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

the basics

rest api concept

It simplifies the development by supporting HTTP methods, error handling, and other RESTFul conventions exposing resources through URLs.
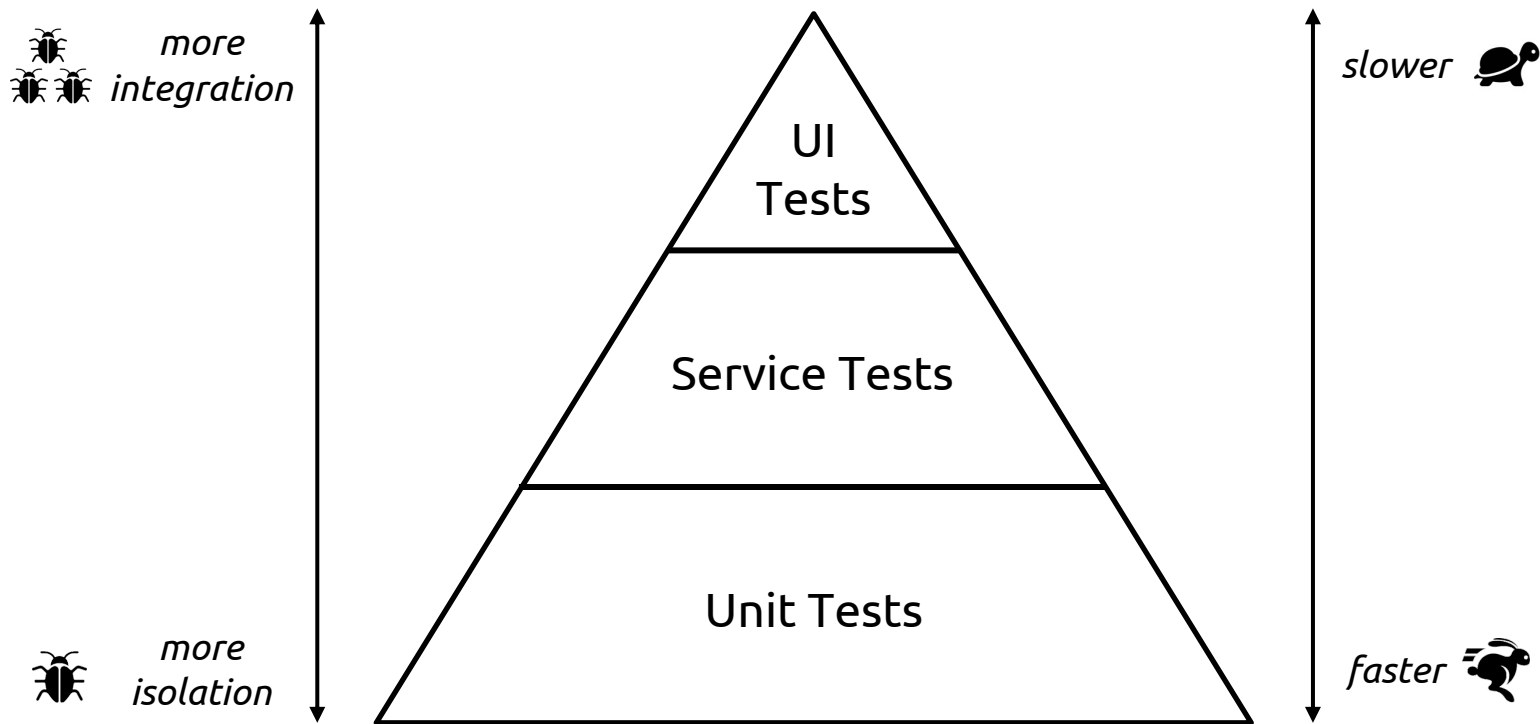
- its programming language independent

- supports different implementations of HTTP methods

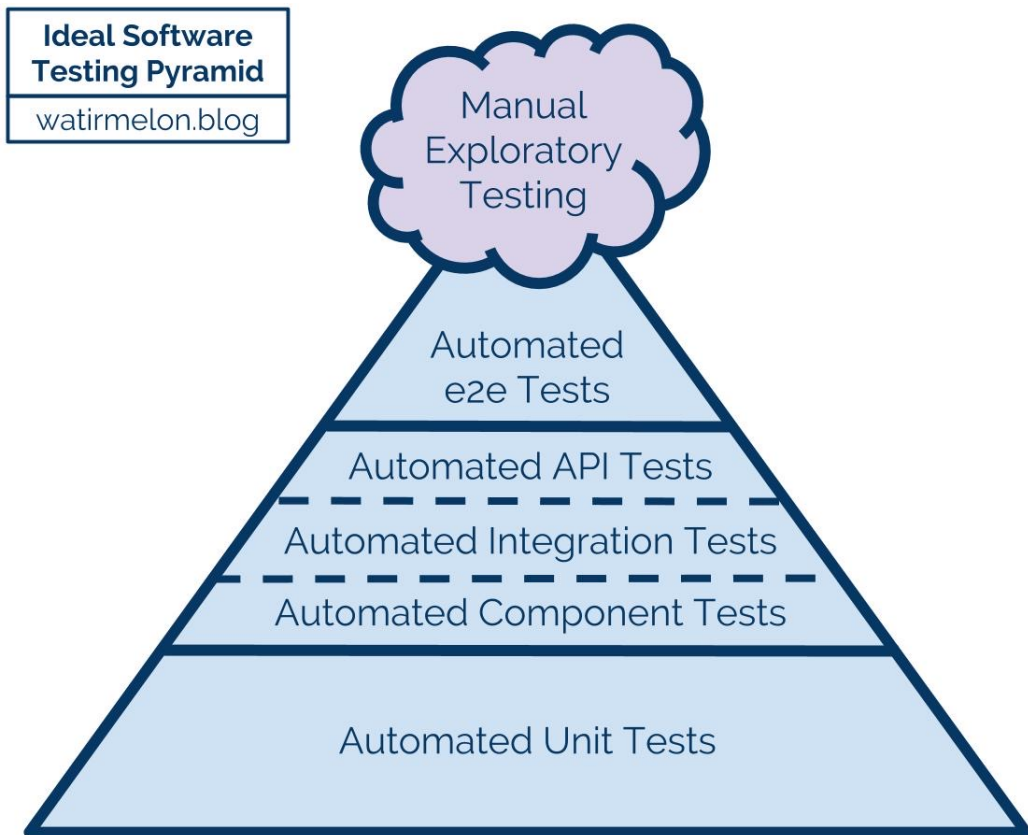- support different representational types (HTML, JSON, XML)

rest api concept

- It has a client-server architecture

- It uses stateless interactions

- Communicates over HTTP using request methods
  - **GET**, HEAD, **POST**, **PUT**, **DELETE**, CONNECT, OPTION, TRACE, PATCH

- Manipulates resources

- Uses HTTP standard response status codes
  - 100 informational
  - 200 success
  - 300 redirection
  - 400 client error
  - 500 server error

eliasnogueira.com

concept

Ideal Test Pyramid

Ideal Software Testing Pyramid

watirmelon.blog

Manual Exploratory Testing

Automated e2e Tests

Automated API Tests

Automated Integration Tests

Automated Component Tests

Automated Unit Tests

eliasnogueira.com

the backend application

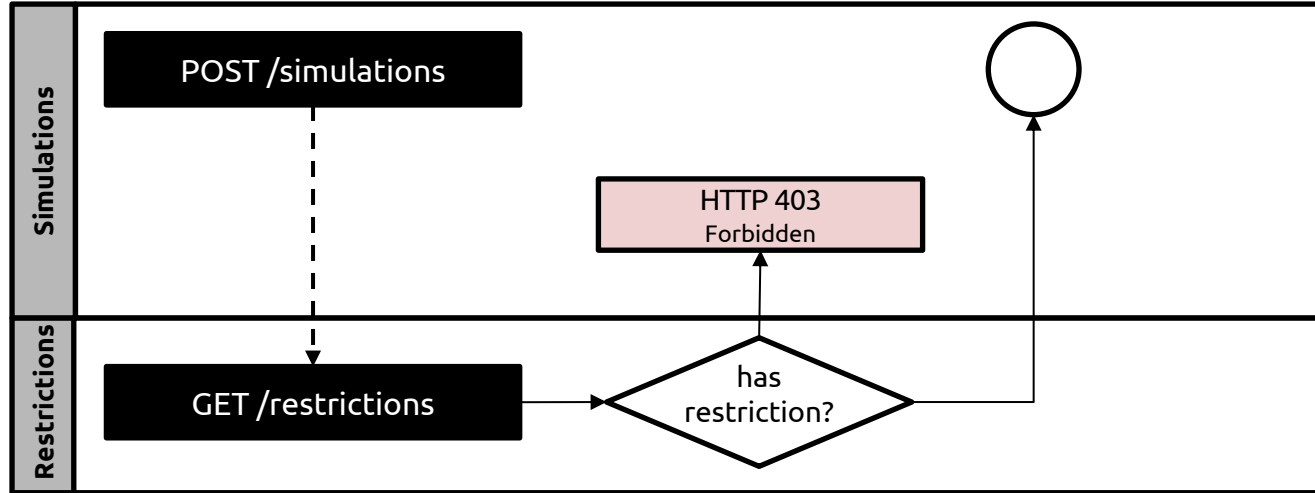# the backend application

The backend is composed of two services:

Simulations API
CRUD operation to record a loan simulation

Restrictions API
GET endpoint to know if a consumer has a restriction, if yes, a simulation is not allowed.

eliasnogueira.com

the backend application

the (non-existent) frontend

ACME Loan   Home   List

ACME Loan   Home   List

Simulate a loan

It is fast and easy! Please fill your CPF.

000.000.000.00   **Search...**

Error!   ✕

This CPF has a restriction

Please, fill in your data

CPF

123.456.789-12

Name

Email

@   john@gmail.com

Amount

$ from $ 1.000 to $ 40.000

Installments

from 2 to 48   installments

Insurance   ⦿ with insurance
              ◯ without insurance

Submit   eliasnogueira.com

API documentation

api documentation

- Usually, we have the API documentation to know any aspect of the API like the **URI**, **parameters**, **headers**, **response**, etc.

- It's important to know how to read this documentation because we will start to understand how we can make the requests and what will be the response

- The documentation is divided into **Resources** and **Models**

api documentation

eliasnogueira.com

api documentation

- We can access the documentation in the same host and port as the application, but the rest of the URL may vary.

