Разработка средств визуализации для анализа покрытия auto API-тестами

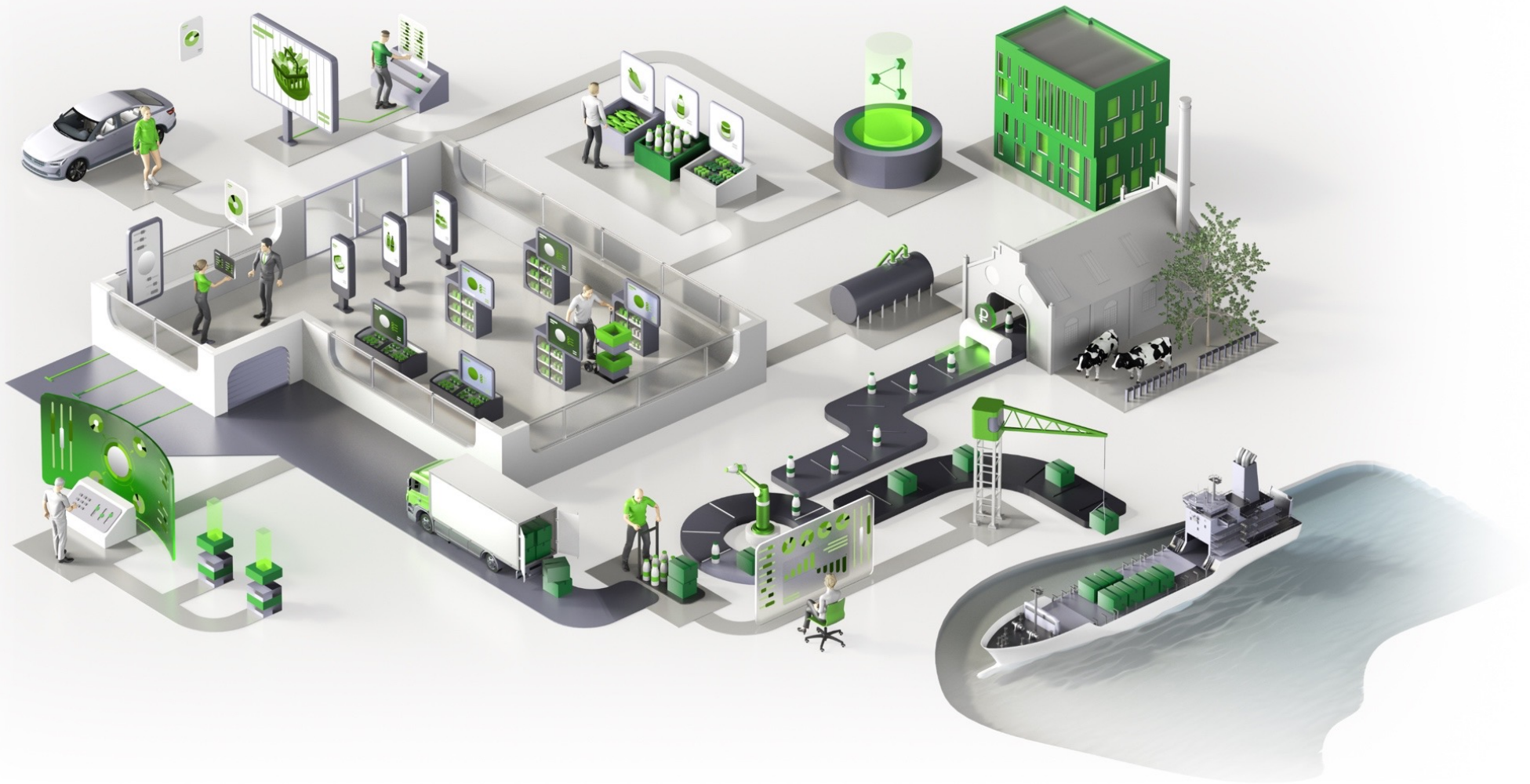
X5 Tech

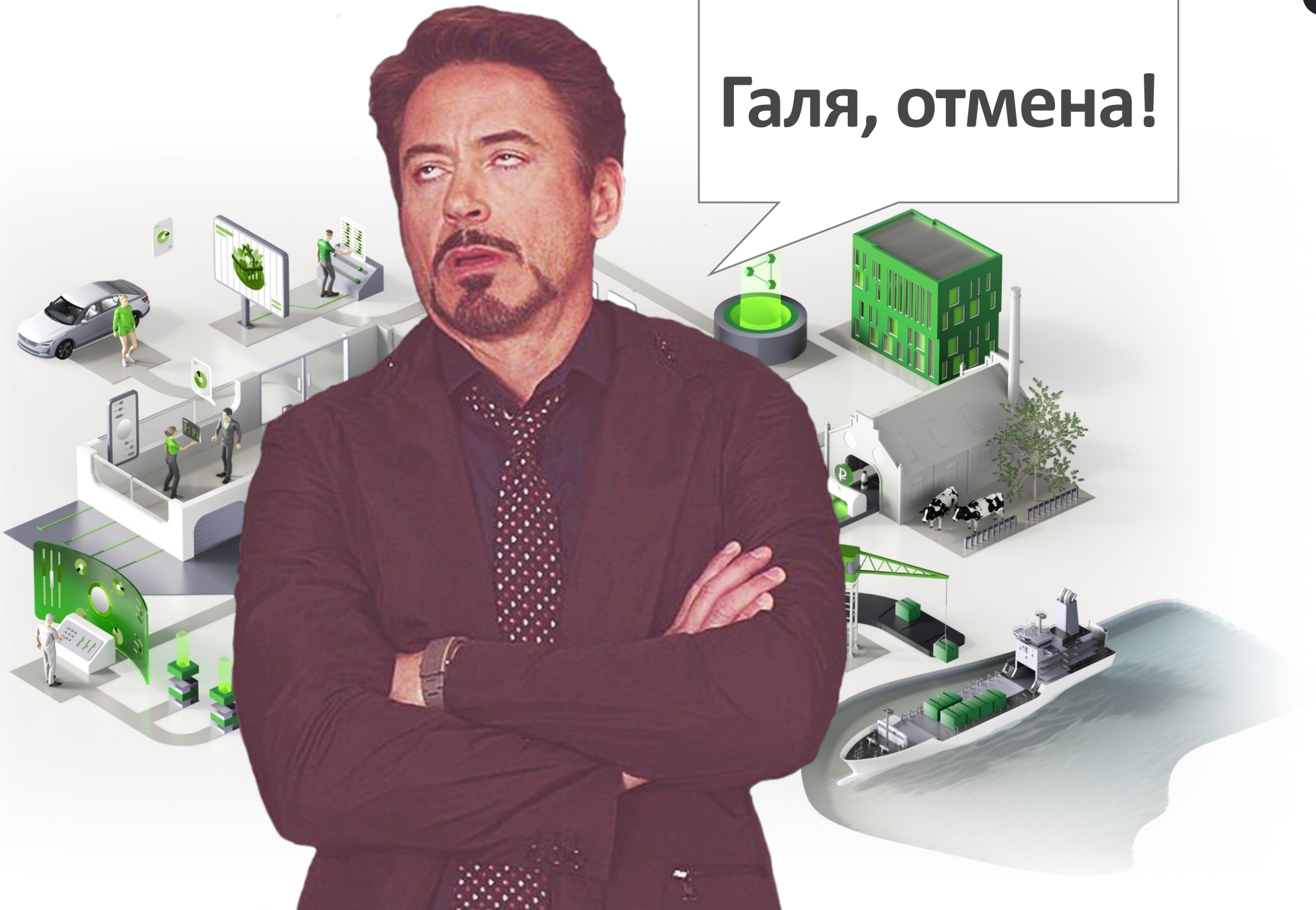
Храбров Кирилл

kirill.khrabrov@yandex.ru

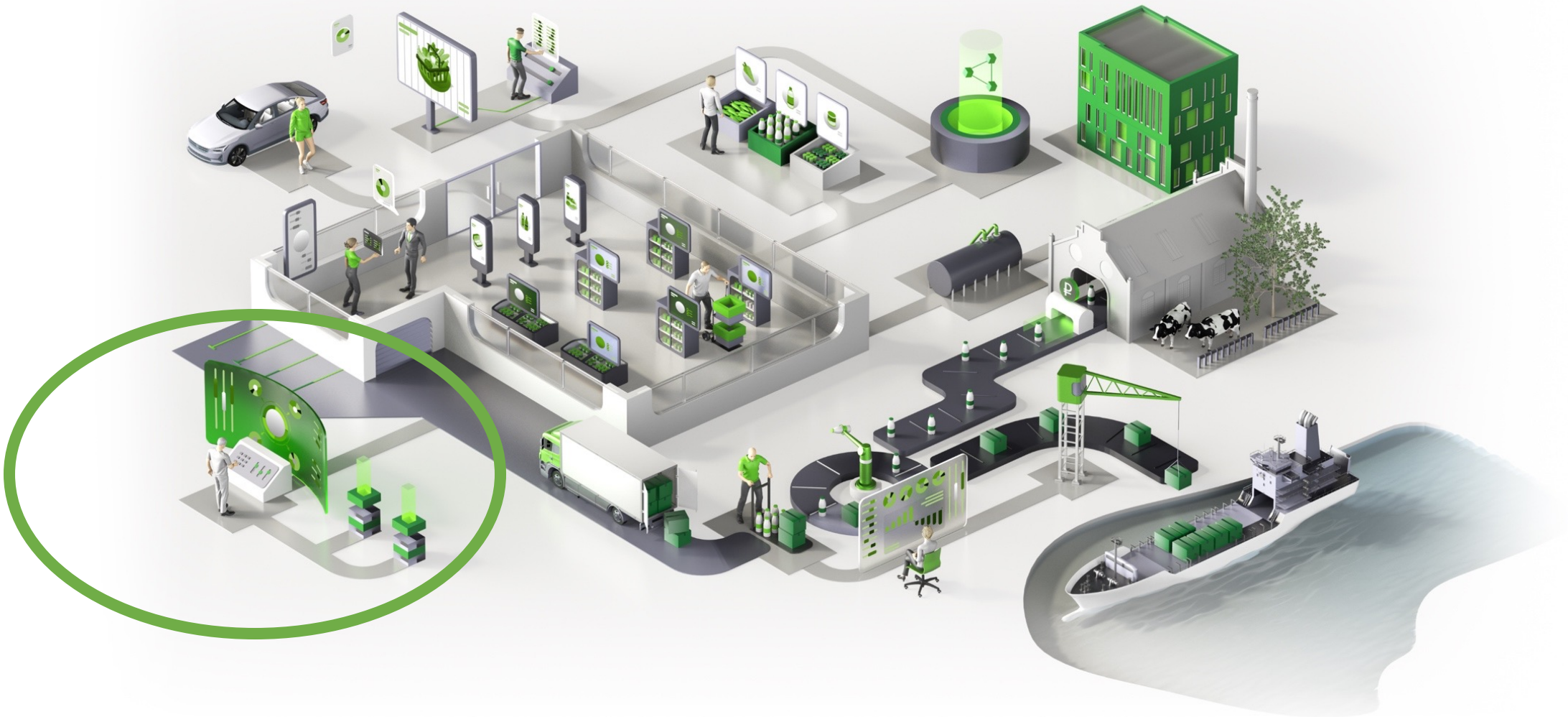
@kirill_khrabrov

Операционные и логистические процессы в магазинах




Галя, отмена!

