



# Mocks

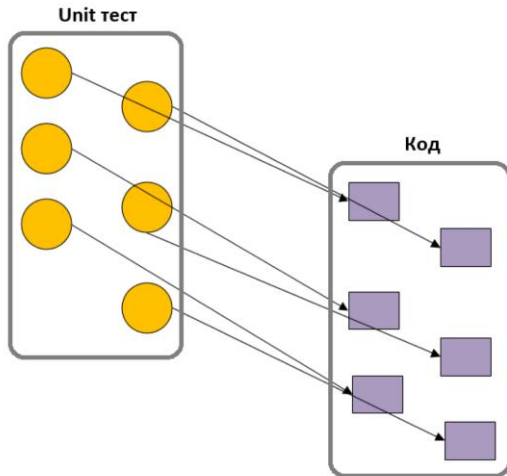
# План занятия

Методология .....	3
Проблематика .....	3
Решаемые проблемы .....	5
Существующие подходы .....	5
Stubs .....	6
Mocks .....	6
Fake .....	7
Python .....	8
Unittest.mock .....	8
Pytest-mock .....	9
Требования к решению .....	11
Выводы .....	11
Mountebank .....	12
Как оно работает .....	13
Примеры .....	14
Как работать с mountebank в Python? .....	15
Заключение .....	17
Материалы .....	17

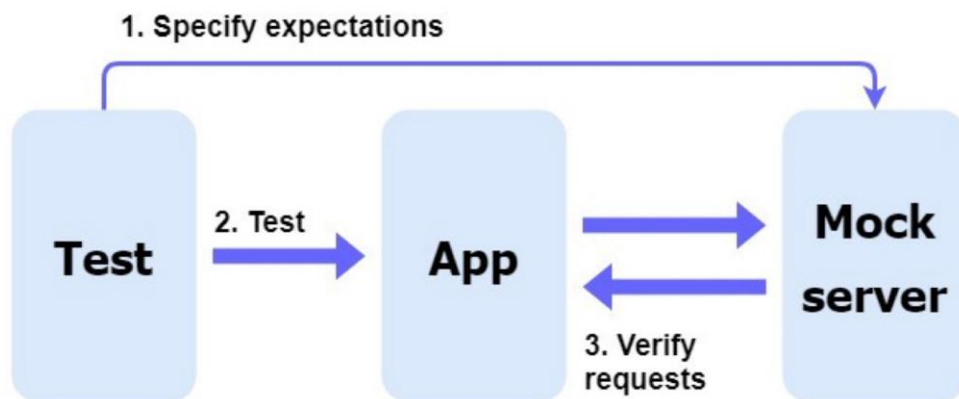
# Методология

## Проблематика

- Медленные тесты.
- Независимый запуск.
- Система без точек входа для запуска набора данных.
- Изменение состояния внешней системы.



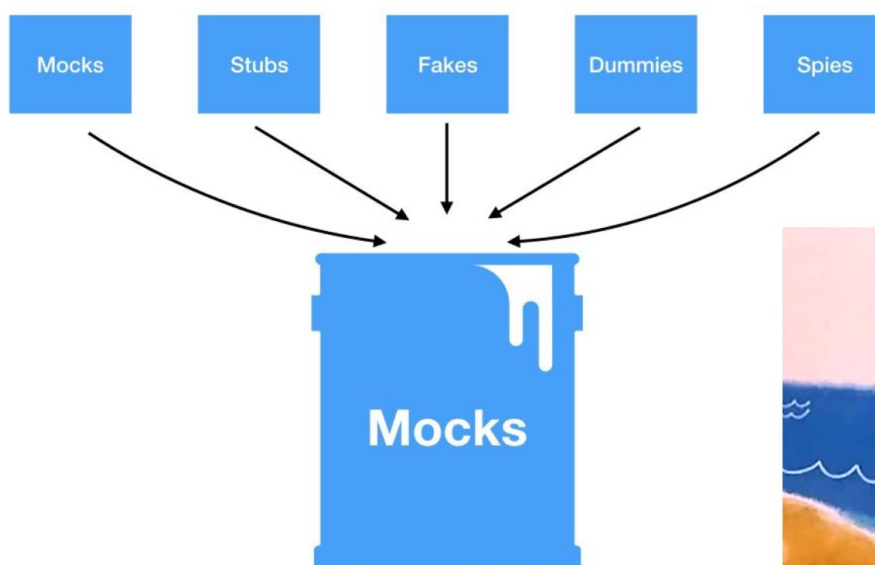
Общий принцип тестирования на мокк состоит в том, чтобы внешнюю зависимость заменить на мокк-объект, состоянием которого мы можем управлять.



