How to create modular microservice test projects

Elias Nogueira

I help professional software engineers (backend, frontend, qa) to develop their quality mindset and deliver bug-free software so they become top-level engineers and get hired for the best positions in the market.

Elias Nogueira

🏢 Backbase
👨‍💼 Principal Software Engineer
📍 Utrecht, the Netherlands
🌐 eliasnogueira.com
🐦 @eliasnogueira

bit.ly/eliasnogueira

What problem do we want to solve?

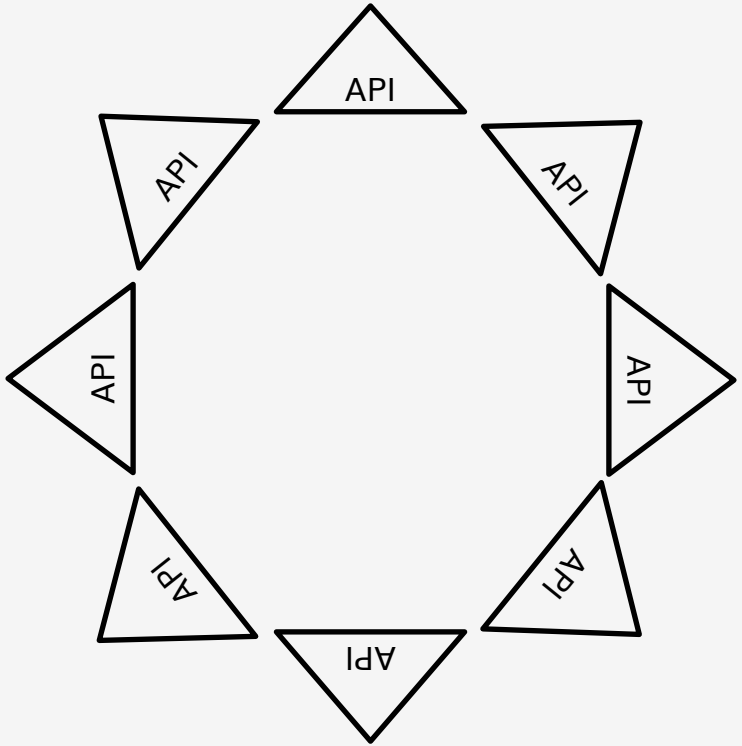
Implicitly we want ...

- Increase confidence in delivery
- Cover key business scenarios
- The certainty that the test will find a possible failure after some modification

API anatomy (testing perspective)

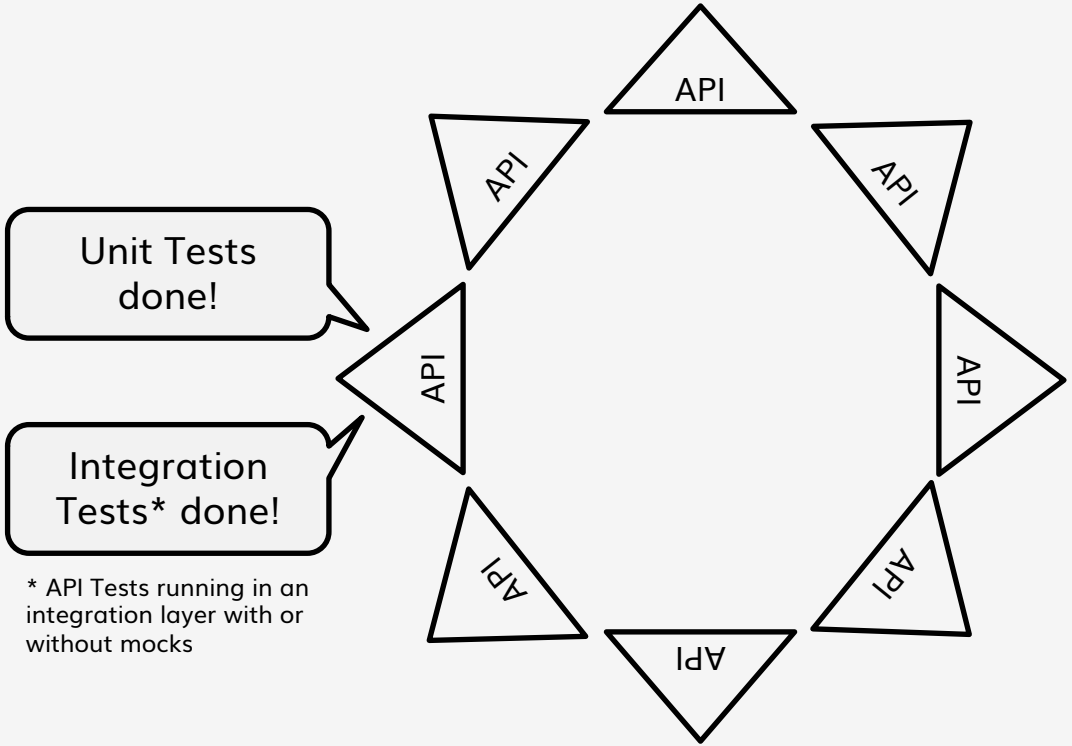
Individual APIs

Each microservice is a small feature that runs in isolation.



