BACKEND TEST AUTOMATION FROM THE SCRATCH

IGOR BALAGUROV

QA AUTOMATION TECH LEAD AT VK

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

EXPERIENCE

- VK
  - Payment systems
  - Internal Development
  - Performance testing
- Startups & Product companies
  - iOS, Android, Web, Backend, Machine learning

PLAN

- 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

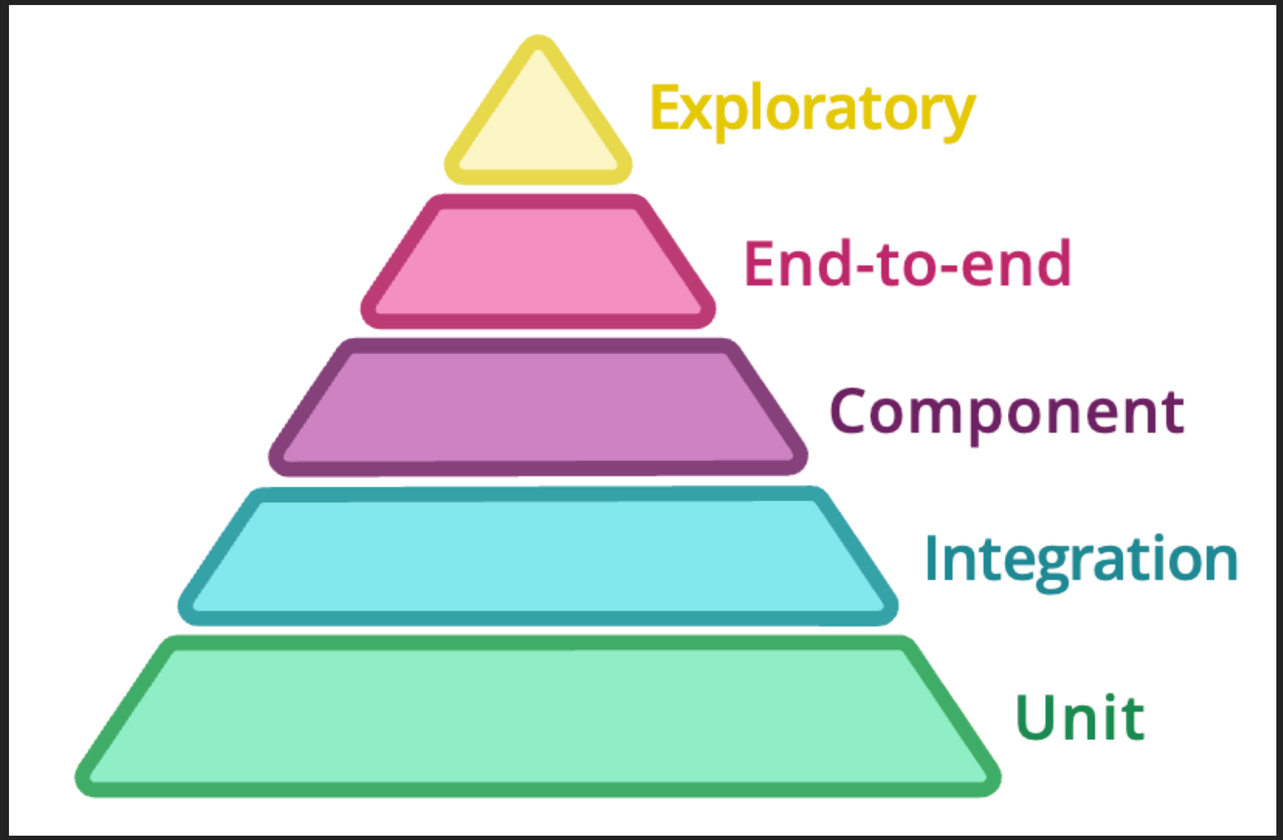
- Application architecture
  - Different approaches, tools and etc