- Always check the correct URL to access the documentation

- The application documentation can be accessed at **http://localhost:8088/swagger-ui/index.html**

eliasnogueira.com

[Setup] setup the project

Please, access **1. Setup - Local Environment** and follow its steps

api documentation

- The **endpoints (path)** are composed of the HTTP methods, path, parameters, body, statuses, and response body related to it

- The **models** are the objects as request or response body, where we can know the attribute names and value types
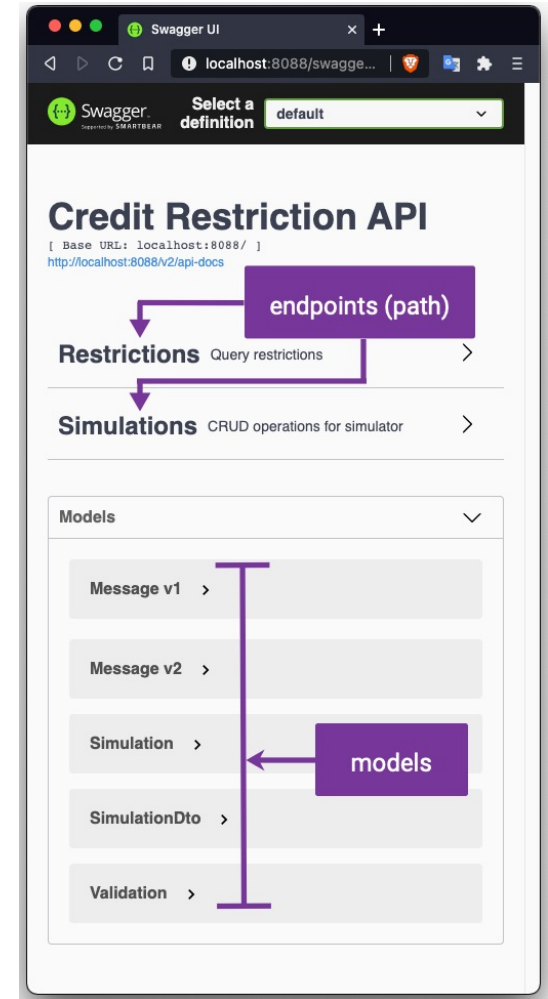
api documentation

- When you open an endpoint (path), like the **Simulations**, you can see all the HTTP Methods and paths.

eliasnogueira.com

api documentation

- Inside a path, you can see
  - HTTP Method
  - URL
  - Parameters
  - Content-type

api documentation

- and the API responses

Responses   Response content type   */*

Code	Description
200	Simulation updated successfully **Example Value** | Model ```{ "amount": 1200, "cpf": 9709323014, "email": "john.doe@gmail.com", "installments": 3, "insurance": true, "name": "John Doe" }```
404	Simulation not found
409	CPF already exists

[Setup] Project and libraries

Please, access **1. Setup - Project** and follow its steps.

REST Assured

REST Assured

http://rest-assured.io

Java DSL for simplifying testing of REST based services.

It has an intuitive code syntax to create an automated test.

Let's express it in a natural language:

```
Given I have a parameter to send

When I send a request using an HTTP Method an URL

Then I can validate the status code and response body
```

NOTE: this is not a BDD approach and I have no intention to apply it

REST Assured

http://rest-assured.io

REST Assured uses the same keywords given, when, then to express it

```java
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

class RestAssuredExampleTest {

    @Test
    void welcomeTest() {
        given().
            param("name", "Elias").
        when().
            post("/register").
        then().
            statusCode(200).
            body("message", is("Hello Elias"));
    }
}
```

eliasnogueira.com

REST Assured

http://rest-assured.io

REST Assured uses the same keywords given, when, then to express it

```java
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

class RestAssuredExampleTest {

    @Test
    void welcomeTest() {
        given().
            param("name", "Elias").
        when().
            post("/register").
        then().
            statusCode(200).
            body("message", is("Hello Elias"));
    }

}
```

import libraries

eliasnogueira.com

REST Assured

http://rest-assured.io

REST Assured uses the same keywords given, when, then to express it

```java
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

class RestA

    @Test
    void welcomeTest() {
        given().
            param("name", "Elias").
        when().
            post("/register").
        then().
            statusCode(200).
            body("message", is("Hello Elias"));
    }

}
```

test method and annotation from JUnit 5

eliasnogueira.com

REST Assured

http://rest-assured.io

REST Assured uses the same keywords given, when, then to express it

```java
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

class RestAssuredExampleTest {

    @Test
    void welcomeTest() {
        given().
            param("name", "Elias").
        when().
            post("/register").
        then().
            statusCode(200).
            body("message", is("Hello Elias"));
    }

}
```

request pre-condition

eliasnogueira.com

REST Assured

http://rest-assured.io

REST Assured uses the same keywords given, when, then to express it

```java
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

class RestAssuredExampleTest {

        param("name", "Elias").
    when().
        post("/register").
    then().
        statusCode(200).
        body("message", is("Hello Elias"));
    }

}
```

action (request)

eliasnogueira.com

REST Assured

http://rest-assured.io

REST Assured uses the same keywords given, when, then to express it

```java
import static io.restassured.RestAssured.*;
import static org.hamcrest.Matchers.*;

class RestAssuredExampleTest {

    @Test
    void welcomeTest() {
        given().

            ").

        st("/register").

    then().
        statusCode(200).
        body("message", is("Hello Elias"));
    }
}
```

assert the response body

eliasnogueira.com

GET requests

GET using path parameters

Commonly, when we need to find a resource, we use unique values as path parameters. This is the case of the `GET /api/v1/restrictions/{cpf}`

GET using path parameters

We could send the path parameter directly in the code request, like in the example below, but we won't because it will increase the code maintainability.

```java
@Test
void getRequestExample() {

    when().
        get("/restrictions/1234567890").

    // code ignored
}
```

GET using path parameters

REST Assured has two ways to help us using the path parameter:

Unnamed parameters
In the requests, directly in the HTTP methods usage.

```
get("/restrictions/${cpf}", "66414919004")
```

Named parameters
Provide a specific method in the pre-condition.

```
given().
    pathParam("cpf", "66414919004").
when().
    get("/restrictions/${cpf}")
```

GET using path parameters

The request must be the same as the endpoint described.

REST Assured will try to find the parameter name in brackets in the request and will be replaced by its value.

```
given().
    pathParam("cpf", "66414919004").
when().
    get("/restrictions/${cpf}").
```

2. the parameter name will be replaced by its value

1. the parameter name must match in the pre-condition and action
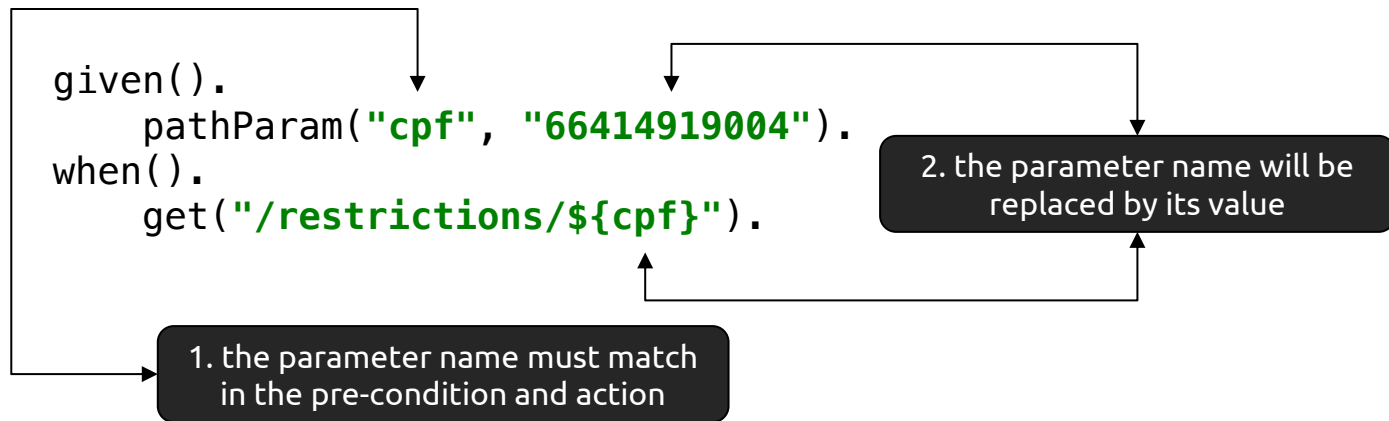
[Lab] BaseTest and first test

Please, access **2. REST Assured Basics -> Lab 1** and follow its steps.

assert the response body

assert the response body

Most of the time[1] the API requests a Response Body, which is the information returned after the request.

The OpenAPI specification will show all the status codes that the request can return, and the response body example associated.

[1] 1xx, 201, 204 and 205 usually do not have a response body returned

assert the response body

The **GET /api/v1/restrictions/{cpf}**
when a restriction is found, returns:

- ○ HTTP 200

- ○ response body with only one
  attribute: `message`

assert the response body

REST Assured help us to validate the response body in the assert step using the **then()** keyword.

This keyword have the **body()** method where we can assert any value from the response body, and it has two parameters:

- Attribute name, which must match with the one in the response
- Hamcrest matcher

assert the response body

```
@Test
void shouldReturnRestriction() {
    given()
        .pathParam("cpf", "62648716050")
    .when()
        .get("/restrictions/{cpf}")
    .then()
        .st

        .body("message", CoreMatchers.is("CPF 62648716050 has a restriction"));
}
```

attribute from the response body

assertion method is() from Hamcrest
to assert the value returned

[Lab] Tests with assertions

Please, access **2. REST Assured Basics -> Lab 2** and follow its steps.

GET all records

The **GET /api/v1/simulations** will retrieve all the data when there's no search term applied. When we need to verify this:

- ○ There're no pre-conditions (`given`)

- ○ The when part (request) is:
  - ■ The use of HTTP Method `GET`
  - ■ The URL `/simulations`

- ○ The then part (response and our validation) is:
  - ■ Status Code `200`
  - ■ Response body as the same as the Example Value from the OpenAPI

GET all records

Response body from the
GET /api/v1/simulations:

- array of objects
- two objects returned

first object

second object

Code	Details
200	**Response body**

```
[
  {
    "id": 1,
    "name": "Tom",
    "cpf": "66414919004",
    "email": "tom@gmail.com",
    "amount": 11000,
    "installments": 3,
    "insurance": true
  },
  {
    "id": 2,
    "name": "John",
    "cpf": "17822386034",
    "email": "john@gmail.com",
    "amount": 20000,
    "installments": 5,
    "insurance": false
  }
]
```

Response headers

```
connection: keep-alive
content-type: application/json
date: Sun,15 Jan 2023 18:22:07 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

GET all records

We already know that we can use the body() method to assert one attribute from the response body, and the return from the Simulations API has several:

```java
@Test
void getAllSimulations() {
// code ignored

// example for the first object returned
then().
    statusCode(200).
    body("cpf", CoreMatchers.equalTo("66414919004")).
    body("name", CoreMatchers.equalTo("Tom")).
    body("email", CoreMatchers.equalTo("tom@gmail.com")).
    body("amount", CoreMatchers.equalTo(11000f)).
    body("installments", CoreMatchers.equalTo(3)).
    body("insurance", CoreMatchers.equalTo(true))
}
```

GET all records

We already know that we can use the body() method to assert one attribute from the response body, and the return from the Simulations API has several:

```java
@Test
void getAllSimulations() {
// code ignored

// example for the first object returned
then().
    statusCode(200).
    body("cpf", CoreMatchers.equalTo("66414919004")).
    body("name", CoreMatchers.equalTo("Tom")).
    body("email", CoreMatchers.equalTo("tom@gmail.com")).
    body("amount", CoreMatchers.equalTo(11000f)).
    body("installments", CoreMatchers.equalTo(3)).
    body("insurance", CoreMatchers.equalTo(true))
}
```

⚠️

This code won't work as we need to specify the object position in the array

GET all records

We must specify the object position when we want to assert all data.
As it is an array, the first position is zero (0)!

