API AUTOTESTS CHECK LIST WITH PYTHON EXAMPLES

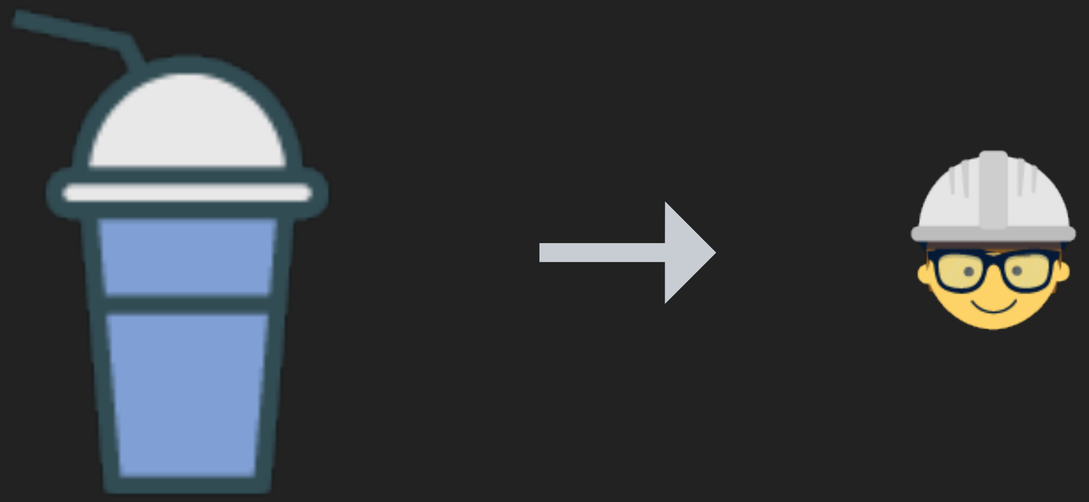
IGOR BALAGUROV

SENIOR AUTOMATION ENGINEER AT VK

- vk.com/korgan
- linkedin.com/in/ibalagurov
- github.com/ibalagurov

DON'T REPEAT YOURSELF: UI-ТЕСТЫ ДЛЯ ВЕБ, IOS И ANDROID ОДНОВРЕМЕННО

РАЗГОВОРЫ ПРО PYTHON



BASED ON...

My life is based on a true story

VK PAY EXPERIENCE

- Infrastructure: MRG → VK → VK Pay
- Backend tech stack: Perl → Go
- 5X dev team growth
- 14M+ wallets in 2 years
- …

VK PAY EXPERIENCE

- Infrastructure: MRG → VK → VK Pay
- Backend tech stack: Perl → Go
- 5X dev team growth
- 14M+ wallets in 2 years
- ...

PLAN

- What to test?
- How to test?
- Tools & Examples
- …

PLAN

- What to test?
- How to test?
- Tools & Examples
- ...
- And Why so?

FUNCTIONAL TESTS

- Positive / Negative (Boundary values and Equivalence partitioning)
  - Actors (Users / Merchant types)
  - Input / Output values
  - Wrong format / nonexistent values
- ...

FUNCTIONAL TESTS

- Positive / Negative (Boundary values and Equivalence partitioning)
  - Actors (Users / Merchant types)
  - Input / Output values
  - Wrong format / nonexistent values
- …
- Required fields

SOMETIMES THE RISK IS TOO HIGH

- Fin-tech
- Sensitive data
- OWASP API Security

BASE SECURITY TESTS

- Privacy / Permissions
- Wrong tokens / hashes / signs / private keys
- Invalid values (overflow attempts, injections)
- Rate & Resource limits (flood control)
- Race conditions

SOMETIMES THE DIFFERENCE IS SMALL

- Negative actors → privacy
- Negative values → resources limits
- …

SOMETIMES THE DIFFERENCE IS SMALL

- Negative actors → privacy
- Negative values → resources limits
- …
- It’s not a problem to extend tests

