BACKEND TEST AUTOMATION FROM THE SCRATCH

IGOR BALAGUROV

QA AUTOMATION TECH LEAD AT VK

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

VK EXPERIENCE

- VK Payment systems
  - Wallets (VK Pay), Card2Card and Card2Wallet transactions
- Internal Development
  - A lot of small internal services
- Performance testing
- Tooling: static code analysis, code formatters, code coverage, CI integration

OVERVIEW TALK

- Introduction
- How to test?
- How to choose tools?
- Examples (Python and REST API)
- What to test in API's next
- We need go deeper
- Conclusions

BACKEND TESTING

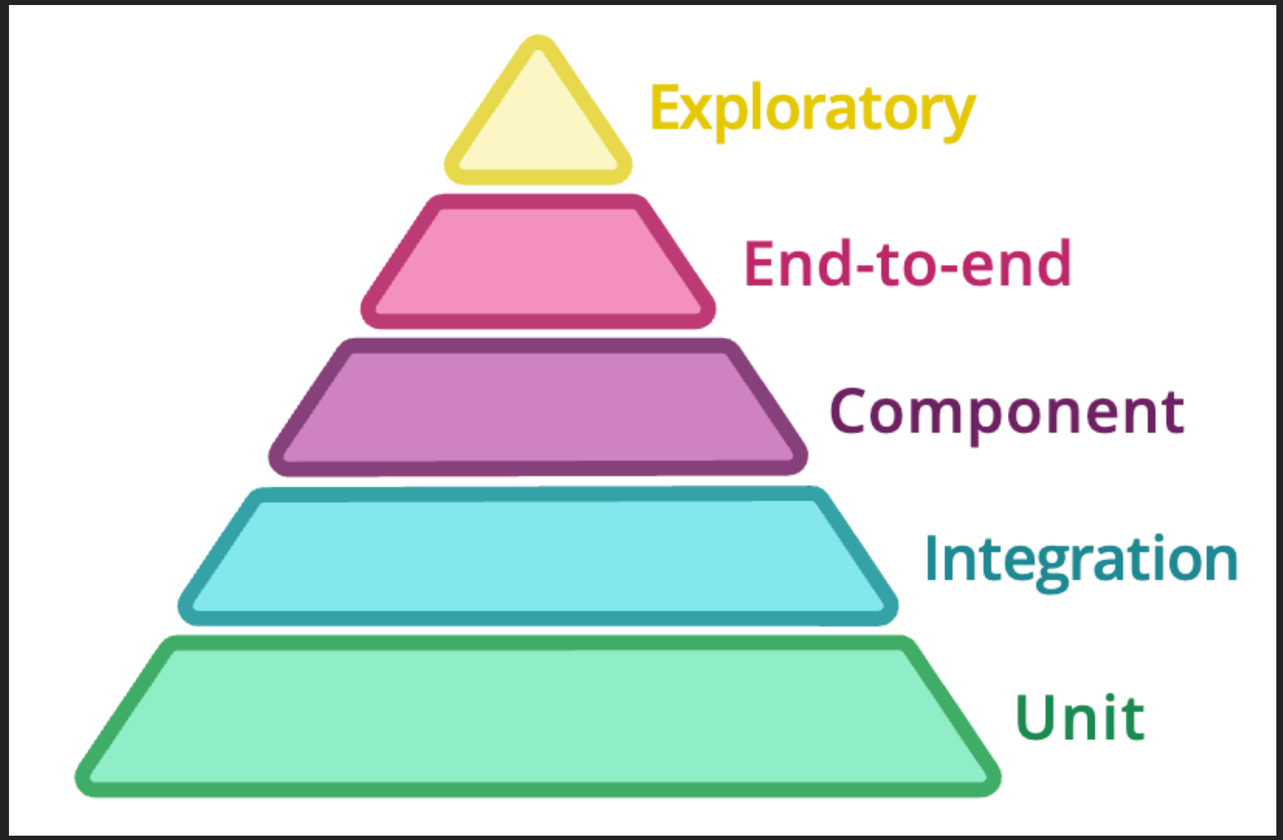
- Application architecture
  - Different approaches, tools and etc

