
OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование

Не забыть включить запись!

Меня хорошо видно && слышно?

Ставьте +, если все хорошо
Напишите в чат, если есть проблемы

Тестирование микросервисов

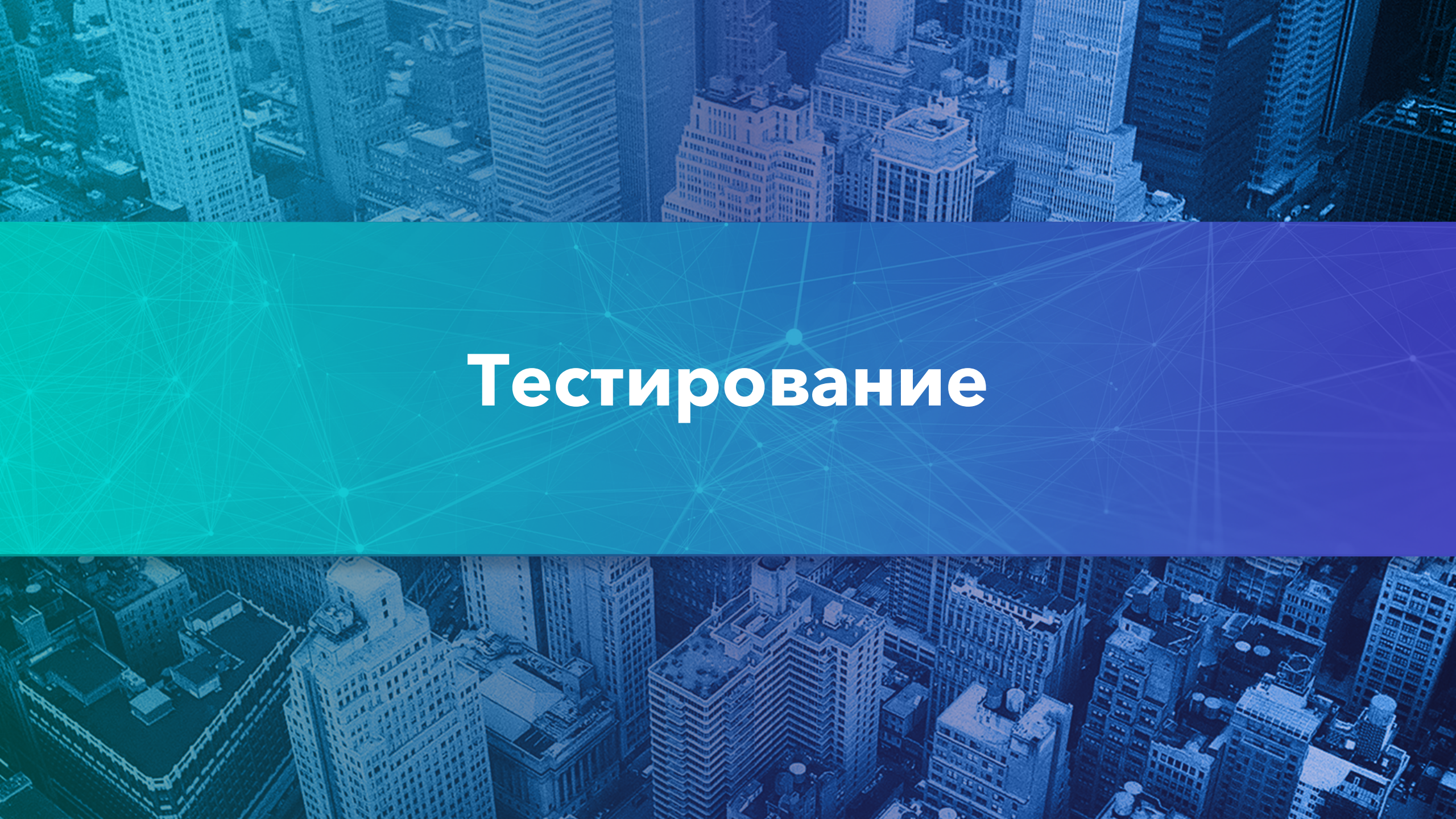
Антон Телышев

Senior Golang Developer at Domclick

t.me/@antonboom

О чём мы сегодня поговорим?