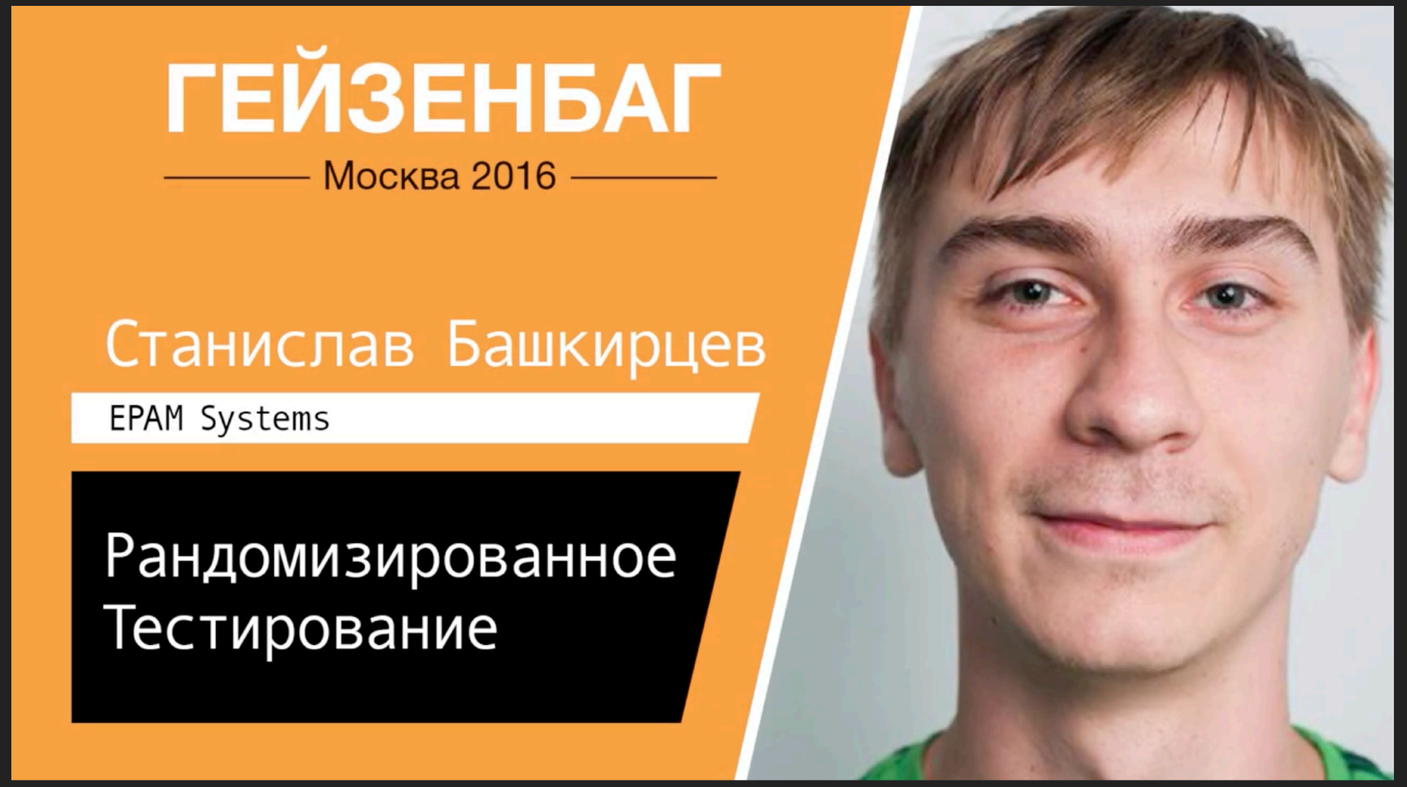
FUNCTIONAL TESTS: POSITIVE / NEGATIVE

- Randomization in getting boundary values and inside equivalence classes
- …

FUNCTIONAL TESTS: POSITIVE / NEGATIVE

- Randomization in getting boundary values and inside equivalence classes
- …
- Helps to be sure
- No need in extra tests for combinations
- Easy to add localization tests / new checks

RANDOMIZATION?!