```
[
  {
    "id": 1,
    "name": "Tom",
    "cpf": "66414919004",
    "email": "tom@gmail.com",
    "amount": 11000,
    "installments": 3,
    "insurance": true
  },
  {
    "id": 2,
    "name": "John",
    "cpf": "17822386034",
    "email": "john@gmail.com",
    "amount": 20000,
    "installments": 5,
    "insurance": false
  }
]
```

position 1 [0] →

```
body("[0].cpf", equalTo("66414919004")).
body("[0].name", equalTo("Tom")).
body("[0].email", equalTo("tom@gmail.com")).
body("[0].amount", equalTo(11000f)).
body("[0].installments", equalTo(3)).
body("[0].insurance", equalTo(true)).
```

position 2 [1] →

```
body("[1].cpf", equalTo("17822386034")).
body("[1].name", equalTo("John")).
body("[1].email", equalTo("john@gmail.com")).
body("[1].amount", equalTo(20000f)).
body("[1].installments", equalTo(5)).
body("[1].insurance", equalTo(false));
```

eliasnogueira.com

[Lab] Tests with assertions

Please, access **2. REST Assured Basics -> Lab 3** and follow its steps for the items 1 and 2.

GET all records

Alternatively to develop reliable tests, it's a good practice to assert the size of the return and elements (data) you know what will be there.

Later we will learn about better assertions, but for now, using Hamcrest you can simply check the array size returned.

The keyword **$** is from JSON Path which means the root object or array.

```java
@Test
void shouldRetrieveAllSimulationsCheckingSize() {
// code ignored

// example for the first object returned
then().
    statusCode(200).
    body("$", Matchers.hasSize(2);
}
```

[Lab] Tests with assertions

Please, access **2. REST Assured Basics -> Lab 3** and follow its steps for the item 3.

hard vs soft assertions

There are two ways to assert the response body values using body:

Hard assertion
Composed by the usage of multiple **body()** methods where the test stops in an assertion error, ignoring the execution of the other **body()** methods.

Soft Assertion
Composed by one single **body()** method and multiples values and matches where all assertions will be performed and the execution stops when the test finishes.
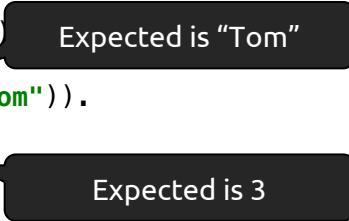
hard vs soft assertions

Hard Assertion

Let's assume that the following test has two validation errors: one in the name and another in the `installments` attribute

```java
@Test
void getAllSimulations() {
// code ignored

// example for the first object returned
then().
    statusCode(200).
    body("cpf", CoreMatchers.equalTo("66414919004")).
    body("name", CoreMatchers.equalTo("Unknown")).   [Expected is "Tom"]
    body("email", CoreMatchers.equalTo("tom@gmail.com")).
    body("amount", CoreMatchers.equalTo(11000f)).
    body("installments", CoreMatchers.equalTo(0)).   [Expected is 3]
    body("insurance", CoreMatchers.equalTo(true))
}
```

hard vs soft assertions

Hard Assertion

The test will fail during the assertion in the name attribute, stop and show the following output:

```
java.lang.AssertionError: 1 expectation failed.
JSON path [0].name doesn't match.
Expected: is "Unknown"
  Actual: Tom
```

hard vs soft assertions

Soft Assertion

Only one **body()** is used where we have an attribute name and its matches as a list.

```java
@Test
void shouldRetrieveAllSimulations() {
    when()
        .get("/simulations/")
    .then()
        .statusCode(HttpStatus.SC_OK)
        .body(
            "[0].id", CoreMatchers.notNullValue(),
            "[0].name", CoreMatchers.is("Unknown"),
            "[0].cpf", CoreMatchers.is("66414919004"),
            "[0].email", CoreMatchers.is("tom@gmail.com"),
            "[0].amount", CoreMatchers.is(new BigDecimal("11000.00")),
            "[0].installments", CoreMatchers.is(0),
            "[0].insurance", CoreMatchers.is(true)
        );
}
```

hard vs soft assertions

Soft Assertion

Only one **body()** is used where we have an attribute name and its matches as a list.

```java
@Test
void shouldRetrieveAllSimulations() {
    when()
        .get("/simulations/")
    .then()
        .statusCode(HttpStatus.SC_OK)
        .body(
            "[0].id", CoreMatchers.notNullValue(),
            "[0].name", CoreMatchers.is("Unknown"),        Expected is "Tom"
            "[0].cpf", CoreMatchers.is("66414919004"),
            "[0].email", CoreMatchers.is("tom@gmail.com"),
            "[0].amount", CoreMatchers.is(new BigDecimal("11000.00")),
            "[0].installments", CoreMatchers.is(0),         Expected is 3
            "[0].insurance", CoreMatchers.is(true)
        );
}
```

eliasnogueira.com

hard vs soft assertions

Soft Assertion

The test will fail during the assertion in the name attribute, but won't stop until all assertions are done. We will get the following output:

```
java.lang.AssertionError: 2 expectations failed.
JSON path [0].name doesn't match.
Expected: is "Unknown"
  Actual: Tom

JSON path [0].installments doesn't match.
Expected: is <0>
  Actual: <3>
```

[Lab] Tests with assertions

Please, access **2. REST Assured Basics -> Lab 3** and follow its steps for the item 4.

delete a record

delete a record

We normally send a delete using the **DELETE** HTTP method sending the key to the object deletion.

The successful return is an empty return with 204 status.

We have the following for the
DELETE /simulations/{cpf}
- parameter named `cpf` as `path`

delete a record

To remove a record, we need send a valid key (**cpf** in this case) which will be deleted from the existing data.

NOTE

The return of the successful data must be always empty and have the **204** status code.

delete a record

To delete a record using Rest-Assured we need to use the method **delete()** and all the knowledge from the previous exercises:

- getting or creating a simulation object
- add as precondition
    - existing **cpf** as **path parameter**
- validate only the status code

delete a record

```java
@Test
void deleteExistingSimu
    given().
        pathParam("cpf", "66414919004").
    when().
        delete("/simulations/{cpf}").
    then().
        statusCode(204);
}
```

existing cpf in the path parameter

delete a record

```java
@Test
void deleteExistingSimulation() {

    path("cpf", "00142536007").
    when().
        delete("/simulations/{cpf}").
    then().
        statusCode(204);
}
```

delete request with the path parameter

delete a record

```java
@Test
void deleteExistingSimulation() {
    given().
                                    4919004").
                                                    validating only the status code
            de   /simulations/{cpf}").
    then().
        statusCode(204);
}
```

validating only the status code

[Lab] Tests with assertions

Please, access **2. REST Assured Basics -> Lab 4** and follow its steps for the item 1.
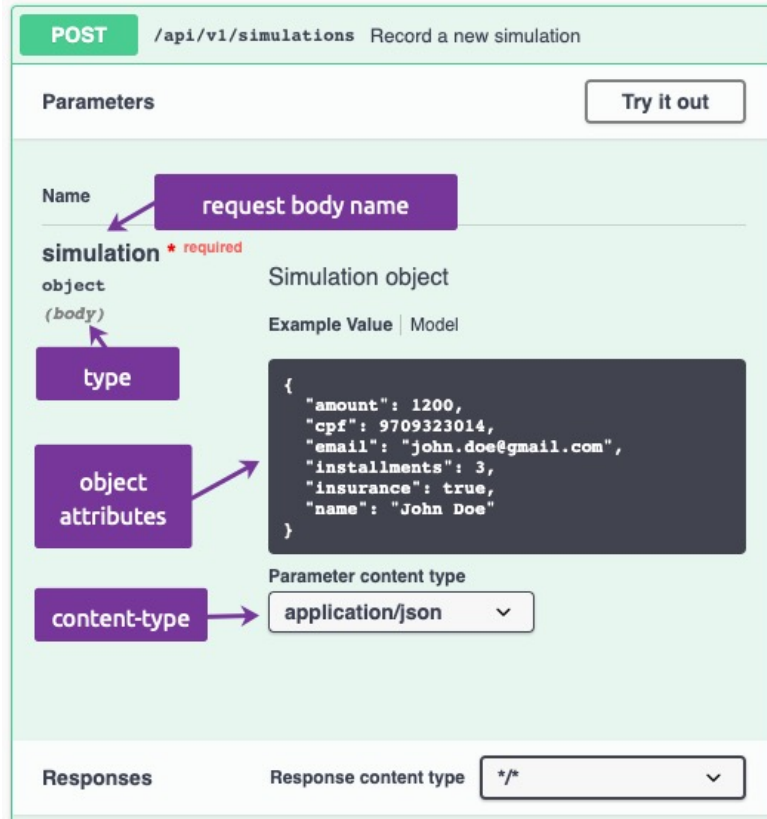
create a record

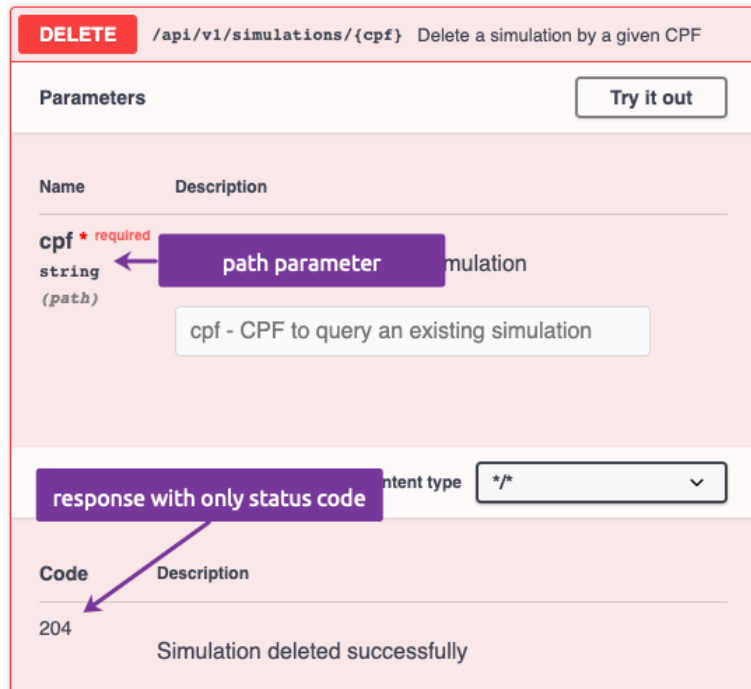# create a record

To post a record (create) we need to know 2 new information's:

- object/request body
- content-type

We have the following for the
POST /api/v1/simulations

- parameter named **simulations**
- parameter type as body
- attributes the body should have
- the **content-type**

create a record

In order to post a record, we need to create a JSON object (because the content-type is `application/json`).

According to the Example Value the JSON object must contain the following attributes (the values are an illustration):

