



## AULA 12

# AUTOMAÇÃO DE TESTE (API)

**Richard Okubo**

Programador Pleno





# Olá, eu sou o Richard Okubo

## PROGRAMADOR

- Formação em Marketing / pós-graduando em Engenharia de Software
- Ex-residência em TIC/Software pelo Serratec (2021.2)
- Trabalho com desenvolvimento e qualidade de software a cerca de 3 anos na T2M.
- MAG Seguros e Petrobrás Transporte S.A.
- Membro dos times de sustentação e evolução dos sistemas da Transpetro, braço logístico da Petrobras.

# OBSERVAÇÕES



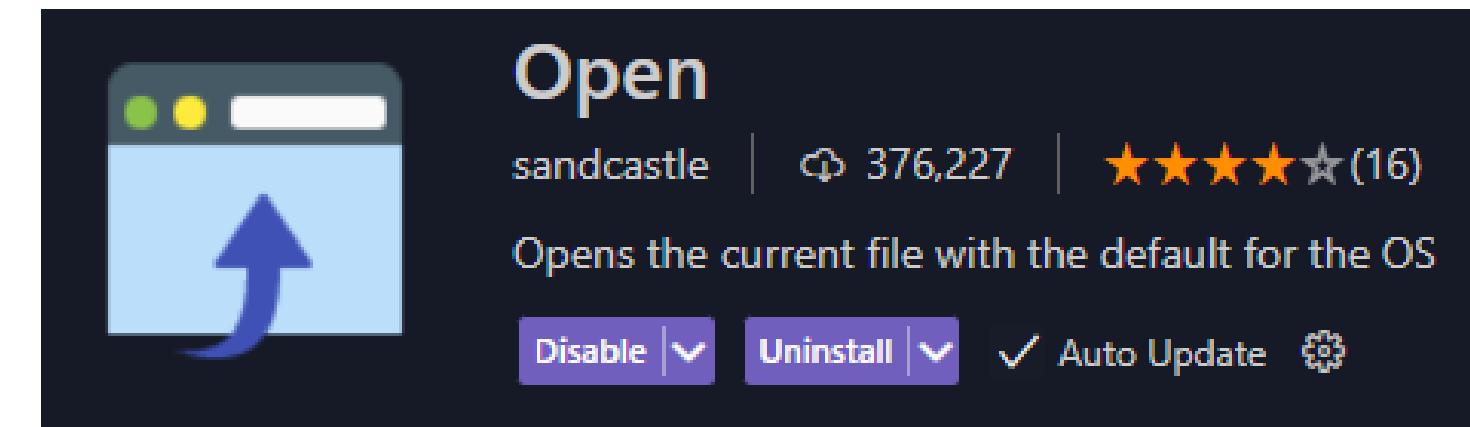
# TÓPICOS DA AULA

- Pré-requisitos
- Mão na massa



# PRÉ-REQUISITOS

```
Prompt de Comando
C:\>python --version
Python 3.13.2
C:\>
```



RobotCode - Robot Framework Support  
Daniel Biehl 🏆 robotcode.io | ⚡ 195,428 | ★★★★★(49) | ❤ Sponsor  
Robot Framework IntelliSense, linting, test execution and debugging, code formatting, refactoring, and many more  
Disable | Uninstall | ✓ Auto Update

# PRÉ-REQUISITOS

```
C:\>pip install robotframework-requests==0.9.7 robotframework-jsonlibrary==0.5
```

```
C:\>pip list
Package           Version
-----
attrs             25.1.0
certifi           2025.1.31
charset-normalizer 3.4.1
idna              3.10
jsonpath-ng       1.7.0
jsonschema         4.23.0
jsonschema-specifications 2024.10.1
pip               24.3.1
ply               3.11
referencing        0.36.2
requests          2.32.3
robotframework     7.2.2
robotframework-jsonlibrary 0.5
robotframework-requests 0.9.7
rplus-py          0.22.3
urllib3            2.3.0

C:\>
```

