<epam>

REST API tests in less than a minute

Marcin Maj

June 2023

Agenda

01 Problems to solve

02 Open API / Swagger

03 Generate clients and servers

04 Demo – client generate and test

05 Integration scenarios

06 GitHub

You are here >>>

01 Problems to solve

02 Open API / Swagger

03 Generate clients and servers

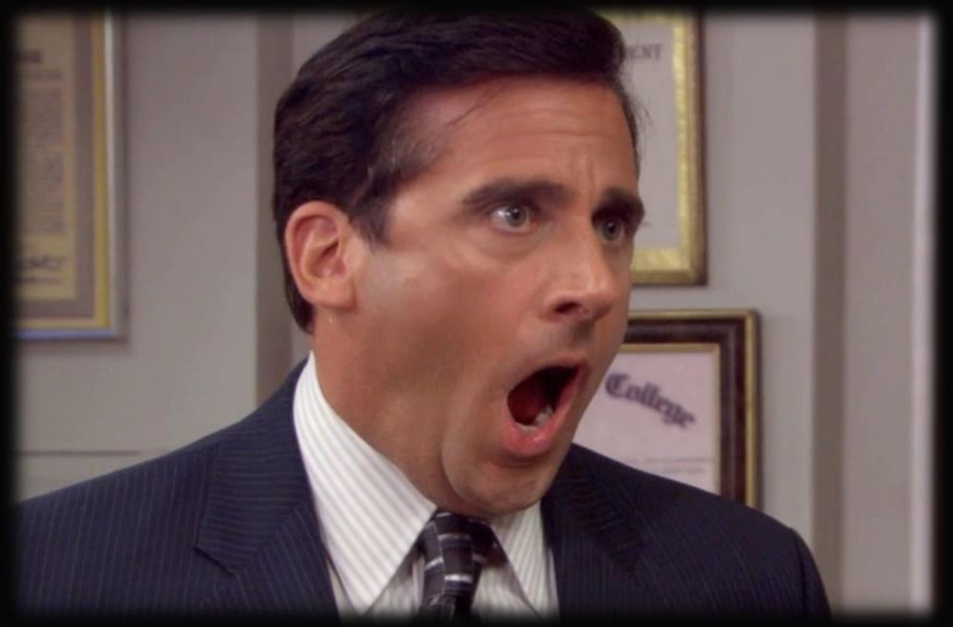
04 Demo – client generate and test

05 Integration scenarios

06 GitHub

Problem: Codeless maintenance

Effort needed to maintain codeless, almost codeless or tools solutions



Name

Get Conversations

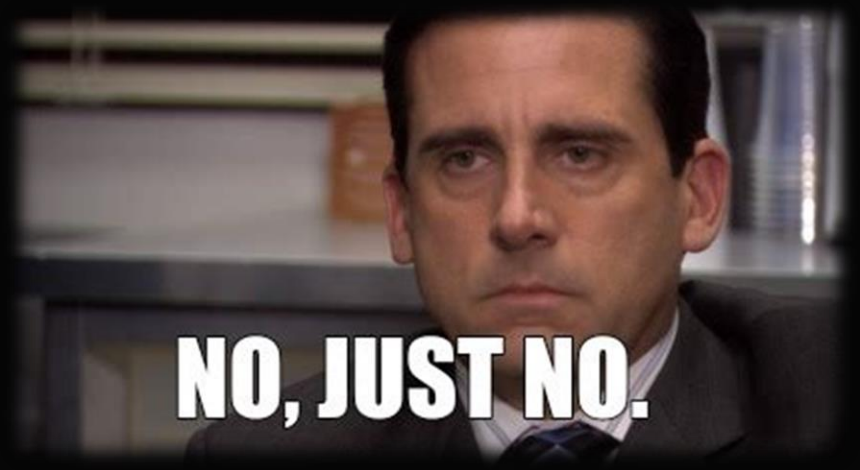
Description | Authorization | Pre-request Scripts | Tests

These tests will execute after every request in this folder. Learn more about Postman's execution order.

```
const jsonBody = pm.response.json().content;

if (responseCode.code === 200  &&  !pm.info.requestName.includes('Count')) {

    pm.test(`${pm.info.requestName}: ` + 'Each object and all nested objects have correct keys.
        ', () => {
        //each object
        jsonBody.forEach(body => {
            pm.expect(body, `${pm.info.requestName}: ` + 'Response body keys are not correct.')
                .to.have.keys('id', 'phone', 'type', 'status', 'userId', 'hasAttachments',
                    'hasSMS', 'interactionsCount', 'unseenCount', 'preview', 'info',
                    'customer', 'createdAt', 'updatedAt', 'lastIncomingAt', 'lastRepliedAt',
                    'lastAssignedAt');});
        //customer objects
        jsonBody.forEach(body => {
            pm.expect(body.customer, `${pm.info.requestName}: ` + 'customer object keys are not
                correct.')
                .to.have.keys('id', 'firstName', 'lastName');});
        //info objects
        var i;
        for (i = 0; i < jsonBody.lengh; i++){
            if (jsonBody[i].type === "call"){
                pm.expect(jsonBody[i].info, `${pm.info.requestName}: ` + 'info object keys are
                    not correct.')
                    .to.have.keys('direction', 'flags', 'callID', 'result');
            }
            else {
                pm.expect(jsonBody[i].info, `${pm.info.requestName}: ` + 'info object keys are
                    not correct.').to.have.keys('direction');
            }
```