```json
{
    "name": "John Doe"
    "cpf": 9709323014,
    "email": "john.doe@gmail.com",
    "amount": 1200,
    "installments": 3,
    "insurance": true
}
```

create a record

There are three ways to create a body:

🚫 **String concatenation**

It consists in a String object concatenating all the necessary data for the request body

Java 8 to 14: regular String concatenation

```java
String requestBody = "{" +
        "\"name\": \"John Doe\" " +
        "\"cpf\": 9709323014," +
        "\"email\": \"john.doe@gmail.com\"," +
        "\"amount\": 1200," +
        "\" installments\": 3," +
        "\"insurance\": true,}\n";
}
```

Java 15+: text block

```java
String requestBody = """
        {
            "name": "John Doe",
            "cpf": 9709323014,
            "email": "john.doe@gmail.com",
            "amount": 1200,
            "installments": 3,
            "insurance": true
        }""";
}
```

eliasnogueira.com

create a record

There are three ways to create a body:

🚫 **HashMap or JSON Object**

It consists in creating a `HashMap` object where the attribute is the key, and the value is the value or a `JSONObject` (`org.json.simple`).

HashMap

```java
var request = new HashMap<>();
request.put("name", "John Doe");
request.put("cpf", "9709323014");
request.put("email", "john.doe@gmail.com");
request.put("amount", "1200");
request.put("installments", "3");
request.put("insurance", "true");
```

JSONObject

```java
var request = new JSONObject<>();
request.put("name", "John Doe");
request.put("cpf", "9709323014");
request.put("email", "john.doe@gmail.com");
request.put("amount", "1200");
request.put("installments", "3");
request.put("insurance", "true");
```

create a record

There are three ways to create a body:

👍 **Object serialization**

It consists in creating a custom object (Model/POJO/DTO) to handle the attributes and data.

This is a feature from REST Assured called [Object Mapping](#) for serialization (request body) and deserialization (response body), supported by the following libraries:

- [default] Jackson 2 (Faster Jackson Databind)
- Jackson (databind)
- Gson
- Johnzon
- JSON-B using Eclipse Yasson

create a record

REST Assured will transform, using the **Object Serialization** approach, the Java object into a JSON object with the same attribute names.

```java
public class Simulation {

    private String name;
    private String cpf;
    private String email;
    private BigDecimal amount;
    private int installments;
    private Boolean insurance;
}
```

Object serialization

```json
{
    "name": "John Doe"
    "cpf": 9709323014,
    "email": "john.doe@gmail.com",
    "amount": 1200,
    "installments": 3,
    "insurance": true,
}
```

eliasnogueira.com

create a record

First, we need to create a Java object with getters, setters, and constructors.

To simplify this creation, we will use the **Lombok** library to remove the code boilerplate using its annotations:

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class Simulation {

    private String name;
    private String cpf;
    private String email;
    private BigDecimal amount;
    private int installments;
    private Boolean insurance;
}
```

[Lab] Object Mapping

Please, access **3. Object Mapping -> Lab 1** and follow its steps for the item 1.

create a record

As we have the **Simulation** class, that will be Serialized by REST Assured and be transformed into a JSON object, we need to add data into it.

The best way to create it is using the **Builder approach** (we added the **@Build** annotation from Lombok in the class). The usage will be like this:

```java
Simulation simulation = Simulation.builder().
        name("Elias").cpf("123456789").email("elias@eliasnogueira.com").
        amount(new BigDecimal(3000)).installments(5).insurance(true).build();
```

create a record

The request body, which is the **Simulation** object must be specified along with its content type. The recommended way is to add the following methos as pre-condition (**given()**):

- **body()**, adding the object as parameter
- **contentType()**, using the **ContentType** enum from REST Assured

```
given().
    body(simulation).
    contentType(ContentType.JSON).
```

create a record

```java
Simulation simulation = Simulation.builder().
        name("Elias").cpf("123456789").email("elias@eliasnogueira.com").
        amount(new BigDecimal(3000)).installments(5).insurance(true).build();

given().
    body(simulation).
    contentType(ContentType.JSON).
when().
    post("/simulations").
```

eliasnogueira.com

create a record

```java
Simulation simulation = Simulation.builder().
    ...email("elias@eliasnogueira.com").
    ...installments(5).insurance(true).build();

given().
    body(simulation).
    contentType(ContentType.JSON).
when().
    post("/simulations").
```

response body object being used into body as a precondition

create a record

```java
Simulation simulation = Simulation.builder().
        name("Elias").cpf("123456789").email("elias@eliasnogueira.com").
        amount(new BigDecimal(3000)).installments(5).insurance(true).build();

given().
    body(simulation).
    contentType(ContentType.JSON).
when().
    post("/simulations").
```

content-type set as JSON

create a record

```java
Simulation simulation = Simulation.builder().
        name("Elias").cpf("123456789").email("elias@eliasnogueira.com").
        amount(new BigDecimal(3000)).installments(5).insurance(true).build();

        contentType(ContentType.JSON).
when().
    post("/simulations");
```

sending the post request

eliasnogueira.com

[Lab] Object Mapping

Please, access **3. Object Mapping -> Lab 1** and follow its steps for the item 2.

create a record

Note that the API shows the response body with the **201** status code and a Header named **location** containing the URL to access the simulation created.

Code	Description
201	Simulation created successfully

Headers:

Name	Description	Type
Location	URI to query the created simulation	string

create a record

Did you realize that we didn't add the validation part?

First, let's look at the response:

Code	Description
201	Simulation created successfully

Headers:

Name	Description	Type
Location	URI to query the created simulation	string

It will return a status code **201** and the `Location` attribute into the header as the `URI` (complete URL) to use if we need to find it (send a get request)

create a record

Example

If you send a post request with **cpf** data as **987654321** the **Location** attribute will have the following

```
connection: keep-alive
content-length: 0
date: Sat,21 Jan 2023 16:26:47 GM
keep-alive: timeout=60
location: http://localhost:8088/api/v1/simulations/987654321
```

To validate the header, we will use the **header()** method instead of **body()** adding into the parameters the attribute name and the assertion method.

create a record

Notice that we are, in the **header()** method

- adding **Location**, that is the attribute name

- Using the **containsString** method to verify it the return contains…

- And using the **simulation.getCpf()** to verify if the data is present in the **URI**

```
then().
    statusCode(201).
    header("Location", containsString(simulation.getCpf()));
```

[Lab] Object Mapping

Please, access **3. Object Mapping -> Lab 1** and follow its steps for the item 3.

update a record

update a record

We normally send an update using the **PUT** HTTP method sending two information's:

- the object key
- object/request body + content-type

We have the following for the **PUT /simulations/{cpf}**

- parameter named `cpf` as `path`
- parameter `simulation` as `body`
- attributes the body should have
- the `content-type`

eliasnogueira.com

update a record

In order to update a record, we need to send a valid key (**cpf** in this case) and create a JSON object with all the data to update.

The simulation object is the same as the previous exercise (**post**).

NOTE

Normally the PUT must send the whole request object to update, even though the update is only in one field. The partial update is made by PATCH HTTP method, where this workshop does not cover.

update a record

The API returns the **200** status code and the updated response body when the request is successful.

Responses

Code   Description

200
            Simulations found

            Example Value | Model

            [
              {
                "amount": 1200,
                "cpf": 9709323014,
                "email": "john.doe@gmail.com",
                "installments": 3,
                "insurance": true,
                "name": "John Doe"
              }
            ]

eliasnogueira.com

update a record

To update a record using Rest-Assured we need to use the method **put()**
and all the knowledge from the previous exercises:

- getting or creating a simulation object
- add as precondition
    - existing **cpf** as **path parameter**
    - the **simulation** object in the **response body**
    - the **content-type** as **JSON**
- validate the status code and the response body

update a record

```java
Simulation simulation = Simulation.builder().
        name("NEW NAME").cpf("66414919004").email("new_name@gmail.com").
        amount(new BigDecimal(3000)).installments(4).insurance(false).build();
given().
    pathParam("cpf", "66414...
    body(simulation).
    contentType(ContentType...
when().
    put("/simulations/{cpf}").
then().
    statusCode(200).
    body("cpf", equalTo("66414919004")).
    body("name", equalTo("NEW NAME")).
    body("email", equalTo("new_name@gmail.com")).
    body("amount", equalTo(3000)).
    body("installments", equalTo(4)).
    body("insurance", equalTo(false));
```

simulation object created

update a record

```java
Simulation simulation = Simulation.builder().
        name("NEW NAME").cpf("66414919004").email("new_name@gmail.com").
        amount(new BigDecimal(3000)).installments(4).insurance(false).build();
given().
    pathParam("cpf", "66414919004").
    body(simulation).
    contentType(ContentType.JSON).
when().
    put(                                        ns/{cpf}").
then().
    stat
    body
    body("name", equalTo("NEW NAME")).
    body("email", equalTo("new_name@gmail.com")).
    body("amount", equalTo(3000)).
    body("installments", equalTo(4)).
    body("insurance", equalTo(false));
```

existing cpf as path param and the whole
simulation object as a request body

update a record

```java
Simulation simulation = Simulation.builder().
        name("NEW NAME").cpf("66414919004").email("new_name@gmail.com").
        amount(new BigDecimal(3000)).installments(4).insurance(false).build();
given().
    pathParam("cpf", "66414919004").
    body(simulation).
    contentType(JSON).
when().
    put("/simulations/{cpf}").
then().
    statusCode(200).
    body("cpf", equalTo("66414919004")).
    body("name", equalTo("NEW NAME")).
    body("email", equalTo("new_name@gmail.com")).
    body("amount", equalTo(3000)).
    body("installments", equalTo(4)).
    body("insurance", equalTo(false));
```

put request with path parameter

eliasnogueira.com

update a record

```java
Simulation simulation = Simulation.builder().
        name("NEW NAME").cpf("66414919004").email("new_name@gmail.com").
        amount(new BigDecimal(3000)).installments(4).insurance(false).build();
given().
    pathParam("cpf", "66414919004").
    body(simulation).
    con
when().
    put("http://...:8080/api/v1/simulations/{cpf}").
then().
    statusCode(200).
    body("cpf", equalTo("66414919004")).
    body("name", equalTo("NEW NAME")).
    body("email", equalTo("new_name@gmail.com")).
    body("amount", equalTo(3000)).
    body("installments", equalTo(4)).
    body("insurance", equalTo(false));
```

status code and response body validation

eliasnogueira.com

[Lab] Object Mapping

Please, access **3. Object Mapping -> Lab 2** and follow its steps for the item 1.

update a record

👍 **Object deserialization**

It consists in matching the attributes present in the response body into a class (Model/POJO/DTO) to have access to all returned data.