TEST PYRAMID

- Too broad conceptions
- Some layers could be unavailable in your application
- 3rd parties could change everything

MICROSERVICE TEST PYRAMID

- martinfowler.com
  - microservice testing
  - practical test pyramid



TEST PYRAMID

- Solve your problems

PROTOCOLS

- HTTP or not
- Text and binary
- Synchronous and asynchronous
- REST, GraphQL, MQ, gRPS, Thrift, DB

EDGE CASES

- Configuration
- BigData
- Machine Learning

PROTOCOLS

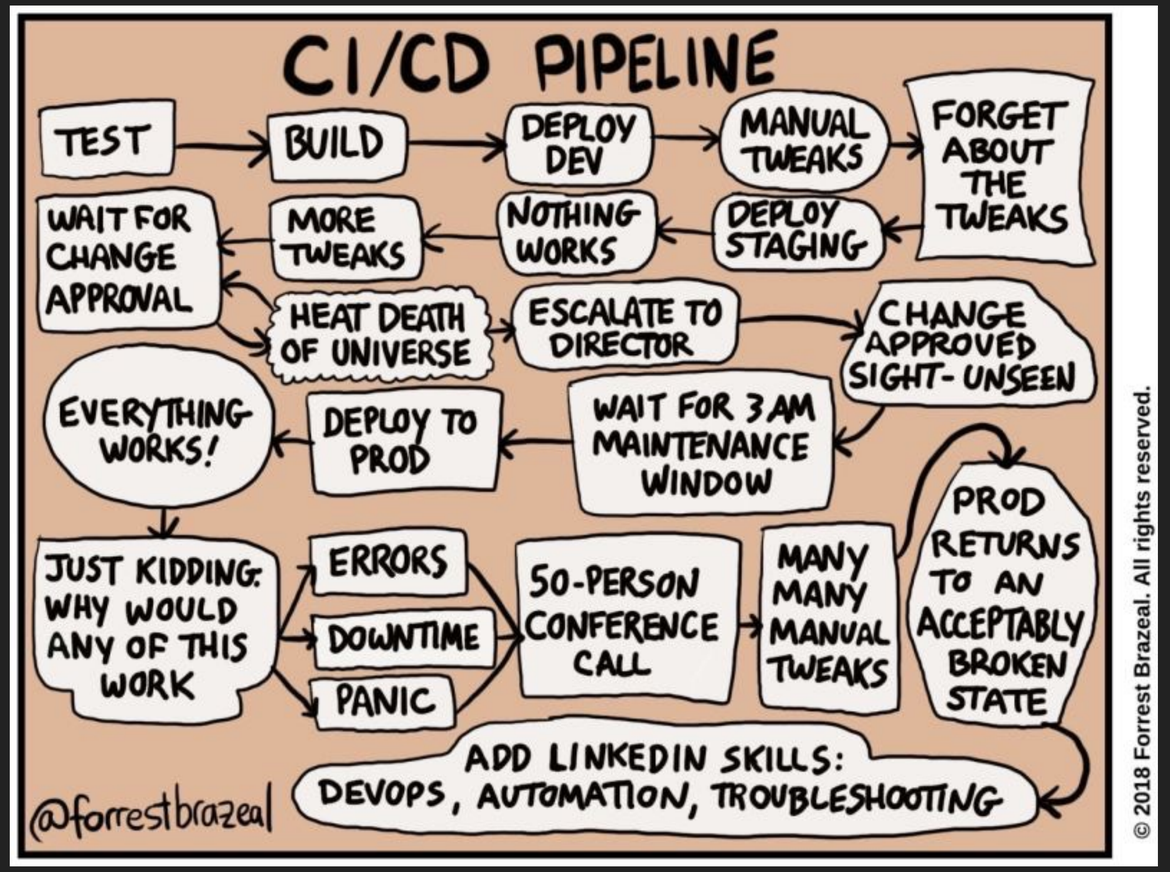
- Interface
  - Input and output data
- Built for programs
  - Fit for automation

MANUALLY

- Requirements / Documentation
  - Usability / UX
  - Testability
- Exploratory
  - Security (Penetration) testing
- Acceptance
  - ?