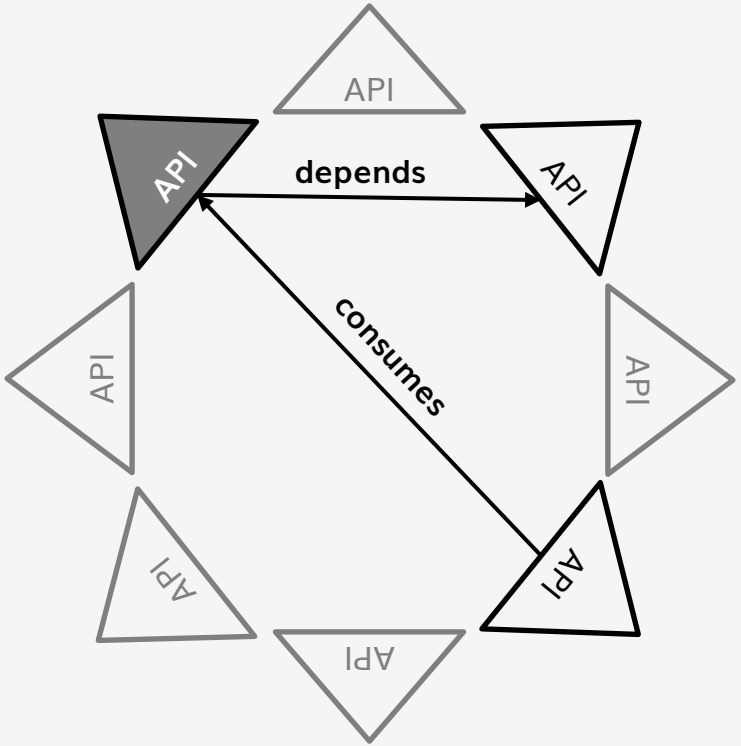
Individual APIs

Each API must have its unit tests and integration.



The problem

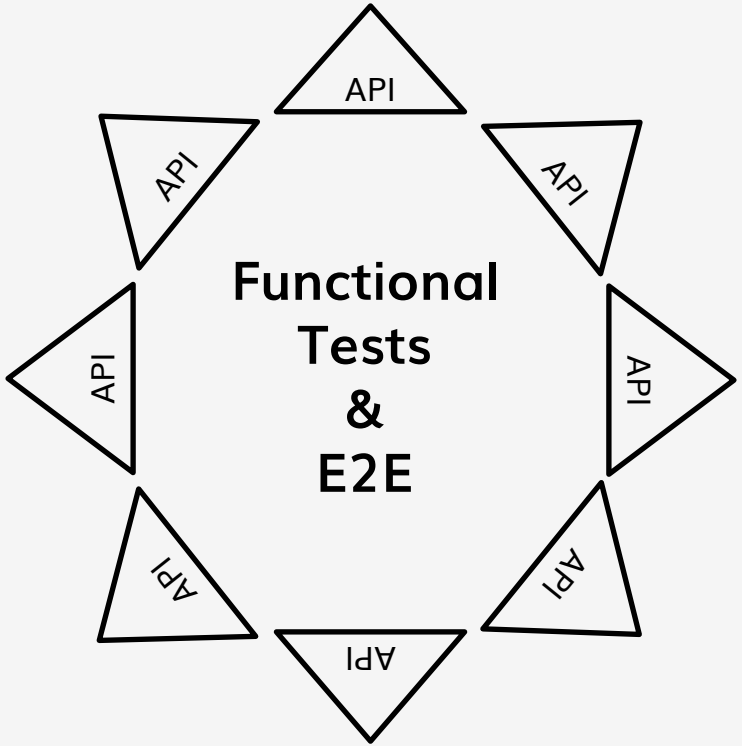
Integration tests won't cover real scenarios of dependent APIs because of the mocks.



