsJson(HttpStatusCode.OK, response);
39
40     return this;
41 }
```

Регистрация импостеров в Mountebank

```
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
53  
54  
55  
56  
57  
58  
59
```

```
public MountebankContainer RegisterImposter<TServiceImposter>(
    Action<TServiceImposter> configureImposter = null)
    where TServiceImposter : Imposter, new()
{
    CheckContainerNotRunning();

    var serviceImposter = new TServiceImposter();

    serviceImposter = (TServiceImposter) _imposters.GetOrAdd(
        serviceImposter.Name, serviceImposter);

    configureImposter?.Invoke(serviceImposter);

    AddPortToBeExposed(serviceImposter.Port);

    return this;
}...

protected override void OnStarted()
{
    var mountebankClient = CreateMountebankClient();

    mountebankClient.Submit(_imposters.Values);
}
```

RabbitMq container

```
7 public class RabbitMqContainer : DockerContainer
8 {
9     private const string DockerImage = "heidiks/rabbitmq-delayed-message-exchange";
10
11     public const int ContainerPort = 5672;
12     private const int ManagementPort = 15672;...
13
14     public RabbitMqContainer(Action<ContainerBuilder> configureBuilder = null)
15         : base(DockerImage, ContainerPort, configureBuilder){...}
16
17     public IModel CreateChannel(Action<IModel> declare, bool startIfNotStarted = false)
18     {
19         CheckContainerRunning(startIfNotStarted);
20
21         if (_rabbitChannel != null)
22         {
23             declare(_rabbitChannel);
24             return _rabbitChannel;
25         }
26
27         var factory = new ConnectionFactory
28         {
29             HostName = "localhost",
30             Port = GetExposedPortFor(ContainerPort)
31         };
32
33         _rabbitConnection = factory.CreateConnection();
34         _rabbitChannel = _rabbitConnection.CreateModel();
35
36         declare(_rabbitChannel);
37
38         return _rabbitChannel;
39     }...
40 }
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
```


Kafka container

```
8 public class KafkaContainer : DockerContainer
9 {
10     private const string DockerImage = "wurstmeister/kafka";
11
12     public const int ContainerPort = 9094;...
13
14
15
16     public KafkaContainer(Action<ContainerBuilder> configureBuilder = null)
17         : base(DockerImage, ContainerPort, configureBuilder){...}
18
19
20
21
22     protected override void BeforeStarted(){...}
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41     public IProducer<Null, string> CreateProducer(){...}
42
43     public IConsumer<Ignore, string> CreateConsumer(){...}
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59     public string GetBootstrapServer(bool startIfNotStarted = false){...}...
60
61
62
63
64
65
66
67
68
69
70
71 }
72
73 class ZookeeperContainer : DockerContainer
74 {
75     private const string DockerImage = "wurstmeister/zookeeper";
76
77     public const int ContainerPort = 2181;
78
79     public ZookeeperContainer(Action<ContainerBuilder> configureBuilder = null)
80         : base(DockerImage, ContainerPort, configureBuilder) {}
81 }
```

Настройка портов Kafka

```
22  ↑ 0
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
```

```
protected override void BeforeStarted()
{
    _zookeeperContainer.Start();