BASE SECURITY TESTS: HOW TO CHECK

- Data validation, schemas
  - Body contains only predefined fields, in appropriate format
  - No additional information: in any case, in any place
  - …

BASE SECURITY TESTS: HOW TO CHECK

- Data validation, schemas
  - Body contains only predefined fields, in appropriate format
  - No additional information: in any case, in any place
  - …
  - There is no any sensitive data

TODO

- Check every request param
  - Randomization
  - Parametrization
- Check every place for sensitive data
  - Data validation
- Check concurrent responses

RANDOMIZATION

- Mimezis, faker, factoryboy
  - Data types
- Seed

RANDOMIZATION: POSITIVE CASES

- Boundary values
- Equivalence classes

RANDOMIZATION: SHOW ME THE CODE

```python
import mimesis

generate = mimesis.Generic(locale="en", seed=100500)

currency = generate.business.currency_iso_code()
# 'USD'

phone = generate.person.telephone()
# '1-115-514-0222'
```

RANDOMIZATION: SHOW ME THE CODE

```
1 import mimesis
2 from mimesis.random import Random
3
4 generate = mimesis.Generic(locale="en", seed=100500)
5 random = Random(x=100500)
6
7 number = generate.numbers.integer_number
8 choice = generate.choice
```

RANDOMIZATION: SHOW ME THE CODE

```
1 # boundary values
2 max_value = ["10000", "10000.0", "10000.00"]
3 min_value = ["1", "1.0", "1.00"]
4 boundary_value = choice(min_value + max_value)
5 # '10000.0'
```

RANDOMIZATION: SHOW ME THE CODE

```
# equivalence classes
decimal_classes = ["{}", "{}#", "{}##"]
fractional_classes = ["", ".#", ".##"]
decimal_template = choice(decimal_classes)
# '{}#'
non_zero_digit = number(start=1, end=9)
# 9
decimal_part = decimal_template.format(non_zero_digit)
fractional_part = choice(fractional_classes)
# ''
mask = decimal_part + fractional_part
equivalence_class = random.custom_code(mask=mask)
# '95'
```

RANDOMIZATION: SHOW ME THE CODE

```
def valid_amount():
    ...
    result = choice([boundary_value, equivalence_class])
    return result

valid_param_dict = {
    "amount": valid_amount()
}

def generate_valid(param):
    return valid_param_dict.get(param)
```

RANDOMIZATION: NEGATIVE CASES

- There are a lot of types

RANDOMIZATION: SHOW ME THE CODE

```python
import mimesis

generate = mimesis.Generic(locale="en", seed=100500)

my_cool_injections = ["' OR 1 == 1"]
nulls = ["null", "nan", "nil", "none"]
overflow_int = "9" * 100
common = [overflow_int, *nulls, *my_cool_injections]

invalid_currency = generate.choice(common)
# "none"
```

RANDOMIZATION: SHOW ME THE CODE

```
invalid_param_dict = {
    "currency": generate.choice(common)
}


def generate_invalid(param):
    return invalid_param_dict.get(param)
```

RANDOMIZATION: SHOW ME THE CODE

```
1 from test_data import generate_invalid
2
3 default_payment_data = {"amount": 1, "currency": "RUB"}
4
5 data = {
6     **default_payment_data,
7     "currency": generate_invalid("currency")
8 }
```

RANDOMIZATION: SHOW ME THE CODE

```
param = "currency"

data = {
    **default_payment_data,
    param: generate_invalid(param)
}
```

PARAMETRIZATION

- Pytest
- ...