Possible solution

Possible solution

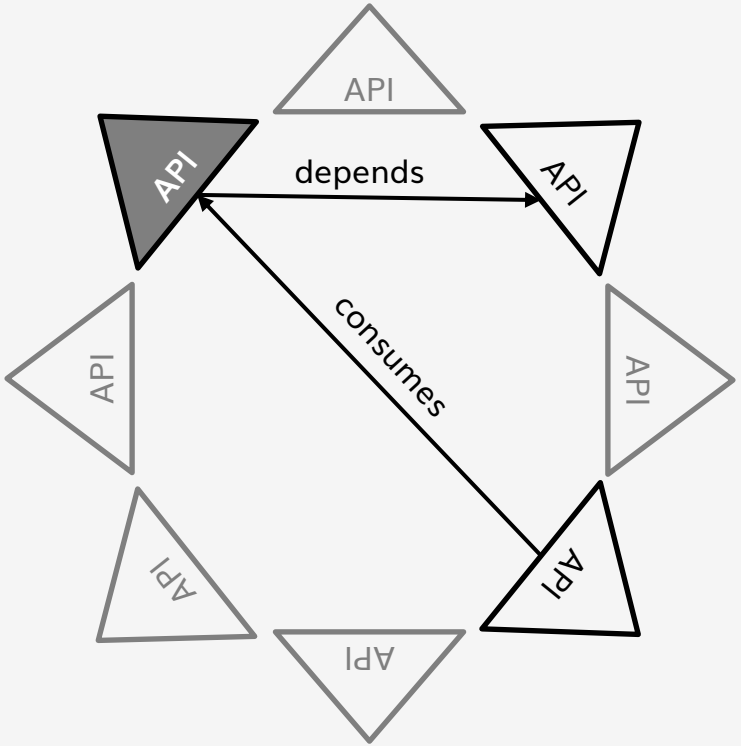
Create functional tests and e2e to cover possible gaps and decrease the test on the UI.