    ContainerBuilder
        .Link(_zookeeperContainer.Name)
        .Mount("/var/run/docker.sock", "/var/run/docker.sock", MountType.ReadWrite)
        .WithEnvironment(
            "KAFKA_ZOOKEEPER_CONNECTION_TIMEOUT_MS=5000",
            "HOSTNAME_COMMAND=\"docker info | grep ^Name: | cut -d' ' -f 2\"",
            $"PORT_COMMAND=\"docker port `hostname` {ContainerPort}/tcp | cut -d: -f 2\"",
            $"KAFKA_LISTENERS=INSIDE://:9092,OUTSIDE://:{ContainerPort}",
            "KAFKA_ADVERTISED_LISTENERS=INSIDE://:9092,OUTSIDE://_{HOSTNAME_COMMAND}:_{PORT_COMMAND}",
            "KAFKA_LISTENER_SECURITY_PROTOCOL_MAP=INSIDE:PLAINTEXT,OUTSIDE:PLAINTEXT",
            "KAFKA_INTER_BROKER_LISTENER_NAME=INSIDE",
            "KAFKA_LEADER_IMBALANCE_CHECK_INTERVAL_SECONDS=1",
            $"KAFKA_ZOOKEEPER_CONNECT={_zookeeperContainer.Name}:{ZookeeperContainer.ContainerPort}");
}
```

Agile Testing

- Общение с владельцами продукта
- Тесное взаимодействие тестировщиков с разработчиками
- Вся команда вовлечена в обеспечение качества

BDD сценарии как критерии приемки

Функционал: Открытие счета ИИС (Позитивные тест-кейсы)

Я как пользователь приложения Мой брокер с генсогом могу открыть ИИС, чтобы получать налоговый вычет.

Сценарий: Создание договора для открытия ИИС

Дано клиент без ИИС

Когда клиент отправил заявку на создание договора

Тогда статус создания договора должен стать 'Succeed'

И генеральное соглашение должно быть
зарегистрировано

И клиент может получить договор

Сценарий: Согласование договора

Дано договор ожидает согласования

Когда клиент согласился подписать договор

Тогда клиенту будет отправлено смс с
кодом подтверждения

Сценарий: Подписание договора

Дано договор ожидает подписания

И клиенту отправлено смс с кодом '111111'

Когда клиент подписал договор с кодом '111111'

Тогда код успешно прошел проверку

И статус заключения договора должен стать 'Succeed'

И генеральное соглашение должно быть подписано

Что из себя представляет шаг BDD сценария?

Сценарий: Подписание договора

Дано договор ожидает подписания

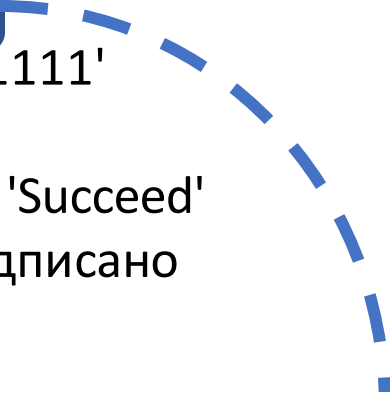
И клиенту отправлено смс с кодом '111111'

Когда клиент подписал договор с кодом '111111'

Тогда код успешно прошел проверку

И статус заключения договора должен стать 'Succeed'

И генеральное соглашение должно быть подписано



```
[Given("клиенту отправлено смс с кодом '(.*)'")]  
public void GivenSmsWasSentToClient(int code)  
{  
    _pushSmsContext.ConfigureCheckOtpFake(code);  
}
```

Пример двух связанных шагов

