

Разработка и управление едиными контрактами API



О себе

Немировский Лев

Руководитель направления разработки
образовательных платформ



«Код самодокументируемый. Если что-то и непонятно, то это проблемы того, кто читает»

Анонимный разработчик

«Зачем документация, если есть тесты? Они и так всё проверяют»

Опытный тестировщик

«Документация устаревает быстрее, чем мы её пишем. Зачем тратить время?»

Известный системный аналитик



Контракт API

представляет собой формальное описание API, которое может включать спецификацию поведения, форматы запросов/ответов и другие технические детали, необходимые для интеграции и взаимодействия с API



Документация API

включает в себя контракт API, но также может содержать руководства, примеры использования, часто задаваемые вопросы и другую информацию, полезную для разработчиков и конечных пользователей



Спецификация API

технический документ, описывающий контракт API в структурированной форме, часто используя стандарты, такие как OpenAPI



Классическая работа над API



Системные аналитики проектируют и создают описание API в WIKI, либо оформляют документацию по факту разработки

Backend и Frontend разработчики создают и имплементируют, описывают, либо модифицируют документацию

Тестировщики используют документацию для тестирования и так же могут модифицировать, выявляя проблемы – с требованием, просто работой, возможно с безопасностью если у ваших тестировщиков есть такие скилы





Проблемы классического процесса



1

Несоответствие между ожидаемым и реализованным функционалом

2

Затруднения в поддержке и масштабировании API из-за отсутствия четкой документации

3

Увеличение времени на разработку и тестирование из-за необходимости постоянного уточнения деталей API

4

Различия в понимании API между командами. Отсутствие формализованных контрактов может привести к различиям в трактовке функциональности API между разработчиками, тестировщиками и системными аналитиками