Possible solution

What

Add tests to any kind of dependencies, as well for the user journey

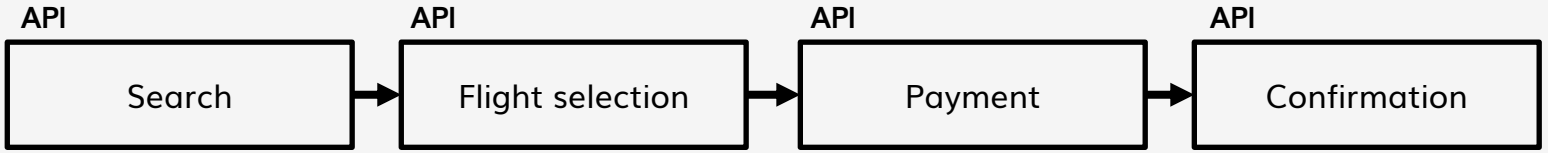


Analogy: buying a flight ticket

Buying a flight ticket

We must make sure each API is self-tested

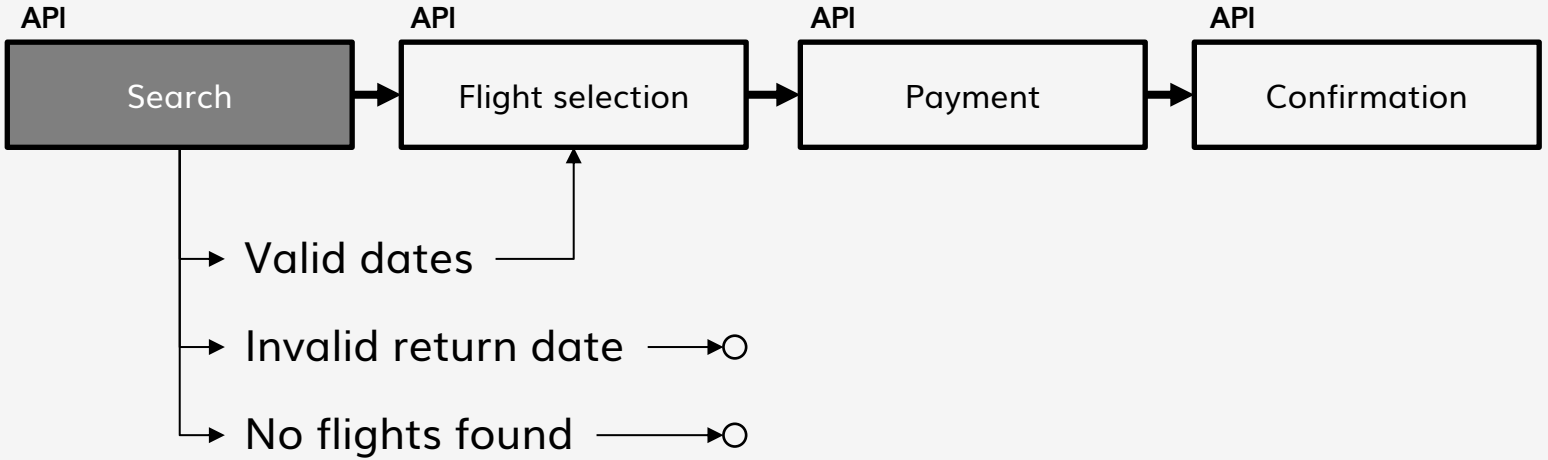
unit -> integration -> functional



Buying a flight ticket

We must make sure each API is self-tested

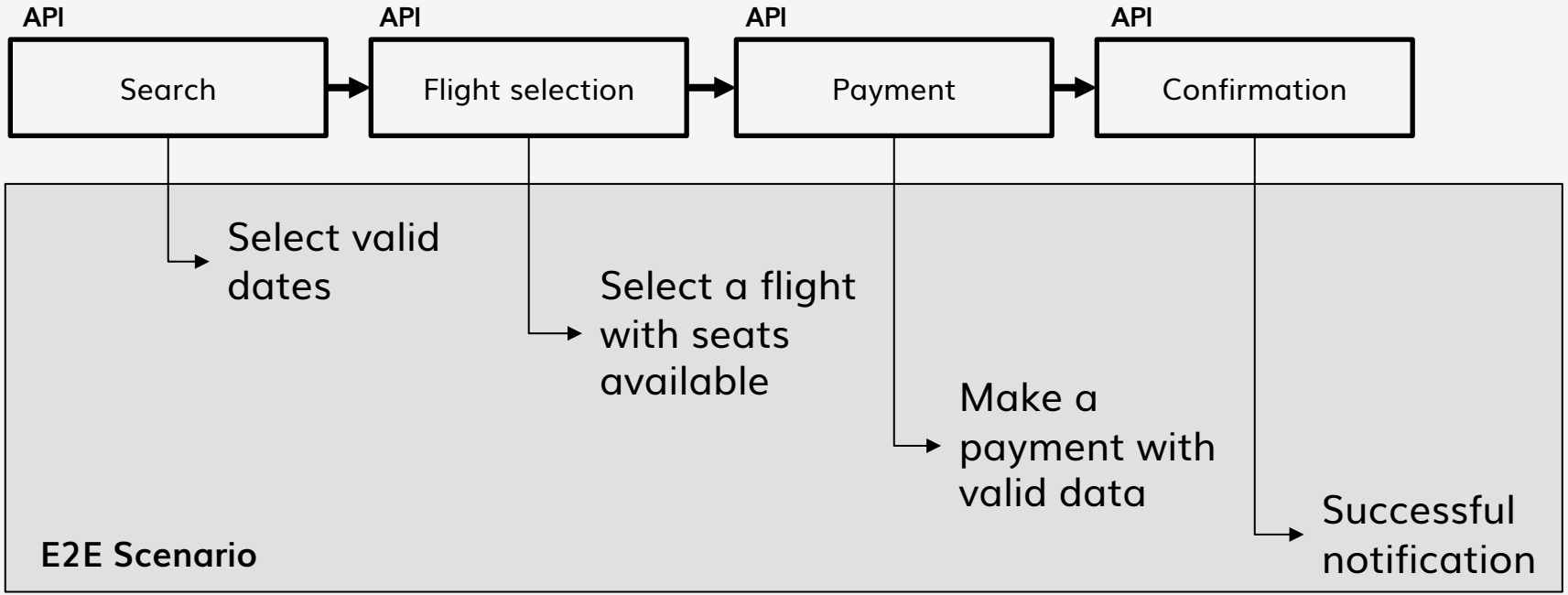
unit -> integration -> functional



Buying a flight ticket

We must make sure each API is self-tested

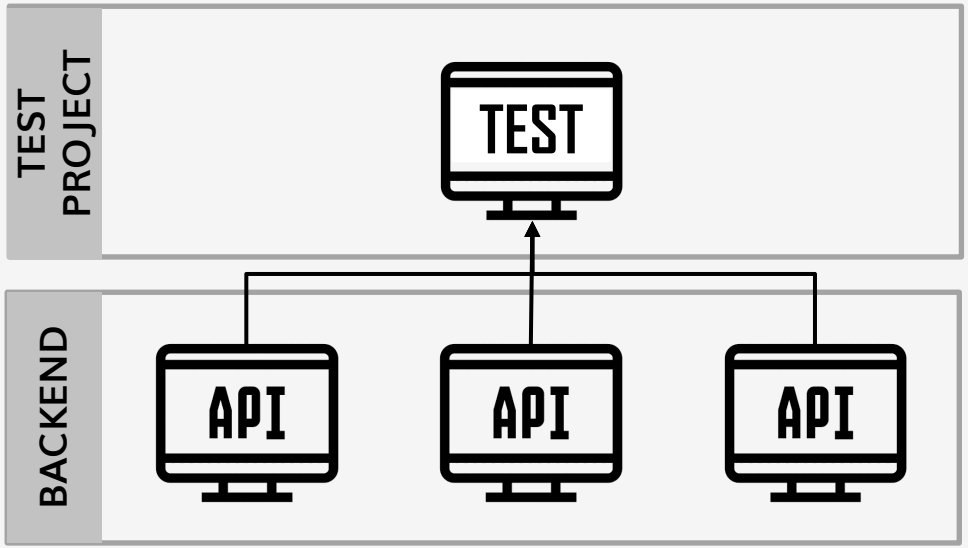
e2e: buy a round-trip flight



How can test projects be created?