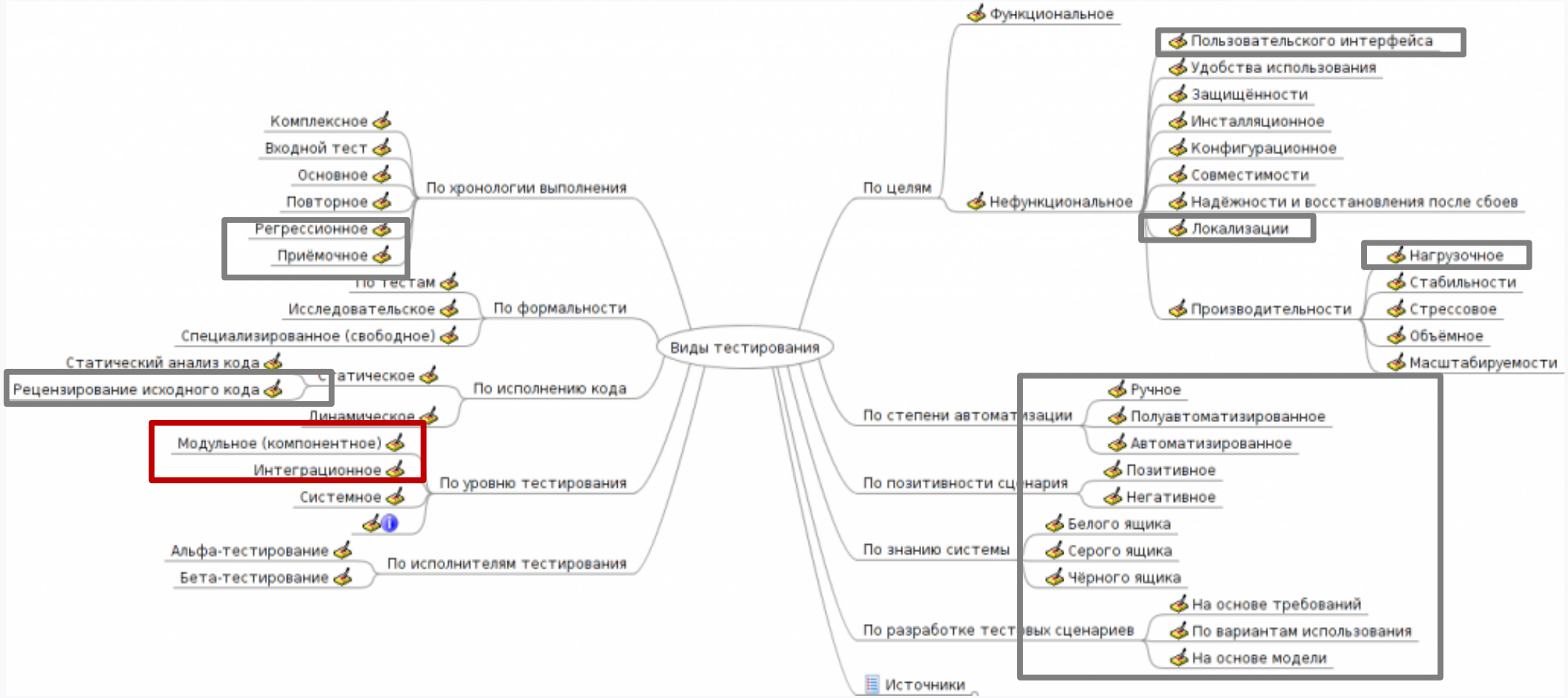
1. Тесты. Какие бывают. Зачем? Отличия?
2. Интеграционное тестирование
3. TDD и BDD
4. Язык Gherkin
5. godog и запуск тестов через docker compose