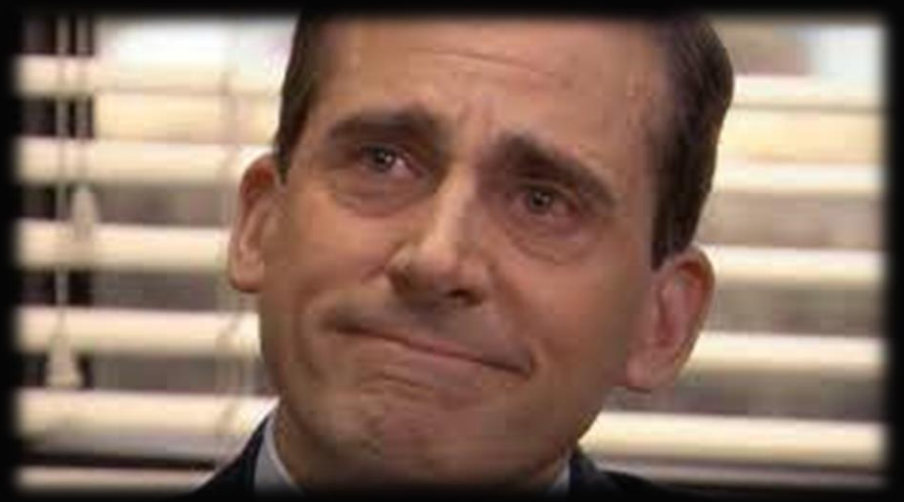
Problem: Boiler plate code

To stop generate boiler plate code. Focus on technical excelence and producivity of testing solutions