5

Сложности в обеспечении обратной совместимости

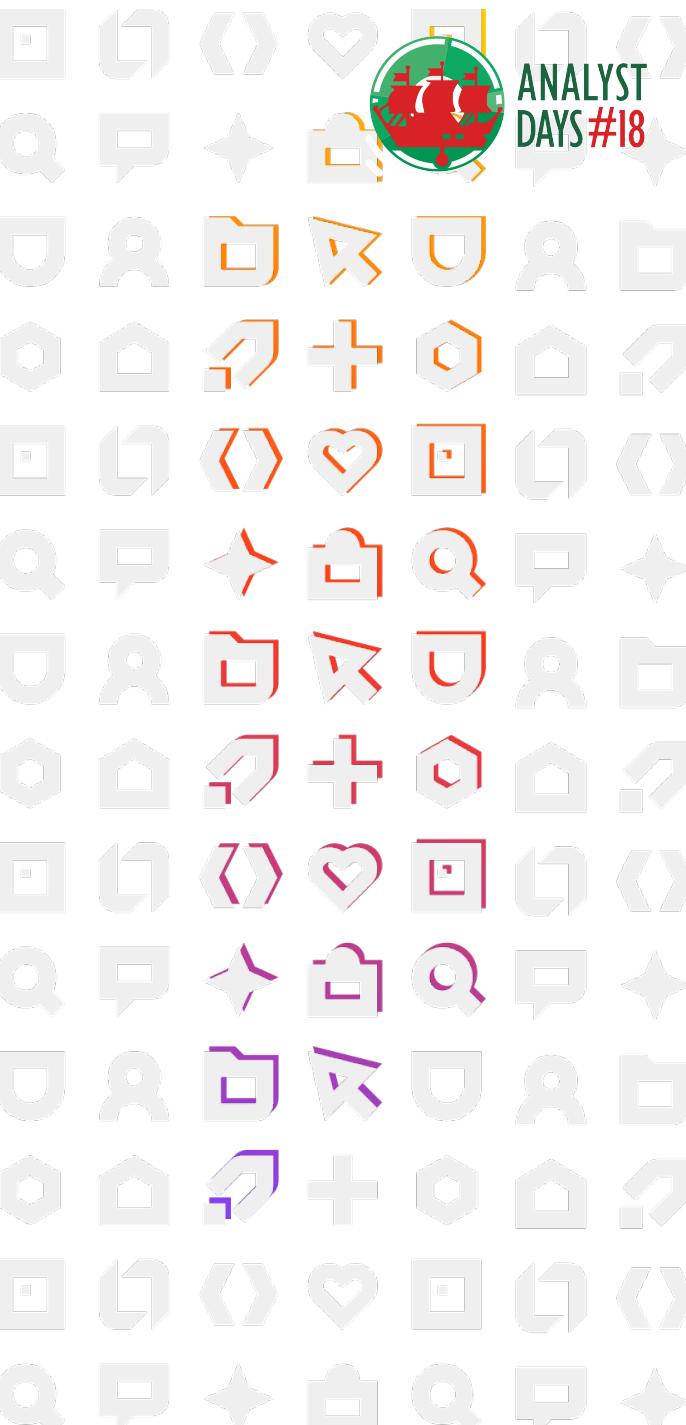
6

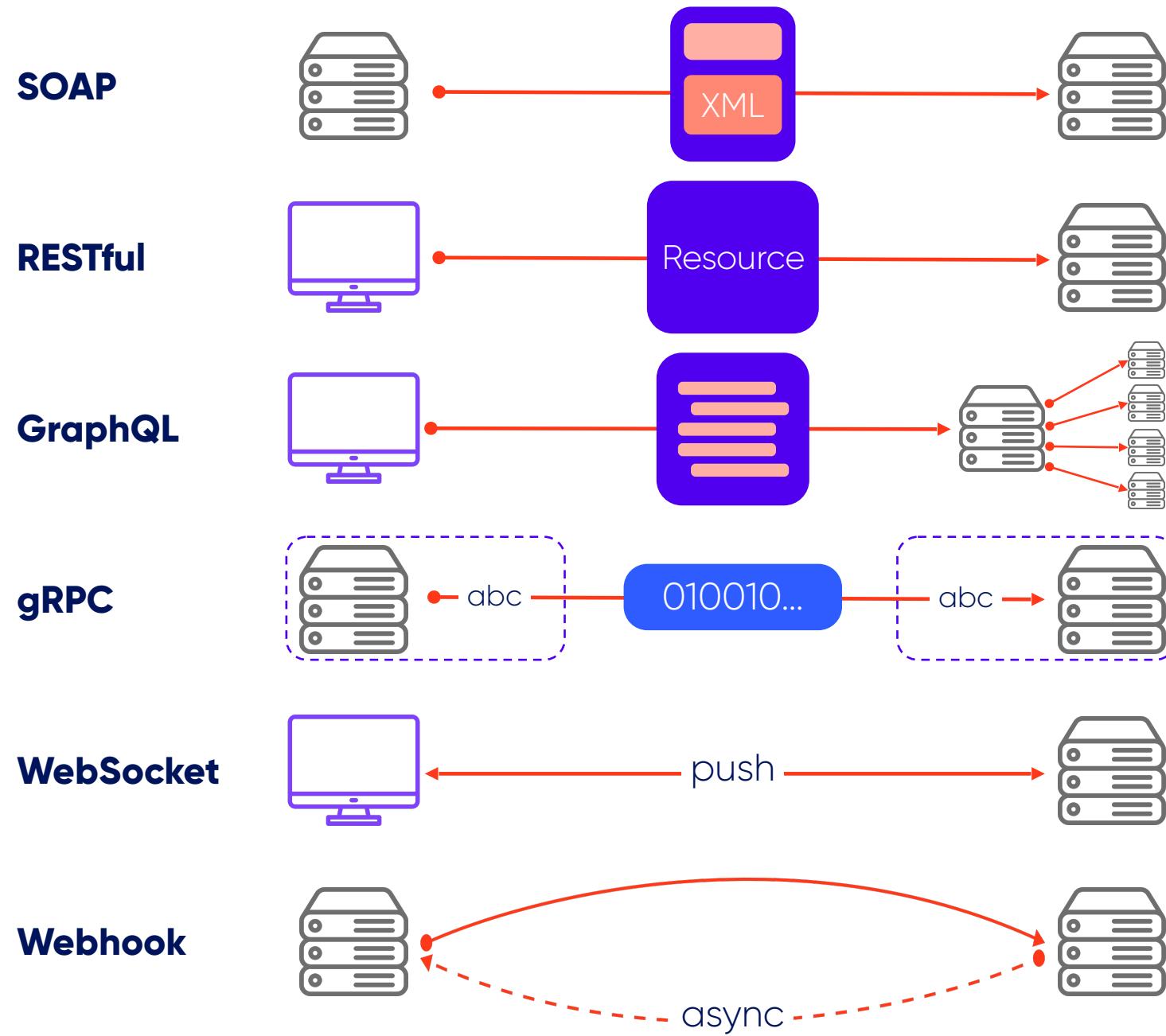
Большое количество дорого времени на коммуникации между участниками, доработки и исправление



Контракты обеспечивают

- Асинхронность разработки
- Четкое понимание требований к API на всех этапах разработки
- Упрощение процесса интеграции и тестирования за счет предсказуемости поведения API
- Возможность автоматизации разработки и тестирования за счет кодогенерации
- Облегчение процесса рефакторинга и обновления
- И многое другое...







SOAP (Simple Object Access Protocol)



SOAP использует XML-сообщения для обмена информацией между системами, независимо от протоколов передачи данных. SOAP обеспечивает высокий уровень безопасности и поддержку транзакций благодаря четко определенному контракту через WSDL (Web Services Description Language)

Преимущества:

- Четкий и строгий контракт с помощью WSDL
- Поддержка сложных операций и транзакций, включая безопасность и надежность обмена сообщениями
- Эффективная обработка ошибок через механизмы SOAP Faults

Недостатки:

- Высокая сложность и многословность XML-сообщений увеличивают объем передаваемых данных
- Требуется больше ресурсов для обработки и анализа сообщений
- Меньшая гибкость и трудности в интеграции с новыми веб-стандартами и технологиями

Использование:

- Идеально подходит для корпоративных и финансовых приложений, где важны безопасность и надежность
- В системах, требующих строгого соответствия контрактам и поддержки сложных бизнес-операций

Контракт с помощью WSDL

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"  
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"  
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
                  targetNamespace="http://example.com/petstore.wsdl">  
  