Тестирование

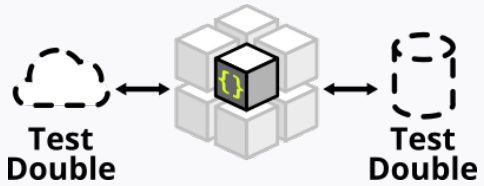
Тестирование

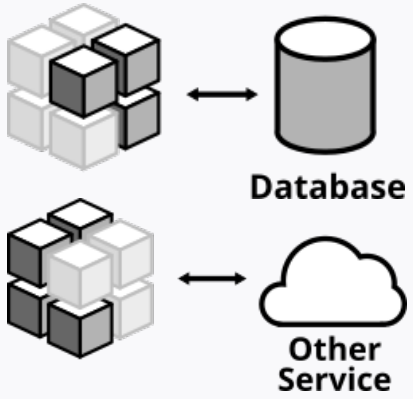
Тестирование программного обеспечения – проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.



https://habr.com/ru/post/279535/

Интеграционные тесты VS Модульные тесты





Тесты	Цель	Требует	Скорость	Сложность	Нужна настройка
Юнит-тесты	класс/метод	исходный код	очень быстро	низкая	нет
Интеграционные тесты	компонент/ сервис	часть работающей системы	медленно	средняя	да

https://habr.com/ru/post/358950/

https://habr.com/ru/post/358178/

Зачем писать тесты?

1. Проверить себя, уменьшить вероятность ошибки
2. Сделать свой код более устойчивым к поспешным изменениям
3. Помочь потомкам работать с твоим кодом, понимать его логику
4. А что думаете вы?

Библиотеки для unit-тестирования в Go

- Testify (assert, mock, require)

https://github.com/stretchr/testify

- gomock (кодогенерация)