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

PROTOCOLS

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

EDGE CASES

- Configuration
- BigData
- Machine Learning

PROTOCOLS

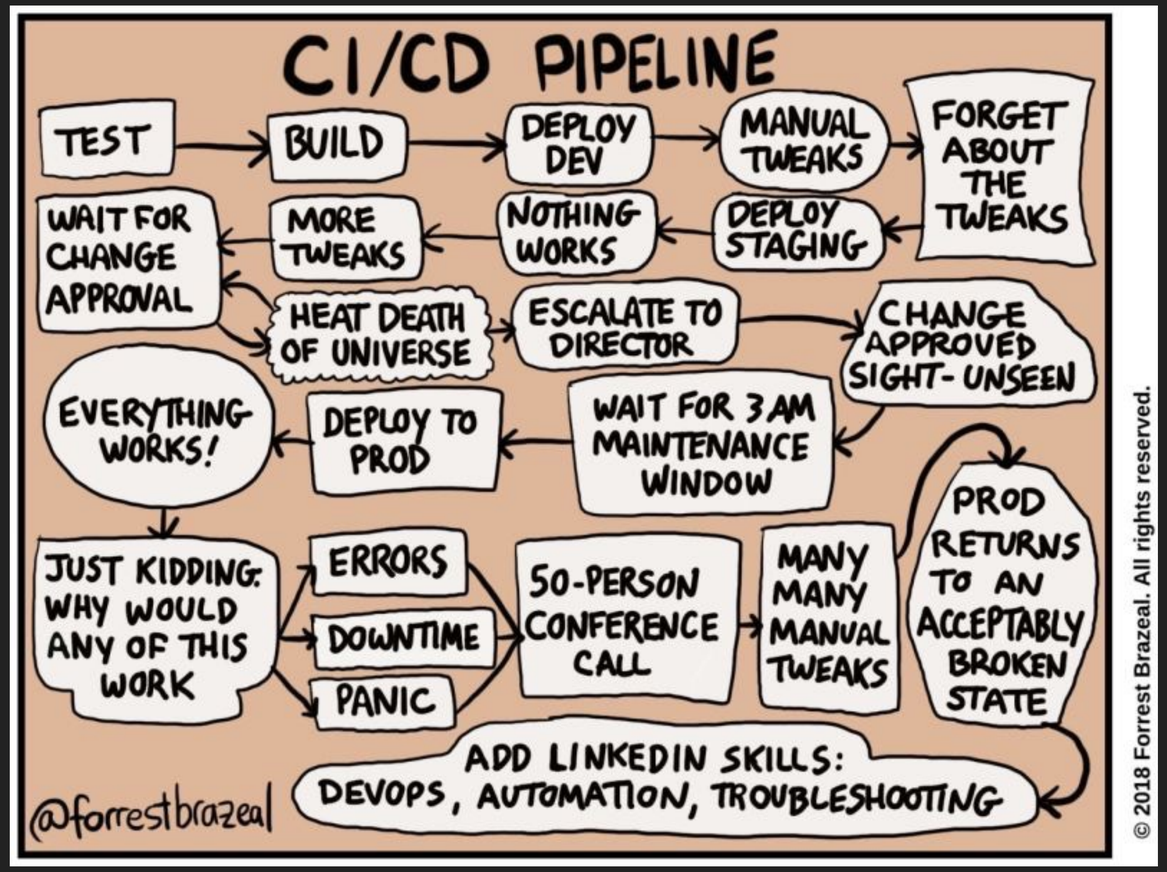
- Interface
  - Input and output data
- How to test?

MANUALLY

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

AUTOMATION

- CI / CD
  - Fast feedback
  - Reduce Time2Market
  - 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

SHORT PROJECT

- Manual
- Postman
- Hello world automation

SHORT PROJECT

- Problems
  - Project became bigger
    - Throw away old approaches

FEATURE TEAMS

- Same repositories
- Same technologies?

FEATURE TEAMS

- Problems
  - Full stack QA Engineers?
  - Huge QA Team inside each feature team

BIG PROJECT + SMALL QA TEAM

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

BIG PROJECT + SMALL QA TEAM

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

BIG PROJECT + BIG QA TEAM

- Same technologies

BIG PROJECT + BIG QA TEAM

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

COMPANY TYPES

- Challenges

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/Kotlin: a new hope

PYTHON AND REST API

- Approaches are language agnostic
- REST API

TEST RUNNER

- Parametrization
- Parallel execution
- Rerun failures
- Soft asserts
- Ability to extend

PYTEST

- Good community
- Nested fixtures (setup+teardown)
- Hook based Plugin Architecture
  - Fast tests development
  - Easy to extend
    - Complex tests, different protocols, many integrations and etc.