    <wsdl:types>  
        <!-- Определения типов -->  
    </wsdl:types>  
    <wsdl:message name="GetPetRequest">  
        <!-- Сообщение запроса -->  
    </wsdl:message>  
    <wsdl:message name="GetPetResponse">  
        <!-- Сообщение ответа -->  
    </wsdl:message>  
    <wsdl:portType name="PetStorePortType">  
        <!-- Операции сервиса -->  
    </wsdl:portType>  
    <wsdl:binding name="PetStoreBinding" type="tns:PetStorePortType">  
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>  
        <!-- Привязка операций -->  
    </wsdl:binding>  
    <wsdl:service name="PetStoreService">  
        <wsdl:port name="PetStorePort" binding="tns:PetStoreBinding">  
            <soap:address location="http://example.com/petstore"/>  
        </wsdl:port>  
    </wsdl:service>  
</wsdl:definitions>
```

GraphQL представляет собой язык запросов для API и среду выполнения запросов к вашим данным. GraphQL позволяет клиентам запрашивать именно те данные, которые им нужны, не более того. Это уменьшает объем передаваемых данных по сети и увеличивает эффективность и скорость приложений

Преимущества:

- Позволяет клиентам точно указывать, какие данные им нужны
- Уменьшает количество запросов и объем передаваемых данных
- Обеспечивает сильную типизацию данных, улучшая надежность и предсказуемость API

Недостатки:

- Сложность и необходимость изучения нового синтаксиса
- Может привести к сложным запросам, увеличивающим нагрузку на сервер
- Отсутствие встроенного механизма кэширования, что может снизить производительность

Использование:

- Подходит для приложений с сложной и взаимосвязанной структурой данных
- Идеально для ситуаций, когда клиентам нужен гибкий доступ к различным данным
- Стандартно – мобильные приложения и фронтенд

Контракт с помощью GraphQL SDL

```
● ● ●

type Query {
    pet(id: ID!): Pet
}

type Pet {
    id: ID!
    name: String!
    type: String!
}

type Mutation {
    addPet(name: String!, type: String!): Pet
}
```

gRPC – это высокопроизводительный фреймворк разработанный компанией Google для вызов удаленных процедур (RPC), работает поверх HTTP/2 для транспорта, Protocol Buffers в качестве языка описания интерфейса, и предлагает функции, такие как аутентификация, нагрузочное распределение, двусторонние потоки и блокировки

Преимущества:

- Высокая производительность и низкая задержка благодаря использованию HTTP/2 и сериализации Protocol Buffers
- Поддержка множества языков программирования
- Встроенные механизмы аутентификации, безопасности, и нагрузочного распределения

Недостатки:

- Более высокий порог входа из-за необходимости работы с Protocol Buffers
- Ограниченнная поддержка браузерами, требует использования gRPC-Web для клиентских приложений
- Меньшая гибкость в сравнении с REST из-за строгой типизации и схемы

Использование:

- Идеально подходит для микросервисной архитектуры и взаимодействия между сервисами внутри системы
- Рекомендуется для приложений, требующих высокой производительности и низкой задержки

Контракт с помощью Protobuf

```
syntax = "proto3";

package petstore;

// Определение сообщений
message PetRequest {
    string id = 1;
}

message Pet {
    string id = 1;
    string name = 2;
    string type = 3;
}

// Определение сервиса
service PetStore {
    rpc GetPetById(PetRequest) returns (Pet) {}
}
```

WebSocket обеспечивает полнодуплексное коммуникационное соединение через одно TCP-соединение. Это позволяет веб-приложениям обмениваться данными с сервером в реальном времени без необходимости постоянно опрашивать сервер

Преимущества:

- Позволяет осуществлять двустороннюю связь между клиентом и сервером в реальном времени
- Уменьшает задержки и повышает производительность по сравнению с традиционным HTTP-поллингом
- Поддерживается большинством современных браузеров

Недостатки:

- Сложность в реализации и поддержке соединения стабильным
- Может быть неэффективным при использовании в сценариях с низкой частотой сообщений из-за постоянного открытого соединения
- Необходимость учитывать меры безопасности, такие как шифрование данных

Использование:

- Идеально подходит для приложений, требующих обмена данными в реальном времени, например, чаты, игры или торговые платформы

Контракт с помощью AsyncAPI

```
● ● ●

asyncapi: '3.0.0'
info:
  title: PetStore WebSocket API
  version: '1.0.0'
  description: |
    This is a sample PetStore server for managing pet
    information through WebSocket.
servers:
  production:
    url: wss://petstore.example.com/ws
    protocol: wss

channels:
  pet/added:
    description: Channel for receiving notifications when a new pet is added
    subscribe:
      summary: Subscribe to pet added events
      operationId: onPetAdded
      message:
        contentType: application/json
        payload:
          type: object
          properties:
            id:
              type: integer
              example: 1
            name:
              type: string
              example: Rex
            species:
              type: string
              example: dog
```

WebHook представляет собой HTTP-коллбэк: простой HTTP POST-запрос, который отправляется на заданный URL в ответ на какое-либо событие. Это позволяет веб-приложениям получать и обрабатывать уведомления в реальном времени о событиях, происходящих в других сервисах

Преимущества:

- Асинхронное получение уведомлений о событиях без необходимости опроса сервера
- Простота реализации и использования
- Возможность мгновенной реакции на события во внешних системах

Недостатки:

- Трудности с обеспечением безопасности данных, передаваемых через WebHook
- Отсутствие стандартизации в формате данных и способах их обработки
- Зависимость от доступности и надежности внешней системы, отправляющей уведомления

Использование:

- Подходит для интеграции различных веб-сервисов и автоматизации реакции на события, например, обновления в базе данных, новые сообщения в чате или изменения в системе управления задачами

Контракт с помощью AsyncAPI