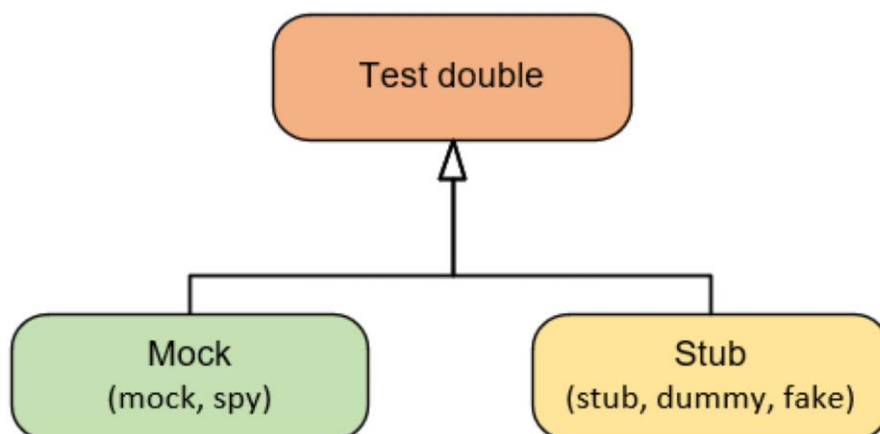
В 2006 году появилась терминология, описывающая виды мокк-объектов, которую ввел Джеррард Месарош. Согласно ей выделяется пять основных групп тестовых двойников:



В актуальной терминологии есть некая путаница: все виды тестовых двойников часто называют одним общим словом **mocks**. Возможно, это связано с тем, что существующие фреймворки зачастую реализуют одновременно несколько mock-объектов.



В современной классификации тестовые дублиры делятся только на два класса, которые в свою очередь делятся на mock, spy, stub, dummy, fake.



## Решаемые проблемы

- Нестабильность внешних сервисов.
- Сложность воспроизведения отдельных сценариев.

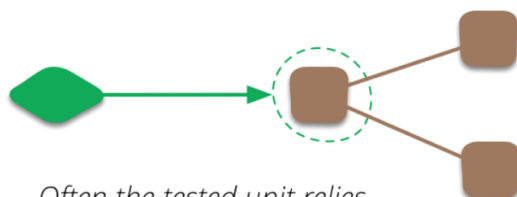


\* - System Under Test

## Существующие подходы

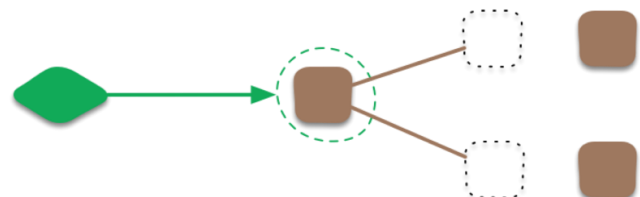
Выделяют две школы — классическую (детройтскую) и мокистскую:

### Sociable Tests



*Often the tested unit relies on other units to fulfill its behavior*

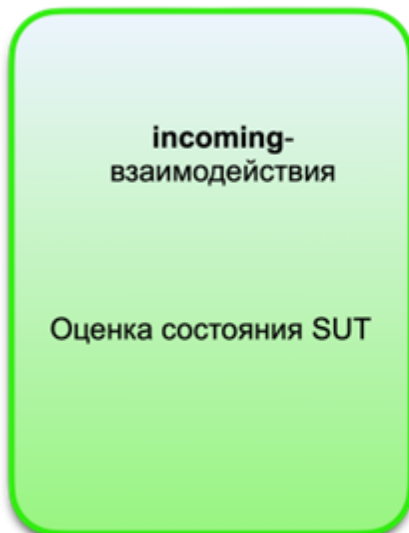
### Solitary Tests



*Some unit testers prefer to isolate the tested unit*

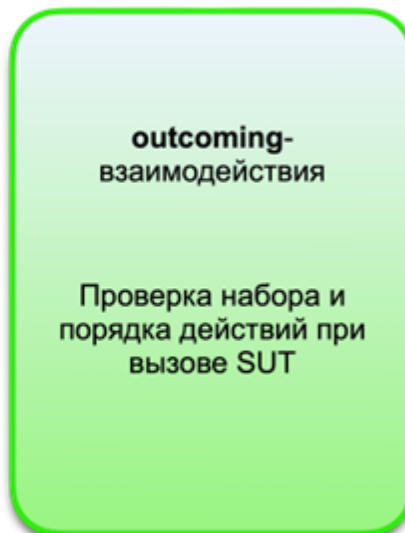
Тесты делятся на два типа. Одни проверяют только состояние объекта (incoming), другие — взаимодействие с внешним окружением (outcoming). В первом случае фальшивый объект называется стабом, во втором — mock-объектом.