HTTP CLIENT

```python
import requests

response = requests.post(
    url="https://your-site.com",
    json={"a": 1}
)

response.request.headers["content-type"]

# 'application/json'
```

REQUESTS

- Good community
- Dead simple client
- API standard

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: 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"
```

RANDOMIZATION: PROS & CONS

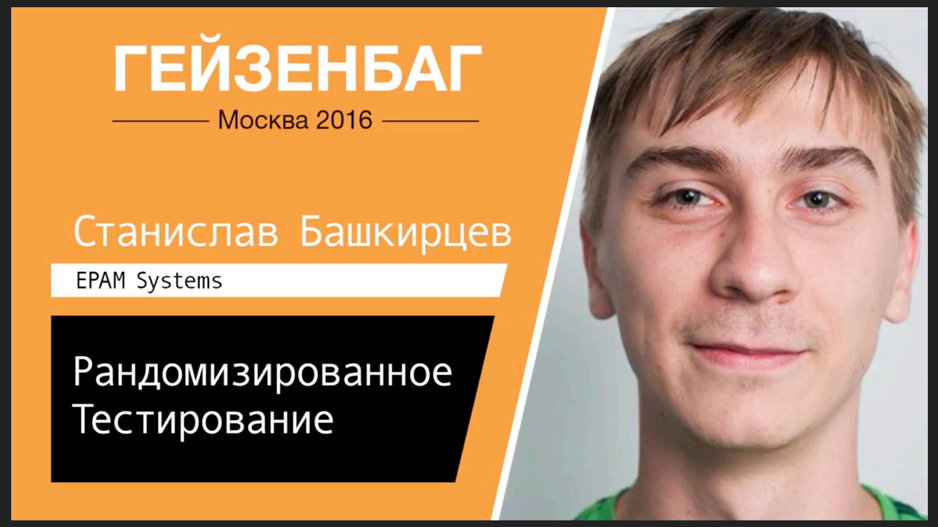
- Reusable values
- It could be used as standalone data generator solution
- Combinations
- ...

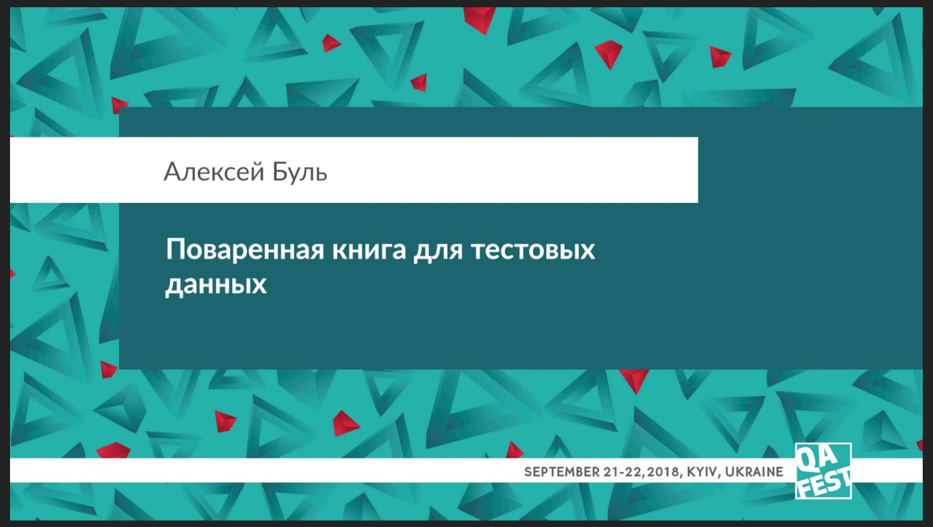
RANDOMIZATION: PROS & CONS

- Reusable values
- 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

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

EXAMPLES

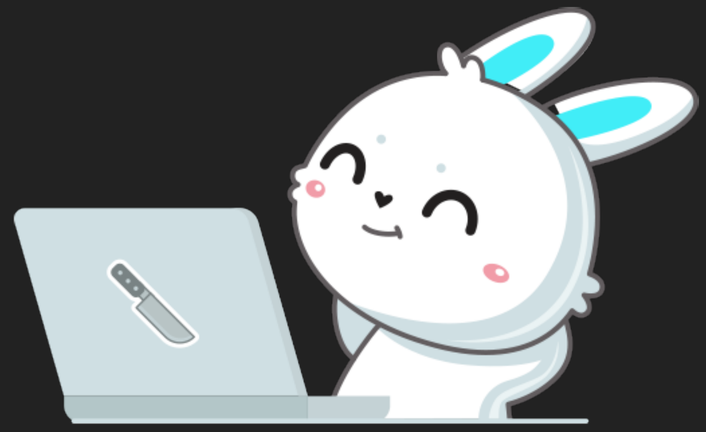
```
data_to_validate = response.json()
# {"data": {"userId": 0, "balance": -0.1}}