```
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

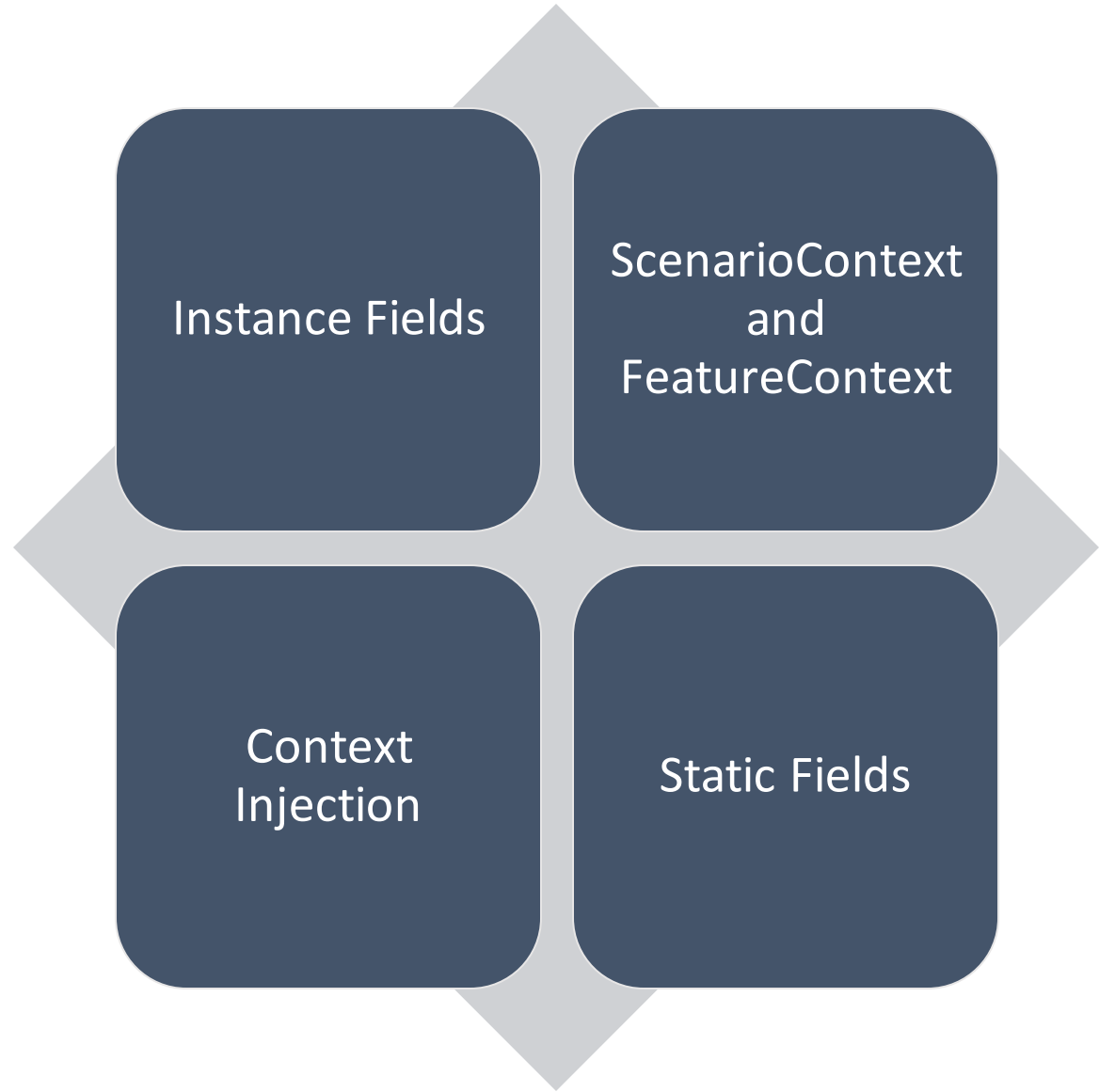
[Binding]
public class SearchSteps
{
    private IEnumerable<Book> _books;

    [When(@"I perform a simple search on '(.*)'")]
    public void WhenIPerformASimpleSearchOn(string searchTerm)
    {
        var controller = new CatalogController();

        _books = controller.Search(searchTerm).Value;
    }

    [Then(@"the book list should exactly contain book '(.*)'")]
    public void ThenTheBookListShouldExactlyContainBook(string title)
    {
        _books.Should().Contain(b => b.Title == title);
    }
}
```

Sharing Data Between Bindings



ScenarioContext

```
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

[Binding]
public class SearchSteps
{
    private readonly ScenarioContext _scenarioContext;

    public SearchSteps(ScenarioContext scenarioContext)
    {
        _scenarioContext = scenarioContext;
    }

    [When(@"I perform a simple search on '(.*)'")]
    public void WhenIPerformASimpleSearchOn(string searchTerm)
    {
        var controller = new CatalogController();