PARAMETRIZATION: SHOW ME THE CODE

```
1 import pytest
2 import requests
3 from test_data import generate_invalid
```

PARAMETRIZATION: SHOW ME THE CODE

```
default_payment_data = {"amount": 1, "currency": "RUB"}

@pytest.mark.parametrize(
    "param", default_payment_data.keys()
)
def test_payment_invalid_param(param):
    data = {
        **default_payment_data,
        param: generate_invalid(param)
    }

    requests.post(url="/payment", data=data)
```

RANDOMIZATION & PARAMETRIZATION: PROS & CONS

- No routine
- Reusable values
  - Common for same parameters
  - It could be used as standalone data generator solution
- ...

RANDOMIZATION & PARAMETRIZATION: PROS & CONS

- No routine
- Reusable values
  - Common for same parameters
  - It could be used as standalone data generator solution
- ...
- Sometimes tests fail → Knowledge mining

DATA VALIDATION

- Cerberus, Pydantic, jsonschema
  - Convenient API: no need in additional conversion
  - All errors in one check
  - Required / Unkown fields
  - Custom rules (any, all, none, dependencies)

DATA VALIDATION: SHOW ME THE CODE

```
 1 from cerberus import Validator
 2
 3 schema = {'name': {'type': 'string'}}
 4
 5 v = Validator(schema)
 6
 7 document = {'name': 'john doe'}
 8
 9 v.validate(document)
10 # True
```

DATA VALIDATION: SHOW ME THE CODE

```python
from src import vk_pay, check
from src import data_service


def test_wallet_to_wallet():
    wallet = data_service.get_wallet()

    response = vk_pay.wallet_balance(wallet=wallet)
    response.json()
    # {"data": {"userId": 1, "balance": 100}}

    check.response_matches_ok_schema(response)
```

DATA VALIDATION: SHOW ME THE CODE

```
wallet_balance = {
    "data": {
        "type": "dict",
        "required": True,
        "require_all": True,
        "schema": {
            "balance": {
                "type": "number", "min": 0, "max": 60000
            },
            "userId": {"type": "integer", "min": 1},
        }
    }
}
```

DATA VALIDATION: SHOW ME THE CODE

```
ok_mapping = {
    "get": {
        "/wallet/balance": wallet_balance,
    }
}

def for_ok_schema(response):
    url = response.request.path_url
    method = response.request.method.lower()
    return ok_mapping.get(method, {}).get(url)
```

DATA VALIDATION: SHOW ME THE CODE

```
 1 from src import schema
 2 from cerberus import Validator
 3
 4 def response_matches_ok_schema(response):
 5     response_schema = schema.for_ok_response(response)
 6     data_to_validate = response.json()
 7     schema_validator = Validator(response_schema)
 8
 9     assert schema_validator.validate(data_to_validate), \
10         schema_validator.errors
```

DATA VALIDATION: SHOW ME THE CODE

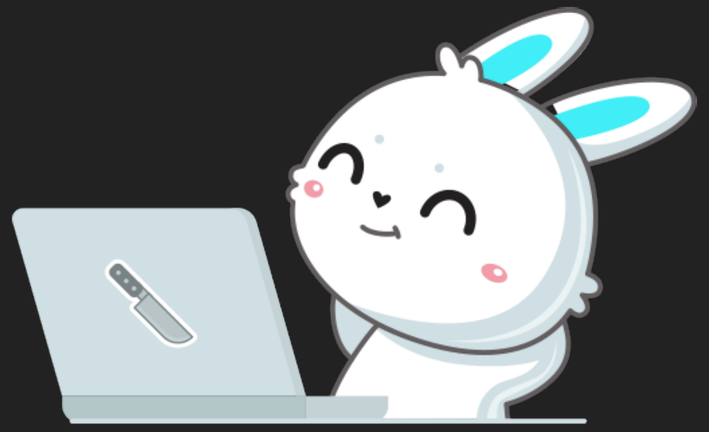
```
 1 data_to_validate = response.json()
 2 # {"data": {"userId": 0, "balance": -0.1}}
 3
 4 schema_validator.validate(data_to_validate))
 5 # False
 6
 7 schema_validator.errors
 8 # {'data':[{
 9 #     'balance': ['min value is 0'],
10 #     'userId': ['min value is 1']
11 # }]}
```