To achieve it we do need:
- a class containing the attributes that match the response body
- tell REST Assured about the deserialization

create a record

REST Assured will transform, using the **Object Serialization** approach, the response body (JSON object) into a Java object.

```java
public class Simulation {

    private String name;
    private String cpf;
    private String email;
    private BigDecimal amount;
    private int installments;
    private Boolean insurance;
}
```

Object deserialization

```json
{
    "name": "John Doe"
    "cpf": 9709323014,
    "email": "john.doe@gmail.com",
    "amount": 1200,
    "installments": 3,
    "insurance": true,
}
```

eliasnogueira.com

create a record

Now the Simulation object can access the values returned from the response body, as the automatic match was done by REST Assured.

```
simulation.getName(); // will return "John Doe" data
simulation.getCpf(); // will return "9709323014" data
simulation.getEmail(); // will return "john.doe@gmail.com" data
simulation.getAmount(); // will return "1200" data
simulation.getInstallments(); // will return "3" data
simulation.getInstallments(); // will return "true" data
```

```json
{
    "name": "John Doe"
    "cpf": 9709323014,
    "email": "john.doe@gmail.com",
    "amount": 1200,
    "installments": 3,
    "insurance": true,
}
```

create a record

In order to do so, we need to extract the response body, associating it with an existing class where the attributes will match.

To extract the data, instead of using the **body()** method to assert its data, we need to use the **extract().as()**, where the **as()** method must have the class that will be automatically deserialized.

create a record

```java
@Test
void shouldUpdateExistingSimulation() {
    String existingCpf = "17822386034";

    var simulation = Simulation.builder().name("Elias").cpf("17822386034")
        .email("elias@eliasnogueira.com").amount(new BigDecimal("3000.00"))
        .installments(5).insurance(true).build();

    var simulation simulationUpdated =
        given()
            .pathParam("cpf", existingCpf)
            .body(simulation)
            .contentType(ContentType.JSON)
        .when()
            .put("/simulations/{cpf}")
        .then()
            .statusCode(HttpStatus.SC_OK)
            .extract().as(Simulation.class);
}
```

create a record

```java
@Test
void shouldUpdateE...
    String existing...

    var simulation = Simulation.builder().name("Elias").cpf("17822386034")
        .email("elias@eliasnogueira.com").amount(new BigDecimal("3000.00"))
        .installments(5).insurance(true).build();

    var simulation simulationUpdated =
        given()
            .pathParam("cpf", existingCpf)
            .body(simulation)
            .contentType(ContentType.JSON)
        .when()
            .put("/simulations/{cpf}")
        .then()
            .statusCode(HttpStatus.SC_OK)
            .extract().as(Simulation.class);
}
```

simulation object with updated data

create a record

```java
@Test
void shouldUpdateExistingSimulation() {
    String existingCpf = "17822386034";

    var simu                                     Elias").cpf("17822386034")
        .ema                                     (new BigDecimal("3000.00"))
        .ins                                     .build();

    var simulation simulationUpdated =
        given()
            .pathParam("cpf", existingCpf)
            .body(simulation)
            .contentType(ContentType.JSON)
        .when()
            .put("/simulations/{cpf}")
        .then()
            .statusCode(HttpStatus.SC_OK)
            .extract().as(Simulation.class);
}
```

new simulation attribute which will have the returned data

eliasnogueira.com

create a record

```java
@Test
void shouldUpdateExistingSimulation() {
    String existingCpf = "17822386034";

    var simulation = Simulation.builder().name("Elias").cpf("17822386034")
        .email("elias@eliasnogueira.com").amount(new BigDecimal("3000.00"))
        .installments(5).insurance(true).build();

    var simulation simulationUpdated =
        given()
            .pathParam("cpf", existingCpf)
            .body(simulation)
        .when()
        .then()
            .statusCode(HttpStatus.SC_OK)
            .extract().as(Simulation.class);
}
```

extraction of the response body matching the Simulation class

create a record

Assertion

As we have extracted the data a new approach for the assertion must be used because. We could still use the **body()** method to assert its data, but it wouldn't make sense as we have extracted it.

We will use the AssertJ library to assert its data, which is a modern assertion library that provides different assertion ways.
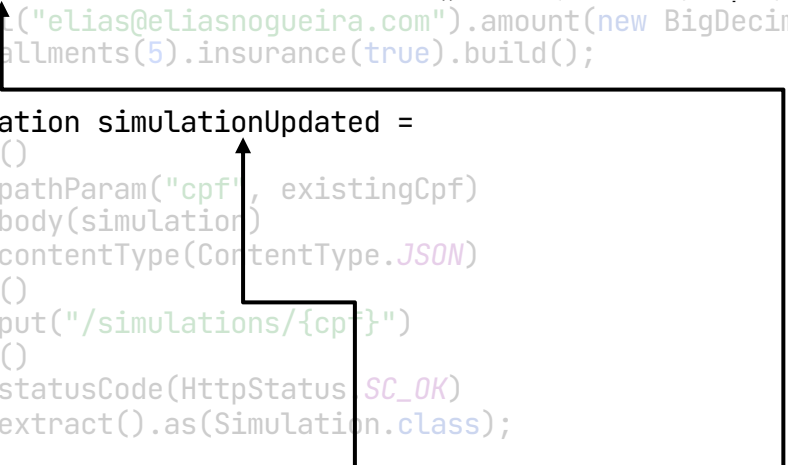
create a record

```java
@Test
void shouldUpdateExistingSimulation() {
    String existingCpf = "17822386034";

    var simulation = Simulation.builder().name("Elias").cpf("17822386034")
            .email("elias@eliasnogueira.com").amount(new BigDecimal("3000.00"))
            .installments(5).insurance(true).build();

    var simulation simulationUpdated =
            given()
                .pathParam("cpf", existingCpf)
                .body(simulation)
                .contentType(ContentType.JSON)
            .when()
                .put("/simulations/{cpf}")
            .then()
                .statusCode(HttpStatus.SC_OK)
                .extract().as(Simulation.class);

    Assertions.assertThat(simulationUpdated).isEqualTo(simulation);
}
```

assertion done by comparing both objects

eliasnogueira.com

[Lab] Object Mapping

Please, access **3. Object Mapping -> Lab 3** and follow its steps for the item 1.

Logging

logging

To help you understand the complete request and response details, REST Assured provides a way to log the details to help you correct to manage the expectations in the console.

There are two types of logging:

- Request logging
- Response logging

logging

Request logging

Will print, in the output, all the request information as parameters, body, headers, cookies, method, and path, or we can log all of them.

It must be placed after the `given()` method.

```
given().log().all(). .. // Log all of below
given().log().params(). .. // Log only the parameters of the request
given().log().body(). .. // Log only the request body
given().log().headers(). .. // Log only the request headers
given().log().cookies(). .. // Log only the request cookies
given().log().method(). .. // Log only the request method
given().log().path(). .. // Log only the request path
```

logging

Response logging

Will print, in the output, the response information as the status, body, headers, and cookies.

It must be placed after the `then()` method.

```
then().log().all(). .. // All of below
then().log().statusLine(). .. // Only log the status line
then().log().headers(). .. // Only log the response headers
then().log().cookies(). .. // Only log the response cookies
```

logging

Only when validation fails

There's also a method present in the request or response that will show it only when the validation fails called **ifValidationFails()**.

But the best way to log it is on the REST Assured global configuration:

```
RestAssured.enableLoggingOfRequestAndResponseIfValidationFails();
```

We have already added it to the **BaseApiConfigurationClass**

logging

This is the example of the request and response log for the **POST /simulations**

```
Request method:   POST
Request URI:      http://localhost:8088/api/v1/simulations/
Proxy:            <none>
Request params:   <none>
Query params:     <none>
Form params:      <none>
Path params:      <none>
Headers:          Accept=*/*
                  Content-Type=application/json
Cookies:          <none>
Multiparts:       <none>
Body:
{
    "name": "Elias",
    "cpf": "123456789",
    "email": "elias@eliasnogueira.com",
    "amount": 3000,
    "installments": 5,
    "insurance": true
}
HTTP/1.1 201
Location: http://localhost:8088/api/v1/simulations/123456789
Content-Length: 0
Date: Tue, 24 Jan 2023 19:43:19 GMT
Keep-Alive: timeout=60
Connection: keep-alive
```

logging

This is the example of the request and response log for the **GET /restrictions/{cfp}**

```
Request method: GET
Request URI:    http://localhost:8088/api/v1/restrictions/62648716050
Proxy:          <none>
Request params: <none>
Query params:   <none>
Form params:    <none>
Path params:    cpf=62648716050
Headers:        Accept=*/*
Cookies:        <none>
Multiparts:     <none>
Body:           <none>


HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Tue, 24 Jan 2023 20:16:10 GMT
Keep-Alive: timeout=60
Connection: keep-alive

{
    "message": "CPF 62648716050 has a restriction"
}
```

filters

Filter allows us to inspect and modify a request and response information before showing it. There are 3 provided filters:

- **RequestLoggingFilter**: print the request details

- **ResponseLoggingFilter**: print the response body details

- **ErrorLoggingFilter**: print the response body when an error occurs

filters

We can also implement our own filters by creating a class implementing `io.restaassured.Filter` interface.

The easiest way to log all the requests and responses during the test execution is to add the default filters to REST Assured global configuration.

ProTip: create a way, using the properties file, to enable/disable it to avoid excessive log in the console, mainly running into CI/CD

filters

One of the possible approaches it to add the filters as a global configuration in the implemented **BaseTest** class.

```
RestAssured.filters(
        new RequestLoggingFilter(),
        new ResponseLoggingFilter()
);
```

filters

One of the possible approaches it to add the filters as a global configuration in the implemented **BaseTest** class.

global REST Assured filter

```
RestAssured.filters(
        new RequestLoggingFilter(),
        new ResponseLoggingFilter()
);
```

filters

One of the possible approaches it to add the filters as a global configuration in the implemented **BaseTest** class.

add the default request
information to the filter

```java
RestAssured.filters(
        new RequestLoggingFilter(),
        new ResponseLoggingFilter()
);
```

eliasnogueira.com

filters

One of the possible approaches it to add the filters as a global configuration in the implemented **BaseTest** class.

```
RestAssured.filters(
        new RequestLoggingFilter(),
        new ResponseLoggingFilter()
);
```

add the default response information to the filter

[Lab] Logging and Filters

Please, access **4. Logging and Filters -> Lab 1** and follow its steps for the item 1.

Specification Re-use

specification re-use

It's the name given for the re-use approach for the request and response, that might be duplicated across the code.

Bot are done by to classes:

- **ResquestSpecBuild**: will carry common request items such as path and query parameters, content type, cookies, headers, and more

- **ResponseSpecBuilder**: will carry the common expectations like expected status code and body

specification re-use

Request Specification

In the current tests, we have a common thing: the path parameter informing the CPF. This is a simple example, but it can show how the Request Specification can be used.

```java
RequestSpecification requestSpecification =
        new RequestSpecBuilder().addPathParams("cpf", "1234567890").build();
```

specification re-use

Request Specification

In the current tests, we have a common thing: the path parameter informing the CPF. This is a simple example, but it can show how the Request Specification can be used.

```
RequestSpecification requestSpecification =
        new RequestSpecBuilder().addPathParams("cpf", "1234567890").build();
```
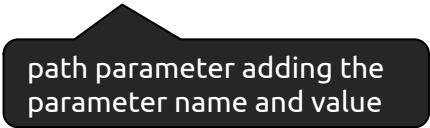
support class to add the specifications

specification re-use

Request Specification

In the current tests, we have a common thing: the path parameter informing the CPF. This is a simple example, but it can show how the Request Specification can be used.

```
RequestSpecification requestSpecification =
      new RequestSpecBuilder().addPathParams("cpf", "1234567890").build();
```
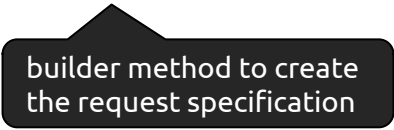
path parameter adding the parameter name and value

specification re-use

Request Specification

In the current tests, we have a common thing: the path parameter informing the CPF. This is a simple example, but it can show how the Request Specification can be used.

```
RequestSpecification requestSpecification =
        new RequestSpecBuilder().addPathParams("cpf", "1234567890").build();
```

builder method to create the request specification

specification re-use

We can add the request specification in the test class, or in any other class. The best approach would be to create it in a different class where we can have a set of shared request specifications.