# MÃO NA MASSA

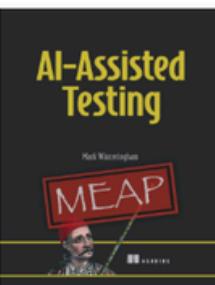
Welcome to Restful-Booker  
An API playground created by [Mark Winteringham](#) for those wanting to learn more about API testing and tools  
[@2bittester](#) | [Website](#) | [Code](#) | [API Docs](#)

Welcome to Restful-booker an API that you can use to learn more about API Testing or try out API testing tools against. Restful-booker is a **Create Read Update Delete** Web API that comes with authentication features and loaded with a bunch of bugs for you to explore. The API comes pre-loaded with 10 records for you to work with **and resets itself every 10 minutes back to that default state**. Restful-booker also comes with [detailed API documentation](#) to help get you started with your API testing straight away.

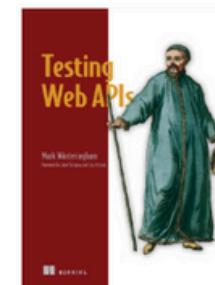
## Support me and Restful-Booker

Restful-booker is a free to use API for practising testing, but if you enjoy using this API please consider supporting me by purchasing one of my books. Alternatively you can support [Ministry of Testing](#) who host this API by going Pro.

[Buy AI-Assisted Testing](#)



[Buy Testing Web APIs](#)



[Go Pro with Ministry of Testing](#)