Стабильность работы нашего продукта – критически важная цель для команды



План

- [x] Тестовое покрытие
- [x] Продумываем архитектуру
- [x] «Фактический результат» (FR) из Allure
- [x] «Ожидаемый результат» (ER) из ELK
- [x] Анализ ER vs FR и его визуализация

Теперь у нас есть какой-то план и мы будем его придерживаться!

HEISENBUG
2019 MOSCOW

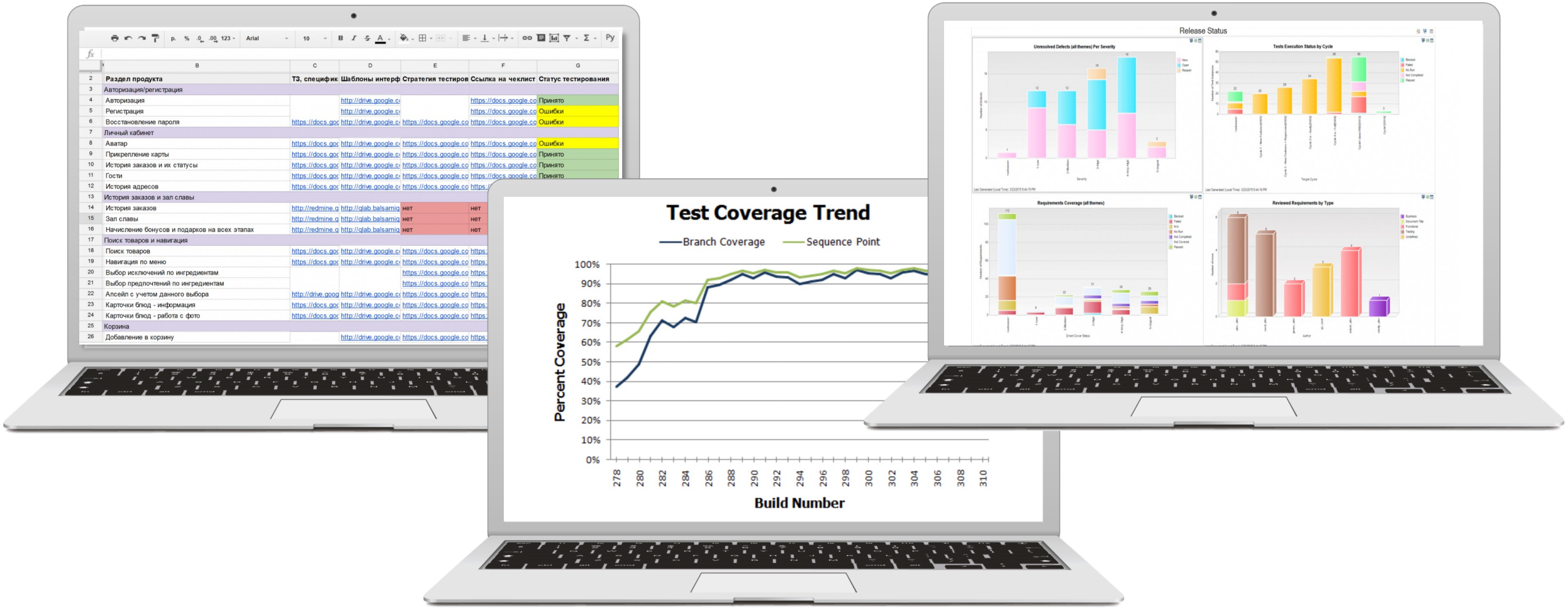
Артем Ерошенко
Qameta Software

Визуализация покрытия автотестов

Тестовое покрытие

- Плотность покрытия тестами требований либо исполняемого кода
- Можно выразить в процентном соотношении покрытия требований к написанным тестам

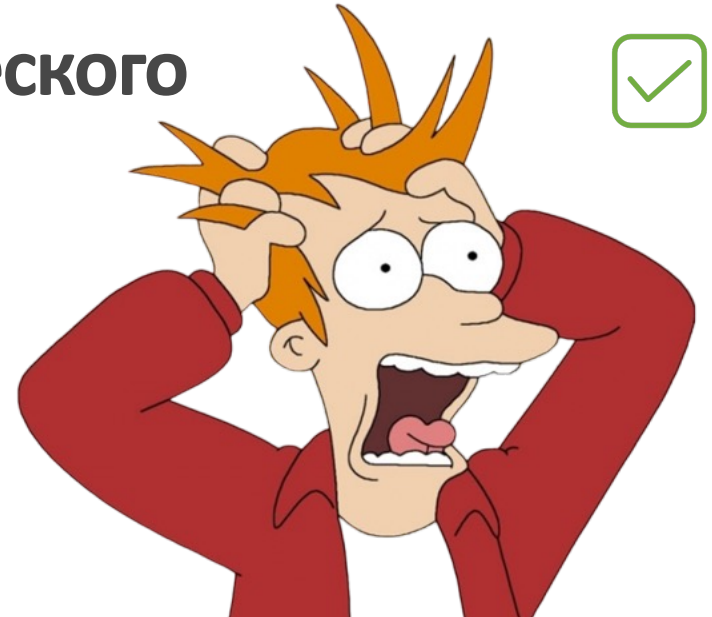
Тестовое покрытие



X5 Tech

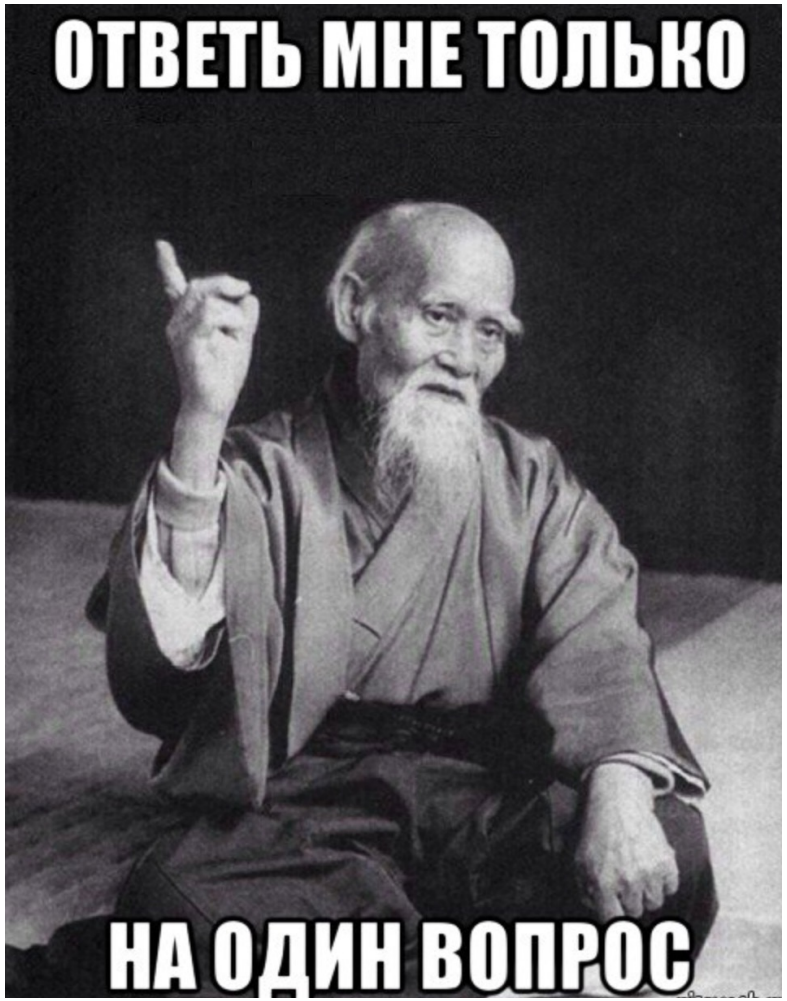
Тестовое покрытие

- Собирать информацию по покрытию требований сложно
- Требования все время меняются
- Зависит от человеческого фактора
- Необходимо составлять матрицы трассировки требований
- Результаты анализа не очевидны
- Требования не атомарны или их нет вообще

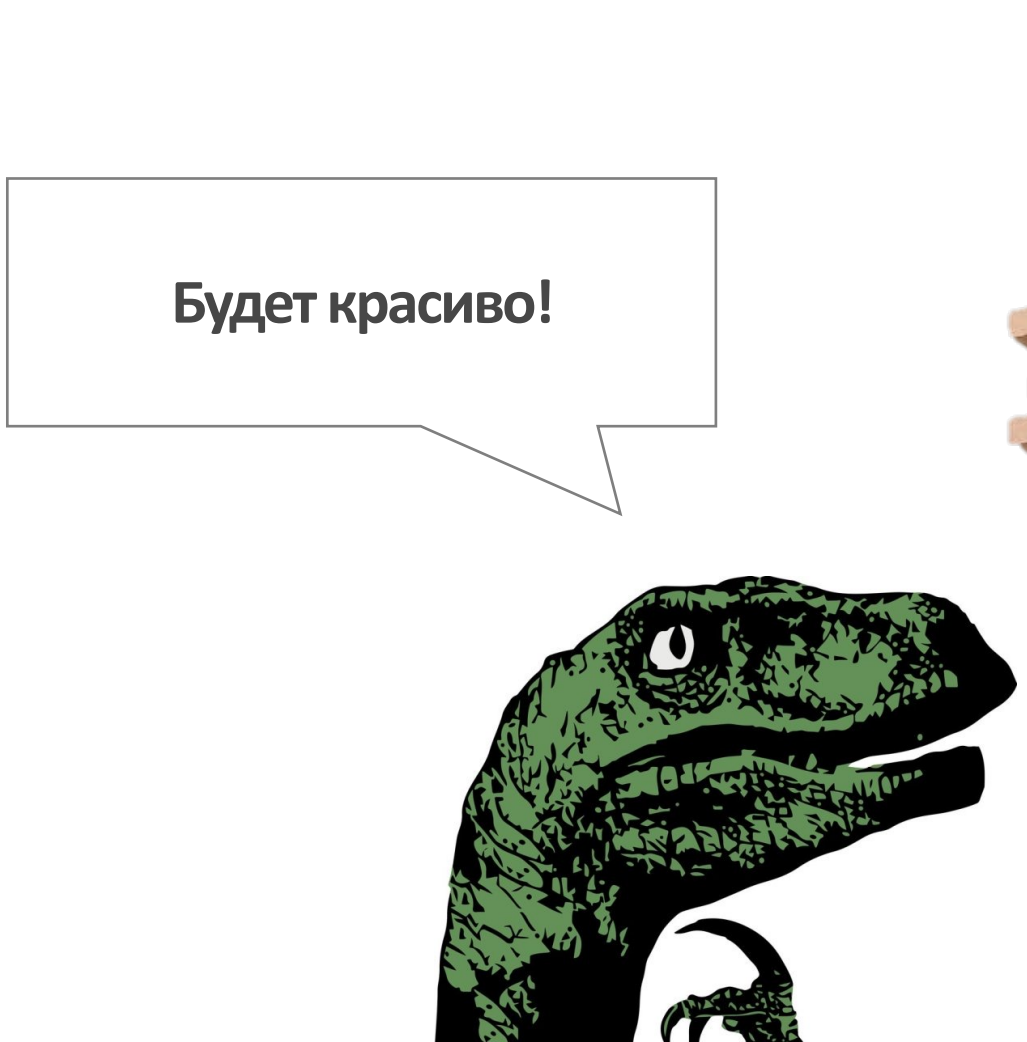


Тестовое покрытие

А покрыты ли у нас тестами методы API, которыми часто пользуются?