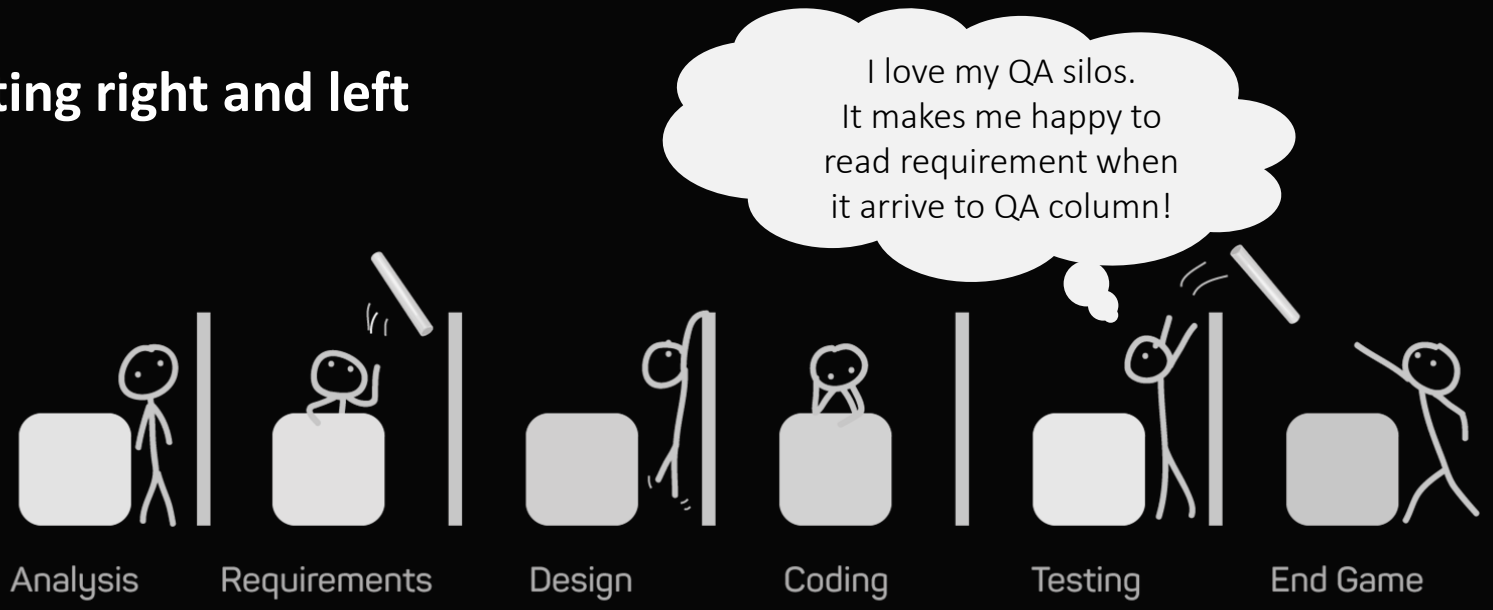
Problem: Weak code quality

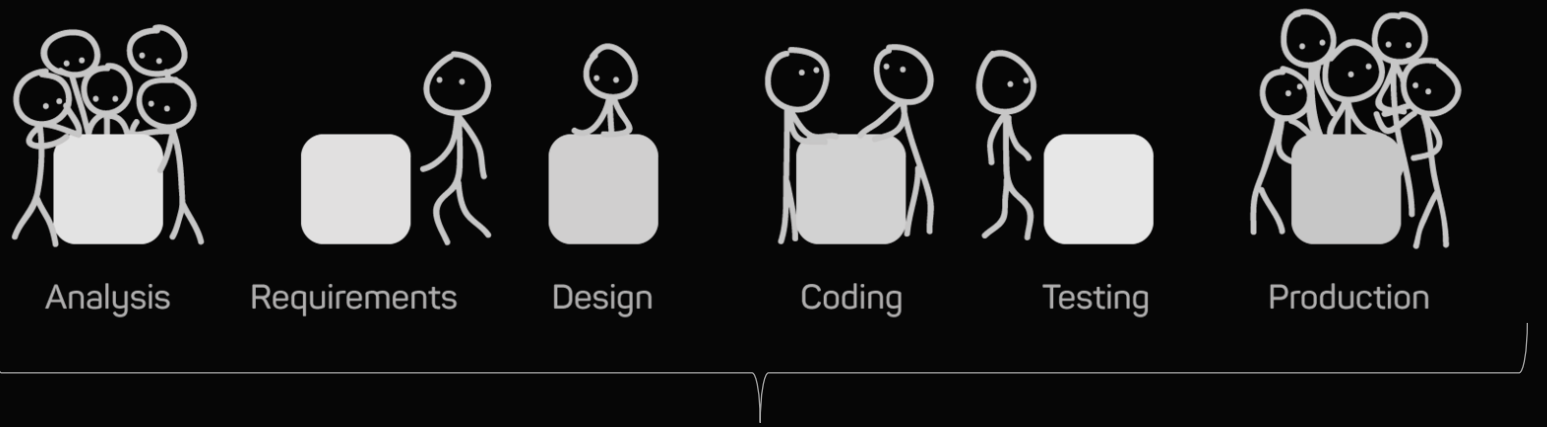
To start using proper tools and create high quality code



```java
package JsonPathForJsonObject;

import io.restassured.path.json.JsonPath;

public class NestedJsonObject {

    public static void main(String[] args) {

        String jsonString = "{\r\n" +
                "  \"firstName\": \"Amod\",\r\n" +
                "  \"lastName\": \"Mahajan\",\r\n" +
                "  \"address\": {\r\n" +
                "    \"houseNo\": 404,\r\n" +
                "    \"buildingName\": \"Not Found\",\r\n" +
                "    \"streetName\": \"Gumnam gali\",\r\n" +
                "    \"city\": \"Bengaluru\",\r\n" +
                "    \"state\": \"Karnataka\",\r\n" +
                "    \"country\": \"India\"\r\n" +
                "  },\r\n" +
                "  \"skills\": {\r\n" +
                "    \"language\": {\r\n" +
                "      \"name\": \"Java\",\r\n" +
                "      \"proficiency\": \"Medium\"\r\n" +
                "    }\r\n" +
                "  }\r\n" +
                "}";

        //Get JsonPath instance of above JSON string
        JsonPath jsonPath = JsonPath.from(jsonString);

        // Since houseNo holds an int value use getInt() method and provide json pa
        int houseNo = jsonPath.getInt("address.houseNo");
        System.out.println("House no is : "+houseNo);

        String name = jsonPath.getString("skills.language.name");
        System.out.println("Name is : "+name);

    }

}
```

Problem: Shifting right and left





Common understanding, prototyping and shifting LEFT

QA Column

You are here >>>

01 Problems to solve

02 Open API / Swagger

03 Generate clients and servers

04 Demo – client generate and test

05 Integration scenarios

06 GitHub

What is Open API Specification

*The **OpenAPI Specification**, previously known as the Swagger Specification,*

*is a specification for a machine-readable **interface definition language** for*

- ***describing,***
- ***producing,***
- ***consuming***
- ***visualizing web services.***

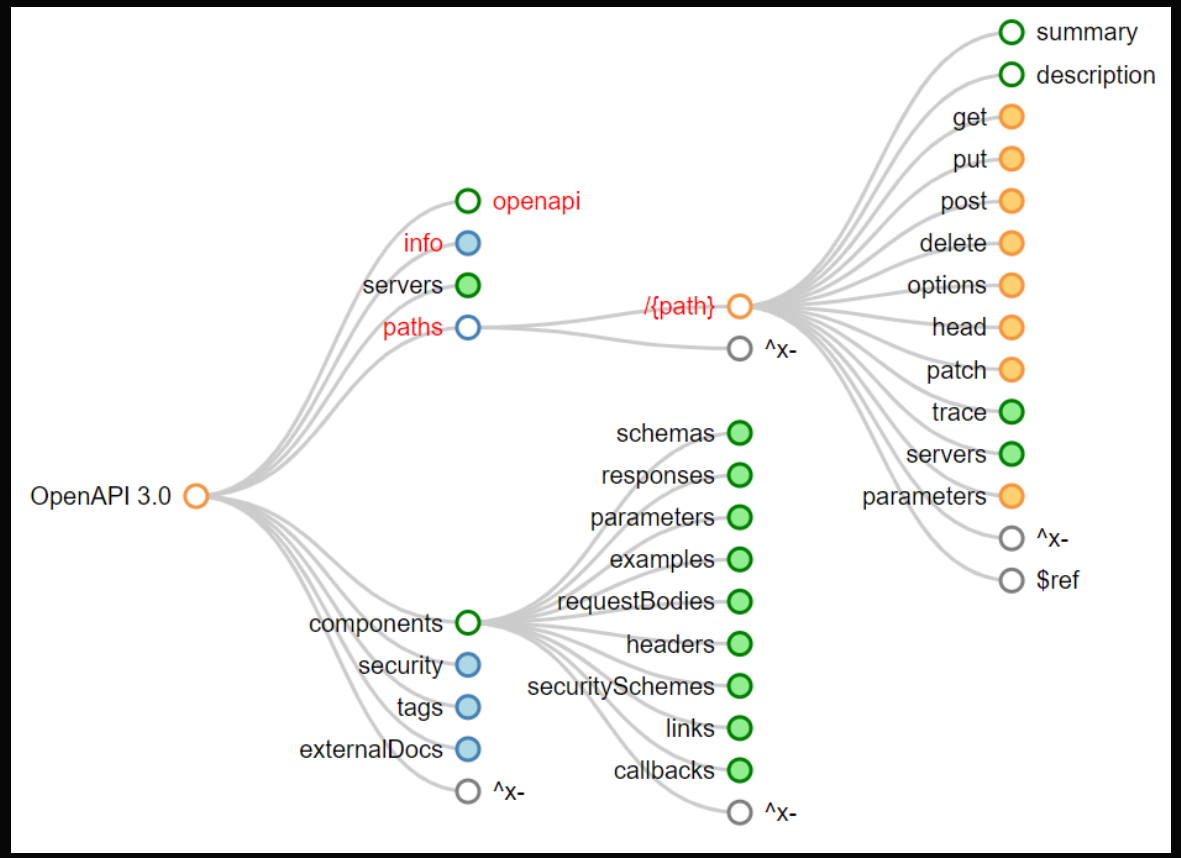
Previously part of the Swagger framework, it became a separate project in 2016, overseen by the OpenAPI Initiative, an open-source collaboration project of the Linux Foundation.