Not talking about where, but about the implementation we can have a method that will have the parameter value as a parameter, making it reusable:

```java
public RequestSpecification cpfPathParameter(String cpf) {
    return new RequestSpecBuilder().addPathParams("cpf",cpf).build();
}
```

[Lab] Request and Response Specs

Please, access **5. Request and Response Specs -> Lab 1** and follow its steps for the item 1.

robust framework

robust framework

You have noticed, so far, we are using the "raw" REST Assured usage through the methods `given()`, `when()`, and `then()`.

They are a great start, but it won't scale as the API and test grow.

One of the possible approaches is to create an abstraction around the main REST Assured methods, having minimal work using the OpenAPI spec.

robust framework

We are using the Open API spec,
but the UI version of it
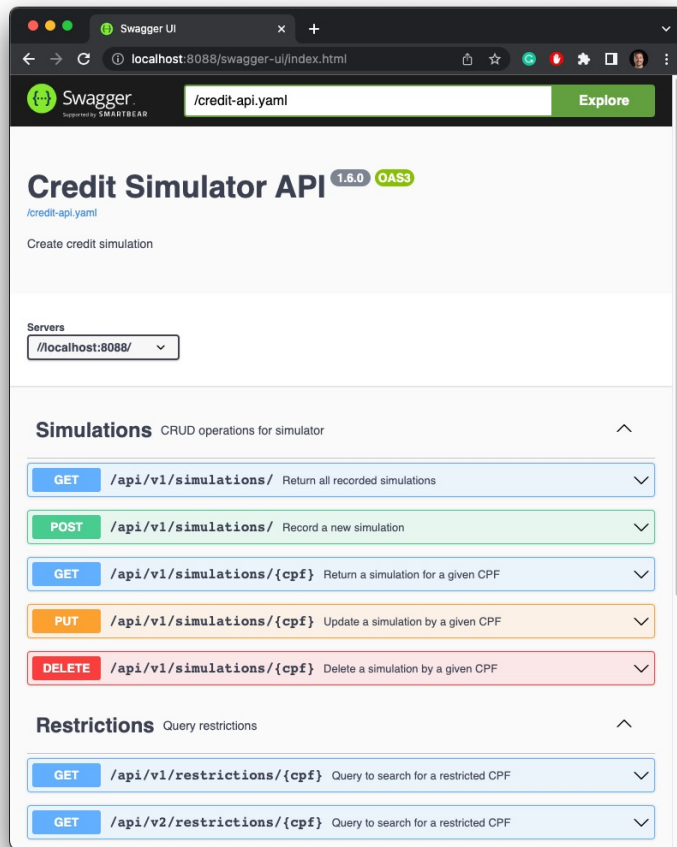
robust framework

Normally we can also have the file and use it to generate the basic structure of the request and response.

There're several tools, and we will use the:

- **`wagon-maven-plugin`**: to download the spec

- **`openapi-generator-maven-plugin`**: to generate the API client based on the spec

openapi-generator

download the spec file

The **wagon-maven-plugin** will help us to download a file to a specific directory.

In general, we will tell the plugin to:

- download a file
- describe the file location
- describe the destination

download the spec file

```xml
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>wagon-maven-plugin</artifactId>
    <version>${wagon-maven-plugin.version}</version>

    <executions>
        <execution>
            <id>download-credit-api-spec</id>
            <goals>
                <goal>download-single</goal>
            </goals>
            <phase>generate-sources</phase>
            <configuration>
                <url>
                    URL-TO-THE-FILE-INCLUDING-FILE-NAME-AND-EXTENSION
                </url>
                <toDir>${project.basedir}/target/openapiSpecs</toDir>
            </configuration>
        </execution>
    </executions>
</plugin>
```

download the spec file

```xml
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>wagon-maven-plugin</artifactId>
    <version>${wag...

    <executions>
        <execution>
            <id>download-credit-api-spec</id>
            <goals>
                <goal>download-single</goal>
            </goals>
            <phase>generate-sources</phase>
            <configuration>
                <url>
                    URL-TO-THE-FILE-INCLUDING-FILE-NAME-AND-EXTENSION
                </url>
                <toDir>${project.basedir}/target/openapiSpecs</toDir>
            </configuration>
        </execution>
    </executions>
</plugin>
```

unique id to identify the execution action, in case of multiple downloads

download the spec file

```xml
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>wagon-maven-plugin</artifactId>
    <version>${wagon-maven-plugin.version}</version>

    <executions>
        <execution>
            <id>download...</id>
            <goals>
                <goal>download-single</goal>
            </goals>
            <phase>generate-sources</phase>
            <configuration>
                <url>
                    URL-TO-THE-FILE-INCLUDING-FILE-NAME-AND-EXTENSION
                </url>
                <toDir>${project.basedir}/target/openapiSpecs</toDir>
            </configuration>
        </execution>
    </executions>
</plugin>
```

goal from the plugin

download the spec file
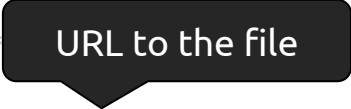
```xml
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>wagon-maven-plugin</artifactId>
    <version>${wagon-maven-plugin.version}</version>

    <executions>
        <execution>
            <id>down
            <goals>
                <goa
            </goals>
            <phase>generate-sources</phase>
            <configuration>
                <url>
                    URL-TO-THE-FILE-INCLUDING-FILE-NAME-AND-EXTENSION
                </url>
                <toDir>${project.basedir}/target/openapiSpecs</toDir>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Maven Build Lifecycle that
will trigger this action

eliasnogueira.com

download the spec file

```xml
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>wagon-maven-plugin</artifactId>
    <version>${wagon-maven-plugin.version}</version>

    <executions>
        <execution>
            <id>download-credit-api-spec</id>
            <goals>
                <goal>download-single</goal>
            </goals>
            <phase>generate-sources
            <configuration>
                <url>
                    URL-TO-THE-FILE-INCLUDING-FILE-NAME-AND-EXTENSION
                </url>
                <toDir>${project.basedir}/target/openapiSpecs</toDir>
            </configuration>
        </execution>
    </executions>
</plugin>
```

URL to the file

download the spec file

```xml
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>wagon-maven-plugin</artifactId>
    <version>${wagon-maven-plugin.version}</version>

    <executions>
        <execution>
            <id>download-credit-api-spec</id>
            <goals>
                <goal>download-single</goal>
            </goals>
            <phase>generate-sources</phase>
            <configuration>
                <url>
                    URL-TO-THE-FI          D-EXTENSION
                </url>
                <toDir>${project.basedir}/target/openapiSpecs</toDir>
            </configuration>
        </execution>
    </executions>
</plugin>
```

file internal location

[Lab] OpenAPI Generator

Please, access **6. OpenAPI Generator -> Lab 1** and follow its steps for the item 1.

generate the Client API

The **openapi-generator-maven-plugin** will help us to generate the Client API and it models based on the Open API file specification.

In general, we will tell the plugin to:

- look at a specific folder to know the spec file
- define the main, api and model packages
- use REST Assured as a support library
- set the serialization library

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <
        </goa
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

unique id in case of many generator

generate the Client API

```xml
<executions>
    <execution>
        <id>generate</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

goal to generate the code

generate the Client API

```xml
<executions>
    <execution>
        <id>ge...
        <goals
            <g...
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

Maven Lifecycle phase that will trigger the execution

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-source
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

Open API spec file location

eliasnogueira.com

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/apispec/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

the package used for the generated invoker (common) objects

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.buil                    -api.yaml
            </inputSpec>
            <invokerPackage>com.elias      ueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

the package used for the generated client api

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build...                    ...api.yaml
            </inputSpec>
            <invokerPackage>com...                          ...nvokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

the package used for the generated models

eliasnogueira.com

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>...it.invoker</invokerPackage>
            <apiPackage>...pi</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>false</generateModelTests>
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

other configurations

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateAp
            <generateMo
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

library used in the client api

generate the Client API

```xml
<executions>
    <execution>
        <id>generate-client-api-code</id>
        <goals>
            <goal>generate</goal>
        </goals>
        <phase>generate-sources</phase>
        <configuration>
            <inputSpec>
                ${project.build.directory}/openapiSpecs/credit-api.yaml
            </inputSpec>
            <invokerPackage>com.eliasnogueira.credit.invoker</invokerPackage>
            <apiPackage>com.eliasnogueira.credit.api</apiPackage>
            <modelPackage>com.eliasnogueira.credit.model</modelPackage>
            <generatorName>java</generatorName>
            <generateApiTests>false</generateApiTests>
            <generateModelTests>f
            <configOptions>
                <library>rest-assured</library>
                <serializationLibrary>jackson</serializationLibrary>
            </configOptions>
        </configuration>
    </execution>
</executions>
```

serialization library

eliasnogueira.com

[Lab] OpenAPI Generator

Please, access **6. OpenAPI Generator -> Lab 2** and follow its steps for the item 1.

architecture

Using the raw features of REST Assured for test creation might not scale as we need to solve possible duplication along its creation.

The changes in the API spec will also influence, increasing the code maintainability.

Thankfully, using abstractions and design patterns we can easily solve these problems.

architecture

Proposed architecture

Client API abstraction

Service abstraction

RestApiClientBuilder → *ApiService ← *Test

build | uses | use

*ApiClient ← *Data ← *Test
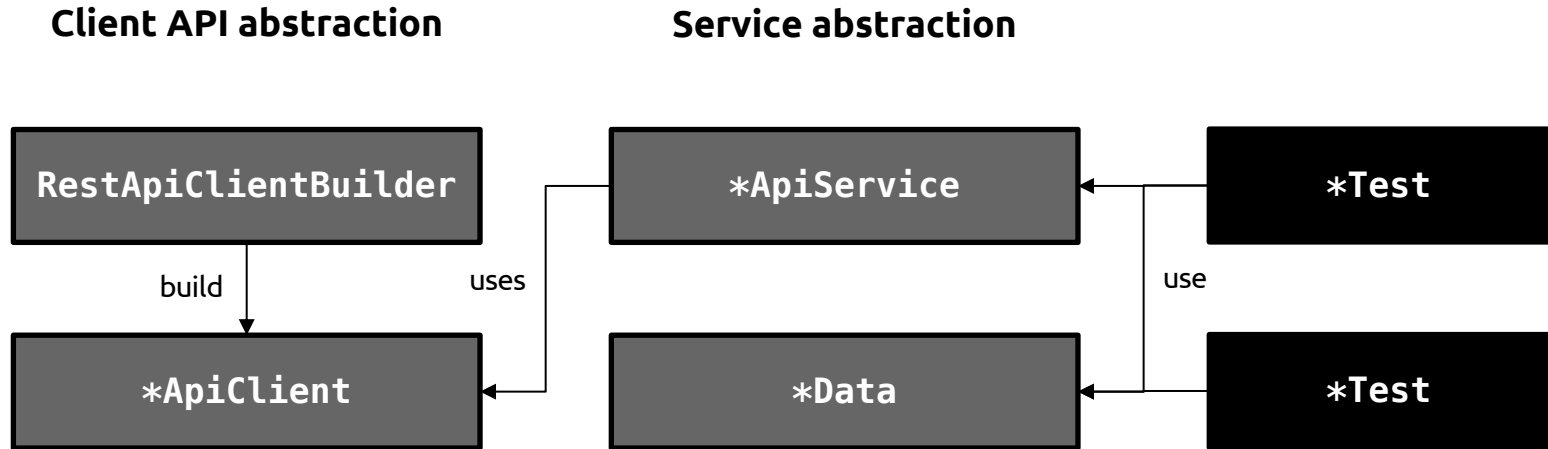
architecture

RestApiClientBuilder

Each Client API generated by the OpenAPI Generator will carry the full path to the related HTTP request.