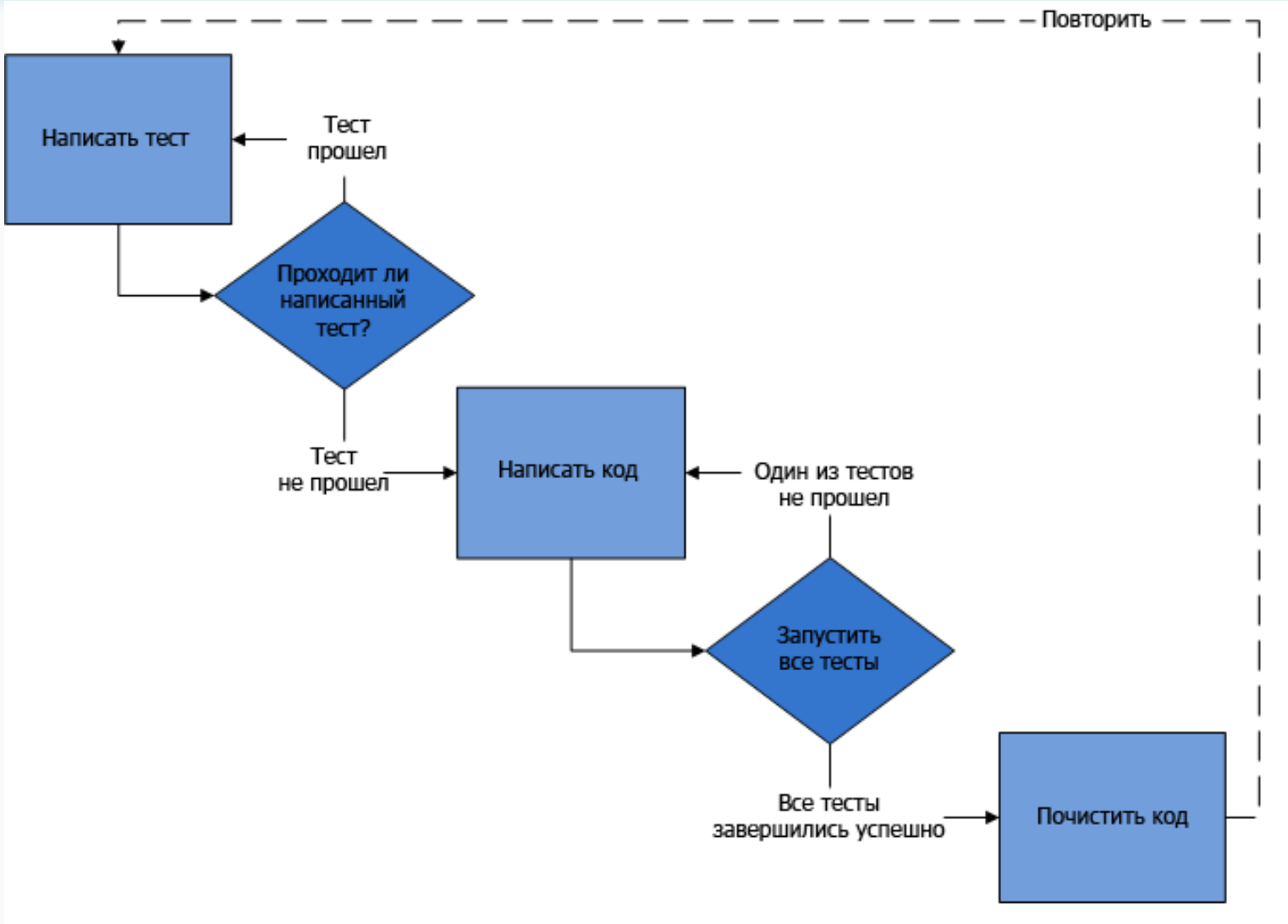
https://github.com/golang/mock

- Monkey (https://en.wikipedia.org/wiki/Monkey_patch)

https://github.com/bouk/monkey


Вопросы?

Test-Driven Development (TDD)



https://ru.wikipedia.org/wiki/Разработка_через_тестирование

TDD: Пример

Задача:

Написать функцию ***Join***, которая склеивает слайс байт в строку, игнорируя пробелы.

TDD: Пример

1. Пишем тесты

2. Запускаем
и получаем ошибку
компиляции

```
$ go test
# test [test.test]
./main_test.go:11:23: undefined: Join
./main_test.go:15:27: undefined: Join
./main_test.go:19:27: undefined: Join
```

```go
package main

import (
    "testing"

    "github.com/stretchr/testify/assert"
)

func TestJoin(t *testing.T) {
    t.Run( name: "Join empty slice", func(t *testing.T) {
        assert.Equal(t,  expected: "", Join(make([]rune, 0)))
    })

    t.Run( name: "Join slice without spaces", func(t *testing.T) {
        assert.Equal(t,  expected: "abcd", Join([]rune{'a', 'b', 'c', 'd'}))
    })

    t.Run( name: "Join slice with spaces", func(t *testing.T) {
        assert.Equal(t,  expected: "abcd", Join([]rune{'a', 'b', ' ', 'c', 'd', ' '}))
    })
}
```

TDD: Пример

3. Реализуем функцию

4. Запускаем тесты

```
$ go test
--- FAIL: TestJoin (0.00s)
    --- FAIL: TestJoin/Join_slice_with_spaces (0.00s)
        main_test.go:29:
                Error Trace:    main_test.go:29
                Error:          Not equal:
                                expected: "abcd"
                                actual  : "ab cd "

                                Diff:
                                --- Expected
                                +++ Actual
                                @@ -1 +1 @@
                                -abcd
                                +ab cd
                Test:           TestJoin/Join_slice_with_spaces
FAIL
exit status 1
FAIL    test    0.012s
```

```go
package main

import (
    "strings"
    "testing"

    "github.com/stretchr/testify/assert"
)

// Join joins rune array into string with spaces ignoring
func Join(arr []rune) string {
    builder := strings.Builder{}
    for _, ch := range arr {
        builder.WriteRune(ch)
    }
    return builder.String()
}

func TestJoin(t *testing.T) {
    t.Run( name: "Join empty slice", func(t *testing.T) {
        assert.Equal(t,  expected: "", Join(make([]rune, 0)))
    })

    t.Run( name: "Join slice without spaces", func(t *testing.T) {
        assert.Equal(t,  expected: "abcd", Join([]rune{'a', 'b', 'c', 'd'}))
    })

    t.Run( name: "Join slice with spaces", func(t *testing.T) {
        assert.Equal(t,  expected: "abcd", Join([]rune{'a', 'b', ' ', 'c', 'd', ' '}))
    })
}
```

TDD: Пример

5. Ищем ошибку: мы забыли игнорировать пробелы!