Link: <https://restful-booker.herokuapp.com/apidoc/index.html>

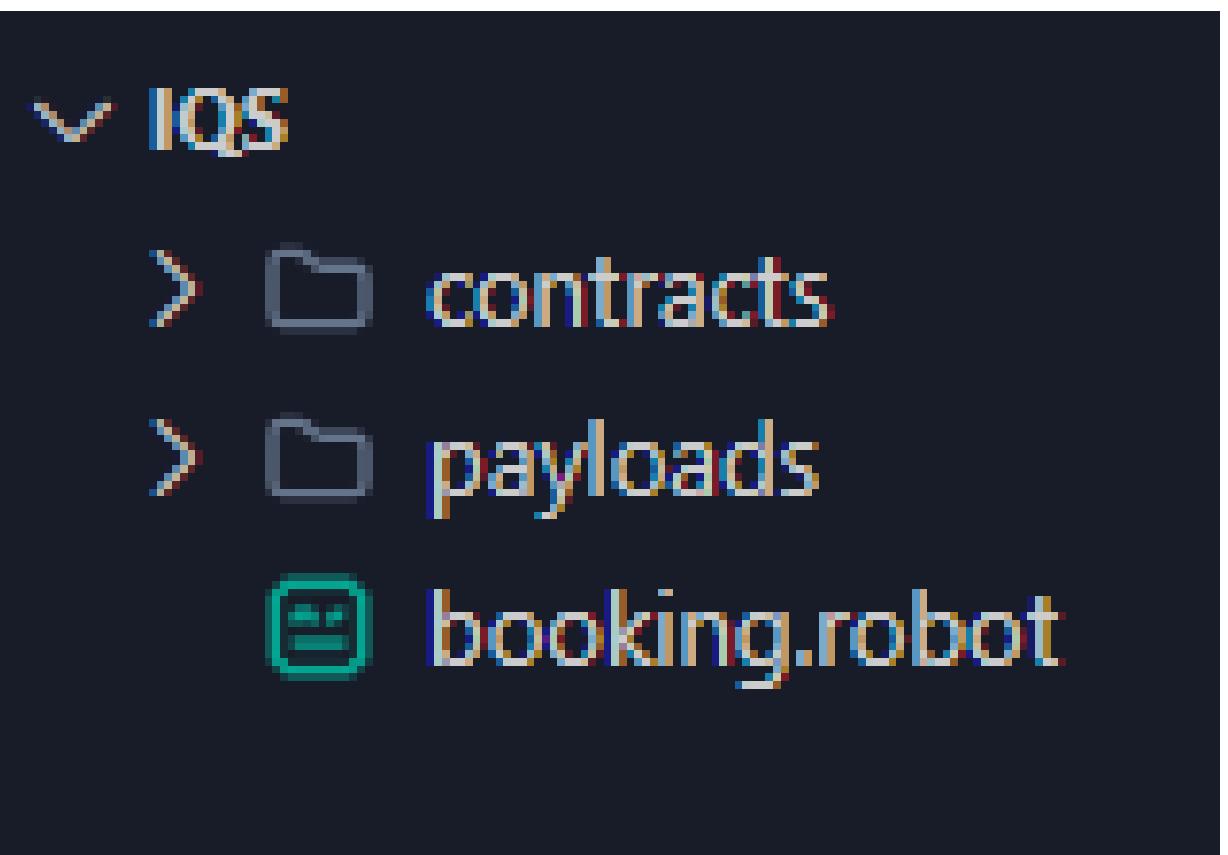
# Estrutura de projeto

Há diferentes formas de estruturar um projeto de automação. Para o nosso caso, iremos usar a seguinte:

**contracts:** pasta onde adicionaremos os nossos contratos (*schemas*)

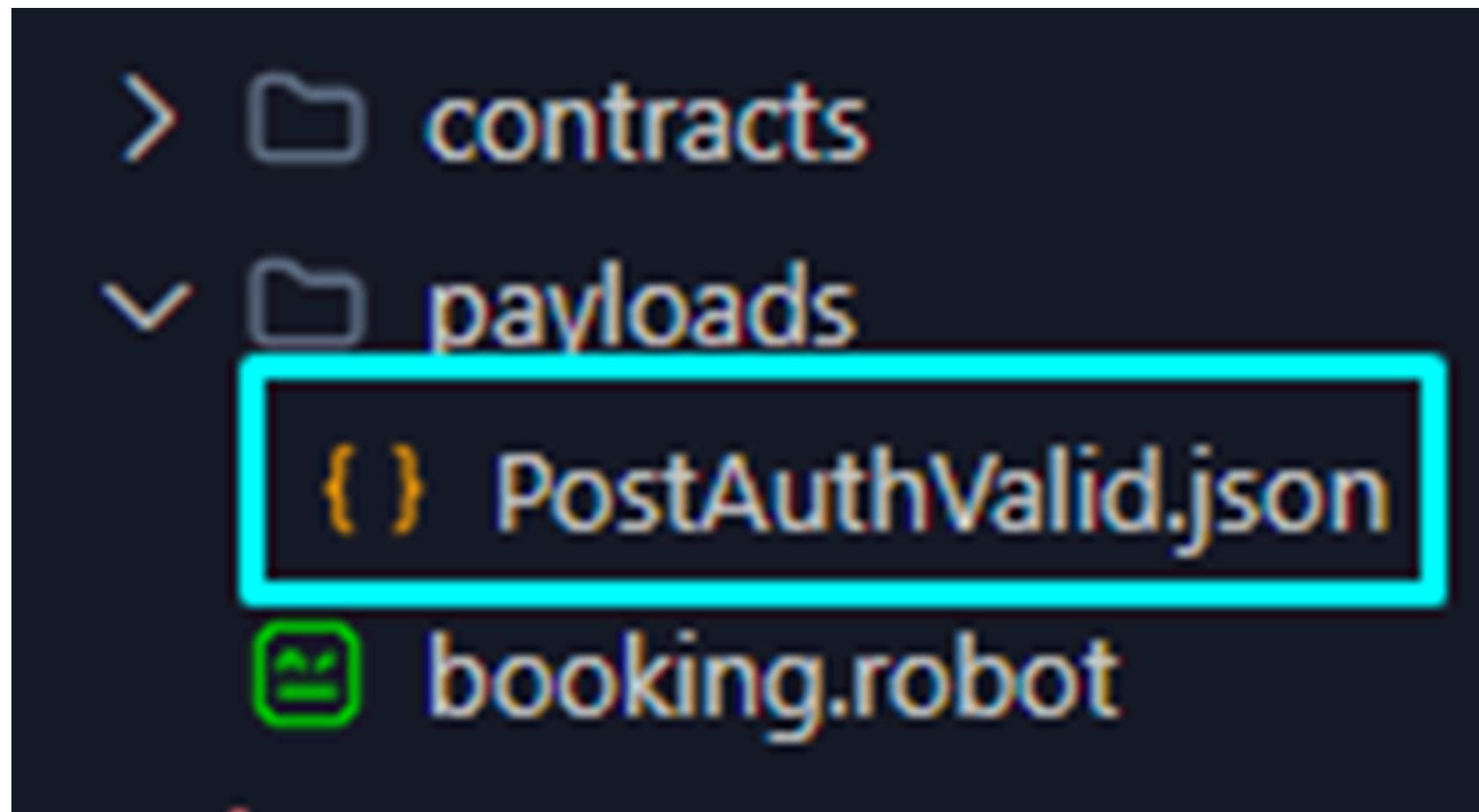
**payloads:** pasta onde adicionaremos os *payloads* (passados no *body* dos métodos POST, PUT e PATCH)