        _scenarioContext.Add("books", controller.Search(searchTerm).Value);
    }

    [Then(@"the book list should exactly contain book '(.*)'")]
    public void ThenTheBookListShouldExactlyContainBook(string title)
    {
        var books = _scenarioContext.Get<IEnumerable<Book>>("books");

        books.Should().Contain(b => b.Title == title);
    }
}
```

Context injection

```
9      public class CatalogSearchContext
10     {
11         public IEnumerable<Book> Books { get; private set; }
12
13         public IEnumerable<Book> Search(string searchTerm)
14         {
15             var controller = new CatalogController();
16
17             return Books = controller.Search(searchTerm).Value;
18         }
19     }
20
21     [Binding]
22     public class SearchSteps
23     {
24         private readonly CatalogSearchContext _catalogSearchContext;
25
26         public SearchSteps(CatalogSearchContext catalogSearchContext)
27         {
28             _catalogSearchContext = catalogSearchContext;
29         }
30
31         [When(@"I perform a simple search on '(.)'")]
32         public void WhenIPerformASimpleSearchOn(string searchTerm)
33         {
34             _catalogSearchContext.Search(searchTerm);
35         }
36
37         [Then(@"the book list should exactly contain book '(.)'")]
38         public void ThenTheBookListShouldExactlyContainBook(string title)
39         {
40             _catalogSearchContext.Books.Should().Contain(b => b.Title == title);
41         }
42     }
```

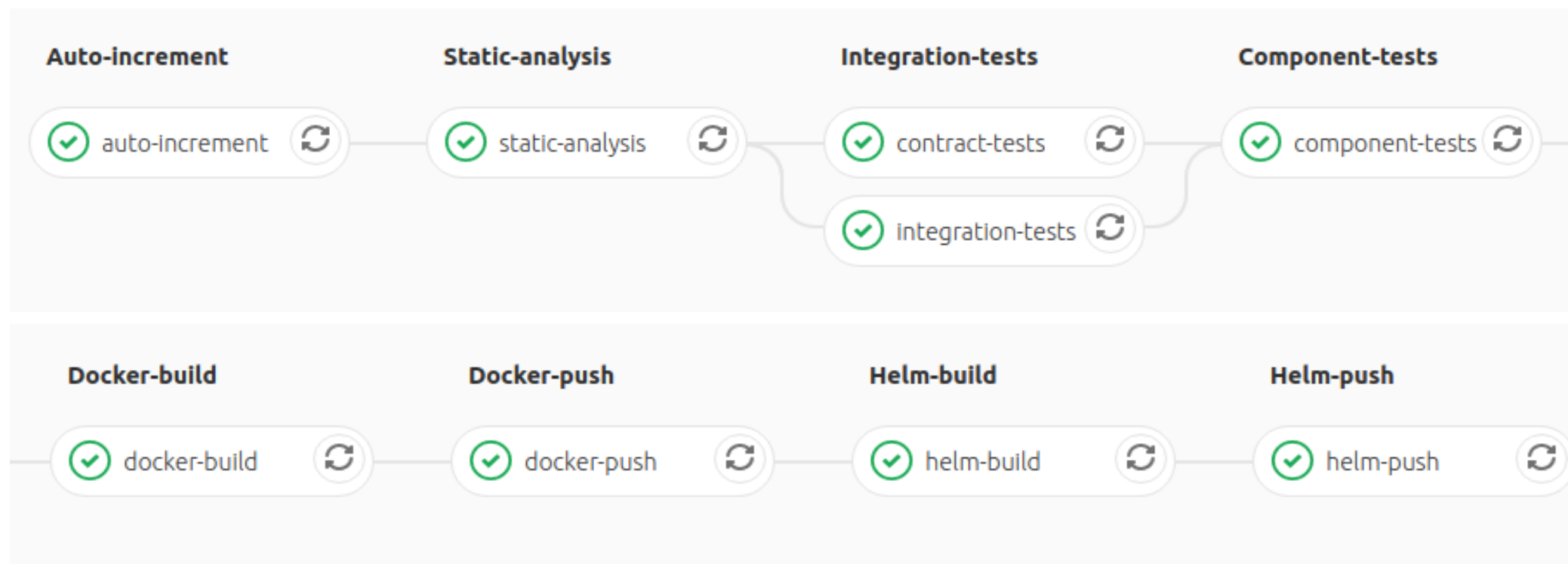
WebApplicationContext