### Состояние объекта

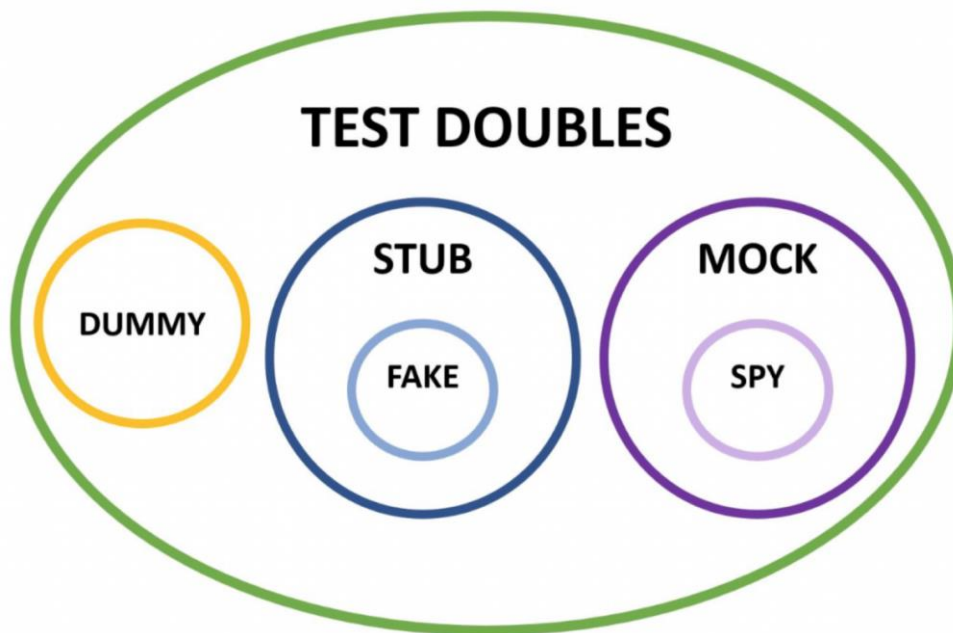


Stubs

### Поведение объекта

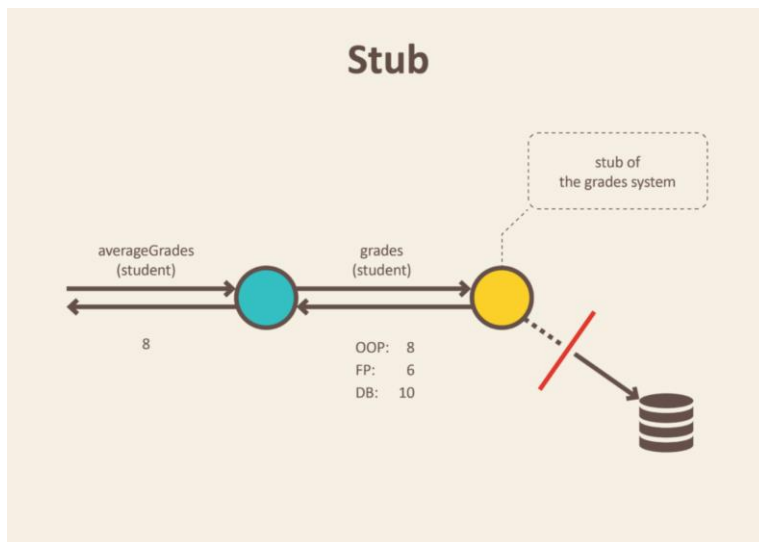


Mocks



### Stubs

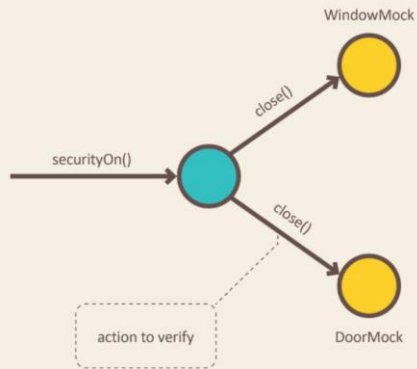
- Жестко зашитый ответ на вызовы.
- Строго определенные данные для конкретного теста.



### Mocks

- Настраиваемые объекты с заданием ожиданий.
- Работающие на верификацию поведения.
- Жестко фиксирующие внутреннюю реализацию -> затрудняющие рефакторинг.