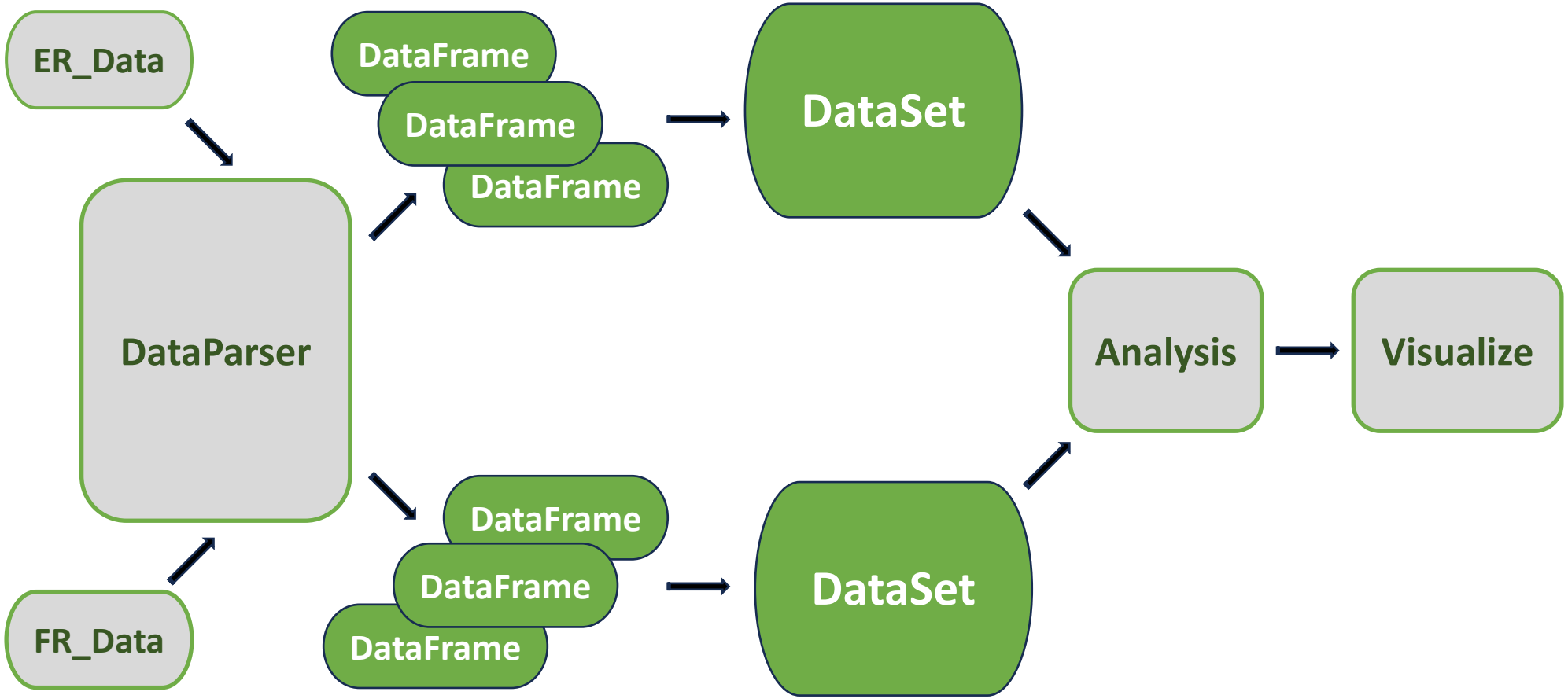
Продумываем архитектуру



Продумываем архитектуру

- [x] Данные об использовании API
- [x] Данные о покрытых тестами методах API
- [x] Парсинг источников данных
- [x] Сохранение результатов парсинга
- [x] Обработка результатов парсинга
- [x] Визуализация результатов обработки
- [x] Вывод результатов визуализации

Продумываем архитектуру



Продумываем архитектуру

DataParser

Паттерны Regexp
Источники данных
Методы чтения источников
Методы обработки данных в источниках

Продумываем архитектуру

DataParser

```
class DataParser:
    PATTERN_HTTP_REQ = r'(GET|POST|PUT|DELETE|PATCH).*(\/api\S*);?'
    PATTERN_UUID4 = r'[0-9a-f]{8}-[0-9a-f]{4}-[0-5][0-9a-f]{3}-[089ab][0-9a-f]{3}-[0-9a-f]{12}'
    PATTERN_OBJ_ID = r'\/[^v]?\d{1,5}\D?\/'

    def __init__(self, fact_results_source=None, expected_results_source=None):
        self.fact_results_source = fact_results_source
        self.expected_results_source = expected_results_source
```

Продумываем архитектуру

DataFrame

API метод
Перечень query параметров
Статус теста
Количество вызовов

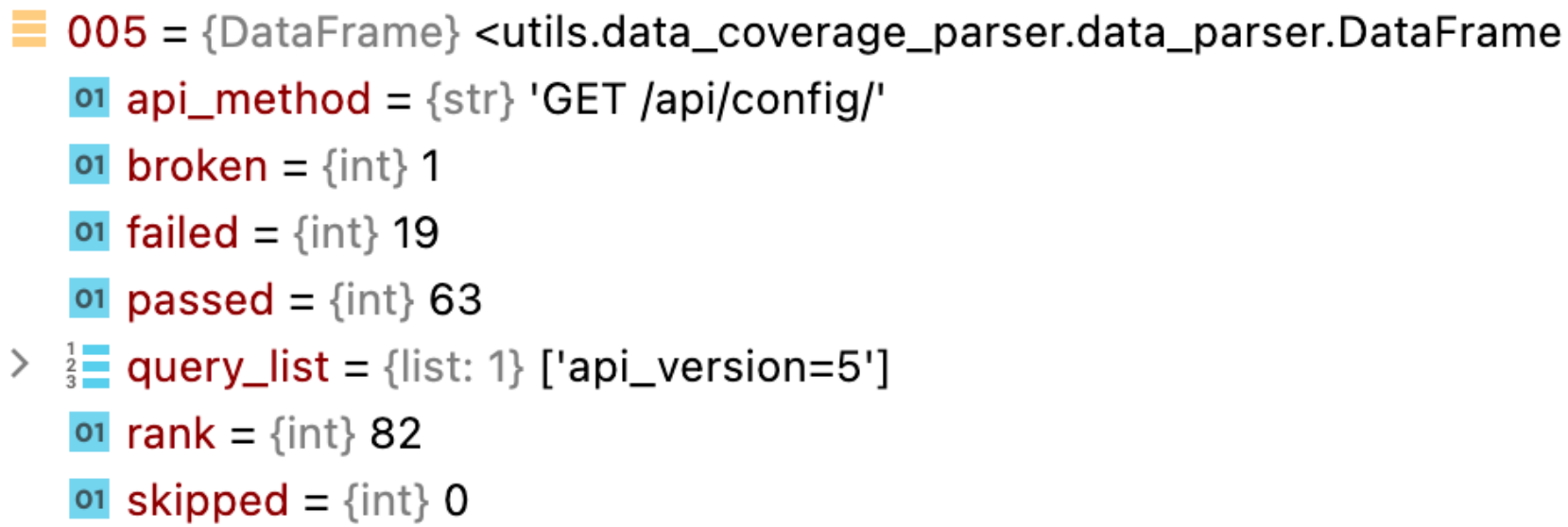
Продумываем архитектуру

DataFrame

```
class DataFrame:
    def __init__(self, api_method: str, test_status=None, query_list=[]):
        self.api_method = api_method
        self.query_list = query_list
        self.rank = 0
        self.passed = 1 if test_status == "passed" else 0
        self.failed = 1 if test_status == "failed" else 0
        self.broken = 1 if test_status == "broken" else 0
        self.skipped = 1 if test_status == "skipped" else 0
```

Продумываем архитектуру

DataFrame



Продумываем архитектуру

DataSet

Массив DataFrame’ов
Методы сохранения DataFrame’ов

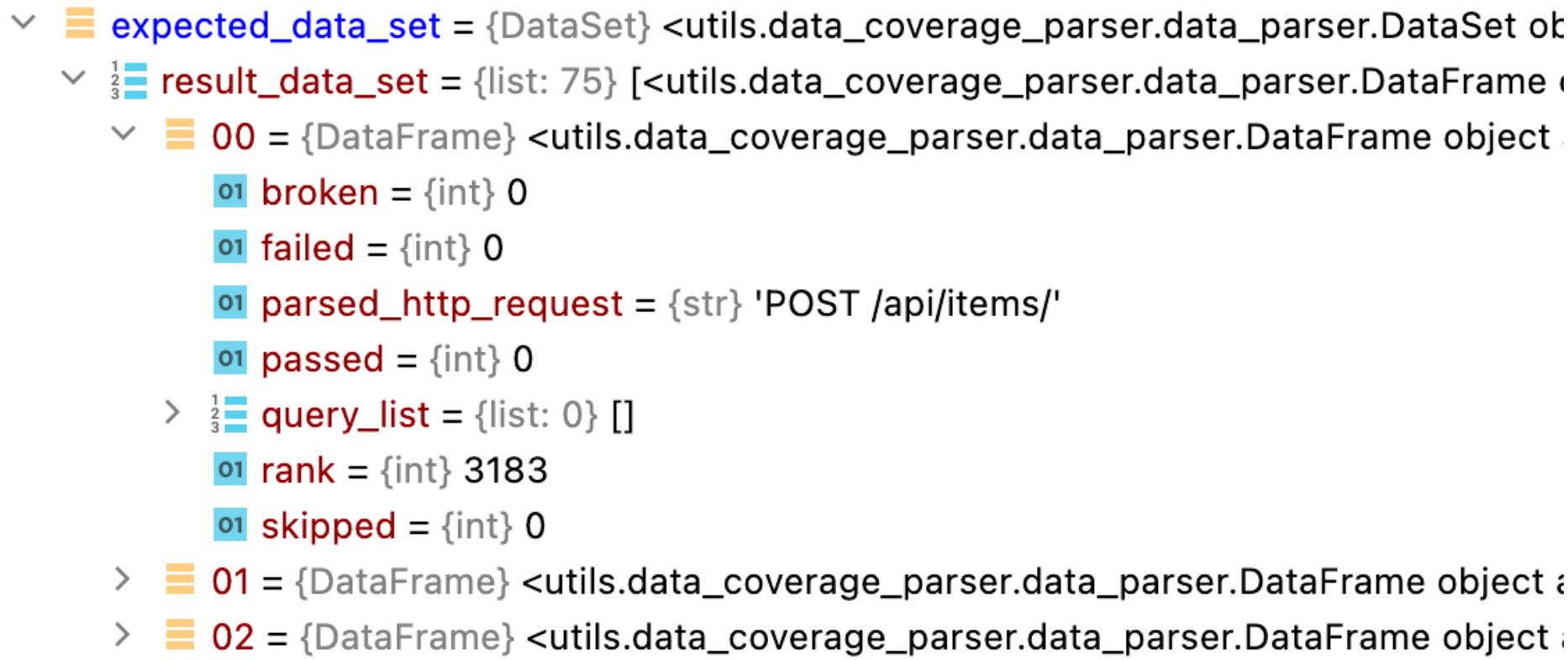
Продумываем архитектуру

DataSet

```
class DataSet:
    def __init__(self):
        self.result_data_set = []

    def _increase_status(self, data_frame: DataFrame):
        pass

    def _increase_rank(self, data_frame: DataFrame):
        pass

    def _append_query_params(self, data_frame: DataFrame):
        pass

    def append_data_frame(self, data_frame: DataFrame):
        pass
```