AUTOMATION

- Fast feedback
- Reduce Time2Market
- CI / CD
- Frequent releases
- Close to development
- UI tests preconditions



TOOLS FOR AUTOMATION

- Same language with development?
- Tools for testing
- Community (StackOverflow 💔)
- Learning curve
- Development performance
- Code performance

EXAMPLES

- Short project
  - Manual
  - Postman
  - Hello world automation

EXAMPLES

- Problems
  - Project became bigger
    - Throw away old approaches

EXAMPLES

- Big project
  - Developers write almost all automated tests
  - Small dedicated QA team
    - Convenient language and tools for QA
    - Same language?

EXAMPLES

- Problems
  - Frequent releases
    - Update tests for new logic
    - Revert?

EXAMPLES

- Feature teams
  - Same repositories
  - Same technologies?

EXAMPLES

- Problems
  - Full stack QA Engineers?

EXAMPLES

- Big project with big QA team(s) (or without QA team at all)
  - Same technologies

EXAMPLES

- Problems
  - Your own frameworks, tools and etc
  - Feature toggles, AB tests

LANGUAGES FOR AUTOMATION (IMHO)

- C#: few tools for testing
- Java: a lot of tools
- Python: easy tools integration, wide community
- JS: a lot of tools, complex tools integration
- Ruby: cool tools, but few candidates
- Go: ?

PYTHON AND REST API

- Approaches are language agnostic
- REST API
- Tools integration examples

PYTEST

- Plugins
  - Rerun failures (pytest-rerunfailures)
  - Parallel execution (pytest-xdist)
  - Soft asserts (pytest-check)
  - Many more
  - Your own

WHY PYTEST

- Plugins
  - Fast tests development
  - Complex tests
- Good community
- Nested fixtures
- Hooks

DEAD SIMPLE CLIENT

```
 1 import requests
 2
 3 response = requests.post(
 4     url="https://your-site.com",
 5     json={"a": 1}
 6 )
 7
 8 response.request.headers["content-type"]
 9
10 # 'application/json'
```

COOKIES AND REDIRECTS

```
 1 from requests import Session
 2
 3 session = Session()
 4
 5 session.post(
 6     url="https://your-site.com/auth",
 7     data={"login": 1, "password": 2}
 8 )
 9
10 response = session.get(
11     url="https://your-site.com/authenticated"
12 )
```

CACHED SESSIONS

```
1 from requests import Session
2 from functools import lru_cache
3
4
5 @lru_cache()
6 def user_session(user: User):
7     return Session()
```

WAIT ASYNC OPERATIONS

```
1 import waiting
2 import requests
3
4 waiting.wait(
5     predicate=requests.get(url="...").json()["a"] == 1,
6     timeout_seconds=15,
7     sleep_seconds=(0.25, 1, 2),
8     expected_exceptions=KeyError,
9 )
```

TESTS

- Risk based scenarios
- Privacy (role based model)
- Input / Output equivalence classes and boundary values
- API methods combinations
- Based on test coverage lower level tests

EXTEND THEM

- Random data generation
- Complex data validation (Response schemas)
- Base security checks
- Property based testing

HIDDEN CORNER CASES

- Pesticide Paradox:
  - Hardcode values in test data
  - Inability to variate data across all tests

APPROACH

- Randomization in getting boundary values and inside equivalence classes
- Randomization helps to test:
  - Combinations
  - Localization
  - New checks
- Seed for reproducible tests

TOOLS

- Mimezis
- Faker

EXAMPLES: POSITIVE CASES

```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'
```