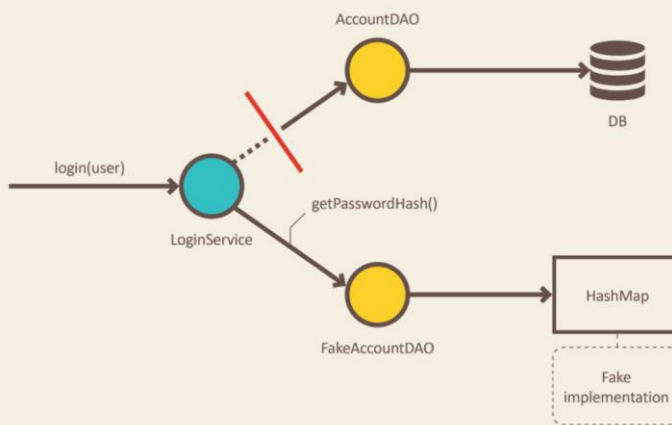
## Mock



## Fake

- Существуют в легковесной версии SUT <sup>[L]</sup><sub>SEP</sub> (реализуют тот же интерфейс).
- Работают на верификацию поведения.
- Имеют рабочую реализацию.
- Удобны для прототипирования.

## Fake




**Важно:** тесты на mocks всегда требуют интеграционного тестирования.

# Python

## Unittest.mock

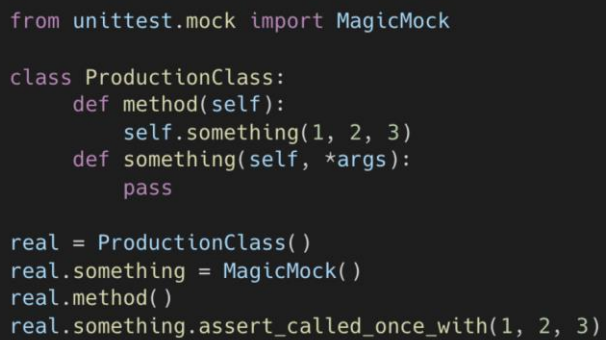
- Mock()



```
from unittest import mock

m = mock.Mock()
m.some_attribute = "hello world"
m.some_attribute
# "hello world"
```

- MagicMock(<sup>SEP</sup>) (более функциональный)



```
from unittest.mock import MagicMock

class ProductionClass:
    def method(self):
        self.something(1, 2, 3)
    def something(self, *args):
        pass

real = ProductionClass()
real.something = MagicMock()
real.method()
real.something.assert_called_once_with(1, 2, 3)
```



- Patch

```
with patch.object(ProductionClass, 'method', return_value=None) as mock_method:
    thing = ProductionClass()
    thing.method(1, 2, 3)

    mock_method.assert_called_once_with(1, 2, 3)
    # It's OK

    mock_method.assert_called_once_with(1, 2, 4)
    # AssertionError: expected call not found.
    # Expected: method(1, 2, 4)
    # Actual: method(1, 2, 3)
```

## Pytest-mock

### pytest-mock 3.8.2

✓ Latest version

Released: Jul 5, 2022

`pip install pytest-mock`

Thin-wrapper around the mock package for easier use with pytest

#### Navigation

- Project description
- Release history
- Download files

#### Project links

- Homepage

#### Statistics

#### Project description

This plugin provides a `mock` fixture which is a thin-wrapper around the patching API provided by the [mock](#) package:

```
import os

class UnixFS:

    @staticmethod
    def rm(filename):
        os.remove(filename)

def test_unix_fs(mock):
    mock.patch('os.remove')
    UnixFS.rm('file')
    os.remove.assert_called_once_with('file')
```

\*[pypi.org/project/pytest-mock/](https://pypi.org/project/pytest-mock/)

- Mocker.spy

```

# for method

def test_spy_method(mock):
    class Foo(object):
        def bar(self, v):
            return v * 2

    foo = Foo()
    spy = mock.spy(foo, 'bar')
    assert foo.bar(21) == 42

    spy.assert_called_once_with(21)
    assert spy.spy_return == 42

```

```

# for function

def test_spy_function(mock):
    # mymodule declares `myfunction` which just returns 42
    import mymodule

    spy = mock.spy(mymodule, "myfunction")
    assert mymodule.myfunction() == 42
    assert spy.call_count == 1
    assert spy.spy_return == 42

```

- **Mocker.stub**

```

def test_stub(mock):
    def foo(on_something):
        on_something('foo', 'bar')

    stub = mock.stub(name='on_something_stub')

    foo(stub)
    stub.assert_called_once_with('foo', 'bar')
    # That's OK

    foo(stub)
    stub.assert_called_once_with('foo', 'bars')
    # E      AssertionError: Expected 'on_something_stub' to be called once. Called 2 times.
    # E      Calls: [call('foo', 'bar'), call('foo', 'bar')].