Model 1

A test project for all microservices



Model 1

Pros

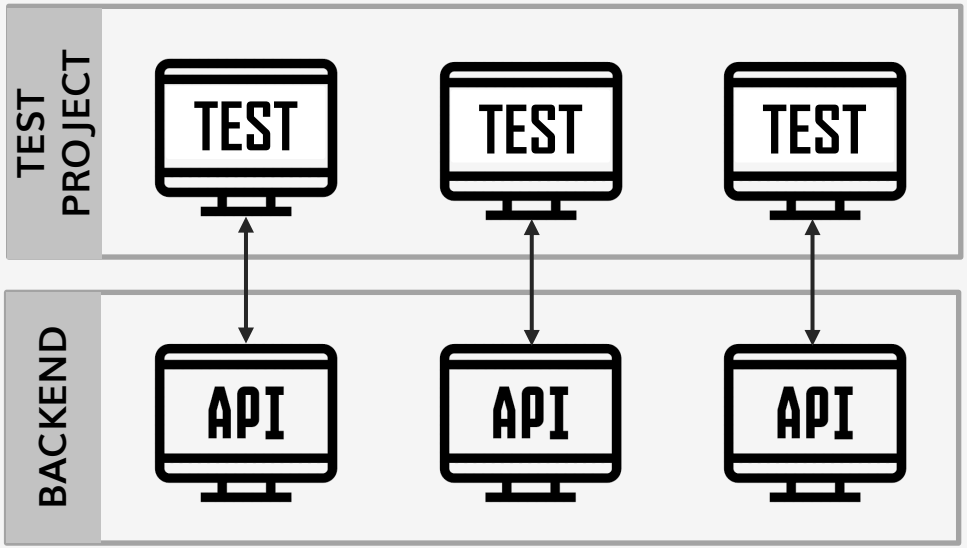
- Centralization of code in a single project
- Speed and agility in creating tests and solving problems

Cons

- Constant changes by several people can have side effects on results

Model 2

A test project for each microservice where interactions with the endpoint (s) will be in the test project.



Model 2

Pros

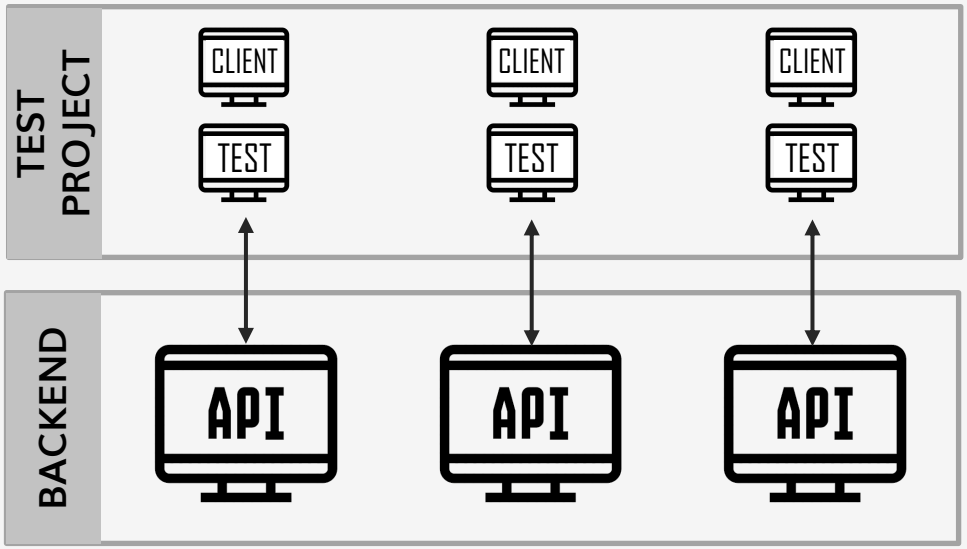
- Organized decentralization of test projects towards an API
- Isolation between APIs

Cons

- Possible code duplication

Model 3

A test project for each microservice divided into **client** and **test** projects



Model 3

Client project

We will put here all the necessary logic to make the requests as:

- Exception return
- Transport Objects
- Request call
- Customizations for validations
- Pointing settings (URI, base path, port)

Model 3

Test Project

We will put here all the methods of using the API we created in the client project, creating the testing logic and validating the results.

In summary, all tests are created here!

Model 3

Pros

- Greater version management (new features, breaking changes)
- Easy use by other teams
  - they just need to consume the customer and create their tests