```java
SimulationsApi.java
public static class DeleteUsingDELETEOper implements Oper {

    public static final Method REQ_METHOD = DELETE;
    public static final String REQ_URI = "/api/v1/simulations/{cpf}";

// code ignored
```

architecture

RestApiClientBuilder

The main necessity of this class is to add, to all requests, the **baseUri**, and the port. The **basePath** is not necessary as the generated client already has it.

```java
public class RestApiClientBuilder {

    public <T> T build(Function<Supplier<RequestSpecBuilder>, T> clientCreator) {
        Supplier<RequestSpecBuilder> requestSpecBuilderSupplier = () -> new RequestSpecBuilder()
                .addRequestSpecification(
                        new RequestSpecBuilder()
                                .setBaseUri("http://localhost")
                                .setPort(8088)
                                .build());

        return clientCreator.apply(requestSpecBuilderSupplier);
    }
}
```

[Lab] OpenAPI Generator

Please, access **7. Better architecture -> Lab 1** and follow its steps for the item 1.

architecture

architecture

*ApiClient

This will abstract the current Client API class generated by the OpenAPI Generator, based on REST Assured, and will use the `RestClientApiBuilder` to add the common request specification.

The generated Client Api class adds an inner class per HTTP request matching the OpenAPI spec.

architecture

```java
public class RestrictionsApi {


    public static RestrictionsApi restrictions(Supplier<RequestSpecBuilder> reqSpecSupplier) {
        return new RestrictionsApi(reqSpecSupplier);
    }


    public OneUsingGETOper oneUsingGET() {
        return new OneUsingGETOper(createReqSpec());
    }
}
```

architecture

```java
public class RestrictionsApi {

    public static RestrictionsApi restrictions(Supplier<RequestSpecBuilder> reqSpecSupplier) {
        return new RestrictionsApi(reqSpecSupplier);
    }


    public OneUsingGETOper oneUsingGET() {
        return new OneUsingGETOper(createReqSpec());
    }
}
```

Will be used by the **RestClientApiBuilder** to build the Client Api

eliasnogueira.com

architecture

```java
public class RestrictionsApi {

    pub                                            r<RequestSpecBuilder> reqSpecSupplier) {

    }

    public OneUsingGETOper oneUsingGET() {
        return new OneUsingGETOper(createReqSpec());
    }
}
```

Inner class with the HTTP method, **basePath** and requests specifics (params)

eliasnogueira.com

architecture

```java
public static class OneUsingGETOper implements Oper {

    public static final Method REQ_METHOD = GET;
    public static final String REQ_URI = "/api/v1/restrictions/{cpf}";

    @Override
    public <T> T execute(Function<Response, T> handler) {
        // magic
    }


    public static final String CPF_PATH = "cpf";


    public OneUsingGETOper cpfPath(Object cpf) {
        reqSpec.addPathParam(CPF_PATH, cpf);
        return this;
    }

}
```

architecture

```java
public static class OneUsingGETOper implements Oper {

    public static final Method REQ_METHOD = GET;
    public static final String REQ_URI = "/api/v1/restrictions/{cpf}";

    @Override
    public <T> T execute(F
        // magic
    }



    public static final String CPF_PATH = "cpf";


    public OneUsingGETOper cpfPath(Object cpf) {
        reqSpec.addPathParam(CPF_PATH, cpf);
        return this;
    }

}
```

HTTP method and baseUri

eliasnogueira.com

architecture

```java
public static class OneUsingGETOper implements Oper {

    public static final Method REQ_METHOD = GET;
    public                                              /{cpf}";

    @Override
    public <T> T execute(Function<Response, T> handler) {
        // magic
    }


    public static final String CPF_PATH = "cpf";


    public OneUsingGETOper cpfPath(Object cpf) {
        reqSpec.addPathParam(CPF_PATH, cpf);
        return this;
    }

}
```
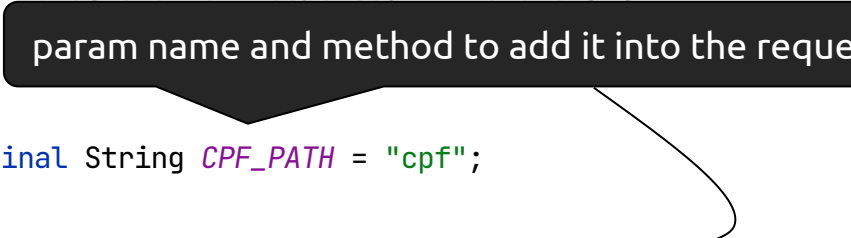
> adds the HTTP method and baseUri to the request

architecture

```java
public static class OneUsingGETOper implements Oper {

    public static final Method REQ_METHOD = GET;
    public static final String REQ_URI = "/api/v1/restrictions/{cpf}";

    @Override
    public <T> T execute(Function<Response, T> handler) {
        // magic
    }


    public static final String CPF_PATH = "cpf";


    public OneUsingGETOper cpfPath(Object cpf) {
        reqSpec.addPathParam(CPF_PATH, cpf);
        return this;
    }

}
```

param name and method to add it into the request

architecture

*ApiClient Creation

We need to use the RestApiClientBuilder to build the Client Api instance to add the URL and port.

Then we add a method per HTTP request. This is a recommended approach to ease any change (even to a different library).

architecture

build of the Client Api using
the RestApiClientBuilder

```java
public class RestrictionsApiClient

    private RestrictionsApi restrictionsApi =
            new RestApiClientBuilder().build(RestrictionsApi::restrictions);


    public Response queryCpf(String cpf) {
        return restrictionsApi.oneUsingGET().cpfPath(cpf).execute(Function.identity());
    }
}
```

architecture

```java
public cl

    priva

public Response queryCpf(String cpf) {
        return restrictionsApi.oneUsingGET().cpfPath(cpf).execute(Function.identity());
    }
}
```

abstracting the internal (ugly) Client Api usage
 - returning a generic response
 - adding a meaningful name
 - adding the path parameter as the method parameter

architecture

```java
public class RestrictionsApiClient {

    private RestrictionsApi
        new RestApiClient                            estrictions);


    public Response queryCpf(String    ) {
        return restrictionsApi.oneUsingGET().cpfPath(cpf).execute(Function.identity());
    }
}
```

using the auto-generated
internal Client Api method

architecture

```java
public class RestrictionsApiClient {

    private RestrictionsApi restrictionsApi =
            new RestApiClientBuilder().build(RestrictionsApi::restrictions);

    public Response queryCpf(String cpf) {
        return restrictionsApi.oneUsingGET().cpfPath(cpf).execute(Function.identity());
    }
}
```

using the path parameter

[Lab] OpenAPI Generator

Please, access **7. Better architecture -> Lab 1** and follow its steps for the item 2.

architecture

*ApiService

The *ApiService abstraction will use the *ApiClient abstraction to consume its methods in different ways. This is the class we will use in the tests.

The service can have one or multiple actions from the *ApiClient and it can be related to the Mediator design pattern, as it encapsulates how a set of objects (methods) interact.

architecture

*ApiService - Example

We do have two test for the Restrictions API:

- ○ Expecting a restriction
- ○ Not expecting a restriction

We will create the request for both, returning the correct response, in the service abstraction.

architecture

```java
public class RestrictionsApiService {

    private RestrictionsApiClient restrictionsApiClient = new RestrictionsApiClient();

    /**
     * Query CPF without a restriction
     */
    public boolean queryCpf(String cpf) {
        restrictionsApiClient.queryCpf(cpf).then().statusCode(HttpStatus.SC_NOT_FOUND);
        return true;
    }

    public MessageV1 queryCpfWithRestriction(String cpf) {
        return restrictionsApiClient.queryCpf(cpf).then().
                statusCode(HttpStatus.SC_OK).extract().as(MessageV1.class);
    }

}
```

architecture

```java
public class RestrictionsApiService {

    private RestrictionsApiClient restrictionsApiClient = new RestrictionsApiClient();

    /**
     * Query CPF without a restriction
     */
    public boolean queryCpf(String cpf) {
        restrictionsApiClient.queryCpf(cpf).then().statusCode(HttpStatus.SC_NOT_FOUND);
        return true;
    }

    public MessageV1 queryCpfWithRestriction(String cpf) {
        return restrictionsApiClient.queryCpf(cpf).then().
                statusCode(HttpStatus.SC_OK).extract().as(MessageV1.class);
    }

}
```

> instance of the abstracted Client Api

architecture

```java
public class RestrictionsApiService {

    private RestrictionsApiClient restrictionsApiClient = new RestrictionsApiClient();

    /**
     * Que
     */
    public boolean queryCpf(String cpf) {
        restrictionsApiClient.queryCpf(cpf).then().statusCode(HttpStatus.SC_NOT_FOUND);
        return true;
    }

    public MessageV1 queryCpfWithRestriction(String cpf) {
        return restrictionsApiClient.queryCpf(cpf).then().
                statusCode(HttpStatus.SC_OK).extract().as(MessageV1.class);
    }

}
```

method to query the cpf expecting no restriction

architecture

```java
public class RestrictionsApiService {

    private RestrictionsApiClient restrictionsApiClient = new RestrictionsApiClient();

    /**
     * Query CPF without a restriction
     */
    public boolean queryCpf(String cpf) {
        restrictionsApiClient.queryCpf(cpf).then().statusCode(HttpStatus.SC_NOT_FOUND);
        return true;
    }

    public                        tion(String cpf) {
                                  ryCpf(cpf).then().
                                  _OK).extract().as(MessageV1.class);
    }

}
```

> returning true because the status code is an HTTP 404
>
> so, we can add an assertions in the test

architecture

```java
public class RestrictionsApiService {

    private RestrictionsApiClient restrictionsApiClient = new RestrictionsApiClient();