DATA VALIDATION: PROS & CONS

- Useful with 3rd parties
- If you want to know about:
  - Every new field
  - Any new format
- …

DATA VALIDATION: PROS & CONS

- Useful with 3rd parties
- If you want to know about:
  - Every new field
  - Any new format
- …
- Sometimes tests fail → Knowledge mining



CONCURRENT RESPONSES

- Standard library (concurrent.futures, threading, asyncio)
- Rate limits
- Race conditions
- Safety
- Fault tolerance

CONCURRENT RESPONSES: SHOW ME THE CODE

```
1 from functools import partial
2 from src import vk_pay, test_data, concurrent
3
4 transaction_id = test_data.create_transaction()
5
6 api_method = partial(
7     func=vk_pay.confirm_transaction,
8     transaction_id=transaction_id
9 )
```

CONCURRENT RESPONSES: SHOW ME THE CODE

```
def test_rate_limits():
    responses = concurrent.execute(
        func=api_method, pause_between=0.1, times=10
    )

    assert any(
        response.status_code == 429
        for response in responses
    )
```

CONCURRENT RESPONSES: SHOW ME THE CODE

```
def test_race_condition():
    responses = concurrent.execute(
        func=api_method, times=10
    )

    ok_responses = [
        response.status_code == 200
        for response in responses
    ]

    assert len(ok_responses) == 1
```

CONCURRENT RESPONSES: SHOW ME THE CODE

```
def test_small_stress():
    responses = concurrent.execute(
        func=api_method, times=100
    )

    long_responses = [
        response for response in responses
        if response.elapsed.total_seconds() >= 2
    ]

    assert not long_responses
```

CONCURRENT RESPONSES: SHOW ME THE CODE

```
1 import time
2 from concurrent.futures import ThreadPoolExecutor
3
4 def execute(func, times=1, pause_between=None):
5     pool = ThreadPoolExecutor(max_workers=times)
```

CONCURRENT RESPONSES: SHOW ME THE CODE

```
1 if pause_between is None:
2     futures = [
3         pool.submit(func) for _ in range(times)
4     ]
5
6     return [f.result() for f in futures]
```

CONCURRENT RESPONSES: SHOW ME THE CODE

```
else:
    futures = []
    for _ in range(times - 1):
        futures.append(pool.submit(func))
        time.sleep(pause_between)
    futures.append(pool.submit(func))

    return [f.result() for f in futures]
```

CONCURRENT RESPONSES: SHOW ME THE CODE

```
if pause_between is None:
    futures = [
        pool.submit(func) for _ in range(times)
    ]
else:
    futures = []
    for _ in range(times - 1):
        futures.append(pool.submit(func))
        time.sleep(pause_between)
    futures.append(pool.submit(func))

return [f.result() for f in futures]
```

CONCURRENT RESPONSES: PROS & CONS

- If you want to prevent or find earlier:
  - Security issues
  - Performance issues
- …

CONCURRENT RESPONSES: PROS & CONS

- If you want to prevent or find earlier:
  - Security issues
  - Performance issues
- …
- Sometimes tests fail → Knowledge mining

REMEMBER ALL

- API tests → Base Functional tests
- Functional tests → Base security tests
- Approaches & Tools & Examples
  - Data generation (Mimesis)
  - Data validation (Cerberus)
  - Concurrent responses (Concurrent.futures)

WHAT IS IT ABOUT?

- Test more → Prevent more
- Explore your system → Knowledge mining
- Make more general solutions → Automate & Chill
- Keep think like QA → Automate like a Boss!

WHAT'S NEXT

- API schemas:
  - Fuzzing
  - Property-base testing

THANK YOU!!!

vk.com

presentation

linkedin.com