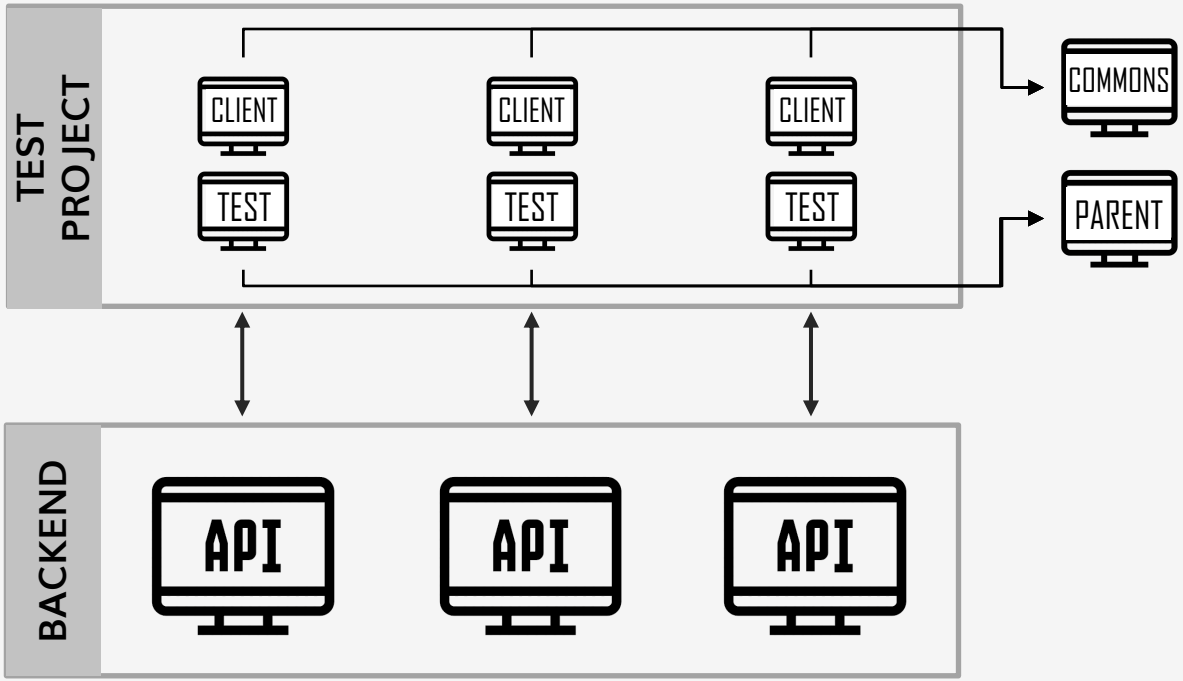
Cons

- Time increase compared to previous models

Implementation exemple

Implementation example

Project separation



Project separation

Project containing all calls to the microservice, whether the calls were successful or simulating exceptions.

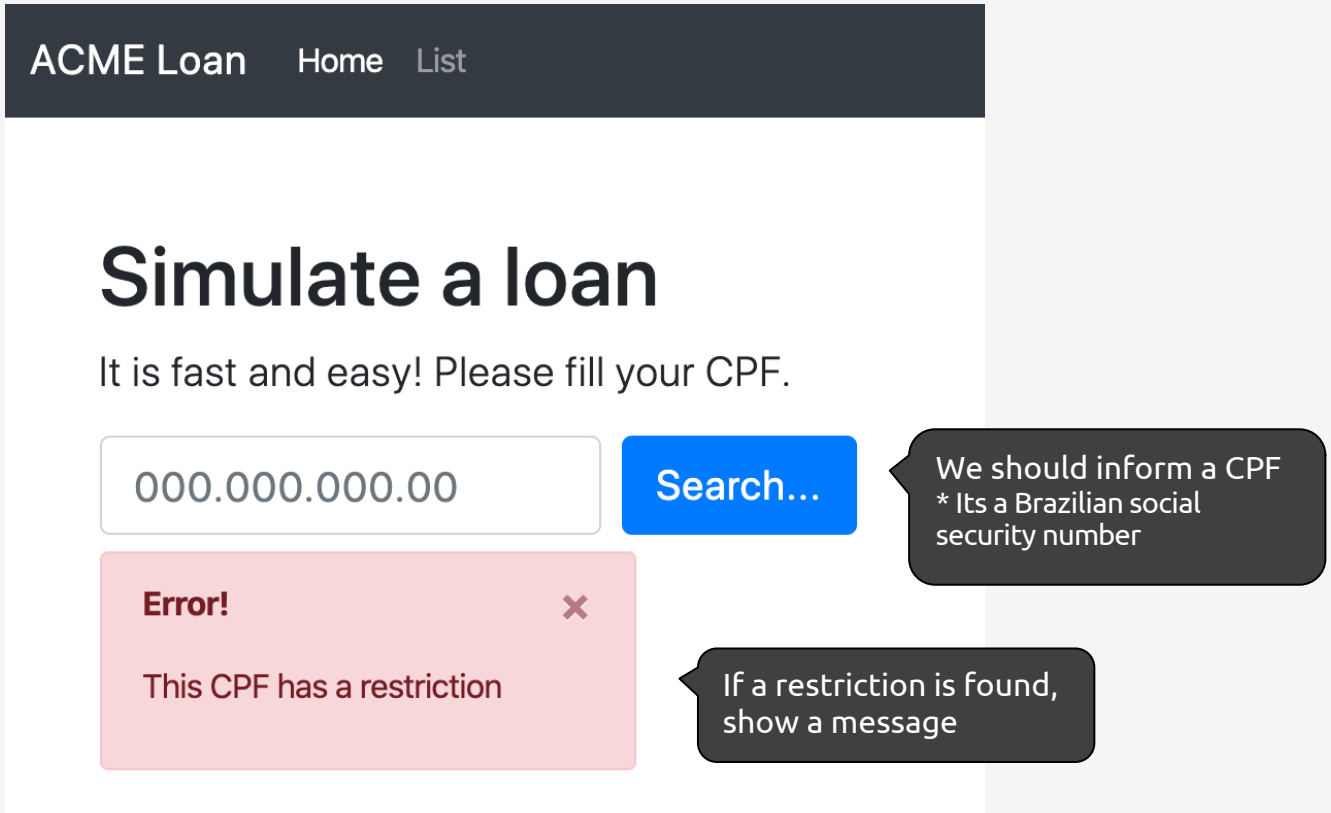
Project containing tests for the microservice, testing its functionality and dependencies with other microservices.