Swagger and some other tools can

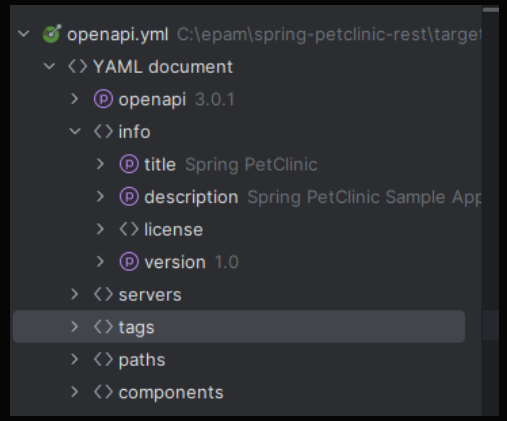
- ***generate code,***
- ***documentation***
- ***and test cases***

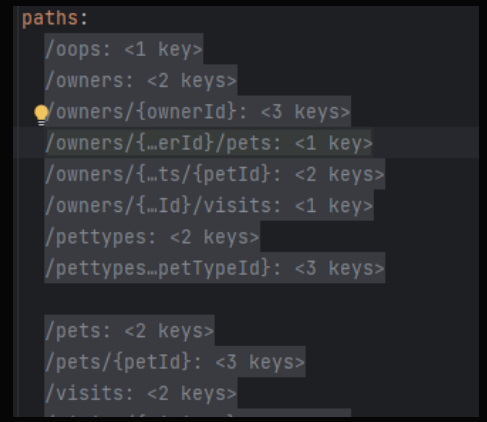
from interface files.