EXAMPLES: NEGATIVE CASES

```python
import mimesis

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

nulls = ["null", "nan", "nil", "none"]
random_word = generate.text.word()
common = [random_word, *nulls]

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

EXAMPLES: PREPARE FOR PARAMETRIZATION

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


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

RANDOMIZATION: PROS & CONS

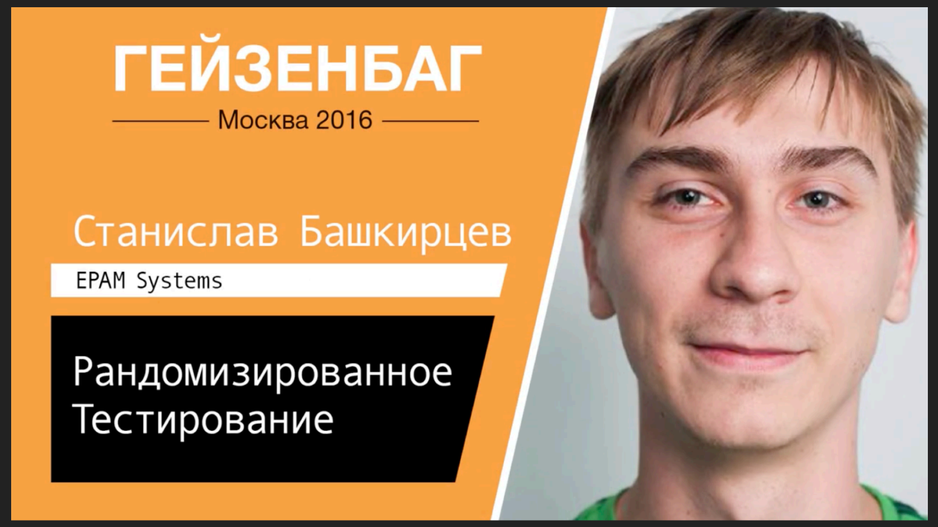
- Reusable values
  - Common for same or similar parameters
  - It could be used as standalone data generator solution
- Combinations
- ...

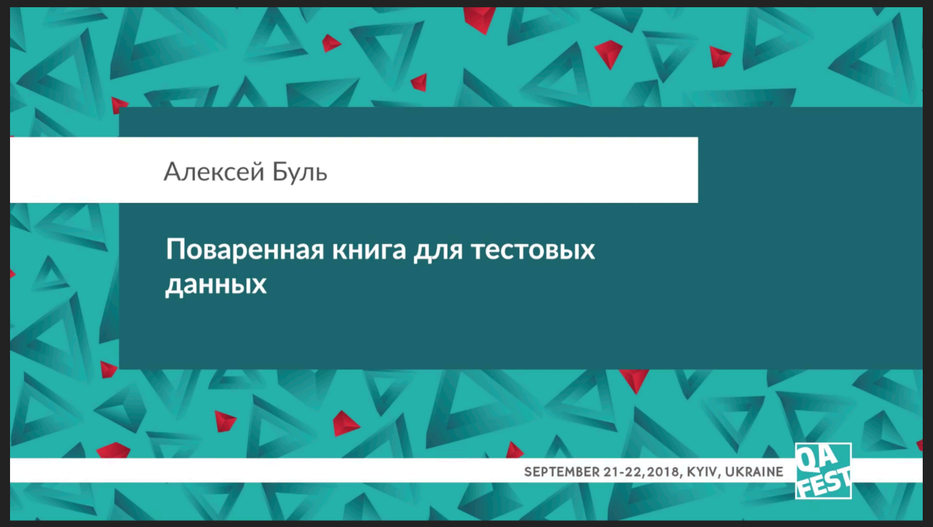
RANDOMIZATION: PROS & CONS

- Reusable values:
  - Common for same parameters
  - It could be used as standalone data generator solution
- Combinations
- Sometimes tests fail → Knowledge mining
- You need the seed to reproduce test

WHAT'S NEXT?

- Mimesis documentation
- Articles (habr)
- Heisenbug 2016 Moscow: Randomized testing
- QA Fest 2018: Поваренная книга для тестовых данных



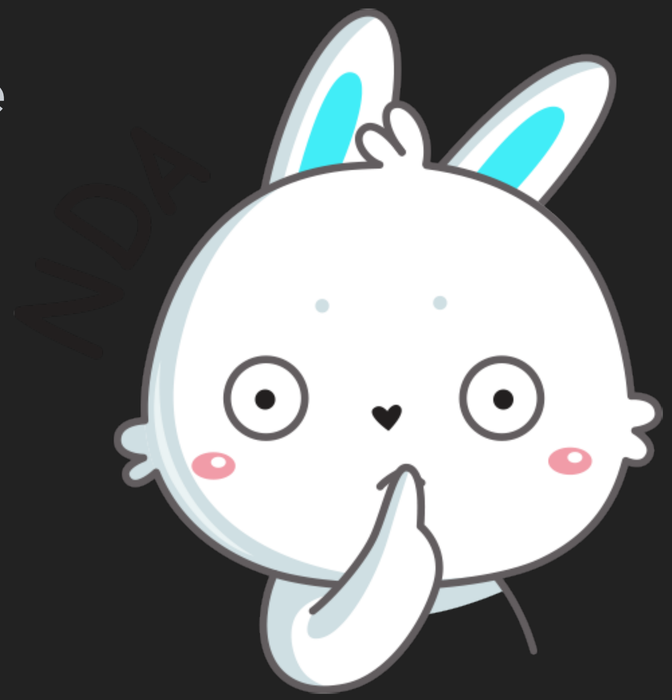


COMPLEX DATA VALIDATION

- 3rd-parties data
- Sensitive data:
  - Any new field is a risk
- Contract:
  - Any new format type is a risk

APPROACH

- 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



TOOLS

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

EXAMPLES