**booking.robot:** o arquivo de teste orópriamente dito (suíte de teste)



# Payload

Crie o arquivo contendo o *payload* que será utilizado no teste para autenticação com credencial válida.



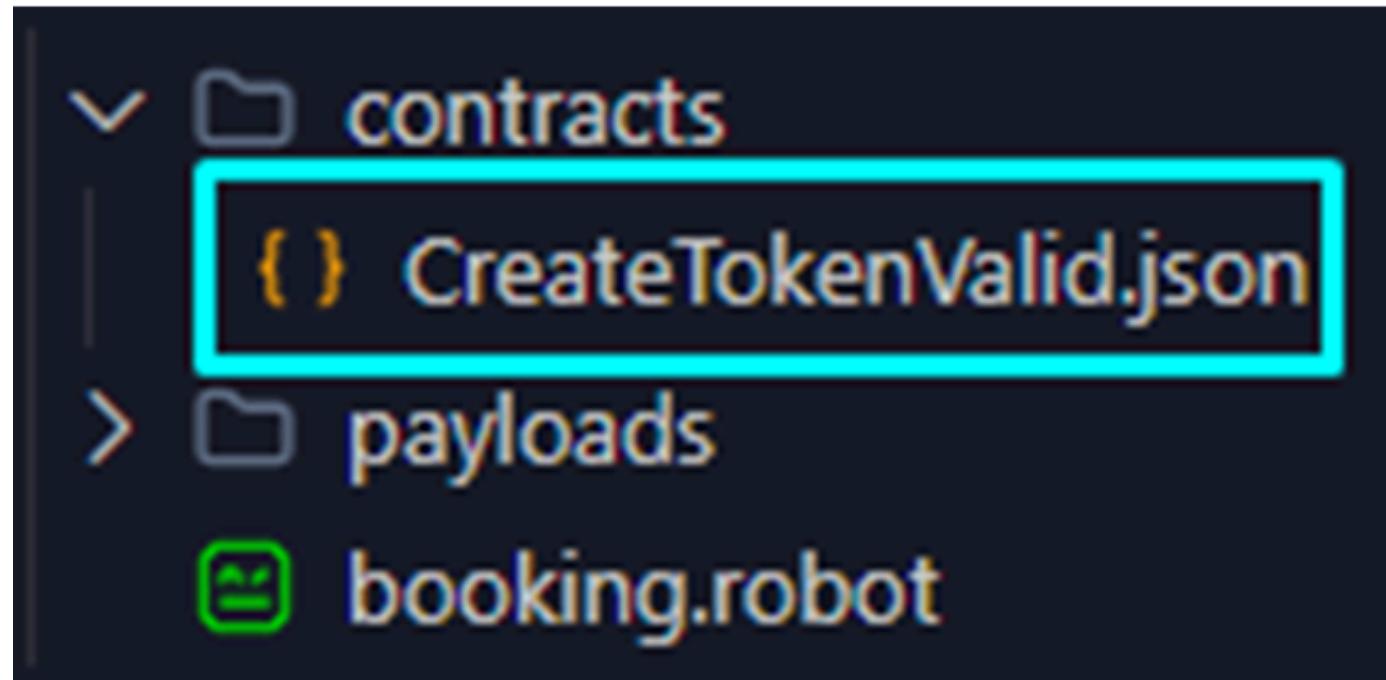
```
{
  "username": "admin",
  "password": "password123"
}
```