```
11 public class WebApplicationContext : WebApplicationContextBase<Startup>
12 {
33     public WebApplicationContext()
34     {
35         ZipkinContainer = new ZipkinContainer();
36         PostgreSQLContainer = new PostgreSQLContainer();
37         MountebankContainer = new MountebankContainer()
38             .RegisterImposter<CatalogServiceImposter>()
39             .RegisterImposter<KeyCloakServiceImposter>(...)
42             .RegisterImposter<SendEmailServiceImposter>(...);
44         KafkaContainer = new KafkaContainer();
45     }
46
47     protected override Dictionary<string, string> ConfigureEnvironments()
48     {
49         ...
56         return new Dictionary<string, string>()
57         {
58             ...
60             ["TECHNICALACCOUNTSETTINGS:AUTHORIZATIONSERVICEHOST"] =
61                 MountebankContainer.GetExposedEndpointFor(KeyCloakServiceImposter.ContainerPort),
62
63             ["TECHNICALACCOUNTSETTINGS:CREDENTIALS:CLIENTID"] = KeyCloakServiceImposter.ClientId,
64             ["TECHNICALACCOUNTSETTINGS:CREDENTIALS:CLIENTSECRET"] = KeyCloakServiceImposter.ClientSecret,
65             ["TECHNICALACCOUNTSETTINGS:CREDENTIALS:USERNAME"] = KeyCloakServiceImposter.UserName,
66             ["TECHNICALACCOUNTSETTINGS:CREDENTIALS:PASSWORD"] = KeyCloakServiceImposter.Password,
67             ...
70             ["ZIPKINSETTINGS:HTTPOURL"] = ZipkinContainer.GetExposedEndpointFor(ZipkinContainer.ContainerPort),
71             ["ZIPKINSETTINGS:SAMPLINGPROBABILITY"] = "1.0",
72             ["ZIPKINSETTINGS:DISABLEREPORTING"] = "true",
73
74             ["DBSETTINGS:PORT"] = PostgreSQLContainer.GetExposedPortFor(PostgreSQLContainer.ContainerPort).ToString(),
75             ["DBSETTINGS:USER"] = PostgreSQLContainer.User,
76             ["DBSETTINGS:PASSWORD"] = PostgreSQLContainer.Password,
77             ["DBSETTINGS:HOST"] = PostgreSQLContainer.Host,
78             ["DBSETTINGS:DBNAME"] = PostgreSQLContainer.DbName,
79
80             ["KAFKASETTINGS:SERVERS"] = KafkaContainer.GetBootstrapServer(),
81             ["KAFKASETTINGS:FAILEDRETRYCOUNT"] = "3",
82             ["KAFKASETTINGS:FAILEDRETRYINTERVAL"] = "60",
83             ["EMAILSETTINGS:SENDER"] = "TuSvc@bcs.ru", ...
86         };
87     }
88 }
```

Specflow Hooks

```
6 [Binding]
7 public class Hooks
8 {
9     private readonly WebApplicationContext _webApplicationContext;
10    private readonly ScenarioContext _scenarioContext;
11
12    public Hooks(WebApplicationContext webApplicationContext, ScenarioContext scenarioContext)
13    {
14        _webApplicationContext = webApplicationContext;
15        _scenarioContext = scenarioContext;
16    }
17
18    [AfterScenarioBlock]
19    public void AfterGivenBlock()
20    {
21        if (_scenarioContext.CurrentScenarioBlock == ScenarioBlock.Given)
22        {
23            _webApplicationContext.Start();
24        }
25    }
26 }
```

Пайплайн CI



Покрытие
микросервиса
тестами