```
 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
```

EXAMPLES

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


def test_wallet_balance():
    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)
```

EXAMPLES

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

EXAMPLES

```
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)
```

EXAMPLES

```
 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
```

EXAMPLES

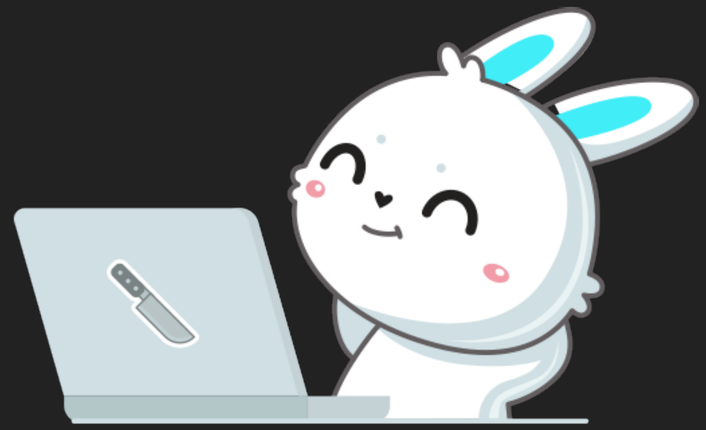
```
data_to_validate = response.json()
# {"data": {"userId": 0, "balance": -0.1}}

schema_validator.validate(data_to_validate))
# False

schema_validator.errors
# {'data':[{
#     'balance': ['min value is 0'],
#     'userId': ['min value is 1']
# }]}
```

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
- You need to describe your schemas



WHAT'S NEXT?

- PiterPy 2018: Cerberus, or Data Validation for Humans
- Cerberus documentation

BASE SECURITY TESTS

- Security risks
- Sensitive data
- Separate security department:
  - It is not so iterative, as it could be

APPROACH

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

APPROACH

- Negative actors → privacy
- Negative values → resources limits
- Wrong headers → tokens / hashes
- …
- It’s not a problem to extend tests

APPROACH

- Check every request param
  - Overflow attempts
  - Injections
- Check concurrent responses
  - Rate limits
  - Race conditions