```
● ● ●

asyncapi: '3.0.0'
info:
  title: PetStore Webhook API
  version: '1.0.0'
  description: |
    This is a sample PetStore server for
    managing pet information through Webhook.
servers:
  production:
    url: https://petstore.example.com/webhook
    protocol: https

channels:
  /webhooks/petAdded:
    description: Webhook for receiving notifications when a new pet is added
    subscribe:
      summary: Subscribe to pet added events
      operationId: onPetAdded
      message:
        contentType: application/json
        payload:
          type: object
          properties:
            id:
              type: integer
              example: 1
            name:
              type: string
              example: Rex
            species:
              type: string
              example: dog
```



REST (Representational State Transfer)



REST API – это архитектурный подход, который устанавливает ограничения для API. Использует стандартные HTTP-методы и ориентирован на работу с ресурсами. Он гибок, легок в реализации и поддерживает масштабируемость

RAML

API Blueprint

OpenAPI



REST (Representational State

REST API – это архитектурный подход, который устанавливает стандартные HTTP-методы и ориентирован на масштабируемость. Он гибок, легок в реализации и поддерживает масштабируемость.

RAML

API Blueprint

OpenAPI

```
#%RAML 1.0
title: PetStore API
version: v1
baseUri: http://api.example.com
/pets:
  get:
    responses:
      200:
        body:
          application/json:
            type: array
            items:
              type: object
              properties:
                id:
                  type: integer
                name:
                  type: string
                tag:
                  type: string
```





REST (Representational State

REST API – это архитектурный подход, который устанавливает стандарты для взаимодействия с API. Использует стандартные HTTP-методы и ориентирован на представление состояния. Он гибок, легок в реализации и поддерживает масштабируемость.

RAML

API Blueprint

OpenAPI



FORMAT: 1A

HOST: http://api.example.com/

PetStore API

Group Pets

Pets Collection [/pets]

List all pets [GET]

+ Response 200 (application/json)

+ Attributes (array[Pet])

Data Structures

Pet (object)

+ id: 1 (number)

+ name: "Doggie" (string)

+ type: "Dog" (string)

REST (Representational State

REST API – это архитектурный подход, который устано
Использует стандартные HTTP-методы и ориентирован
Он гибок, легок в реализации и поддерживает масштаб

RAML

API Blueprint

OpenAPI

```
openapi: 3.0.0
info:
  title: PetStore API
  version: "1.0"
paths:
  /pets:
    get:
      summary: Возвращает список всех питомцев
      responses:
        '200':
          description: Список питомцев
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Pet'
components:
  schemas:
    Pet:
      type: object
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
        tag:
          type: string
```



← Test ▾ 1.0.0 ▾

```
1  openapi: 3.1.0
2  info:
3    title: Swagger Petstore - OpenAPI 3.1
4    description: |-
5      This is a sample Pet Store Server based on the OpenAPI 3.1 specification. You can
6      find out more about
7      Swagger at [https://swagger.io](https://swagger.io). In the third iteration of the
8      pet store, we've switched to the design first approach!
9
10     You can now help us improve the API whether it's by making changes to the definition
11     itself or to the code.
12
13     That way, with time, we can improve the API in general, and expose some of the new
14     features in OAS3.
15
16     Some useful links:
17       - [The Pet Store repository](https://github.com/swagger-api/swagger-petstore)
18       - [The source API definition for the Pet Store](https://github.com/swagger-api/
19         swagger-petstore/blob/master/src/main/resources/openapi.yaml)
20
21     termsOfService: http://swagger.io/terms/
22     contact:
23       email: apiteam@swagger.io
24     license:
25       name: Apache 2.0
26       url: http://www.apache.org/licenses/LICENSE-2.0.html
27     version: 1.0.0
28     externalDocs:
29       description: Find out more about Swagger
30       url: http://swagger.io
31     servers:
32       - url: https://petstore3.swagger.io/api/v3
33     tags:
34       - name: pet
35       description: Everything about your Pets
36     externalDocs:
37       description: Find out more
38       url: http://swagger.io
39     - name: store
40       description: Access to Petstore orders
41     externalDocs:
42       description: Find out more about our store
43       url: http://swagger.io
44     - name: user
45       description: Operations about user
46     paths:
47       /pet:
48         put:
49           tags:
50             - pet
51             summary: Update an existing pet
52             description: Update an existing pet by Id
53             operationId: updatePet
54             requestBody:
55               description: Update an existent pet in the store
56               content:
57                 application/json:
```

Last Saved: 9:29:40 pm - Feb 18, 2024

VALID ▾

SAVE ▾

GET /pet/{petId} Find pet by ID

Returns a single pet

Parameters

Name Description

petId * required ID of pet to return

integer(\$int64)

(path)

Responses

Code	Description	Links
200	successful operation	No links
400	Invalid ID supplied	No link

Media type

application/json

Controls Accept header.

Example Value | Schema

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 9007199254740991,
      "name": "string"
    }
  ],
  "status": "available"
}
```