Продумываем архитектуру

DataSet



Продумываем архитектуру

Analysis

Методы обработки DataSet’ов
Методы получения результирующих DataSet’ов

Продумываем архитектуру

Analysis

```
class CoverageVisualizer:
    def __init__(self, fact_data_set: DataSet, expected_data_set: DataSet):
        self.fact_data_set = fact_data_set.result_data_set
        self.expected_data_set = expected_data_set.result_data_set

    def get_top_n_api_methods(data_set: DataSet, top_n: int):
        pass

    def get_tested_unused_api_methods(self, expected_data_set=None, fact_data_set=None):
        pass

    def get_untested_used_api_methods(self, expected_data_set=None, fact_data_set=None):
        pass

    def get_tested_api_methods(self, expected_data_set=None, fact_data_set=None):
        pass

    def get_fact_stat(self, expected_data_set: list):
        pass
```

Продумываем архитектуру

Visualize

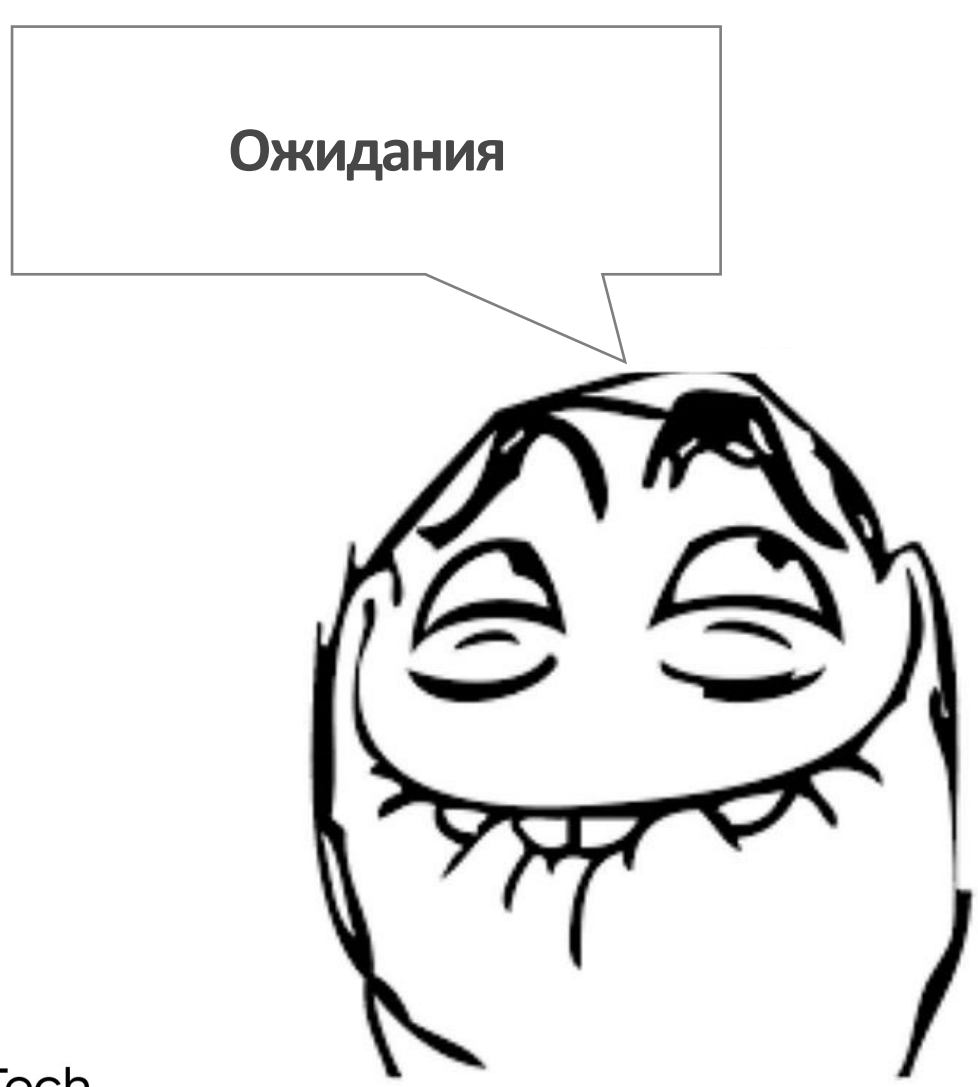
Методы визуализации результатов обработки DataSet’ов

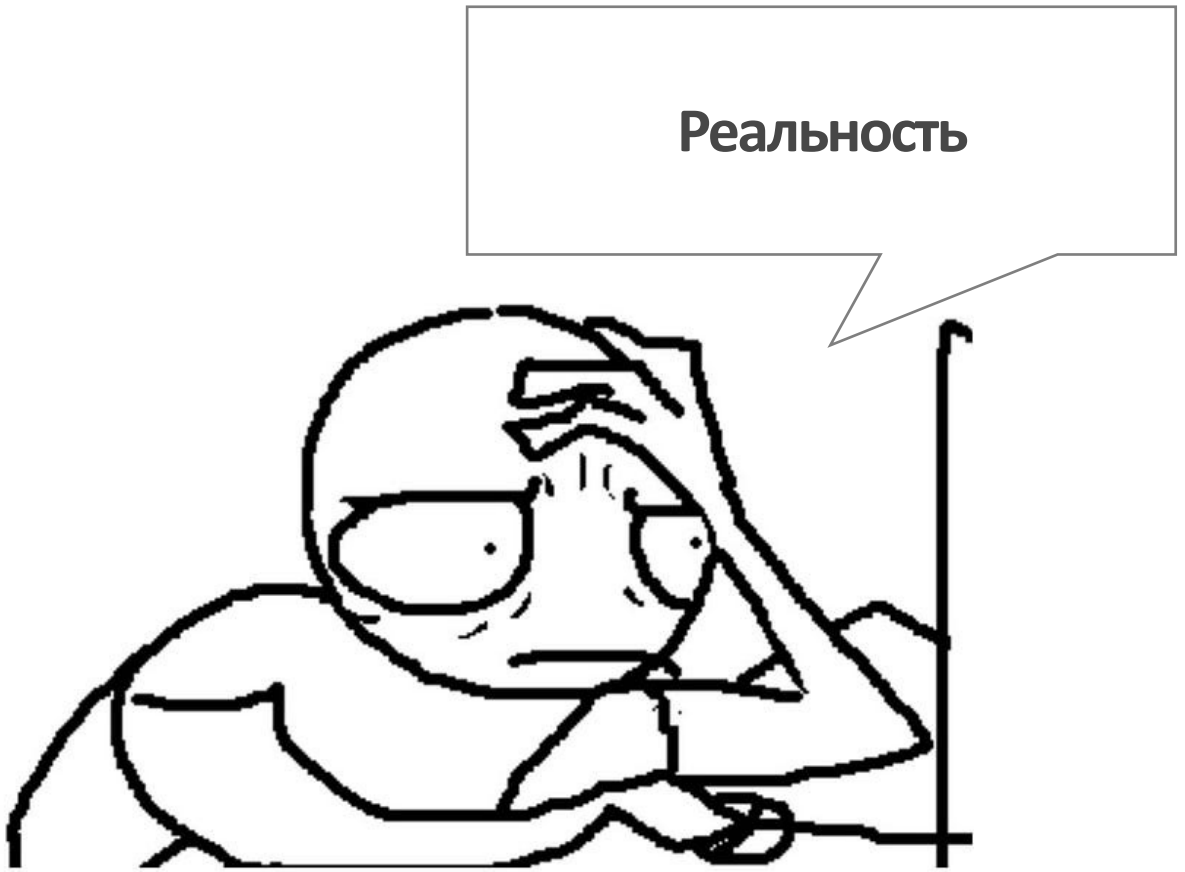
Продумываем архитектуру

Visualize

```
class CoverageVisualizer:
    def __init__(self, fact_data_set: DataSet, expected_data_set: DataSet):
        self.fact_data_set = fact_data_set.result_data_set
        self.expected_data_set = expected_data_set.result_data_set

    def build_info_table(**kwargs)::
        pass

    def build_info_gist_on_code_coverage(**kwargs):
        pass

    def build_info_barh_on_code_coverage(width=0.8, **kwargs):
        pass
```

«Фактический результат» (FR) из Allure

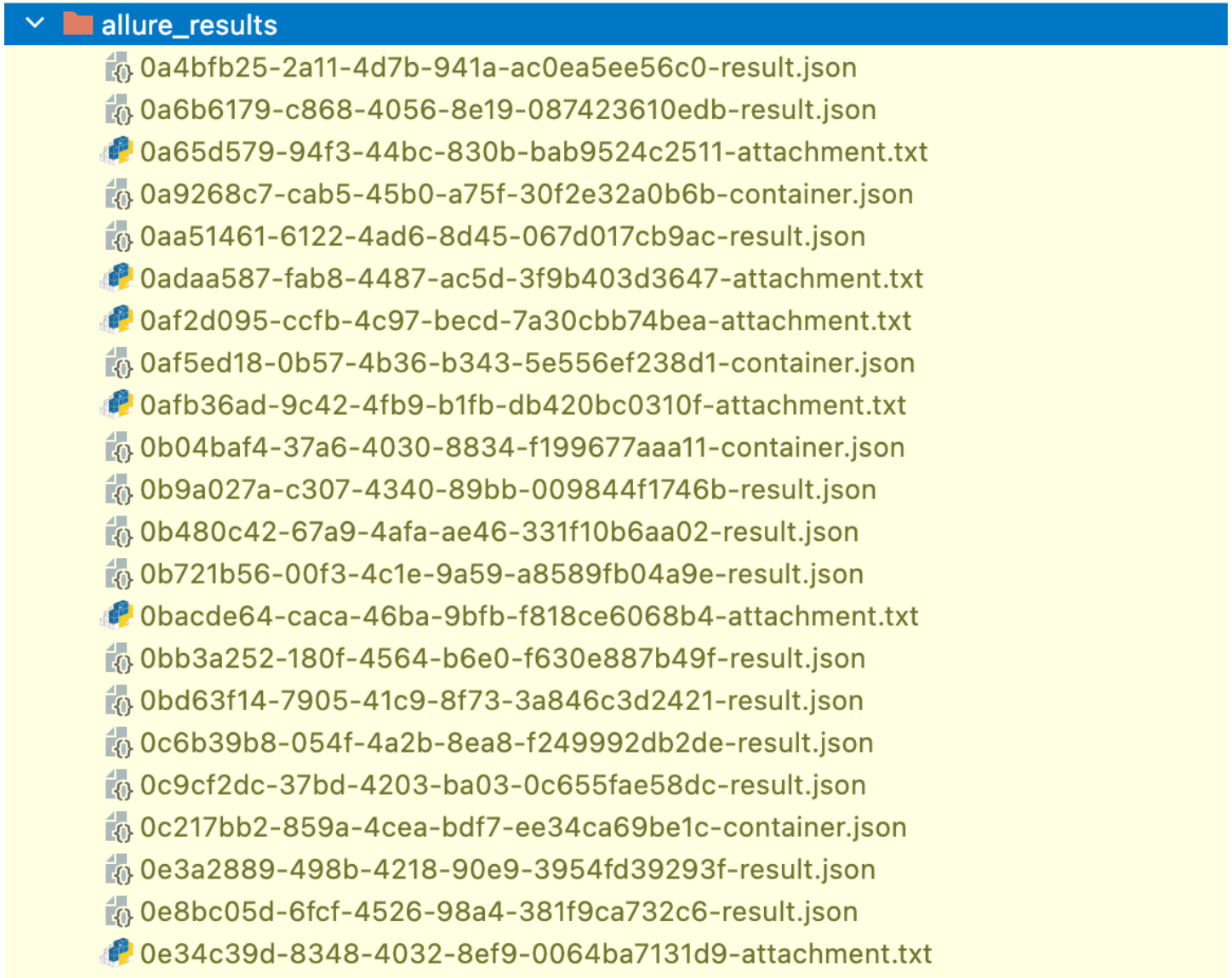




«Фактический результат» (FR) из Allure

```
pytest --alluredir=$ALLURE_RESULTS
```