# JSON Schema

O JSON Schema nos ajuda a definir a estrutura e o formato dos documentos JSON. Fornece uma maneira padronizada de validar o conteúdo de documentos JSON, assegurando que eles sigam um formato específico.

**Link:** <https://transform.tools/json-to-json-schema>

Crie o arquivo que conterá o JSON Schema do retorno da requisição POST para o endpoint /auth nos testes de autenticação com credencial válida.



```
{  
  "$schema": "http://json-schema.org/draft-07/schema#",  
  "type": "object",  
  "properties": {  
    "token": {  
      "type": "string"  
    },  
    "required": [  
      "token"  
    ]  
}
```

# Bibliotecas

Na seção "Settings", importamos duas bibliotecas:

- **RequestsLibrary**: para fazer requisições HTTP;
- **JSONLibrary**: para manipular e validar dados JSON.

```
*** Settings ***
Library     RequestsLibrary
Library     JSONLibrary

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response}    POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenValid.json
```

# Test Case

Na seção "Test Cases", definimos o nosso primeiro caso de teste. Iremos realizar uma verificação no serviço de autenticação com usuário válido.

```
*** Settings ***
Library ... RequestsLibrary
Library ... JSONLibrary

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    Create Session ... alias=iqs ... url=https://restful-booker.herokuapp.com
    ${payload} ... Load Json From File ... ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response} ... POST On Session ... alias=iqs ... url=/auth ... json=${payload} ... expected_status=200
    Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenValid.json
```

# Session

**Create Session:** funcionalidade fornecida pela RequestsLibrary, que permite criar uma sessão HTTP persistente. Passamos um alias (nome da sessão) e a URL base como argumento da palavra-chave.

```
*** Settings ***
Library    RequestsLibrary
Library    JSONLibrary

*** Test Cases ***
TC1: realizar autenticacão com usuário válido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response}    POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenValid.json
```

# Carregando JSON

Load Json From File: carrega o payload (dados do corpo da requisição) de um arquivo JSON para o nosso caso de teste.  
A variável especial \${CURDIR} é a representação do diretório atual.  
A variável especial \${/} representa o separador de diretórios da plataforma que o script estará rodando.

```
*** Settings ***
Library ... RequestsLibrary
Library ... JSONLibrary

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response}    POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenValid.json
```

# POST

**POST On Session:** envia uma requisição POST usando a sessão iqspara endpoint /auth com o payload JSON carregado anteriormente. Também adicionamos uma verificação para o status da resposta (200 OK). Por fim, armazenamos o retorno da requisição na variável \${response}.

```
*** Settings ***
Library ... RequestsLibrary
Library ... JSONLibrary

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    Create Session ... alias=iqs ... url=https://restful-booker.herokuapp.com
    ${payload} ... Load Json From File ... ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response} ... POST On Session ... alias=iqs ... url=/auth ... json=${payload} ... expected_status=200
    Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenValid.json
```

# Verificação do JSON Schema

**Validate Json By Schema File:** utilizado para validar a estrutura e o conteúdo de uma resposta JSON de acordo com um esquema JSON predefinido. Isso é essencial para garantir que a API responda com dados no formato esperado, aderindo a uma especificação definida. Link: <https://json-schema.org/>

`${response.json()}: representa o JSON a ser validado.`

```
*** Settings ***
Library     RequestsLibrary
Library     JSONLibrary

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}        Load Json From File    ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response}       POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenValid.json
```

DÚVIDAS?