6. Исправляем код

7. Запускаем тесты

```
$ go test
PASS
ok      test    0.012s
```

```go
package main

import (
    "strings"
    "testing"

    "github.com/stretchr/testify/assert"
)

// Join joins rune array into string with spaces ignoring
func Join(arr []rune) string {
    builder := strings.Builder{}
    for _, ch := range arr {
        if ch != ' ' {
            builder.WriteRune(ch)
        }
    }
    return builder.String()
}

func TestJoin(t *testing.T) {
    t.Run( name: "Join empty slice", func(t *testing.T) {
        assert.Equal(t,  expected: "", Join(make([]rune, 0)))
    })

    t.Run( name: "Join slice without spaces", func(t *testing.T) {
        assert.Equal(t,  expected: "abcd", Join([]rune{'a', 'b', 'c', 'd'}))
    })

    t.Run( name: "Join slice with spaces", func(t *testing.T) {
        assert.Equal(t,  expected: "abcd", Join([]rune{'a', 'b', ' ', 'c', 'd', ' '}))
    })
}
```

Test-Driven Development (TDD)

- Часто можно увидеть в резюме
- Ломает мышление - тяжело писать тесты на то, чего еще нет ☺
- Помогает рассмотреть задачу с нескольких сторон, выделить основные рабочие сценарии, определить интерфейсы взаимодействия
- Необходимо применять при работе с малоизученным кодом, а также при починке бага

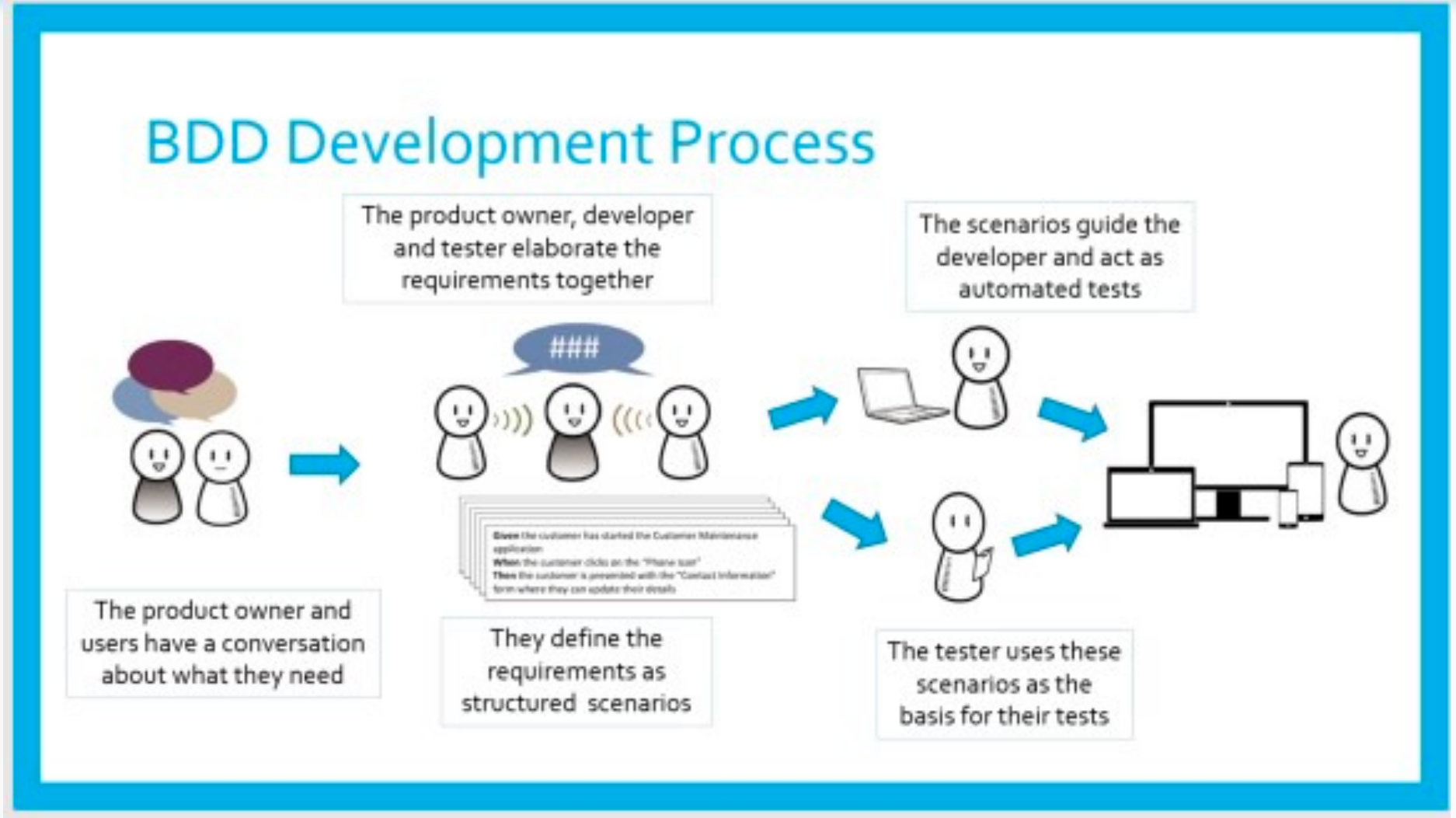
https://ru.wikipedia.org/wiki/Разработка_через_тестирование