«Фактический результат» (FR) из Allure



«Фактический результат» (FR) из Allure

```
*-result.json
```

«Фактический результат» (FR) из Allure

```
{
    "name": "Имя тестового кейса",
    "status": "passed",
    "steps": [
        {
            "name": "Имя шага теста № 1",
            "status": "passed",
            "parameters": [...],
        },
        ...
    ],
    "attachments": [
        {
            "name": "log",
            "source": "***-attachment.txt",
            "type": "text/plain"
        }
    ],
    "fullName": "tests.api-web.group_tasks...",
    "labels": [
        {
            "name": "...",
            "value": "..."
        },
        ...
    ]
}
```

«Фактический результат» (FR) из Allure

```
{
    "name": "Имя тестового кейса",
    "status": "passed",
    "steps": [
        {
            "name": "Имя шага теста № 1",
            "status": "passed",
            "parameters": [...],
        },
        ...
    ],
    "attachments": [
        {
            "name": "log",
            "source": "***-attachment.txt",
            "type": "text/plain"
        }
    ],
    "fullName": "tests.api-web.group_tasks...",
    "labels": [
        {
            "name": "...",
            "value": "..."
        },
        ...
    ]
}
```

«Фактический результат» (FR) из Allure

```
{
    "name": "Имя тестового кейса",
    "status": "passed",
    "steps": [
        {
            "name": "Имя шага теста № 1",
            "status": "passed",
            "parameters": [...],
        },
        ...
    ],
    "attachments": [
        {
            "name": "log",
            "source": "***-attachment.txt",
            "type": "text/plain"
        }
    ],
    "fullName": "tests.api-web.group_tasks...",
    "labels": [
        {
            "name": "...",
            "value": "..."
        },
        ...
    ]
}
```

«Фактический результат» (FR) из Allure

```
class Logger:
    def __init__(self):
        self.logger = logging.getLogger()
        self.logger.setLevel(logging.INFO)
        self.consoleHandler = logging.StreamHandler()
        self.consoleHandler.setLevel(logging.WARNING)
        self.logger.addHandler(self.consoleHandler)

    def write_text_log(self, level: str, message: str) -> None:
        if level.lower() == 'info':
            self.logger.info(message)
        elif level.lower() == 'warning':
            self.logger.warning(message)
        elif level.lower() == 'error':
            self.logger.error(message)
```

«Фактический результат» (FR) из Allure

```
class BaseApi:
    @allure.step("Базовый метод GET")
    def get_method_call(
            self,
            host: str,
            path: str,
            token: object
    ) -> requests.Response:

        """Базовый метод GET"""
        response = requests.request(
            "GET",
            url=host + path,
            headers=headers,
            data={},
            verify=False
        )

        logger.write_rest_log(response)
```

«Фактический результат» (FR) из Allure

```
{
    "name": "Имя тестового кейса",
    "status": "passed",
    "steps": [
        {
            "name": "Имя шага теста № 1",
            "status": "passed",
            "parameters": [...],
        },
        ...
    ],
    "attachments": [
        {
            "name": "log",
            "source": "***-attachment.txt",
            "type": "text/plain"
        }
    ],
    "fullName": "tests.api-web.group_tasks...",
    "labels": [
        {
            "name": "...",
            "value": "..."
        },
        ...
    ]
}
```

«Фактический результат» (FR) из Allure

```
*-attachments.txt
```

«Фактический результат» (FR) из Allure

[32mINFO [0m root:logger.py:20 POST: /api-web/group-tasks/; 0:00:00.778308; data: b'{"priority": 0, "name": "AT \\u0421\\u043e\\u0437\\u0434\\u0430\\u043d\\u0438\\u0435 \\u0437\\u0430\\u0434\\u0430\\u0447\\u0438 VoCTPAjbAV", "description": "khYJfFoLpQsxfAkFqHNuhxXzZRfEJorbUWsdwARYQhZHPIVqxHmDrSwCzNEDUBG dPqbXjbKAwafvUEingNeJOTuCiewjGETqdQbi", "deadline": "2023-09-23T10:18:24.000000Z", "assign_to": "counter", "is_attachment_required": false, "is_commentary_required": true, "stores": [], "regions": [101], "divisions": [], "attachments": []}'
[32mINFO [0m root:logger.py:20 GET: /api-web/group-tasks/c0ff8864-e29f-41b1-867d-a741b76be6ce/; 0:00:00.283921; data: None
[32mINFO [0m root:logger.py:20 DELETE: /qa-test-api/web/grouptask/c0ff8864-e29f-41b1-867d-a741b76be6ce/; 0:00:01.174668; data: None
[32mINFO [0m root:logger.py:27 result: status: 204; text:

«Фактический результат» (FR) из Allure

```
PATTERN_HTTP_REQ = r'(GET|POST|PUT|DELETE|PATCH).*(\/api\S*);?'

re.findall(DataParser.PATTERN_HTTP_REQ, log_content)
```

«Фактический результат» (FR) из Allure


```
[32mINFO [0m root:logger.py:20 POST: /api-web/group-tasks/;
0:00:00.778308; data: b'{"priority": 0, "name": "AT
\\u0421\\u043e\\u0437\\u0434\\u0430\\u043d\\u0438\\u0435
\\u0437\\u0430\\u0434\\u0430\\u0447\\u0438 VoCTPAjbAV",
"description":
"khYJfFoLpQsxfAkFqHNuhxXzZRfEJorbUWsdwARYQhZHPIVqxHmDrSwCzNEDUBG
dPqbXjbKAwafvUEingNeJOTuCiewjGETqdQbi", "deadline": "2023-09-
23T10:18:24.000000Z", "assign_to": "counter",
"is_attachment_required": false, "is_commentary_required": true,
"stores": [], "regions": [101], "divisions": [], "attachments":
[]}'
[32mINFO [0m root:logger.py:20 GET: /api-web/group-
tasks/c0ff8864-e29f-41b1-867d-a741b76be6ce/; 0:00:00.283921;
data: None
[32mINFO [0m root:logger.py:20 DELETE: /qa-test-
api/web/grouptask/c0ff8864-e29f-41b1-867d-a741b76be6ce/;
0:00:01.174668; data: None
[32mINFO [0m root:logger.py:27 result: status: 204; text:
```

«Фактический результат» (FR) из Allure

https://regex101.com/

MATCH INFORMATION

Match 1	36-64	POST: /api-web/group-tasks/;
Group 1	36-40	POST
Group 2	42-64	/api-web/group-tasks/;
Match 2	1773-1837	GET: /api-web/group-tasks/c0ff8864-e29f-41b1-867d-a741b76be6ce/;
Group 1	1773-1776	GET
Group 2	1778-1837	/api-web/group-tasks/c0ff8864-e29f-41b1-867d-a741b76be6ce/;

«Фактический результат» (FR) из Allure


```
http_path = '/api-web/stores/c0ff8864-e29f-41b1-867d-a741b76be6ce/'
```

«Фактический результат» (FR) из Allure

```
PATTERN_UUID4 = r'[0-9a-f]{8}-[0-9a-f]{4}-[0-5][0-9a-f]{3}-[089ab][0-9a-f]{3}-[0-9a-f]{12}'

http_after_masked_uuid = re.sub(
        DataParser.PATTERN_UUID4,
        '{id}',
        http_path
)
```

«Фактический результат» (FR) из Allure


```
http_after_masked_uuid = '/api-web/group-tasks/{id}/'
```

«Фактический результат» (FR) из Allure

```
http_path = '/api-web/stores/B15444/dashboards/'
```

«Фактический результат» (FR) из Allure

```
PATTERN_OBJ_ID = r'\/[^v]?\d{1,5}\D?\/'

http_after_masked_id = re.sub(
      DataParser.PATTERN_OBJ_ID,
      '{obj_id}',
      http_path
)
```

«Фактический результат» (FR) из Allure


```
http_after_masked_id = '/api-web/stores/{obj_id}/dashboards/'
```

«Фактический результат» (FR) из Allure

```
GET /api-web/stores/B15444/dashboards/c0ff8864-e29f-41b1-867d-
a741b76be6ce/?created_from=21-09-2023&created_to=22-09-2023
```

«Фактический результат» (FR) из Allure

```
GET /api-web/stores/B15444/dashboards/{id}/?created_from=21-09-2023&created_to=22-09-2023
```

«Фактический результат» (FR) из Allure

```
GET /api-web/stores/{obj_id}/dashboards/{id}/?created_from=21-09-2023&created_to=22-09-2023
```

«Фактический результат» (FR) из Allure

```
GET /api-web/stores/{obj_id}/dashboards/{id}/?created_from=21-09-2023&created_to=22-09-2023


http_method = 'GET'

http_path = '/api-web/stores/{obj_id}/dashboards/{id}/'

query_params = [
      'created_from=21-09-2023',
      'created_to=22-09-2023'
]
```

«Фактический результат» (FR) из Allure

005 = {DataFrame} <utils.data_coverage_parser.data_parser.DataFrame object

- api_method = {str} 'GET /api/config/'
- broken = {int} 1
- failed = {int} 19
- passed = {int} 63
- query_list = {list: 1} ['api_version=5']
- rank = {int} 82
- skipped = {int} 0

«Ожидаемый результат» (ER) из ELK

А что я вообще ожидал?