What's inside



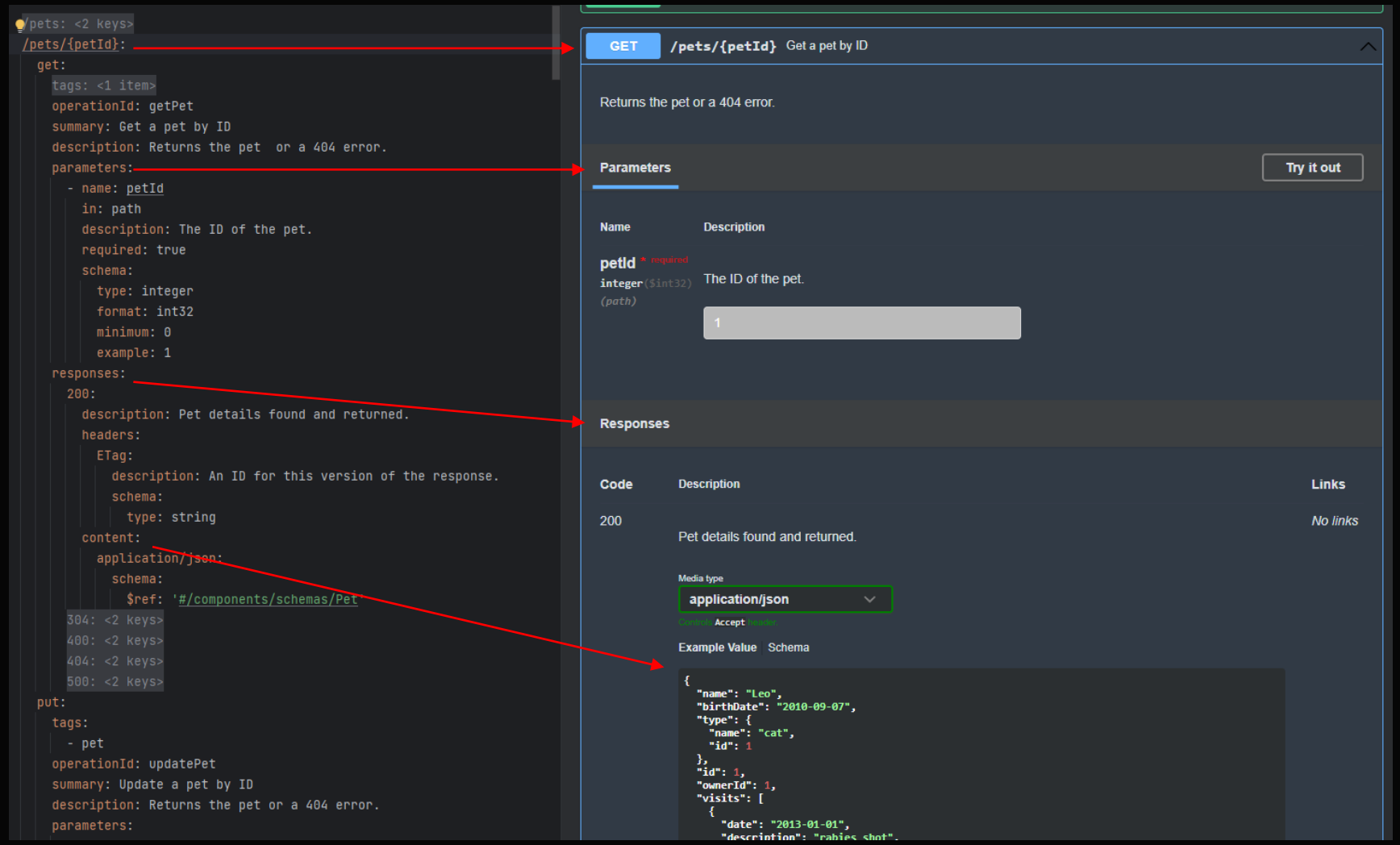

```yaml
openapi: 3.0.1
info:
  title: Spring PetClinic
  description: Spring PetClinic Sample Application.
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0
  version: '1.0'
servers:
  - url: http://localhost:9966/petclinic/api
```



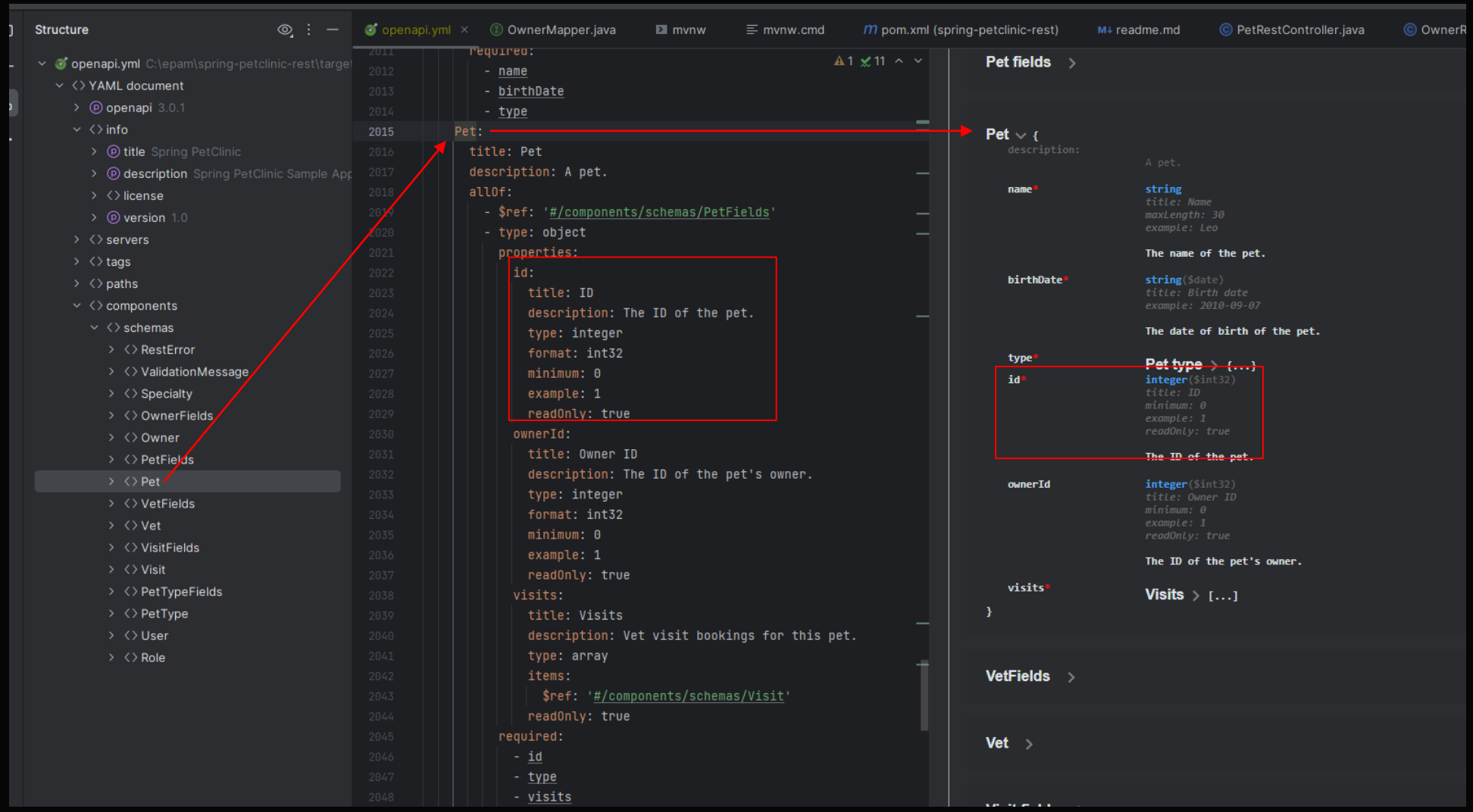


https://swagger.io/specification/
https://spec.openapis.org/oas/v3.1.0
https://openapi-map.apihandyman.io/?version=3.1