```

## Требования к решению

```
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/balance/9260219812', methods=['GET'])
def payment():
    response = {
        'balance': '100'
    }
    return jsonify(response), 200
```

- Конфигурация mocks-автотестов.
- Hot reload (перенастройка).
- Логирование.
- Работа в прокси-режиме.
- Поддержка различных протоколов (не только HTTP).
- Низкий порог входа.
- И т.д.

### Create a mock server

1. Select collection to mock    2. Configuration

Create a new collection    Select an existing collection

Enter the requests you want to mock. Optionally, add a request body by clicking on the (...) icon.

Request Method	Request URL	Response Code	Response Body	...
GET	URL	200	Response Body	


- У QA должна быть экспертиза в написании сервисов.
- Дополнительная инфраструктура / много лишнего кода в проекте.
- На QA держится вся поддержка этой инфраструктуры.

## Выводы

Мокирование на уровне кода не очень эффективно. Ниже перечень потенциальных проблем.


- Проблемы с комбинаторикой.
- Ненужный тестовый код внутри приложения.
- Не проверяется взаимодействие с реальной системой.
- Не тестируются исключительные ситуации.

# Mountebank



mountebank - over the wire test doubles

Search...

[home](#) [imposters](#) [logs](#) [config](#) [metrics](#) 

the apothecary

[getting started](#)  
[mental model](#)  
[client libraries](#)  
[command line](#)  
[security](#)  
[faq](#)  
[support](#)

api:

[overview](#)  
[contracts](#)  
[mock verification](#)  
[stub responses](#)  
[proxies](#)  
[injection](#)  
[behaviors](#)  
[faults](#)  
[stub predicates](#)  
[xpath](#)  
[json](#)  
[jsonpath](#)  
[errors](#)

builtin protocols:

[http](#)  
[https](#)  
[tcp](#)  
[smtp](#)

## Welcome, friend

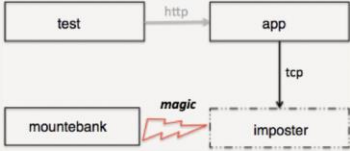
mountebank is the first open source tool to provide cross-platform, multi-protocol test doubles over the wire. Simply point your application under test to mountebank instead of the real dependency, and test like you would with traditional stubs and mocks.

mountebank is the most capable open source service virtualization tool in existence, and will cure what ails you, guaranteed.

### How it works

mountebank employs a legion of *imposters* to act as on-demand test doubles. Your test communicates to mountebank over http using the *api* to set up *stubs*, *record* and *replay proxies*, and verify *mock expectations*. In the typical use case, each test will start an imposter during test setup and stop an imposter during test teardown, although you are also welcome to configure mountebank at startup using a *config file*.

mountebank employs several types of imposters, each responding to a specific protocol. Typically, your test will tell the imposter which port to bind to, and the imposter will open the corresponding socket.



View the [getting started guide](#) for a quick introduction.

v2.7.0 was released this month



## mountebank

2.3.3 - ozon

Mountebank

## Mountebank

### TL;DR;

```
helm repo add ozon https://charts.s.oz3.ru
$ helm install mountebank
```

### How to use at Ozon

Add in .o3/k8s/requirements.yaml

```
dependencies:
- name: mountebank
  version: 1.0.0
  repository: https://charts.s.oz3.ru
```

## Install

Helm CLI

### Add ozon repository

helm repo add o:



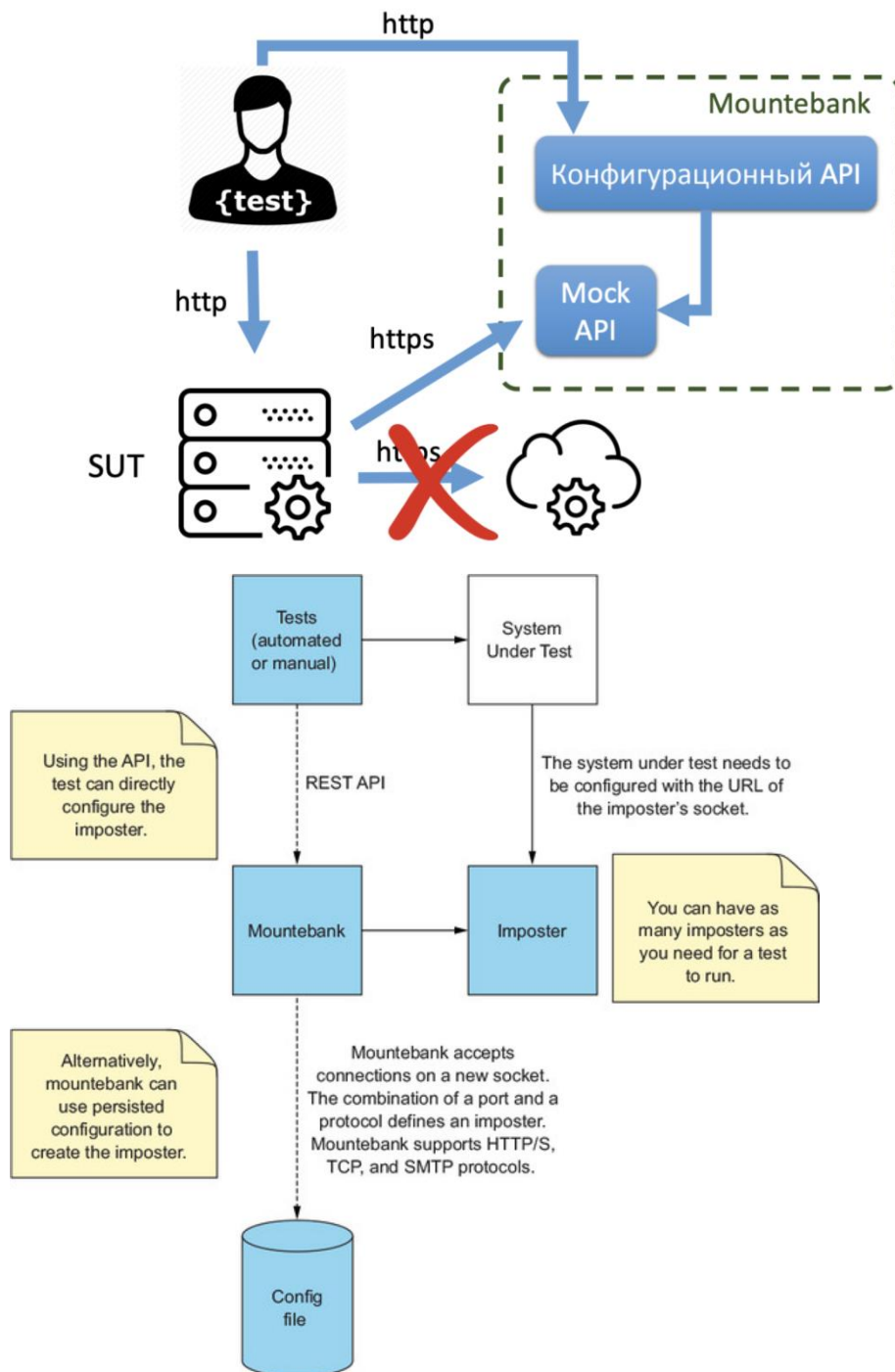
### Install chart

helm install ozon



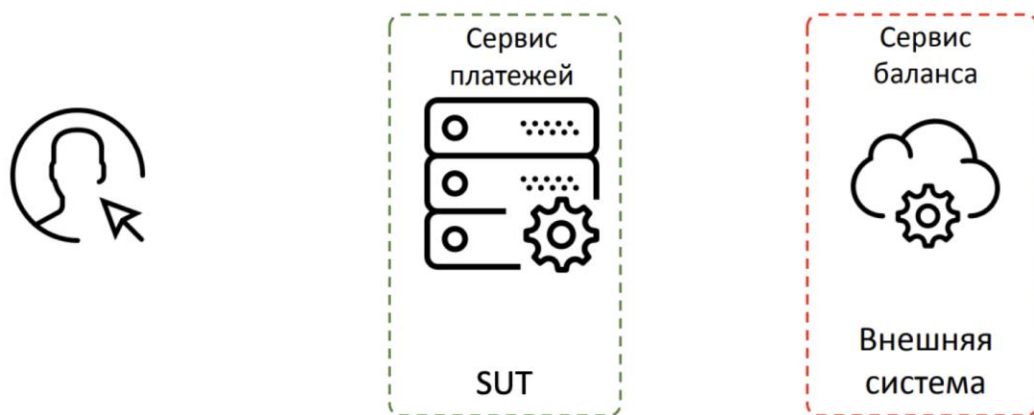
[Need Helm?](#)