# Payload

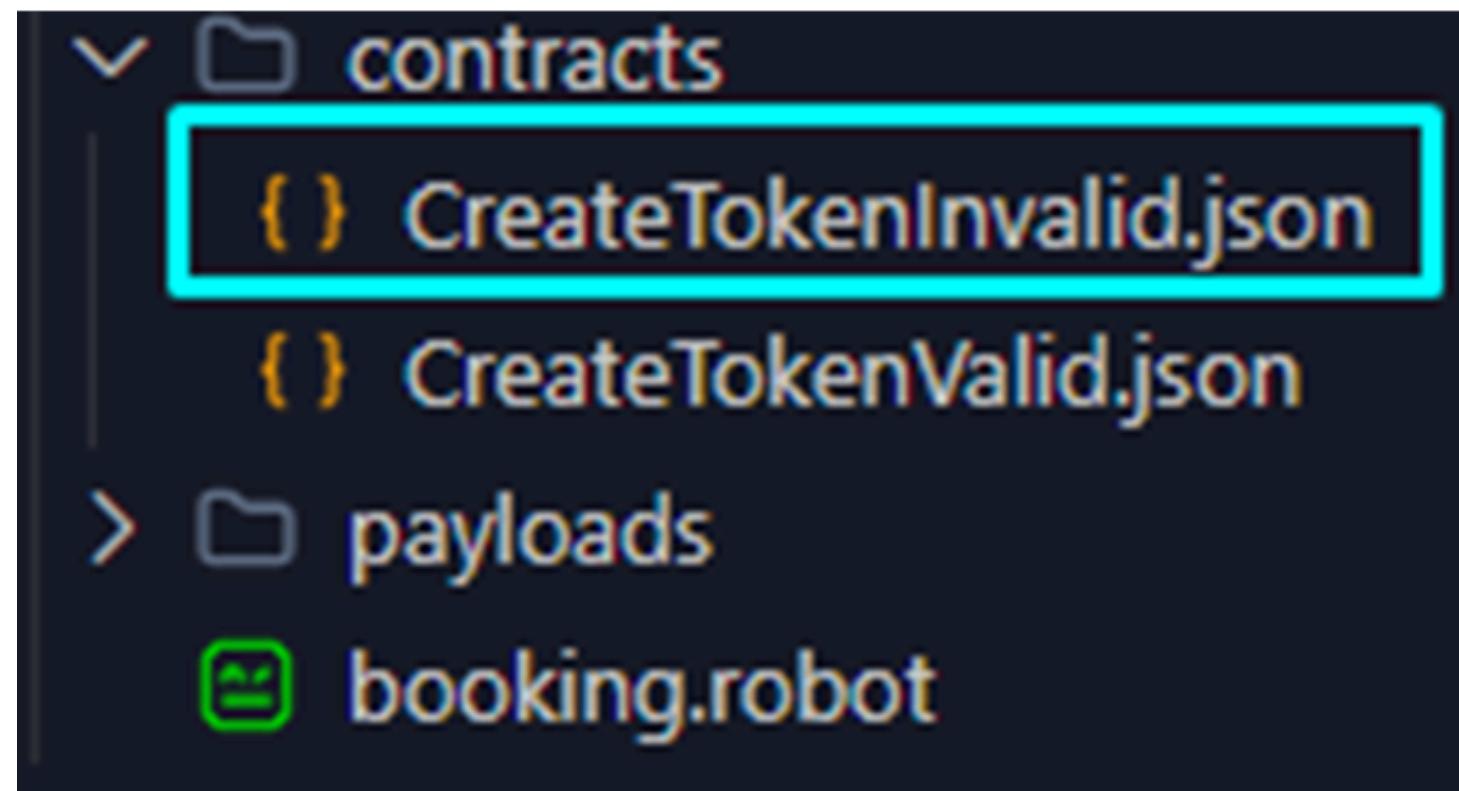
Crie o arquivo contendo o payload que será utilizado no teste para autenticação com credencial inválida.

```
> contracts
< payloads
  { } PostAuthInvalid.json
  { } PostAuthValid.json
  booking.robot
```

```
{
  "username": "invalid",
  "password": "invalid"
}
```