Open API Spec and „Swagger view”



Open API Spec and „Swagger view”



Question

Do we have all needed data?

Can we use it to make testing easier?

You are here >>>

01 Problems to solve

02 Open API / Swagger

03 Generate clients and servers

04 Demo – client generate and test

05 Integration scenarios

06 GitHub & QA

Open API Generator

Client Generators (66 technologies!)

- CLI version
- Maven Plugin
- and others

https://openapi-generator.tech/docs/generators

```
> $$("#server-generators + ul a").map(g => g.innerText).length
< 57
> $$("#client-generators + ul a").map(g => g.innerText).length
< 66
```

Server Generators (57 technologies!)

We are not going to cover it today

You can generate not only client but also API itself

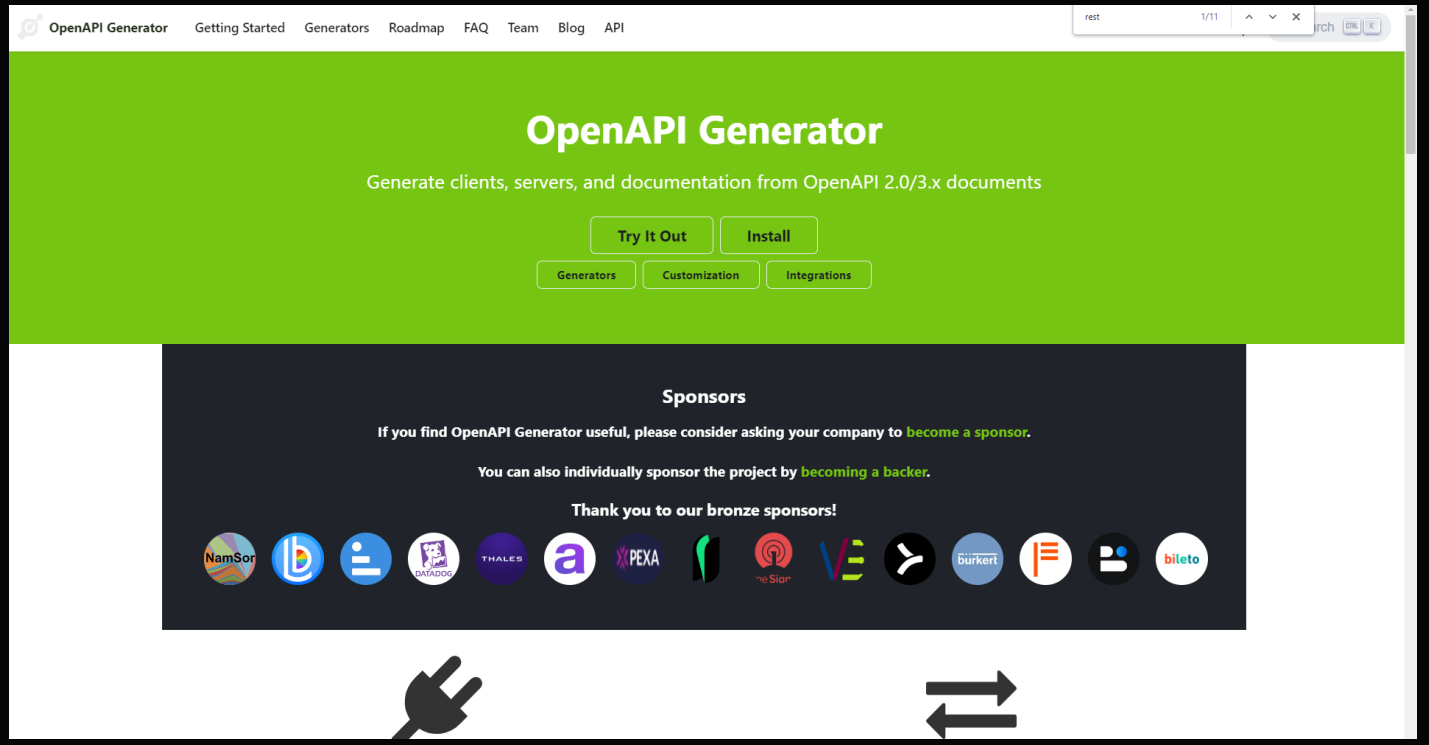
Use Case:

- You need a mocked version of one of your APIs in QA env

Solution

- Generate server from Open API spec
- Put dummy / mocked data as response
- Build & Deploy it ☺

Open API Generator – no screens, just explore web page!