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

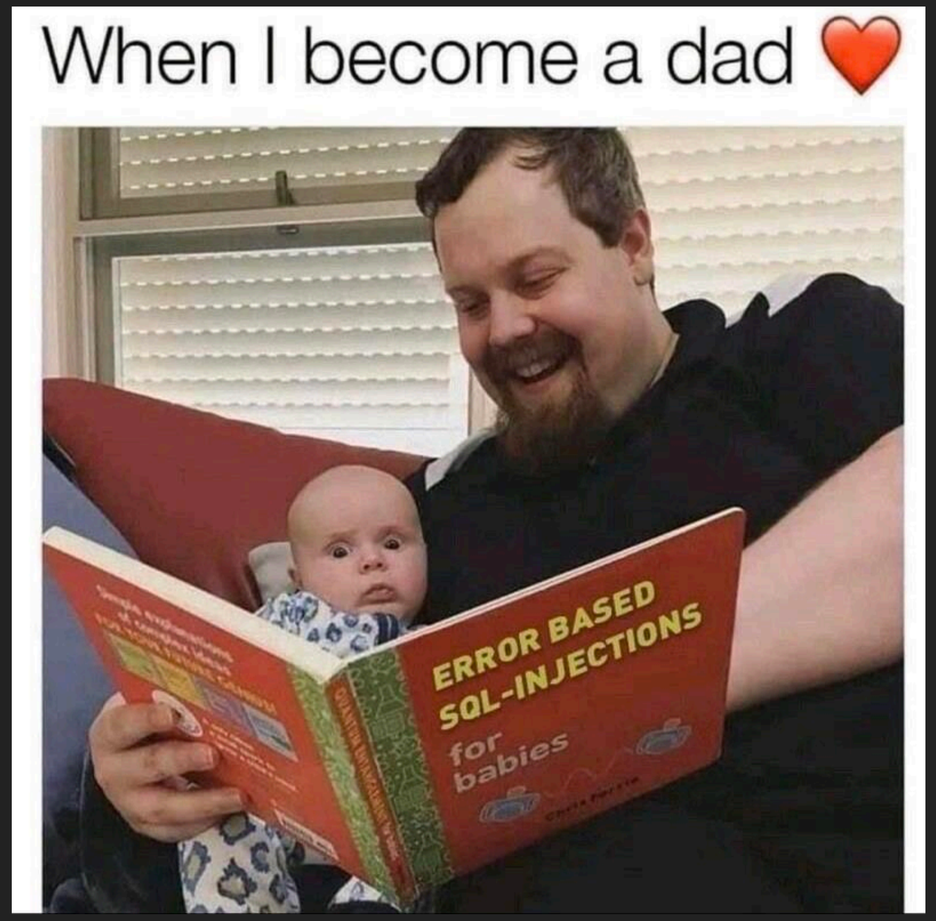
- Check every request param
  - Overflow attempts
  - Injections
- Check concurrent responses
  - Rate limits
  - Race conditions

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

```python
from functools import partial
from src import vk_pay, test_data, parallel

transaction_id = test_data.create_transaction()

api_method = partial(
    func=vk_pay.confirm_transaction,
    transaction_id=transaction_id
)
```

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

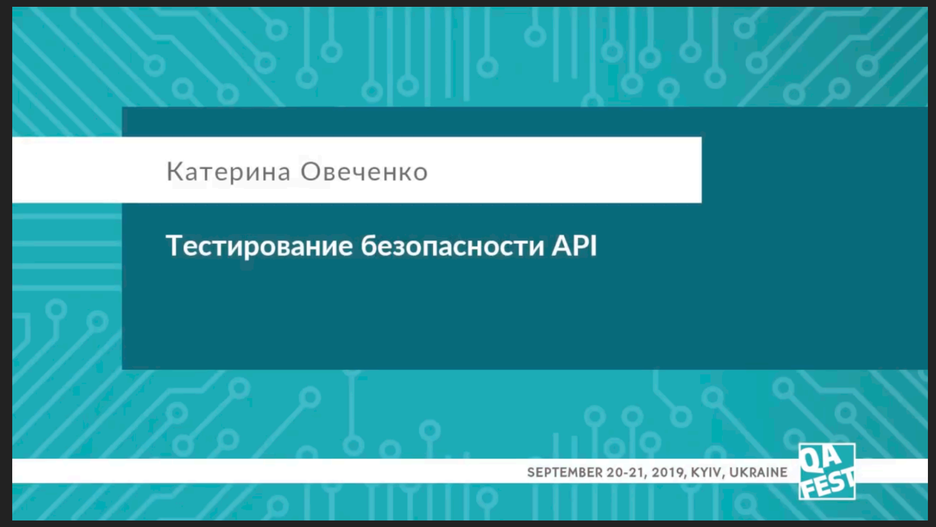
- If you want to prevent or find earlier:
  - Security issues
- …