# JSON Schema

Crie o arquivo que conterá o JSON Schema do retorno da requisição POST para o endpoint /auth nos testes de autenticação com credencial inválida.



```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "reason": {
      "type": "string"
    }
  },
  "required": [
    "reason"
  ]
}
```

# Test Case

Na seção "Test Cases", definimos outra caso de teste. Agora, iremos realizar uma verificação no serviço de autenticação com usuário inválido.

Note que o código é ligeramente semelhante ao caso de teste anterior, com a diferença de que estaremos passando desta vez os arquivos com o payload e outro com o contrato (JSON Schema) referente a este cenário com credencial inválida.

## TC2: realizar autenticação com usuário inválido

```
.... Create Session .... alias=iqs .... url=https://restful-booker.herokuapp.com

.... ${payload} .... Load Json From File .... ${CURDIR}${/}payloads${/}PostAuthInvalid.json
.... ${response} .... POST On Session .... alias=iqs .... url=/auth .... json=${payload} .... expected_status=200

.... Validate Json By Schema File .... ${response.json()} .... ${CURDIR}${/}contracts${/}CreateTokenInvalid.json
```

# Refatorando

Lógica semelhante entre os testes.

```
*** Test Cases ***

TC1: realizar autenticação com usuário válido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}        Load Json From File    ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response}       POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenValid.json

TC2: realizar autenticação com usuário inválido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}        Load Json From File    ${CURDIR}${/}payloads${/}PostAuthInvalid.json
    ${response}       POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenInvalid.json
```

# Refatorando

Lógica semelhante entre os testes.

```
*** Test Cases ***

TC1: realizar autenticação com usuário válido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response}    POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenValid.json

TC2: realizar autenticação com usuário inválido
    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}PostAuthInvalid.json
    ${response}    POST On Session    alias=iqs    url=/auth    json=${payload}    expected_status=200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenInvalid.json
```

# Refatorando

**Suite Setup:** usado para definir ações que devem ser executadas uma vez antes de qualquer caso de teste.

```
*** Settings ***
Library ... RequestsLibrary
Library ... JSONLibrary

Suite Setup ... Create Session ... alias=iqs ... url=https://restful-booker.herokuapp.com

*** Test Cases ***
TC1: realizar autenticação com usuário válido
... Create Session ... alias=iqs ... url https://restful-booker.herokuapp.com

... ${payload} ... Load Json From File ... ${CURDIR}${/}payloads${/}PostAuthValid.json
... ${response} ... POST On Session ... alias=iqs ... url=/auth ... json=${payload} ... expected_status=200

... Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenValid.json

TC2: realizar autenticação com usuário inválido
... Create Session ... alias=iqs ... url https://restful-booker.herokuapp.com

... ${payload} ... Load Json From File ... ${CURDIR}${/}payloads${/}PostAuthInvalid.json
... ${response} ... POST On Session ... alias=iqs ... url=/auth ... json=${payload} ... expected_status=200

... Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenInvalid.json
```

# Refatorando

Criamos uma palavra-chave que irá abstrair a ação de requisição POST nos nossos testes.