Фиксим баги через TDD

- Вам сообщили о проблеме и вы экстренно её решили.
- Оставив всё как есть, вы написали тест на решение и он, конечно же, прошёл.
- Через какое-то время проблема повторилась. Почему?

https://ru.wikipedia.org/wiki/Разработка_через_тестирование

Behavior-Driven Development (BDD)



https://en.wikipedia.org/wiki/Behavior-driven_development

BDD

- В чём TDD? В том, что перед написанием теста необходимо описать желаемый результат от новой фичи на предметно-ориентированном, часто естественном, языке

- Описание идёт через спецификацию поведения:
  Заголовок, Описание, Сценарии

- Стандарт для спецификации de facto – язык Gherkin

- Наиболее известная компания, продвигающая фреймворки для BDD - **Cucumber**

- Вроде бы BDD позволяет бизнесу быть ближе к программистам, но что думаете сами? ☺

Язык Gherkin

```
# Comment
@tag
Feature: Eating too many cucumbers may not be good for you

  Eating too much of anything may not be good for you.

  Scenario: Eating a few is no problem
    Given Alice is hungry
    When she eats 3 cucumbers
    Then she will be full
```

https://cucumber.io/docs


Вопросы?

Возвращаясь к интеграционному тестированию

Для интеграционных тестов нужна вся (или частично) работающая система. Какие варианты?

1. Долго и мучительно поднимаем сервисы, базу, кеши и пр. локально

2. У нас есть виртуалка или админы любезно предоставили нам тестовое окружение, куда мы можем раскатиться

3. **docker-compose:**
   - поднимаем всю инфраструктуру без особого труда
   - не храним состояние между запусками, если не хотим
   - остается проблема с сервисами, которые ходят во внешнюю сеть (стороннее API и пр.), как решить?

https://ru.wikipedia.org/wiki/Разработка_через_тестирование

docker-compose

Полезные команды:

- `docker-compose [-f file] up [-d] [-build] [--exit-code-from service]`
- `docker-compose [-f file] down`
- `docker-compose logs [-f service]`
- `docker-compose ps [-a]`
- `docker-compose [-f file] run service [command]`
- `docker-compose [-f file] exec service [command]`

https://ru.wikipedia.org/wiki/Разработка_через_тестирование

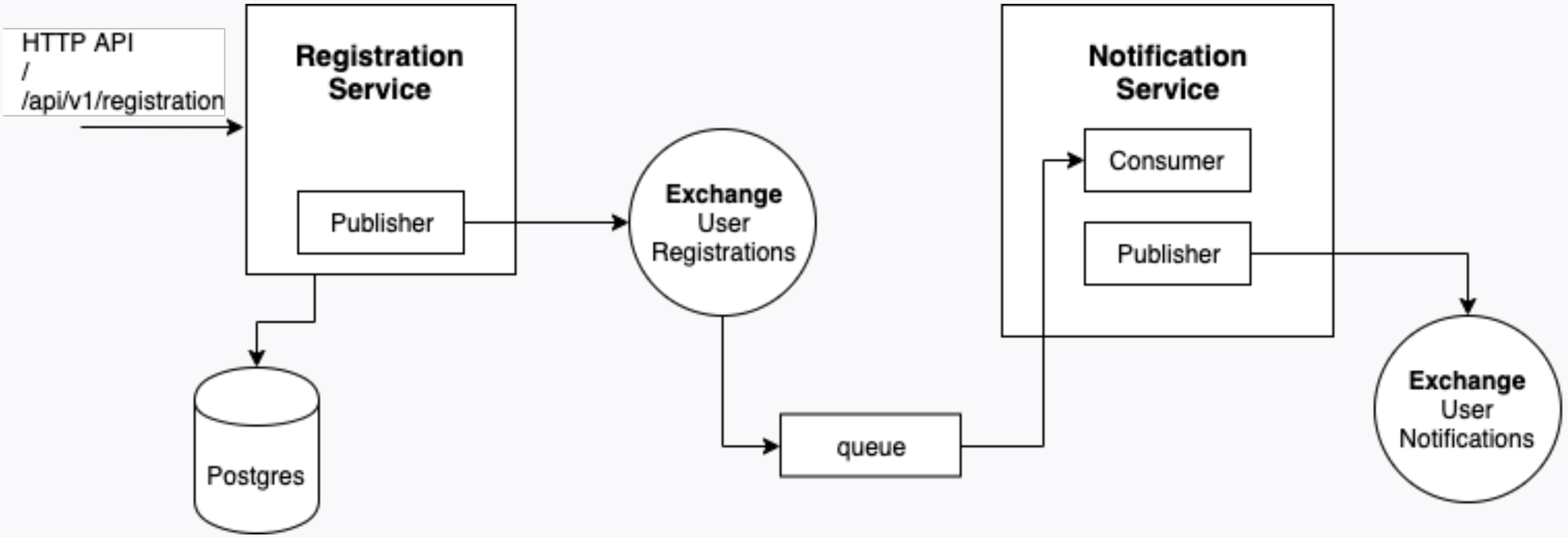
Пример

1. Клиент API посылает запрос на регистрацию пользователя в RegistrationService

2. RegistrationService сохраняет пользователя в базу и публикует событие, что произошла новая регистрация

3. NotificationService уведомляет пользователя о регистрации (например смс, email и пр.) и публикует событие, что такой-то пользователь был проинформирован

Пример


HTTP API
/
/api/v1/registration
Registration Service
Publisher
Postgres
Exchange User Registrations
queue
Notification Service
Consumer
Publisher
Exchange User Notifications

Как выглядит возможный тест на Gherkin?

```
История: Отсылка email-уведомления

  Как клиент API сервиса регистрации
  Чтобы понимать, что пользователю приходит подтверждение регистрации
  Я хочу получать события из соответствующей очереди

  Сценарий: Получаем событие от сервиса уведомлений
       Когда я отсылаю POST-запрос с пользовательским JSON в сервис регистрации
       Тогда ответ от сервиса должен быть 200 OK
       И я должен получить событие из очереди, содержащее email-пользователя
```

Реализуем?

Реализация примера:
https://github.com/Antonboom/go-integration-testing-example

Для BDD используем godog (читайте внимательно README):
https://github.com/DATA-DOG/godog

Пройдите опрос

https://otus.ru/polls/4918/

До новых встреч!
Приходите на следующие занятия

Антон Телышев
Senior Golang Developer at Domclick
t.me/@antonboom