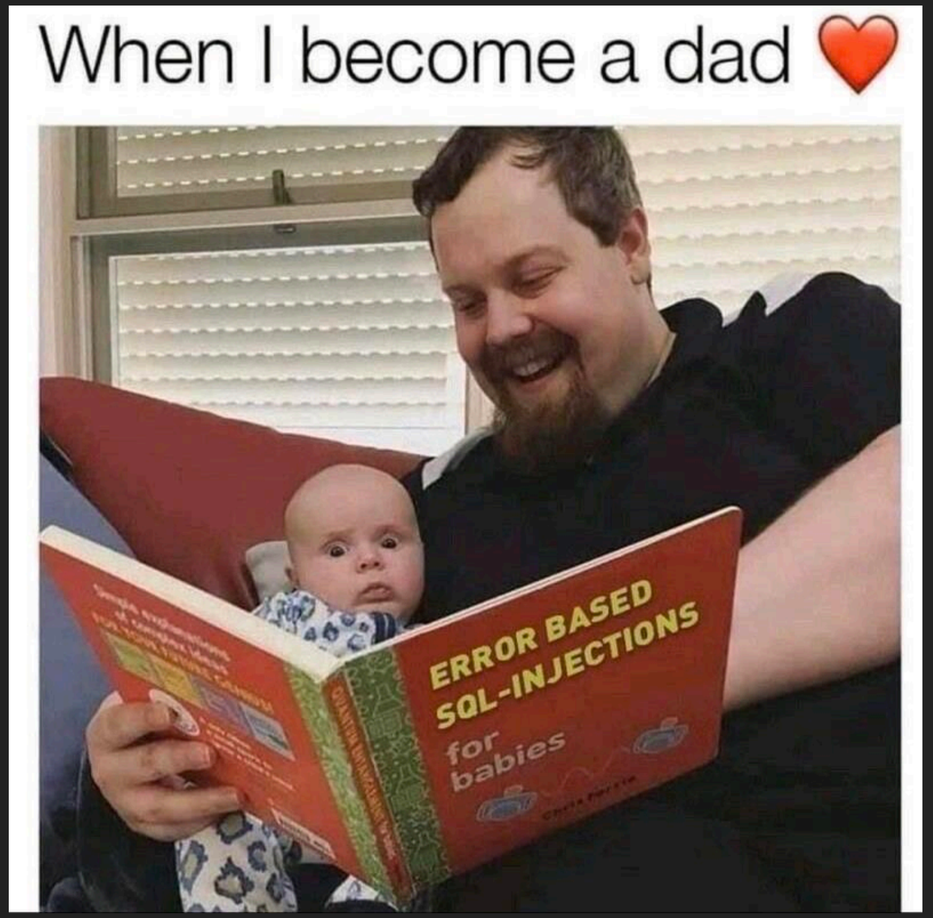
EXAMPLES: NEGATIVE CASES

- Nulls
- Invalid format values
- Non-existent values

EXAMPLES: NEGATIVE CASES

- Nulls
- Invalid format values
- Non-existent values

- Overflow attempts
- Injections



EXAMPLES: NEGATIVE CASES

```python
import mimesis

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

nulls = ["null", "nan", "nil", "none"]
random_word = generate.text.word()
common = [random_word, *nulls]

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

EXAMPLES: NEGATIVE CASES

```
1 overflow_int = "9" * 100
2
3 my_cool_injections = ["' OR 1 == 1"]
4
```

EXAMPLES: NEGATIVE CASES

```python
import mimesis

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

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

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

APPROACH

- Check every request param
  - Overflow attempts
  - Injections
- Check concurrent responses
  - Rate limits
  - Race conditions

APPROACH

- Check concurrent responses
  - Rate limits
  - Race conditions

TOOLS

- Standard library:
  - concurrent.futures
  - threading
  - asyncio
  - and etc.

EXAMPLES

```
1 from functools import partial
2 from src import vk_pay, test_data, parallel
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 )
```