## Как оно работает

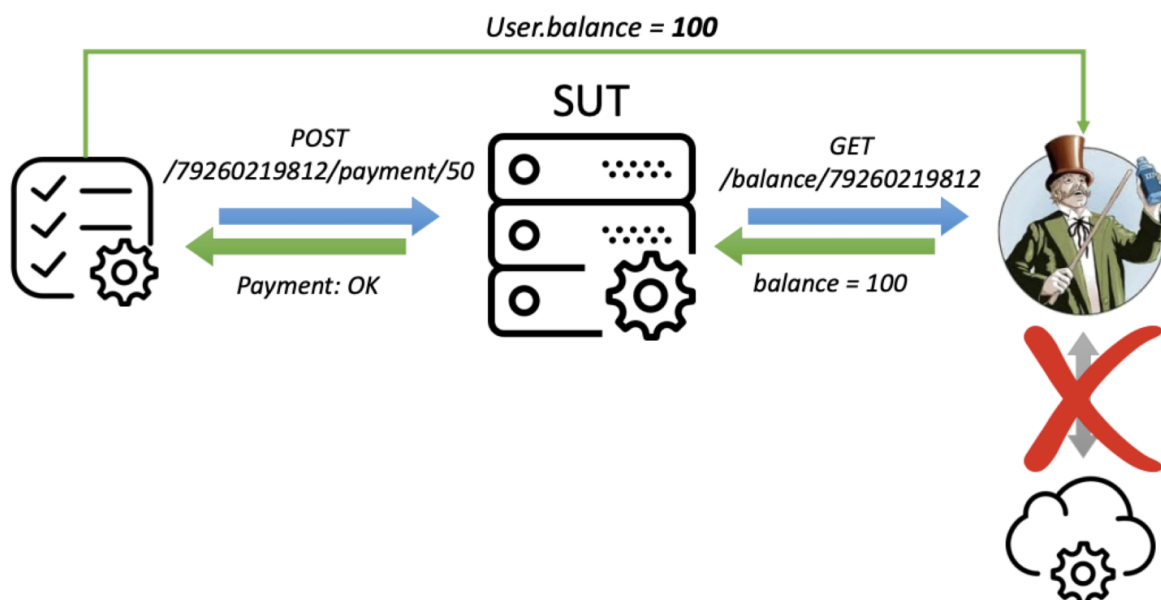
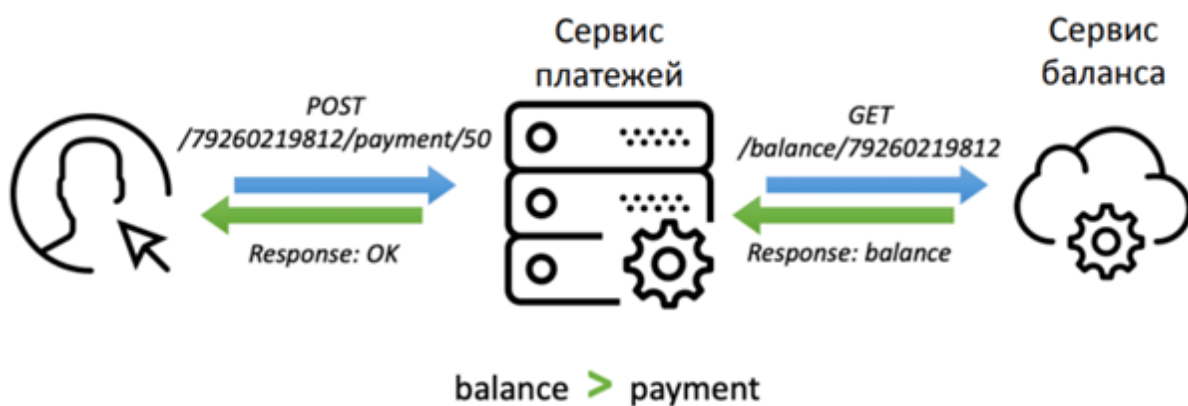


- Поддержка http/s, smtp, tcp из «коробки».
- Низкий порог входа.
- Не нужно поддерживать код mocks.
- Отсутствие дополнительного кода в приложении.
- Хорошая документация.
- Open source решение.