5

ANALYST
DAYS #18

New API / Tets

Language Go - Native

Fork | 0 View Collection

GET Find pet by ID

Open request →

<https://petstore3.swagger.io/api/v3/pet/:petId>

Returns a single pet

Authorization API Key

Key api_key

Value {{apiKey}}

Path Variables

petId <long>
(Required) ID of pet to return

Example successful operation

Request

```
Go - Native
package main

import (
    "fmt"
    "net/http"
    "io/ioutil"
)

func main() {
    View More
}
```

Response

Body Headers (1) 200 OK

```
json
{
    "name": "<string>",
    "photoUrls": [
        "<string>",
        "<string>"
    ]
}
```

Online Find and replace Console

JUMP TO

Introduction

> pet

> store

> user



ANALYST
DAYS #18

JSON:API

```
{  
  "data": {  
    "type": "blog",  
    "id": "1",  
    "attributes": {  
      "title": "JSON:API быстрый старт",  
      "created": "2018-10-05T00:00:00+00:00"  
    },  
    "relationships": {  
      "author": {  
        "links": {  
          "self": "/blogs/1/author",  
          "related": "/peeps/1"  
        }  
      },  
      "comments": {  
        "links": {  
          "self": "/blogs/1/comments"  
        }  
      }  
    }  
  }  
}
```

// основной объект данных
// тип единичного объекта ресурса
// идентификатор единичного объекта ресурса
// атрибуты
// атрибут на основе строки
// отношения
// именованное отношение
// ссылка на отношение между блогом и автором
// независимый объект
// другое именованное отношение
// ссылка на отношение к зависимым объектам

**BlogPostResponse** ^ Collapse all **object****data** ^ Collapse all **object****type** > Expand all **string****id** > Expand all **string****attributes** ^ Collapse all **object****title** > Expand all **string****created** > Expand all **string** **date-time****relationships** ^ Collapse all **object****author** ^ Collapse all **object****links** ^ Collapse all **object****self** ^ Collapse all **string***Example* ["/blogs/1/author"]**related** > Expand all **string****comments** > Expand all **object**



Code	Description	Links
200	<p>successful operation</p> <p>Media type</p> <p>application/json ▾</p> <p>Controls <code>Accept</code> header.</p> <p>Example Value Schema</p> <div><p>Pet ^ Collapse all <code>object</code></p><p>id ^ Collapse all <code>integer</code> int64</p><p><i>Example</i> <code>10</code></p><p>name* ^ Collapse all <code>string</code></p><p><i>Example</i> <code>"doggie"</code></p><p>category ^ Collapse all <code>object</code></p><p>id > Expand all <code>integer</code> int64</p><p>name > Expand all <code>string</code></p><p>XML > Expand all <code>object</code></p><p>photoUrls* > Expand all <code>array<string></code></p><p>tags > Expand all <code>array<object></code></p><p>status > Expand all <code>string</code></p><p>XML > Expand all <code>object</code></p></div>	<i>No links</i>



Структура OpenAPI



Четкое разделение по микросервисам/сервисам

Каждый микросервис/сервис имеет свою директорию, что упрощает навигацию и изоляцию компонентов

Версионирование

Для каждой версии API создается отдельная поддиректория ('v1', 'v2' и т.д.), что позволяет параллельно поддерживать несколько версий API

Централизованное хранение моделей

Модели данных размещаются в поддиректории 'models' внутри каждой версии, что облегчает их переиспользование и обновление

Осмысленное именование файлов

Имена файлов должны четко отражать их содержимое, например, 'user.yaml' для модели пользователя

Использование общих определений для ответов и параметров

Вынос общих ответов и параметров в отдельные файлы ('responses.yaml', 'parameters.yaml') упрощает поддержку и обновление API

Выделение общих компонентов

Общие модели ошибок, параметры пагинации и другие переиспользуемые компоненты хранятся в общей директории ('shared' или 'common'), что предотвращает дублирование

Например:

```
api-specifications/
└── microservice-1/
    ├── v1/
    │   ├── openapi.yaml      # Основная спецификация OpenAPI для v1
    │   └── models/
    │       ├── user.yaml     # Модели данных для v1
    │       └── product.yaml
    └── v2/
        ├── openapi.yaml      # Основная спецификация OpenAPI для v2
        └── models/
            ├── user.yaml     # Модели данных для v2
            └── order.yaml
    └── microservice-2/
        ├── v1/
        │   ├── openapi.yaml      # Основная спецификация OpenAPI для v1
        │   └── models/
        │       ├── account.yaml
        │       └── transaction.yaml
        └── common/              # Общие модели для всех версий
            ├── errorModel.yaml
            └── pagination.yaml
    └── shared/                # Переиспользуемые компоненты и общие определения
        ├── responses.yaml
        └── parameters.yaml
```

Описание (description)

```
parameters:  
  - name: petId  
    in: path  
    description: ID of the pet to fetch  
    required: true  
    schema:  
      type: string
```

В параметрах

```
paths:  
  /pets/{petId}:  
    get:  
      summary: Get a pet by its ID  
      description: Returns a single pet
```

В paths

```
Pet:  
  type: object  
  description: A pet object  
  properties:  
    id:  
      type: string  
      description: Unique identifier of the Pet
```

В схеме

```
name:  
  type: string  
  description: Name of the Pet
```

В атрибуте

Разбиение файла



```
components:
```

```
schemas:
```

```
Pet:
```

```
$ref: 'definitions/pet.yaml'
```

Секци info



```
openapi: 3.0.0
```

```
info:
```

```
  title: PetStore API
```

```
  version: "1.0"
```

```
  description: |
```

Это API для магазина домашних животных.