«Ожидаемый результат» (ER) из ELK

Активность пользователя можем собирать с помощью средств анализа веб трафика

«Ожидаемый результат» (ER) из ELK

Воспроизвести наиболее частые пользовательские сценарии, понять какие методы АПИ при этом используются и обратить наше внимание на них

«Ожидаемый результат» (ER) из ELK

Активность вызовов клиентом API методов может собираться в логах

«Ожидаемый результат» (ER) из ELK


```
POST https://{es_source}/internal/search/es
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/search-search.html

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [...],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [...],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [...],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [{
                            "match_phrase": {"event": "REQUEST_SUCCESS"}
                        }],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [...],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    ...,
                    "filter": [{
                            "range": {
                                "@timestamp": {
                                    "gte": "2023-08-23T21:31:57.000000Z",
                                    "lte": "2023-08-30T21:31:57.000000Z",
                                    "format": "strict_date_optional_time"
                                }
                            }
                        }],
                    ...                    }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [...],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [...],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    ...,
                    "must_not": [
                        {
                            "match_phrase": {"path_template": "health/"}
                        },
                        {
                            "match_phrase": {"path_template": "/metrics"}
                        }
                    ]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
{
    "params": {
        "index": "{log_index}",
        "body": {
            "query": {
                "bool": {
                    "must": [...],
                    "filter": [...],
                    "should": [...],
                    "must_not": [...]
                }
            },
            "size": 10000
        }
    }
}
```

«Ожидаемый результат» (ER) из ELK

```
response.json()["rawResponse"]["hits"]["hits"]
```

«Ожидаемый результат» (ER) из ELK


```
{
    ...,
    "rawResponse": {...},
        "hits": {
            ...,
            "hits": [{
                ...,
                "_source": {
                    ...,
                    "path": "/biosmart/",
                    "kubernetes": {...},
                    "full_path": "/biosmart/",
                    "status_code": 200,
                    "event": "REQUEST_SUCCESS",
                    "level": "info",
                    "path_template": "biosmart/",
                    "method": "POST",
                    "time": "2023-10-01T21:41:44.212579439Z",
                }
            }]
        }
    },
    ...
}
```

«Ожидаемый результат» (ER) из ELK


```
{
    ...,
    "rawResponse": {...},
        "hits": {
            ...,
            "hits": [{
                ...,
                "_source": {
                    ...,
                    "path": "/biosmart/",
                    "kubernetes": {...},
                    "full_path": "/biosmart/",
                    "status_code": 200,
                    "event": "REQUEST_SUCCESS",
                    "level": "info",
                    "path_template": "biosmart/",
                    "method": "POST",
                    "time": "2023-10-01T21:41:44.212579439Z",
                }
            }]
        }
    },
    ...
}
```

«Ожидаемый результат» (ER) из ELK


```
{
    ...,
    "rawResponse": {...},
        "hits": {
            ...,
            "hits": [{
                ...,
                "_source": {
                    ...,
                    "path": "/biosmart/",
                    "kubernetes": {...},
                    "full_path": "/biosmart/",
                    "status_code": 200,
                    "event": "REQUEST_SUCCESS",
                    "level": "info",
                    "path_template": "biosmart/",
                    "method": "POST",
                    "time": "2023-10-01T21:41:44.212579439Z",
                }
            }]
        }
    },
    ...
}
```

«Ожидаемый результат» (ER) из ELK


```
{
    ...,
    "rawResponse": {...},
        "hits": {
            ...,
            "hits": [{
                ...,
                "_source": {
                    ...,
                    "path": "/biosmart/",
                    "kubernetes": {...},
                    "full_path": "/biosmart/",
                    "status_code": 200,
                    "event": "REQUEST_SUCCESS",
                    "level": "info",
                    "path_template": "biosmart/",
                    "method": "POST",
                    "time": "2023-10-01T21:41:44.212579439Z",
                }
            }]
        }
    },
    ...
}
```

«Ожидаемый результат» (ER) из ELK

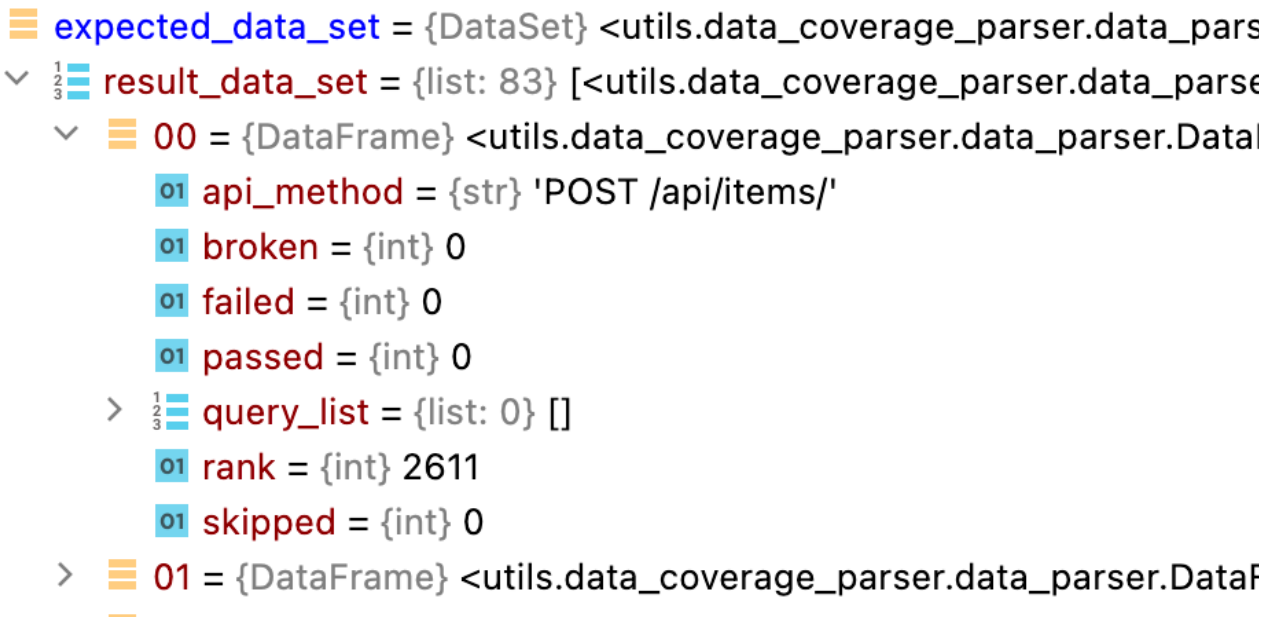

```
{
    ...,
    "rawResponse": {...},
        "hits": {
            ...,
            "hits": [{
                ...,
                "_source": {
                    ...,
                    "path": "/biosmart/",
                    "kubernetes": {...},
                    "full_path": "/biosmart/",
                    "status_code": 200,
                    "event": "REQUEST_SUCCESS",
                    "level": "info",
                    "path_template": "biosmart/",
                    "method": "POST",
                    "time": "2023-10-01T21:41:44.212579439Z",
                }
            }]
        }
    },
    ...
}
```

«Ожидаемый результат» (ER) из ELK

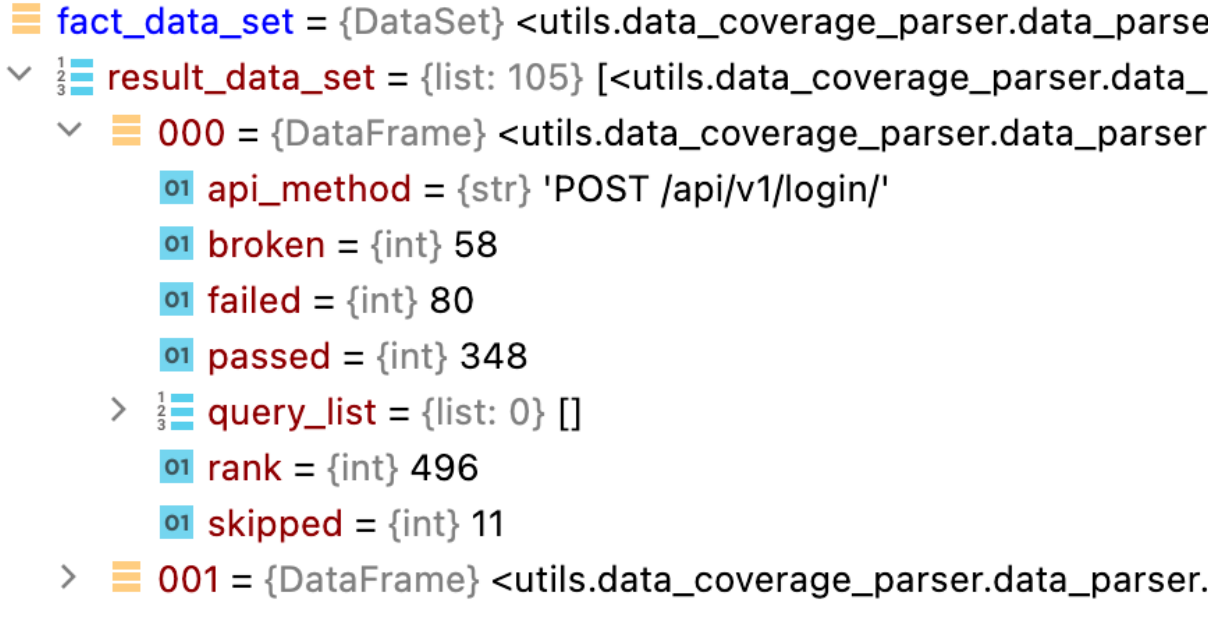
27 = {DataFrame} <utils.data_coverage_parser.data_parser.DataFrame object
- api_method = {str} 'PATCH /api/v2/basket/items/'
- broken = {int} 0
- failed = {int} 0
- passed = {int} 0
- query_list = {list: 1} ['type=write_off']
- rank = {int} 20
- skipped = {int} 0