Code and libraries common to clients such as URL definitions, authentication, and any other shared code.

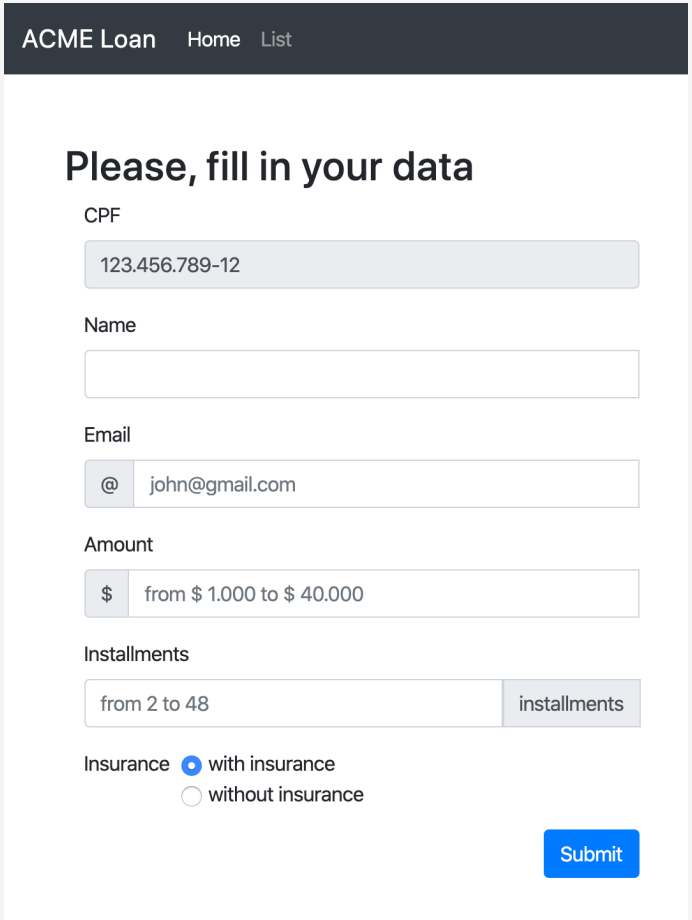
Code and libraries common to tests such as data mass, support functions, or any code common to tests.

Microservice examples

Possible graphical interface: Constraint check by CPF



Microservice examples



Possible graphical interface:
Loan Simulation

Fill in loan information

Microservice examples

We will put here all the necessary logic to make the requests as:

Constraint Query API

- **GET** /api/v1/restricrions/{cpf}
- **GET** /api/v2/restricrions/{cpf}

Credit Simulation API

- **GET** /api/v1/simulations/
- **GET** /api/v1/simulations/{cpf}
- **POST** /api/v1/simulations/
- **PUT** /api/v1/simulations/{cpf}
- **DELETE** /api/v1/simulations/{cpf}

Client's implementation

Methods will be created to simulate all possible calls from each HTTP method

Constaint Query
API Client

- **`GET /api/v1/restricrions/{cpf}`**
  - **Method** - find CPF
  - **Method** - find CPF and return an exception

Client's implementation

Methods will be created to simulate all possible calls from each HTTP method

Credit Simulation
Client API

- GET **/api/v1/simulations/**
  - **Method** – find all simulations
- **GET** **/api/v1/simulations/{cpf}**
  - **Method** – find a simulation by a given CPF
  - **Method** – find a simulation by a given CPF and return not found
- **POST** **/api/v1/simulations/**
  - **Method** – submit a simulation
  - **Method** – submit a simulation and return unprocessable entity
  - **Method** – submit a simulation and return conflict