    /**
     * Query CPF without a restriction
     */
    public boolean queryCpf(String cpf) {
        restrictionsApiClient.queryCpf(cpf).then().statusCode(HttpStatus.SC_NOT_FOUND);
        return t
    }

    public MessageV1 queryCpfWithRestriction(String cpf) {
        return restrictionsApiClient.queryCpf(cpf).then().
                statusCode(HttpStatus.SC_OK).extract().as(MessageV1.class);
    }

}
```

method to query the cpf expecting a restriction

architecture

```java
public class RestrictionsApiService {

    private RestrictionsApiClient restrictionsApiClient = new RestrictionsApiClient();

    /**
     * Query CPF without a restriction
     */
    public boolean queryCpf(String cpf) {
        restrictionsApiClient.queryCpf(cpf).then().statusCode(HttpStatus.SC_NOT_FOUND);
        return true;
    }

    public MessageV1 queryCpfWithRestriction(String cpf) {
        return restrictionsApiClient.queryCpf(cpf).then().
                statusCode(HttpStatus.SC_OK).extract().as(MessageV1.class);
    }

}
```

it returns the expected response body

eliasnogueira.com

[Lab] OpenAPI Generator

Please, access **7. Better architecture -> Lab 1** and follow its steps for the item 3.

architecture

Test

Now the tests will be created using only the Service class.

The different is that we don't need to use the raw REST Assured methods anymore, relying only in the Service.

The structure of precondition, action and assert will be always present in the new way to create tests.

architecture

Raw REST Assured Test

```java
@Test
void shouldReturnRestriction() {
    given()
        .spec(SharedRequestSpecs.cpfPathParameter("62648716050"))
    .when()
        .get("/restrictions/{cpf}")
    .then()
        .statusCode(HttpStatus.SC_OK)
        .body("message", CoreMatchers.is("CPF 62648716050 has a restriction"));
}
```

architecture

Raw REST Assured Test

```java
@Test
void shouldReturnRestriction() {
    given()
        .spec(SharedRequestSpecs.cpfPathParameter("62648716050"))
    .when()
        .get("/restrictions/{cpf}")
    .then()
        .statusCode(HttpStatus.SC_OK)
        .body("message", CoreMatchers.is("CPF 62648716050 has a restriction"));
}
```

REST Assured Test using Client – Service abstraction

```java
@Test
void shouldReturnRestriction() {
    RestrictionsApiService restrictionsApiService = new RestrictionsApiService();

    MessageV1 message = restrictionsApiService.queryCpfWithRestriction("60094146012");

    Assertions.assertThat(message.getMessage()).contains("60094146012");
}
```

architecture

Test

The usage of the service abstraction add more readability and help us to decrease the maintainability as we will have only one place to change it behaviour (client or service).

```java
@Test
void shouldReturnRestriction() {
    RestrictionsApiService restrictionsApiService = new RestrictionsApiService();

    MessageV1 message = restrictionsApiService.queryCpfWithRestriction("60094146012");

    Assertions.assertThat(message.getMessage()).contains("60094146012");
}
```

architecture

```java
@Test
void shouldReturnRestriction() {

    RestrictionsApiService restrictionsApiService = new RestrictionsApiService();

    MessageV1 message = restrictionsApiService.queryCpfWithRestriction("60094146012");

    Assertions.assertThat(message.getMessage()).contains("60094146012");
}
```

architecture

```java
@Test
void shouldReturnRestriction() {

    RestrictionsApiService restrictionsApiService = new RestrictionsApiService();

    MessageV1 message = restrictionsApiService.queryCpfWithRestriction("60094146012");

    Assertions.assertThat(message.getMessage()).contains("60094146012");
}
```

instance to use the Service abstraction

architecture

```java
@Test
void shouldReturnRestriction() {

    RestrictionsApiService restrictionsApiService = new RestrictionsApiService();

    MessageV1 message = restrictionsApiService.queryCpfWithRestriction("60094146012");

    Assertions.assertThat(message.getMessage()).contains("60094146012");
}
```

usage of the method in the service

architecture

```
@Test
void shouldReturnRestriction() {

    RestrictionsApiService restrictionsApiService = new RestrictionsApiService();

    MessageV1 message = restrictionsApiService.queryCpfWithRestriction("60094146012");

    Assertion                  (message.getMessage()).contains("60094146012");
}
```

associating the correct return (response body)

architecture

```java
@Test
void shouldReturnRestriction() {

    RestrictionsApiServi                        piService();

    MessageV1 message = restric                ice.queryCpfWithRestriction("60094146012");

    Assertions.assertThat(message.getMessage()).contains("60094146012");
}
```

assertion using the response body object

[Lab] OpenAPI Generator

Please, access **7. Better architecture -> Lab 2** and follow its steps for the item 1 and 2.

architecture

BaseApiConfiguration

Within the new approach, the general configuration is now being applied using the `RestClientApiBuilder` class, as it created a common request specification for all the requests.

We can either move the previous configurations from the `BaseApiConfiguration` class to the `RestClientApiBuilder` or continue to use the `BaseApiConfiguration` without the `baseUri`, `basePath`, and `port`.

architecture

Recommendation

A better approach is to use the RestClientApiBuilder to deal only with the global request actions and the BaseApiConfiguration with the configurations related to the test.

Both classes do different things, and they must have a single responsibility.

[Lab] OpenAPI Generator

Please, access **7. Better architecture -> Lab 2** and follow its steps for the item 3.
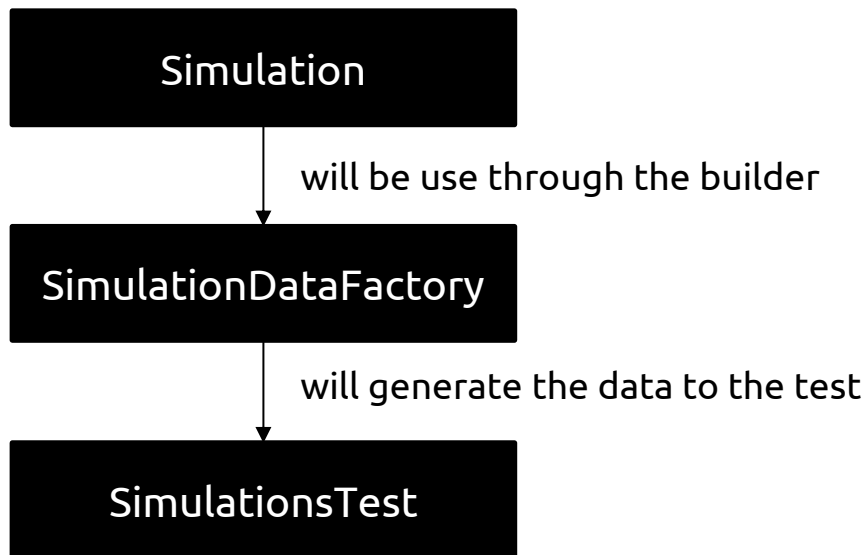
data

architecture

Data is one of the most painful aspects of testing, if not done correctly!

We can use it in different ways… some can be hard-coded, but some must have a meaning, as it can influence the test result.

We will apply a combination of some patterns to create a new one for the data perspective(!!!)

architecture

Basically, we need a Model which with the Builder ability that will be used by a generator class used by the tests to generate data.

```
┌─────────────────────────────┐
│         Simulation          │
└─────────────────────────────┘
              │
              │  will be use through the builder
              ▼
┌─────────────────────────────┐
│    SimulationDataFactory     │
└─────────────────────────────┘
              │
              │  will generate the data to the test
              ▼
┌─────────────────────────────┐
│       SimulationsTest        │
└─────────────────────────────┘
```

architecture

Factory class

The idea is to have a Factory class implementation (or sort of) to generate data from different requirements.

architecture

```java
public final class SimulationDataFactory {

    private SimulationDataFactory() {}

    public static Simulation validSimulation() {
        return Simulation.builder().
                cpf("0987654321").name("Robert").email("robert@gmail.com").
                amount(new BigDecimal("2.000")).
                installments(5).insurance(false).build();
    }

    public static Simulation notValidEmail() {
        return Simulation.builder().
                cpf("95746263958").name("Lucas").email("not-valid-email").
                amount(new BigDecimal("10.000")).
                installments(7).insurance(true).build();
    }
}
```

architecture

```java
public final class SimulationDataFactory {
```

class cannot be extended

```java
    public static Simulation validSimulation() {
        return Simulation.builder().
                cpf("0987654321").name("Robert").email("robert@gmail.com").
                amount(new BigDecimal("2.000")).
                installments(5).insurance(false).build();
    }

    public static Simulation notValidEmail() {
        return Simulation.builder().
                cpf("95746263958").name("Lucas").email("not-valid-email").
                amount(new BigDecimal("10.000")).
                installments(7).insurance(true).build();
    }
}
```

architecture

```java
public final class SimulationDataFactory {

    private SimulationDataFactory() {}

    public static Simulation validSimulation() {
        return ...
                                        ).email("robert@gmail.com").
                amount(new BigDecimal("2.000")).
                installments(5).insurance(false).build();
    }

    public static Simulation notValidEmail() {
        return Simulation.builder().
                cpf("95746263958").name("Lucas").email("not-valid-email").
                amount(new BigDecimal("10.000")).
                installments(7).insurance(true).build();
    }
}
```

restrict the object creation

architecture

```java
public fina                  creation of a valid simulation using
                             the Builder from the Simulation class
    private Simula        actory() {}

    public static Simulation validSimulation() {
        return Simulation.builder().
                cpf("0987654321").name("Robert").email("robert@gmail.com").
                amount(new BigDecimal("2.000")).
                installments(5).insurance(false).build();
    }

    public static Simulation notValidEmail() {
        return Simulation.builder().
                cpf("95746263958").name("Lucas").email("not-valid-email").
                amount(new BigDecimal("10.000")).
                installments(7).insurance(true).build();
    }
}
```

eliasnogueira.com

architecture

```java
public final class SimulationDataFactory {

    private SimulationDataFactory() {}

    public static Simulation validSimulation() {
        return Simulation.builder().
                cpf("0987654321").name("Robert").email("robert@gmail.com").
                                                                    ld();
    }

    public static Simulation notValidEmail() {
        return Simulation.builder().
                cpf("95746263958").name("Lucas").email("not-valid-email").
                amount(new BigDecimal("10.000")).
                installments(7).insurance(true).build();
    }
}
```

creation of an invalid valid simulation

architecture

Approach

The previous approach is not the best one for data generation, as we still have hard-coded data that might show a false-positive error.

We must have a way to generate different data during each call from the methods, and for this, we can use the DataFaker library.

architecture

```java
public final class SimulationDataFactory {

    private static Faker faker = new Faker();

    private SimulationDataFactory() {}

    public static Simulation validSimulation() {

        return Simulation.builder().
                cpf(faker.number().digits(11)).
                name(faker.name().fullName()).
                email(faker.internet().emailAddress()).
                amount(new BigDecimal(faker.number().numberBetween(100, 40000))).
                installments(faker.number().numberBetween(2, 48)).
                insurance(faker.bool().bool())
            .build();
    }
}
```

[Lab] Data

Please, access **8. Data -> Lab 1** and follow its steps for the item 1.

what's next

what's next

You have the Extra section to exercise (here or at home!)

Thank you for you time and don't hesitate giving any feedback about this session!