«Ожидаемый результат» (ER) из ELK

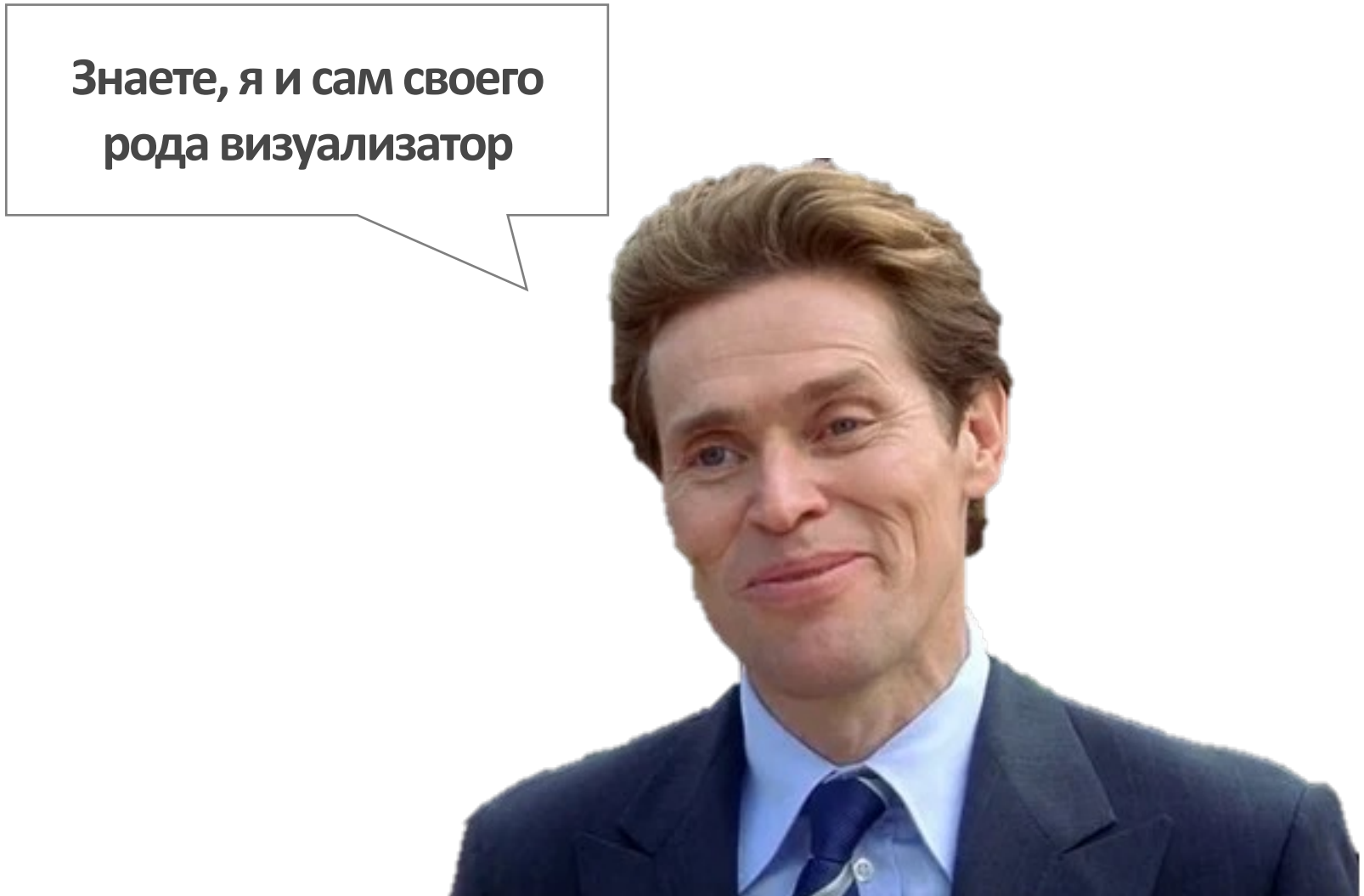
Expected



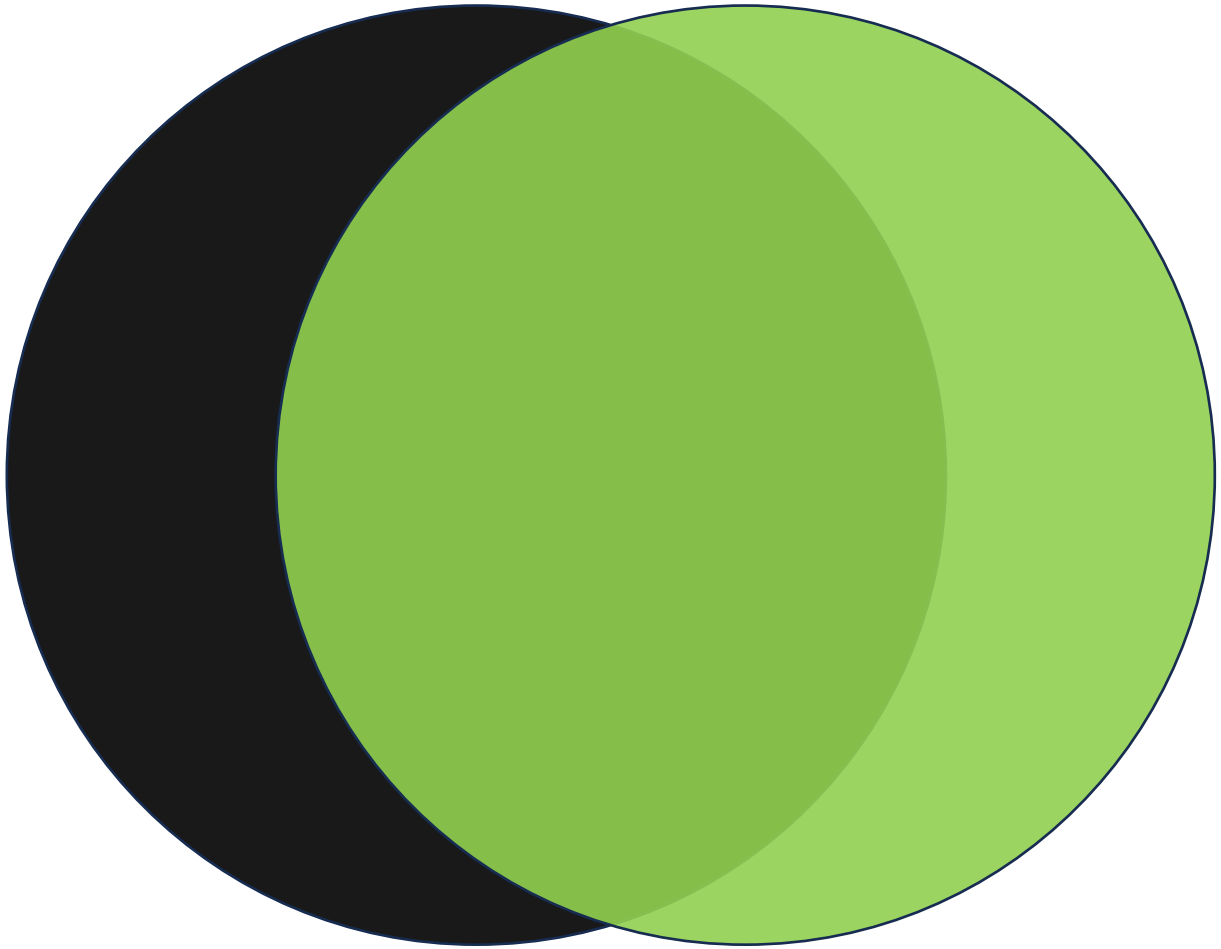
Fact



Анализ ER vs FR и его визуализация



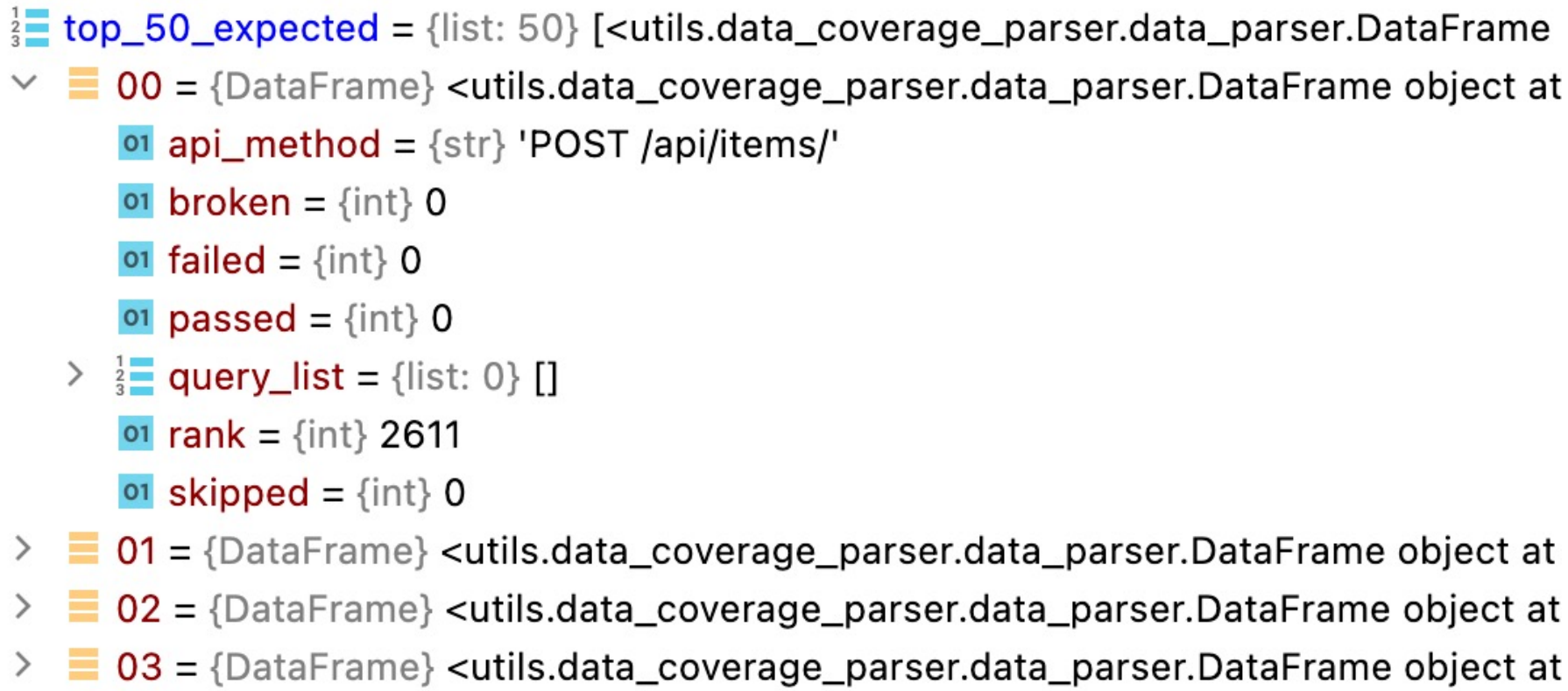
Анализ ER vs FR и его визуализация



Анализ ER vs FR и его визуализация


```
# Непротестированные используемые API методы
untested_used_methods = coverage_visualizer.get_untested_used_api_methods()


# Протестированные API используемые методы
tested_used_methods = coverage_visualizer.get_tested_api_methods(top_50_fact)


# Протестированные API неиспользуемые методы
tested_unused_from_top_50 = coverage_visualizer.get_tested_unused_api_methods()
```

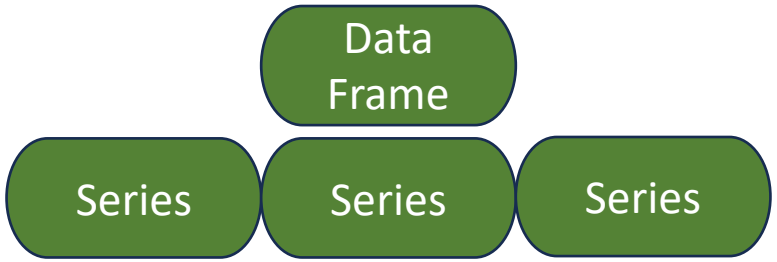
Анализ ER vs FR и его визуализация



Анализ ER vs FR и его визуализация

Pandas - Python библиотека для анализа данных

Одномерный массив с набором ассоциированных меток (индексов), вдоль каждого элемента из списка.



DataFrame является табличной структурой данных, столбцы, строки - объекты Series

Анализ ER vs FR и его визуализация

- ТОП 50 используемых API методов
- ТОП 50 протестированных API методов
- Непротестированные используемые API методы
- Статистика ТОП 15 протестированных API методов
- Общая диаграмма покрытия

Анализ ER vs FR и его визуализация

Непротестированные используемые API методы

```
# untested API Methods from top

untested_from_top_50_df = PData(
    data=untested_used_methods,
    columns=["api_method", "rank"],
    sort_by="rank"
).df
```

Анализ ER vs FR и его визуализация

ТОП 50 используемых API методов

```
# top 50 expected API Methods

top_50_expected_df = PData(
    data=top_50_expected,
    columns=["api_method", "rank"],
    sort_by="rank"
).df
```

Анализ ER vs FR и его визуализация

Статистика ТОП 15 протестированных API методов

```
# stat for top tested API methods

stats_for_tested_from_top_df = PData(
    data=fact_stat_tested_15_from_top,
    columns=[
        "api_method",
        "skipped",
        "broken",
        "passed",
        "failed",
        "rank"
    ],
    sort_by="rank"
).df
```

Анализ ER vs FR и его визуализация

ТОП 50 протестированных API методов

```
# top 50 fact API Methods

top_50_fact_df = PData(
    data=top_50_fact,
    columns=[
        "api_method",
        "skipped",
        "broken",
        "passed",
        "failed",
        "rank"
    ],
    sort_by="rank"
).df
```