Позволяет получать информацию о питомцах, их добавление и обновление.

```
servers:
```

```
  - url: https://example.com/api
```

```
    description: Основной сервер PetStore API
```

Переиспользуйте схемы



Pet:

 type: object

 required:

 - id

 - name

 properties:

 id:

 type: string

 name:

 type: string

Примеры (example)



Pet:

`type: object`

`example:`

`id: 1`

`name: Doggie`

`type: Dog`

Теги



```
paths:  
  /pets:  
    get:  
      tags:  
        - pet  
      summary: Возвращает всех питомцев  
      responses:  
        '200':  
          description: успешный ответ
```

Авторизация



```
components:  
  securitySchemes:  
    OAuth2:  
      type: oauth2  
      flows:  
        authorizationCode:  
          authorizationUrl: https://example.com/oauth/authorize  
          tokenUrl: https://example.com/oauth/token  
          scopes:  
            read: Grants read access  
            write: Grants write access
```

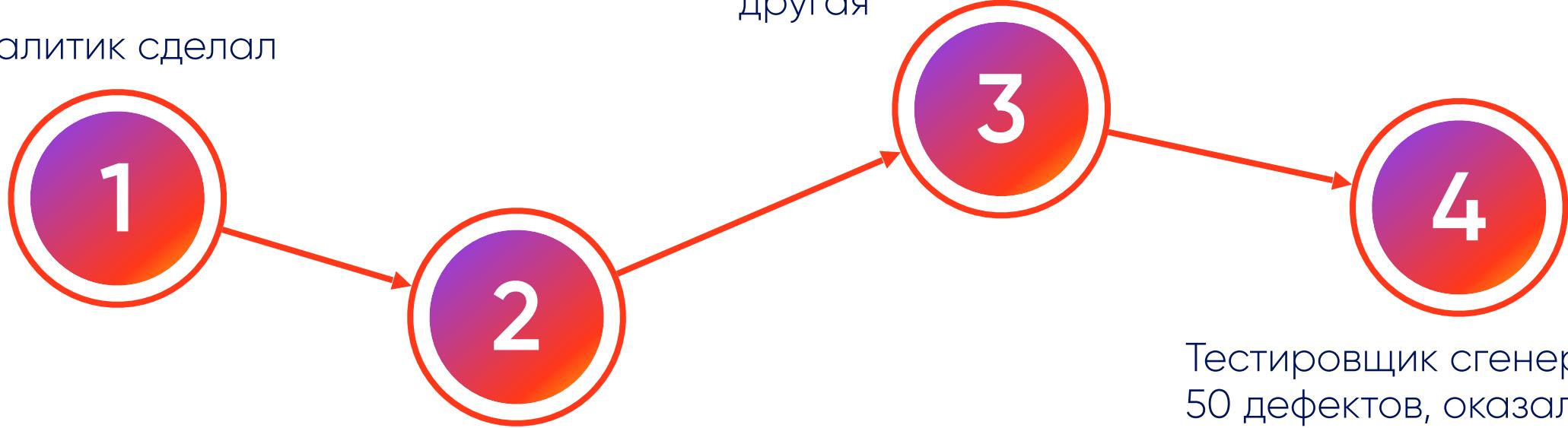
Указание scopes

```
● ● ●  
  
/pets/{petId}:  
  get:  
    summary: Получает информацию о питомце по ID  
    operationId: getPetById  
    security:  
      - OAuth2: [read]  
    parameters:  
      - name: petId  
        in: path  
        required: true  
        schema:  
          type: string  
    responses:  
      '200':  
        description: Информация о питомце успешно получена  
        content:  
          application/json:  
            schema:  
              $ref: '#/components/schemas/Pet'
```



Как не надо работать

Аналитик сделал



Разработчик решил,
что схема не очень

Тестировщик сгенерировал
50 дефектов, оказалось,
что схема другая, негодует,
что нет источника правды