EXAMPLES

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

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

    assert len(ok_responses) == 1
```

EXAMPLES

```
1 from concurrent.futures import ThreadPoolExecutor
2
3
4 def execute(func, times=1):
5     pool = ThreadPoolExecutor(max_workers=times)
6     futures = [pool.submit(func) for _ in range(times)]
7     return [f.result() for f in futures]
```

PROS & CONS

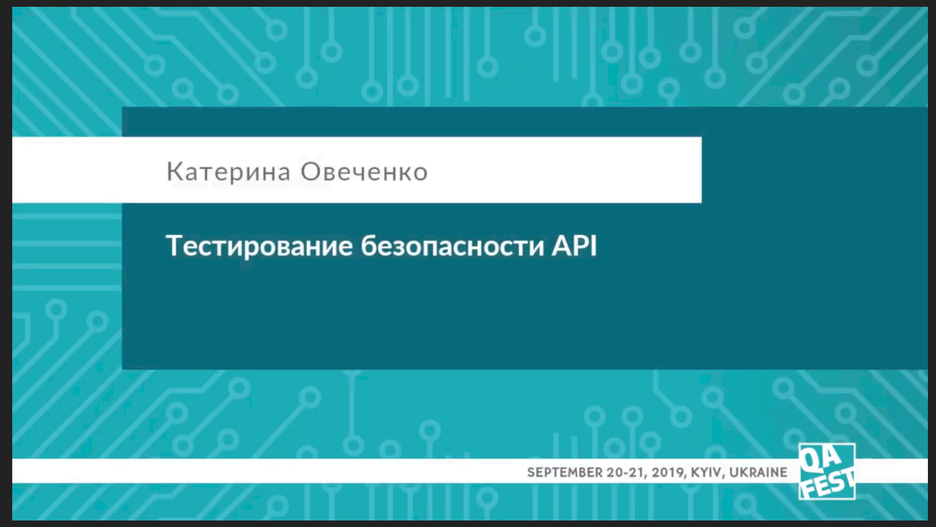
- If you want to prevent or find earlier:
  - Security issues
- ...

PROS & CONS

- If you want to prevent or find earlier:
  - Security issues
- ...
- Sometimes tests fail → Knowledge mining
- Learning and investigation time

WHAT'S NEXT?

- OWASP API Security
- QA Fest 2019: Security API Testing



PROBLEM: A LOT OF MANUALLY MADE AUTOTESTS

- Every new service require much time

APPROACH

- API Schema (Swagger, GraphQL, Something custom*)
- Parsing
- Generate data for each type
- Fuzzing (send requests)
  - Each method
  - Each field
  - Check responses

APPROACH AND TOOLS

- Parsing schema
  - Swagger
- Property-based testing
  - Hypothesis

TOOLS

- Schemathesis (hypothesis-jsonschema)
  - Swagger 2.0 / OpenAPI 3.0
  - GraphQL
    - Full-featured support - in progress
  - Requests
  - Pytest (parametrization)

EXAMPLES: CONNECT TO SWAGGER SCHEMA

```python
import schemathesis

schema = schemathesis.from_uri(
    "https://my-cool-service.com/v2/api-docs",
    verify=False
)


@schema.parametrize()
def test_no_server_errors(case):
    ...
```

EXAMPLES: QUICK START FROM DOCUMENTATION

```python
@schema.parametrize()
def test_no_server_errors(case):
    # `requests` will make an appropriate call
    response = case.call(verify=False)
    # You could use built-in checks
    # case.validate_response(response)
    # Or assert the response manually
    assert response.status_code < 500
```

EXAMPLES: REAL WORLD

```python
@pytest.mark.parametrize("user_role", [...])
@schema.parametrize()
def test_no_server_errors(case, arrange_user, user_role):
    user = arrange_user(role=user_role)
    client = http_client.auth_session(user=user)

    response = case.call(
        headers=client.headers, cookies=client.cookies
    )

    assert response.status_code < 500, "Server error"
```

EXAMPLES: PROPERTY CONFIGURATION