Анализ ER vs FR и его визуализация

- [x] Таблица
- [x] Круговая диаграмма
- [x] Столбчатая гистограмма

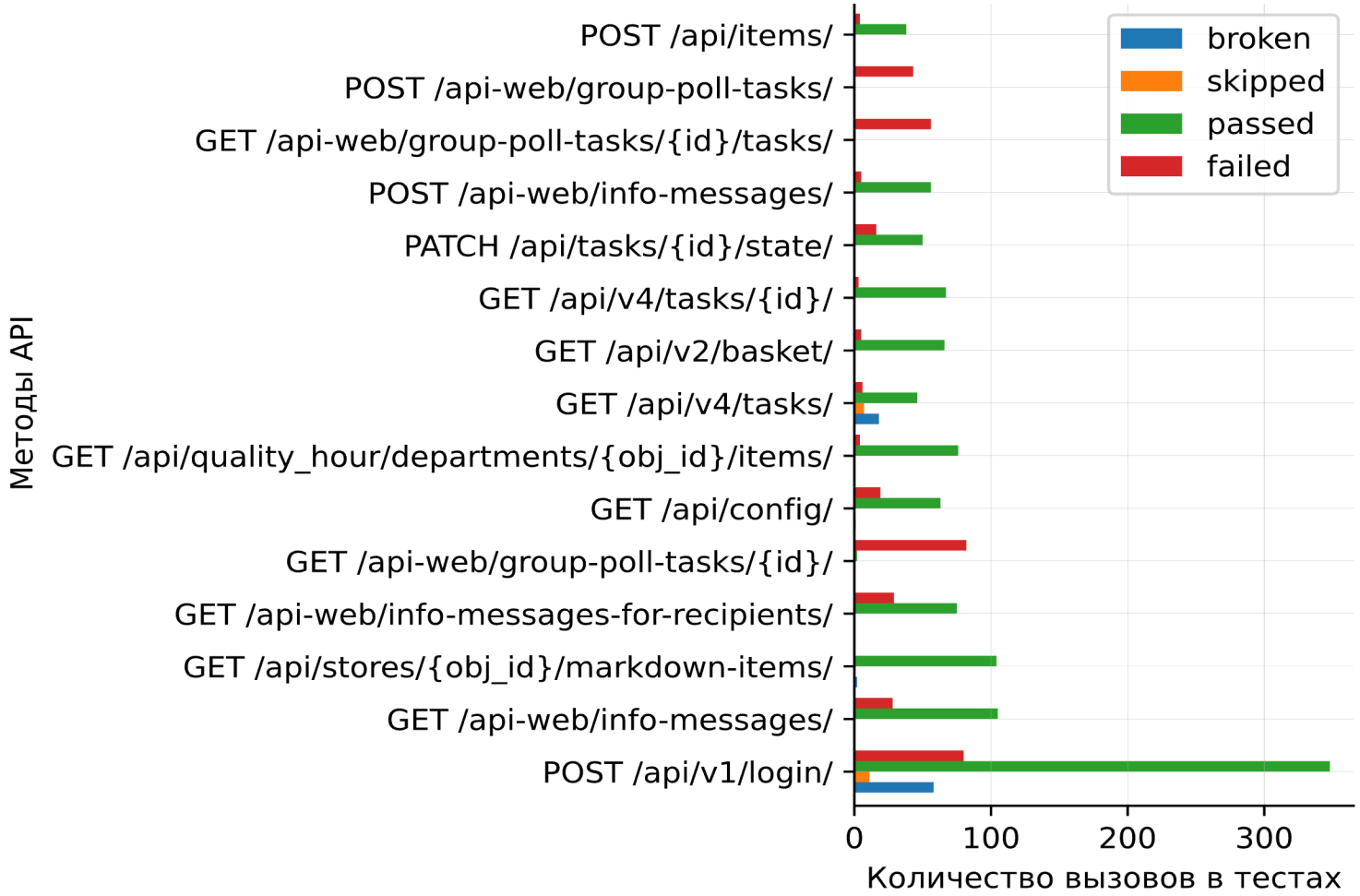
Анализ ER vs FR и его визуализация


```
class CoverageVisualizer:
    def __init__(self, fact_data_set: DataSet, expected_data_set: DataSet):
        self.fact_data_set = fact_data_set.result_data_set
        self.expected_data_set = expected_data_set.result_data_set

    def build_info_table(**kwargs):
        pass

    def build_info_gist_on_code_coverage(**kwargs):
        pass

    def build_info_barh_on_code_coverage(width=0.8, **kwargs):
        pass
```

Структура результатов тестов самых популярных API методов

```
barh_stats_for_tested_from_top = CoverageVisualizer.build_info_barh(
    df=stats_for_tested_from_top_df,
    title="Структура результатов тестов самых популярных API методов",
    xlabel="Количество вызовов в тестах",
    ylabel="Методы API",
    x="api_method",
    y=["broken", "skipped", "passed", "failed"]
)
```

Структура результатов тестов самых популярных API методов



Топ 50 наиболее популярных API методов

```
CoverageVisualizer.build_info_table(
    df=top_50_expected_df,
    title="Топ 50 наиболее популярных API методов"
)
```

Топ 50 наиболее популярных API методов

API method	Rank
POST api/items/	2750
GET api/work_schedules/	1283
GET api/v5/main/	1281
GET api/photos/	813
POST api/stores/{obj_id}/markdown-items/	507
POST api/items_printed/	335
POST api/employees/	327
GET api/config/	304
POST api/devices/	226
GET api/stores/{obj_id}/markdown-items/	133
GET api-web/log-doc-view/	106
GET api-web/regions/	105
GET api-web/stores/	98
GET api-web/counters/	89
POST api/comments/	83

Топ 50 покрытых тестами API методов

```
CoverageVisualizer.build_info_table(
    df=top_50_fact_df,
    title="Топ 50 покрытых тестами API методов"
)
```

Топ 50 покрытых тестами API методов

API method	Passed	Failed	Skipped	Broken
POST api/v1/login/	348	80	11	58
GET api-web/info-messages/	105	28	0	0
GET api/stores/{obj_id}/markdown-items/	104	0	0	2
GET api-web/info-messages-for-recipients/	75	25	0	0
GET api-web/tasks/group-poll-tasks/{id}	82	0	0	0
GET api/config/	61	3	0	1
GET api/v4/tasks/	46	18	5	3
GET api/v2/basket/	66	5	0	0
POST api-web/info-messages/	56	5	0	0
GET api-web/statistics/	49	0	1	3
DELETE api/comment/{id}/	47	3	0	0
GET api-web/stores/	42	0	2	0
POST api-web/tasks/	36	3	0	0
GET api/v1/planned-tasks/	35	0	2	0
GET api-web/stores/{obj_id}/dashboards/	30	1	5	0

Перечень не покрытых тестами API методов

```
CoverageVisualizer.build_info_table(
    df=untested_from_top_50_df,
    title="Перечень не покрытых тестами API методов"
)
```

Перечень не покрытых тестами API методов

API method	Rank
POST api/photos/	813
POST api/devices/	226
GET api-web/log-doc-view/	106
GET api-web/counters/	89
POST api/comments/	83
POST api-web/info-messages/catalogs/	75
PATCH api/config/	74
GET api/basket-placing/	69
GET api/segments/	62
POST api/segments/	59
GET api/segments/{obj_id}/	54
GET api/department/	45
GET api/regions/	39
POST api/v2/items/	25
GET api/v2/items/	15

Диаграмма покрытия API методов авто тестами

```
all_untested = coverage_visualizer.get_untested_used_api_methods()
all_tested = coverage_visualizer.get_tested_api_methods()

gist = CoverageVisualizer.build_info_gist_on_code_coverage(
    values=[len(all_tested), len(all_untested)],
    labels=["Покрыто", "Не Покрыто"],
    title="Диаграмма покрытия API методов авто тестами"
)
```

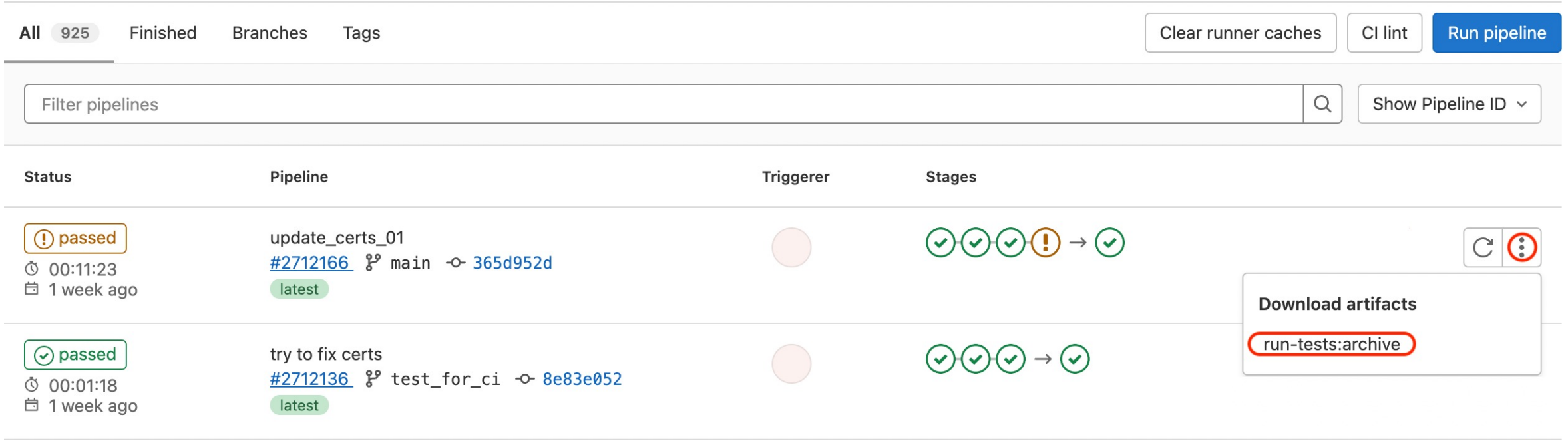
Диаграмма покрытия API методов авто тестами

Анализ ER vs FR и его визуализация

.gitlab-ci.yml

```yaml
stages:
  - linting
  - build
  - run-tests
  - coverage

coverage_vis:
   stage: coverage
   script:
     - ls
     - mkdir pandas_results
     - python coverage_visualizer.py
   artifacts:
     paths:
       - ./pandas_results/*
   rules:
     - if: $CI_PIPELINE_SOURCE == "merge_request_event" ||
$CI_PIPELINE_SOURCE == "push"
       when: manual
     - when: always
```

Анализ ER vs FR и его визуализация

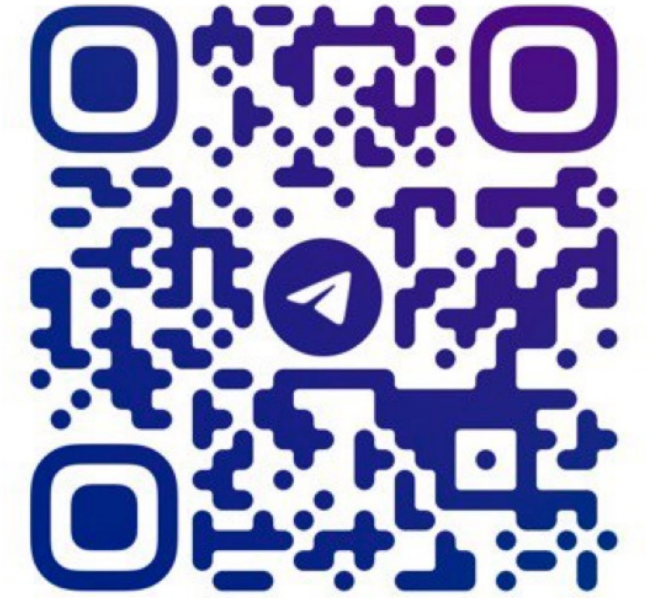


Итоги

Пришло время подводить итоги ☺

Итоги

- Получить информацию о пользовательской активности использования API
- Быстро получить информацию о соотношении покрытых / не покрытых авто тестами API методах
- Сконцентрироваться на покрытии реально используемых API методах
- Сократить количество не покрытых тестами API методов



@KIRILL_KHRABROV

X5 Tech