https://openapi-generator.tech/

You are here >>>

01 Problems to solve

02 Open API / Swagger

03 Generate clients and servers

04 Demo – client generate and test

05 Integration scenarios

06 GitHub & QA

Code time ☺

You are here >>>

01 Problems to solve

02 Open API / Swagger

03 Generate clients and servers

04 Demo – client generate and test

05 Integration scenarios

06 GitHub & QA

SCENARIO NO. „0”

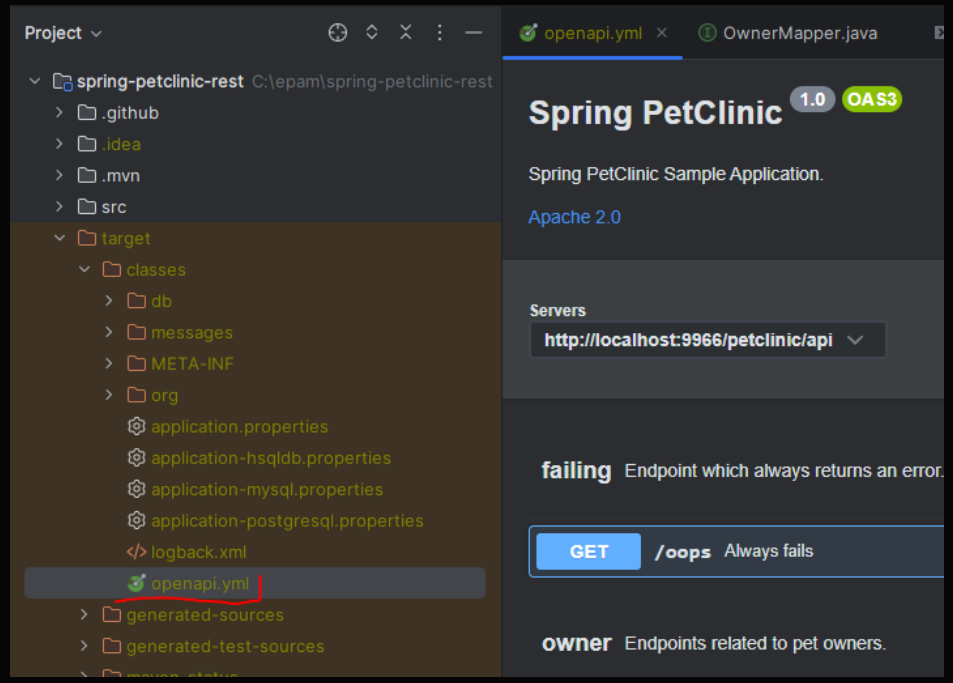
No Open API Specification

Dev

- No action

QA

- Take code of API
- Add documentation or generate
- Build client
- Implement tests
- Run/Report




```
const swaggerAutogen = require('swagger-autogen')()

const outputFile = './swagger_output.json'
const endpointsFiles = ['./routers/personRouter.js']

swaggerAutogen(outputFile, endpointsFiles)
```

```xml
<dependency>
   <groupId>org.springdoc</groupId>
   <artifactId>springdoc-openapi-ui</artifactId>
   <version>1.7.0</version>
</dependency>
```

SCENARIO NO. 1

One timer – (generated project)

Dev

- Deploy Swagger
- Publish OAS on SwaggerHub

QA

- Take Open Api Secification
  - Or take ready client code from Swagger Hub
- Generate client(s)
- Implement Tests
- Run/Report

Cons

- Bolier plate code in repository
- Have to be careful with configuration
- Hard to maintain on frequent changes

Pros

- Super quick to generate
- Test stubs are ready to use and run
- Ready to run on CI

SCENARIO NO. 2

Popular – dependency to client on TAF

Dev

- Deploy Swagger 
- Publish OAS on SwaggerHub

QA

- Take Open Api Secification (or URL)
- Generate client(s) – as separate project or module

  You can use Maven Open API Generator plugin
- Test are implemented as part of TAF
- TAF just holds dependency to client
- Run/Report

Cons

- After change in API commit is needed (spec) – if used as file
- Changes is spec are not tracked in repository – if used as URL
- Test are run against currently deployed version

Pros

- Super quick to generate
- Test stubs are ready to use and run
- Ready to run on CI

SCENARIO NO. 6

Automated and Integrated

Dev

- Deploy OAS (yaml or json) to artificat repository as part of CI
- Deploy Swagger
- Publish OAS on SwaggerHub

QA

- Take any version of Open Api Secification
- Generate client(s) – during each build
  - **You can use Maven Open API Generator plugin (put URL in pom.xml)**
- Test are implemented as part of TAF
- TAF just holds dependency to client
- Run/Report

Maven

Cons

- ?

Pros

- No Bolier plate code in repository
- No commit on spec change (spec is downloaded from Swagger during build)
- Super quick to generate
- Test stubs are ready to use and run
- Ready to run on CI

01 Problems to solve

02 Open API / Swagger

03 Generate clients and servers

04 Demo – client generate and test

05 Integration scenarios

You are here >>> **06** GitHub & QA

FAQ (and problems)

We do not have Swagger.

Or our swagger is old, depracated or not reflect current state.

- With PO/PM/SM put testability requirement into backlog to make it usable and deployable
- Do not forgot that it is also project documentation, so you impove documentationa and testability
- As mentioned, – take ownership and generate spec on you own

We already have a test suites for API

- Like always - small steps
- Like every software also TAF's and test suites getting depracated or obsolete
- They produce or are tech debt
- Or with growing complexity of project are not valid / hard to maintain.