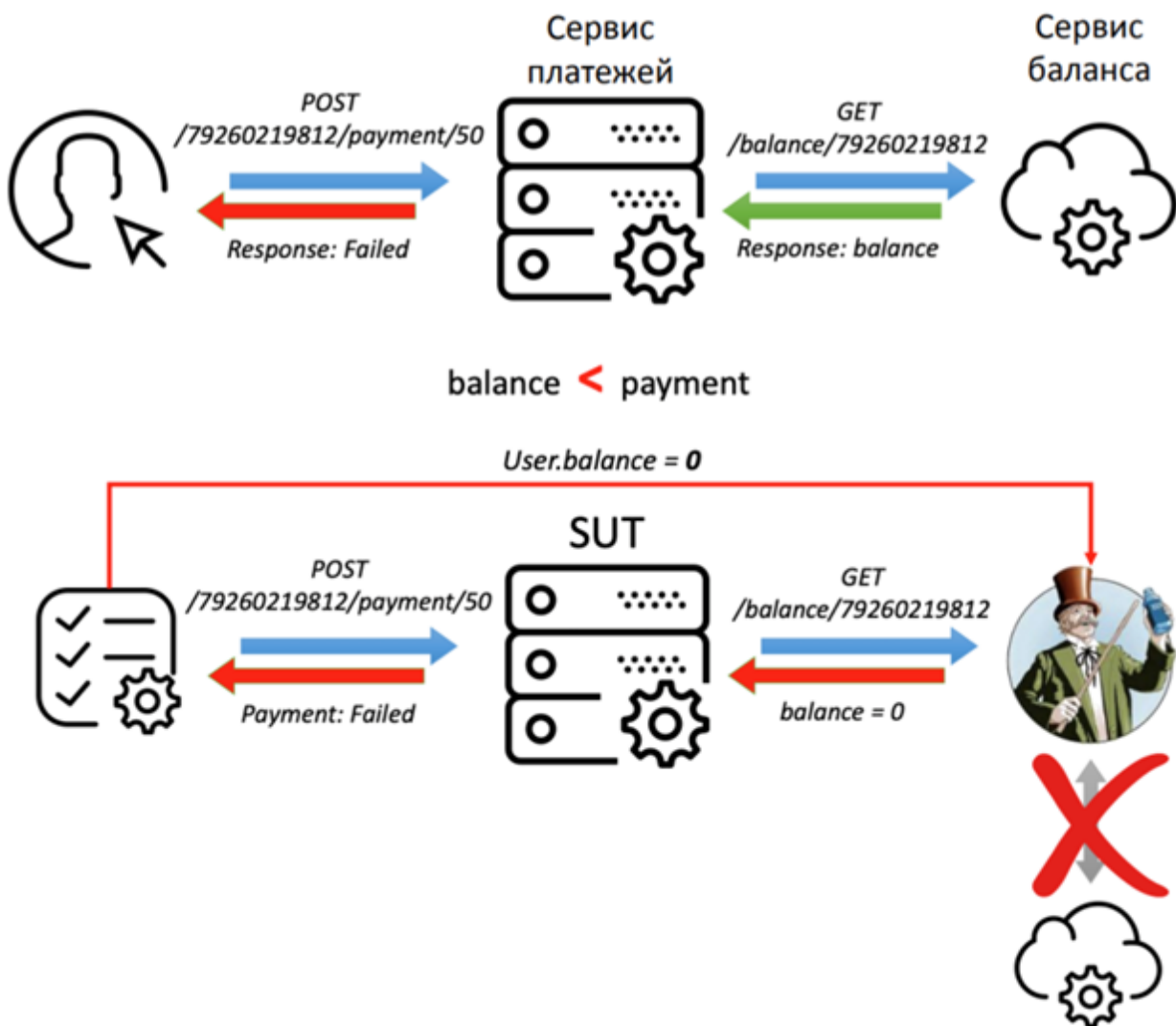
## Примеры



1. Кейс 1: клиент с балансом в 100 у.е. пытается провести операцию в 50 у.е.



2. Кейс 2: клиент с нулевым балансом пытается провести операцию в 50 у.е.



## Как работать с mountebank в Python?

- Mountepy

```
# https://pypi.org/project/mountepy/

def test_mountebank_simple_imposter():
    test_port = port_for.select_random()
    test_response = 'Just some reponse body (that I used to know)'
    test_body = 'Some test message body'

    with Mountebank() as mb:
        imposter = mb.add_imposter_simple(
            port=test_port,
            method='POST',
            path='/some-path',
            status_code=201,
            response=test_response
        )

        response = requests.post(f'http://localhost:{test_port}/some-path', data=test_body)
        assert response.status_code == 201
        assert response.text == test_response
        assert imposter.wait_for_requests()[0].body == test_body
```

- Mbtest

```
# https://mbtest.readthedocs.io/en/latest/

import requests
from hamcrest import assert_that
from brunns.matchers.response import is_response
from mbtest.matchers import had_request
from mbtest.imposters import Imposter, Predicate, Response, Stub

def test_request_to_mock_server(mock_server):
    imposter = Imposter(Stub(Predicate(path="/test"), Response(body="sausages")))

    with mock_server(imposter):
        response = requests.get(f"{imposter.url}/test")
        assert_that("We got the expected response",
                    response, is_response().with_status_code(200).and_body("sausages"))
        assert_that("The mock server recorded the request",
                    imposter, had_request().with_path("/test").and_method("GET"))
```



# Заключение

На занятии мы обсудили терминологию, принятую в mock-тестировании; узнали, как создавать mock-объекты средствами Python и познакомились с mock-сервером Mountebank, который используется в Ozon.

# Материалы

1. xUnit Test Patterns: Refactoring Test Code (Gerard Meszaros)
2. Unit Testing Principles, Practices, and Patterns (Владимир Хориков)
3. [www.mbtest.org](http://www.mbtest.org)