```
1 from hypothesis import HealthCheck
2
3 @hypothesis.settings(
4     suppress_health_check=HealthCheck.all(),
5     max_examples=config.FUZZING_MAX_EXAMPLES
6 )
7 def test_no_server_errors(...):
8     ...
```

EXAMPLES: BUILT IN VALIDATION RULES

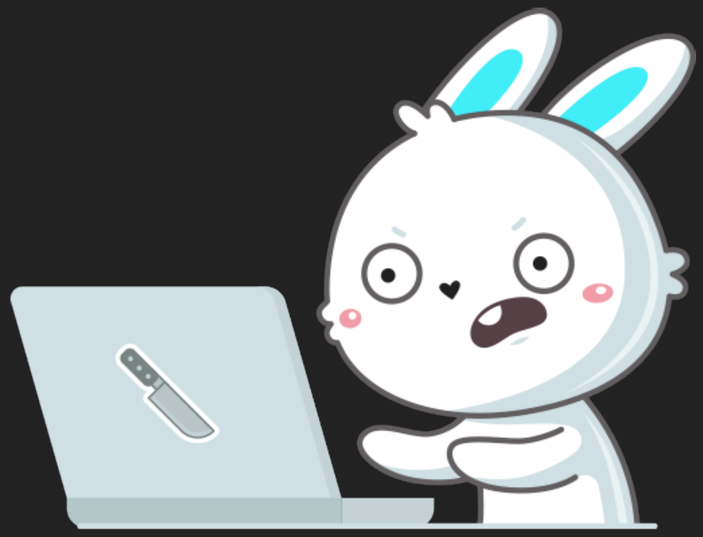
```
1 @pytest.mark.parametrize("user_role", [...])
2 @schema.parametrize()
3 def test_no_server_errors(case, arrange_user, user_role):
4     user = arrange_user(..)
5     client = http_client.auth_session(...)
6
7     response = case.call(...)
8
9     case.validate_response(response=response)
```

PROS

- Rapid start testing for new services:
  - Schema → few minutes → several 5xx errors
- Declarative approach:
  - Describe rules that should be validated
- Lazy test automation:
  - No need to write tests on every new API method

CONS

- Send all required fields and data that fits types
- Need to suppress warnings from Hypothesis
- Require up-to-date swagger schema
  - Could fail on schema parsing
- Pytest-xdist parallel issues
- Many layers → issues require time
- Many dependencies

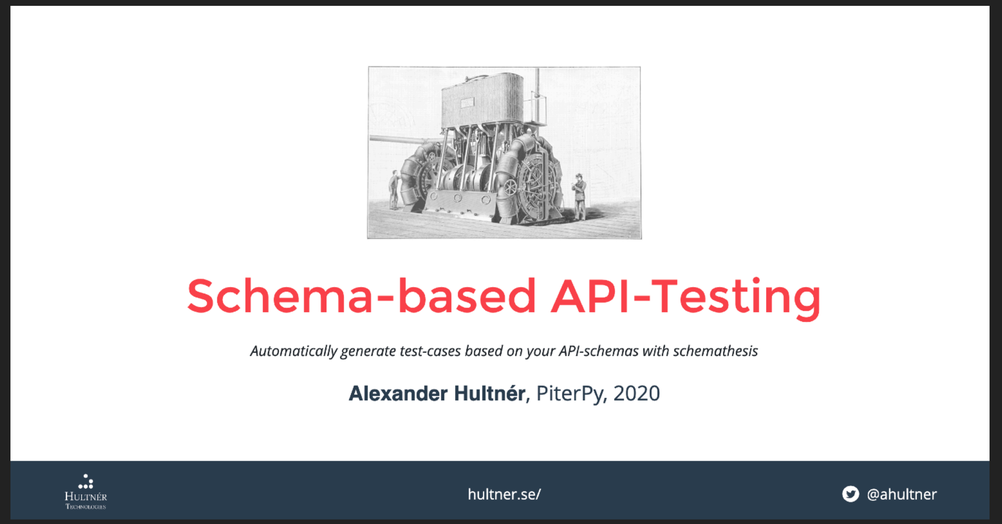


WHAT'S NEXT?

- Introduction to property-based testing (Hypothesis)
- Choosing properties for property-based testing (F#)
- Hypothesis documentation
- Heisenbug 2019 Piter: Property testing: Strategic automation for devs and SDETs

WHAT'S NEXT?

- PiterPy 2020: Schema-based API-Testing – Automatically generate test-cases with schemathesis
- Progress report and plan (2020-10-05)
- Habr (Russian)
- Medium (English)



REMEMBER ALL

- Architecture of your application
  - Approaches and tools
- Common approaches (Risk based scenarios)

REMEMBER ALL

- Next approaches
  - Random data generation
  - Data validation based on schemas
  - Base security checks
  - API Fuzzing

OUTSIDE THE TALK SCOPE

- Contract testing
- Security testing
- Performance testing
- 3rd party integration testing

QUALITY IS ABOUT WHOLE DEVELOPMENT PROCESS

- Shift-left & white box
  - Code review (at least unit tests) & code coverage analysis
  - Static code analysis, code formatters
  - Testability is one of software architecture quality attributes
- Shift-right & SRE
  - AB-testing, Canary releases, Feature toggles

THANK YOU!!!

vk.com

presentation

linkedin.com