Как надо работать



Аналитик создал
документацию
с контрактом

Лиды: тестирования,
backend, frontend
верифицировали

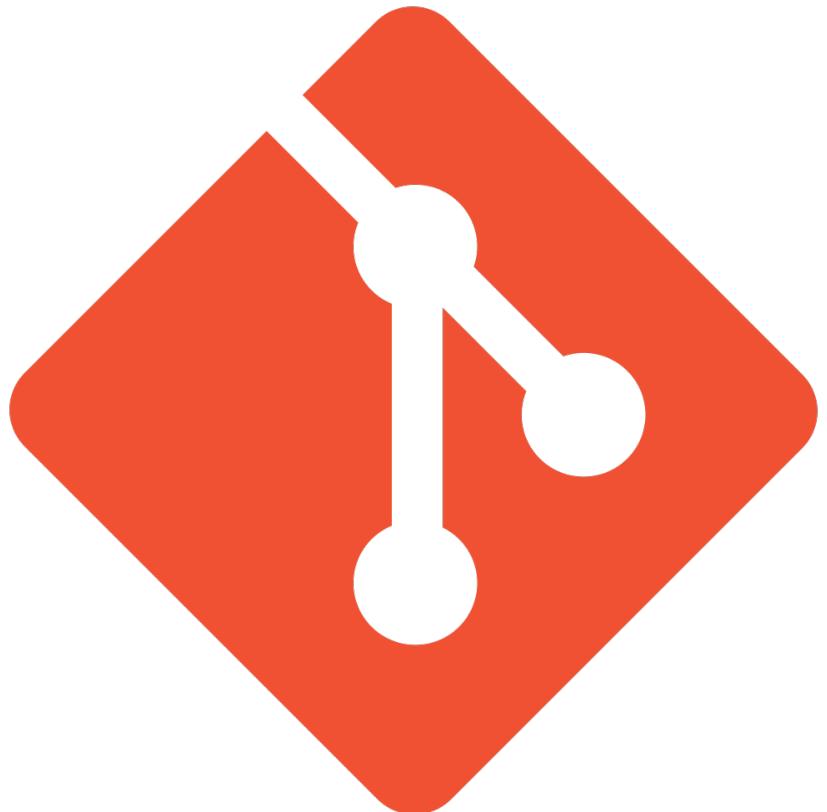
Backend и Frontend
параллельно имплементируют,
тестировщик пишет автотесты

**Если по ходу работы необходимо внести изменения в контракт – вноситься и
валидируется аналогично п.2**

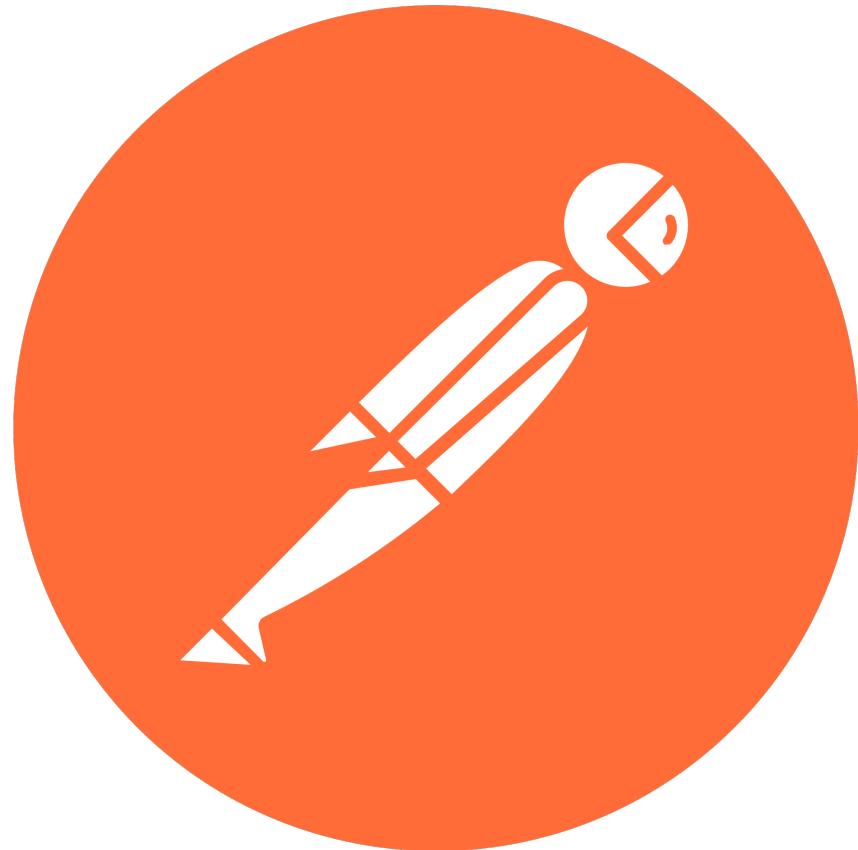
После реализации просто прогоняются автотесты