Test implementation

A test method will be created for one or more use of the methods on the client

Credit Simulation
Client API

POST **/api/v1/simulations/**

Method – submit a simulation

Credit Simulation
Test Project

should create a successful simulation **- Test**

Test implementation

A test method will be created for one or more use of the methods on the client

Credit Simulation
Client API

Credit Simulation
Test Project

POST **/api/v1/simulations/**

Method – submit a simulation

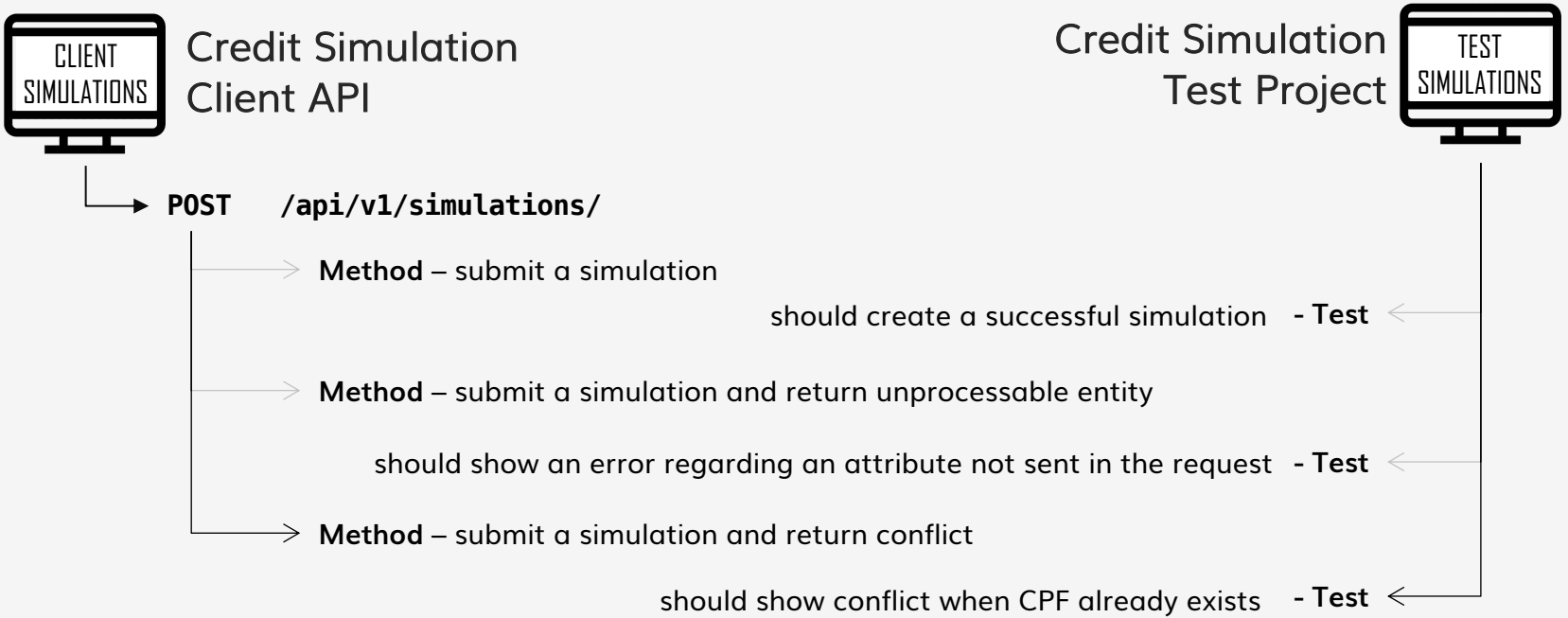
should create a successful simulation **- Test**

Method – submit a simulation and return unprocessable entity

should show an error regarding an attribute not sent in the request **- Test**

Test implementation

A test method will be created for one or more use of the methods on the client



How to use client <-> test?

How to use client <-> test?

Through the client version

- We must add the client dependency in the test

Java + Maven example

```xml
<dependency>
    <groupId>com.eliasnogueira</groupId>
    <artifactId>simulations-client</artifactId>
    <version>1.2.5</version>
</dependency>
```

Typescript + Node example

```
"devDependencies": {
  "@eliasnogueira/simulations-client": ”1.2.5",
}
```

How to use client <-> test?

Pros: association of the client with the microservice version

Just as the microservice evolves the client and the test evolve. Adding a changelog with the association of the client version with the microservice version can help fix bugs.

Also tags helps if you would like to track the evolution.

Versões

Simulations Client	Simulations API
Unreleased	2.7.2
2.0.3	2.6.56
1.2.5	1.2.31
1.2.4	1.2.26
1.2.3	1.2.20
1.2.2	1.2.17
1.2.1	1.2.12
1.2.0	1.2.7
1.1.25	1.1.42

Recap

Recap

- We created a robust model for
  - test a microservice in isolation (functional test)
  - test a microservice and its dependencies (e2e)
- We modularize the creation of projects
  - client: the entire API call for a microservice
  - test: test project using the microservice client
  - commons: client codes and libraries
  - parent: test code and libraries

Thank you!

You can follow me on Twitter **@eliasnogueira** and find the practical examples of this presentation. Be sure to simulate the examples in the code!

SCAN ME

Projects

- https://github.com/eliasnogueira/test-parent
- https://github.com/eliasnogueira/test-commons
- https://github.com/eliasnogueira/restrictions-client
- https://github.com/eliasnogueira/restrictions-test
- https://github.com/eliasnogueira/simulations-client
- https://github.com/eliasnogueira/simulations-test

Important note

To be able to run the Project locally, after you clone it, you must configure the GitHub Packages in your `pom.xml` file.

You can find the example here: https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-apache-maven-registry

Replace the `OWNER` by `eliasnogueira`