- Nothing should be set in store
- Continous refactoring

What in case of changes in API

- Nonbreaking changes
  - Regression still works
  - You must cover new code with test cases

- Breaking changes, like:
  - Removed endpoint
  - Change in model (i.e. removed field)
  - Your client will be generated and compiled
  - BUT tests fail on compilation = you must adjust to new state of API!

FAQ (and problems)

What about versioning?

- When you have OAS in your TAF repository it's just versioned with your code
- When you user Nexus or Artifactory you can run against any version with comination of
  - Git tag or commit
  - Specified version of artifact

What about negative tests?

- Seems to be tricky?
  - When generate client enums are fixed, you cannot use not existing entry!

BUT

- You can still serialize/deserialize and change payload
- You can user RestAssured filters to modify payload
- You can use API client and connectivity fom Open API and work on raw paylods using i.e. JsonPaths
- It can be mixed!

Me and my 50 microservices

	C1	C2	C3	C4
1	group	name	specUrl	schemas
2	pet	pet_service	http://{{env}}.docs.example.com/swagger.json	http://{{env}}.docs.example.com/schema.json
3	pet	pet_owners	http://{{env}}.docs.example.com/swagger.json	http://{{env}}.docs.example.com/schema.json
4	pet	pet_types	http://{{env}}.docs.example.com/swagger.json	http://{{env}}.docs.example.com/schema.json
5	pet	pet_visits	http://{{env}}.docs.example.com/swagger.json	http://{{env}}.docs.example.com/schema.json
6	pet	pet_service	http://{{env}}.docs.example.com/swagger.json	http://{{env}}.docs.example.com/schema.json
7	pet	pet_service	http://{{env}}.docs.example.com/swagger.json	http://{{env}}.docs.example.com/schema.json

Problem

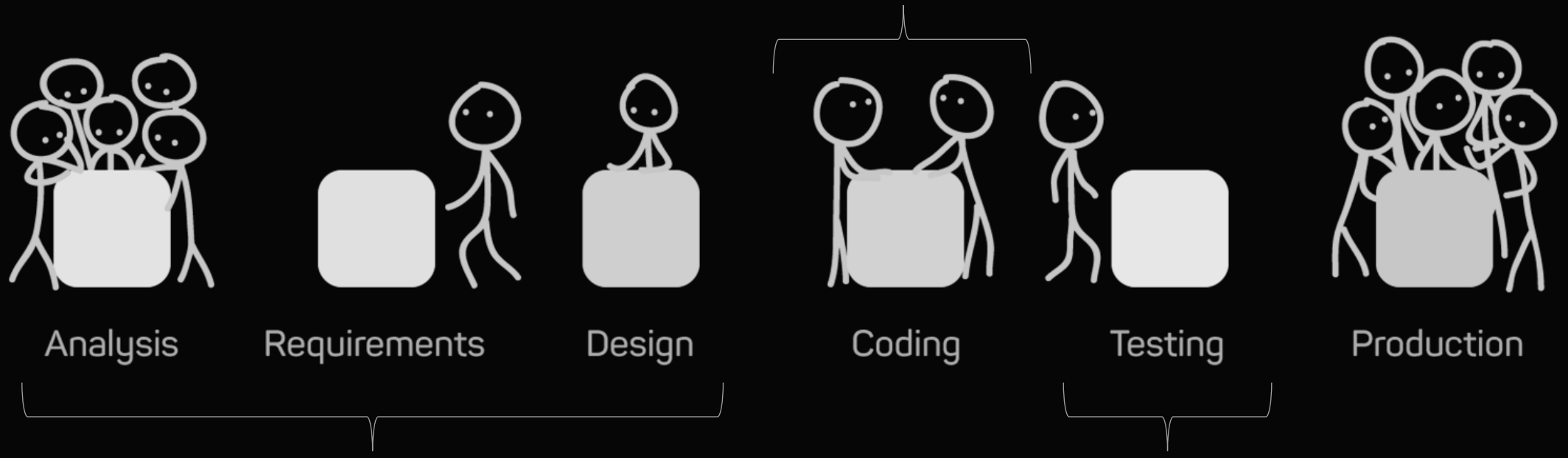
- I have 50 microservices in my project to test
- Each service has own Swagger instance and specification
- I have to handle 50 specs and

Solution

- Build tools have specified structure **– use that fact!**
- Collect all of them to CSV file (as examle)
- Before building TAF – generate structure of you soulution
- Using the Mave you need
  - For each service direcotry with
    - pom.xml (with open-api-generator-maven-plugin)
    - Open Api specs (json or yaml)
    - .gitignore (to ignore all generated code)
- Create small python script which
  - Create project structure
  - Add modules to <modules> section in parent pom.xml
- Then build all the clients – Maven Reactor will handle it!

BDD/ATDD/TDD & shifted left friends with OAS

- OAS prototype is created
- Automation goes first as TDD
- Or automation in parallel with development
- Devs can run new test cases with new code



- Collect Requirements
- Create OAS as prototype
- Create / plan test cases

- Final manual confirmation
- Exploratory
- Regression

Finally – Q and A

https://github.com/majran/tech_talks_sample_petstore

Now we are asking ☺

Name two „core” elements of Open API Specifiction structure?