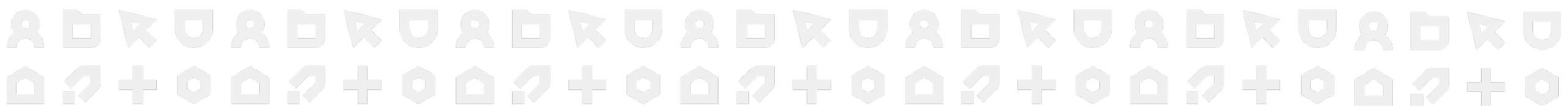
Основные инструменты



Git

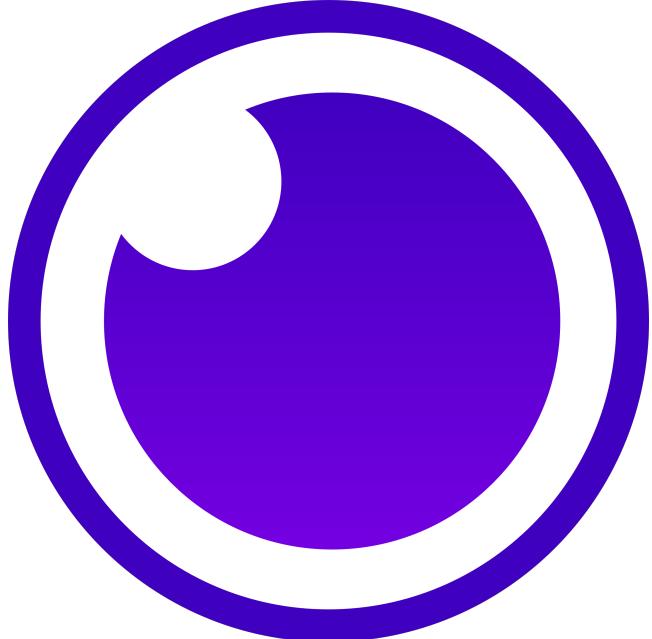


Postman

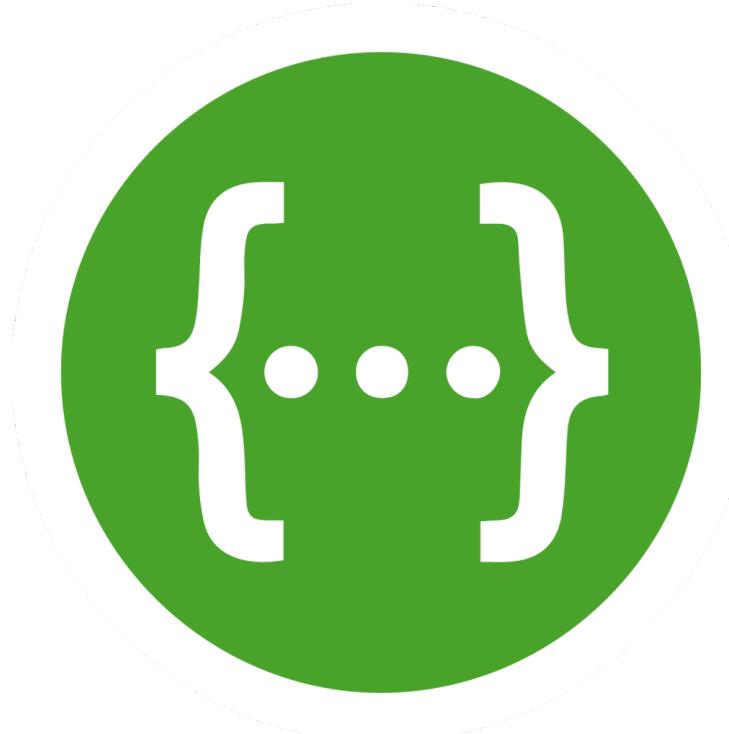




А еще

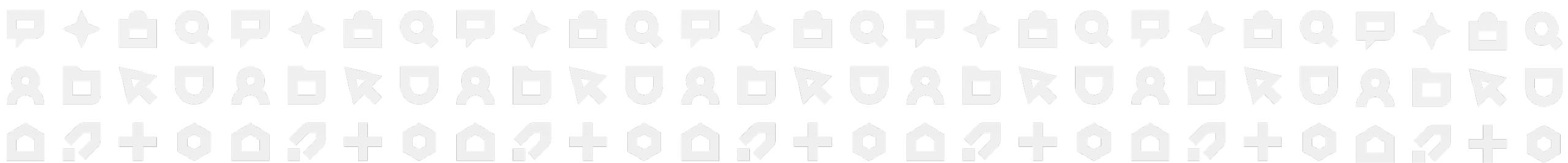


Insomnia



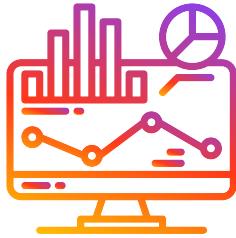
Swagger

Redoc, Apicurio Studio,
Stoplight Studio, Speccy и
非常多 других...

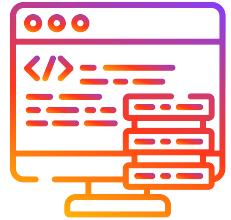




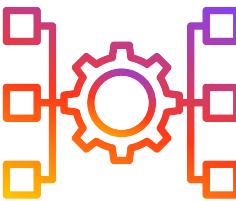
Итоги



Четкое определение интерфейсов и ожидаемого поведения, уменьшая вероятность недопонимания и ошибок в интеграции



Упрощение процесса разработки и тестирования благодаря автоматизации создания кода и тестов



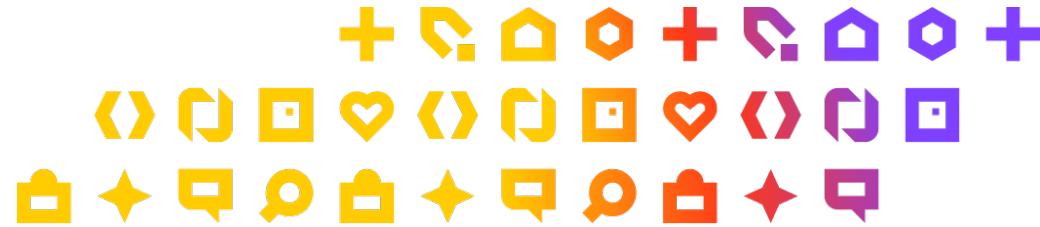
Улучшение совместимости и масштабируемости систем за счет стандартизации подходов к описанию API



Актуальность документации и гармония между проектом и реализацией



А еще...



**В команде наступает мир, счастье и
полное понимание друг друга :)**



Спасибо за внимание!



Обсудим доклад?



ANALYST
DAYS #18



Бодрый Кодер @bodrcoder