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



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"
)


@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()
    # You could use built-in checks
    # case.validate_response(response)
    # Or assert the response manually
    assert response.status_code < 500
```

EXAMPLES: REAL WORLD

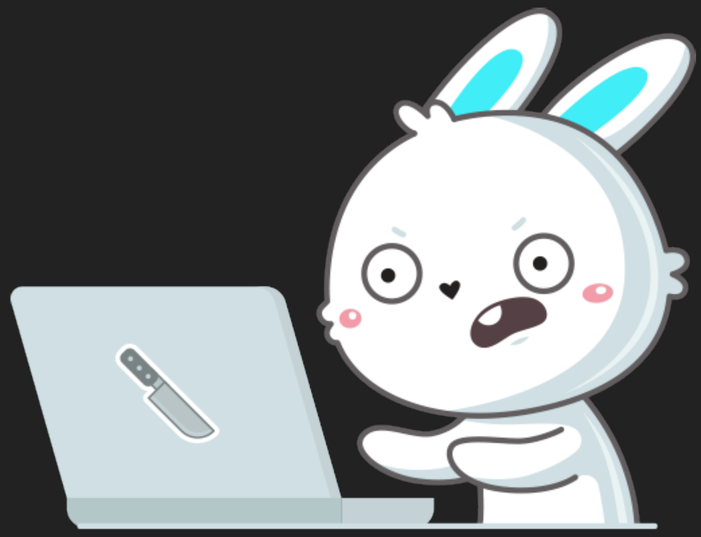
```python
@pytest.mark.parametrize("user_role", [...])
@schema.parametrize()
def test_no_server_errors(case, arrange_user, user_role):
    user = arrange_user(role=user_role)
    client = http_client.auth_session(user=user)

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

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

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
- Require up-to-date swagger schema
  - Could fail on schema parsing
- Pytest-xdist parallel issues
- Many layers → issues require time
- Many dependencies

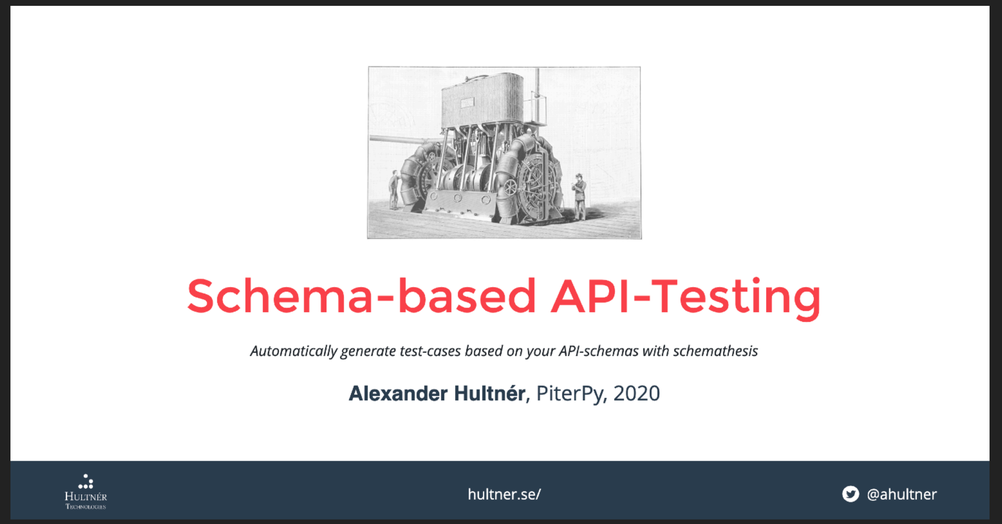


WHAT'S NEXT?

- 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
  - Property based 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