**[Arguments]**: passagem de argumento para dentro da palavra-chave

**RETURN**: conteúdo retornado da palavra-chave

```
*** Settings ***
Library ... RequestsLibrary
Library ... JSONLibrary

Suite Setup ... Create Session ... alias=iqs ... url=https://restful-booker.herokuapp.com

*** Keywords ***
POST /${endpoint}
[Arguments]    ${filename}    ${expected_status}
${payload}    Load Json From File    ${CURDIR}${/}payloads${/}${filename}
${response}    POST On Session ... alias=iqs ... url=${endpoint} ... json=${payload} ... expected_status=${expected_status}
RETURN    ${response}

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}PostAuthValid.json
    ${response}    POST On Session ... alias=iqs ... url=/auth ... json=${payload} ... expected_status=200
    Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenValid.json

TC2: realizar autenticação com usuário inválido
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}PostAuthInvalid.json
    ${response}    POST On Session ... alias=iqs ... url=/auth ... json=${payload} ... expected_status=200
    Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenInvalid.json
```

# Refatorando

Agora podemos substituir o código semelhante entre os testes pela chamada da palavra-chave que acabamos de definir. O uso de \${endpoint} indica que o endpoint será um argumento passado para a palavra-chave.

```
*** Settings ***
Library ... RequestsLibrary
Library ... JSONLibrary

Suite Setup ... Create Session alias=iqs url=https://restful-booker.herokuapp.com

*** Keywords ***
POST /${endpoint}
    [Arguments] ... ${filename} ... ${expected_status}
    ${payload} ... Load Json From File ... ${CURDIR}${/}payloads${/}${filename}
    ${response} ... POST On Session alias=iqs url=${endpoint} json=${payload} expected_status=${expected_status}
    RETURN ... ${response}

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    ${response} ... POST /auth ... PostAuthValid.json ... 200
    Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenValid.json

TC2: realizar autenticação com usuário inválido
    ${response} ... POST /auth ... PostAuthInvalid.json ... 200
    Validate Json By Schema File ... ${response.json()} ... ${CURDIR}${/}contracts${/}CreateTokenInvalid.json
```

# Refatorando

De maneira semelhante, também podemos abstrair a validação do JSON Schema por uma palavra-chave que simplifique o processo.

```
*** Settings ***
Library    RequestsLibrary
Library    JSONLibrary

Suite Setup    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com

*** Keywords ***
POST /${endpoint}
    [Arguments]    ${filename}    ${expected_status}
    ${payload}    Load Json From File    ${CURDIR}${/}payloads${/}${filename}
    ${response}    POST On Session    alias=iqs    url=${endpoint}    json=${payload}    expected_status=${expected_status}
    RETURN    ${response}

Validate Json
    [Arguments]    ${response}    ${filename}
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}${filename}

*** Test Cases ***
TC1: realizar autenticação com usuário válido
    ${response}    POST /auth    PostAuthValid.json    200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenValid.json

TC2: realizar autenticação com usuário inválido
    ${response}    POST /auth    PostAuthInvalid.json    200
    Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}CreateTokenInvalid.json
```

# Refatorando

Agora só precisamos chamar a nova palavra-chave passando o \${response} e o arquivo que contém o JSON Schema do retorno da requisição.

```
*** Settings ***
Library    RequestsLibrary
Library    JSONLibrary

Suite Setup    Create Session    alias=iqs    url=https://restful-booker.herokuapp.com

*** Keywords ***
POST /${endpoint}
[Arguments]    ${filename}    ${expected_status}
${payload}    Load Json From File    ${CURDIR}${/}payloads${/}${filename}
${response}    POST On Session    alias=iqs    url=${endpoint}    json=${payload}    expected_status=${expected_status}
RETURN    ${response}

Validate Json
[Arguments]    ${response}    ${filename}
Validate Json By Schema File    ${response.json()}    ${CURDIR}${/}contracts${/}${filename}

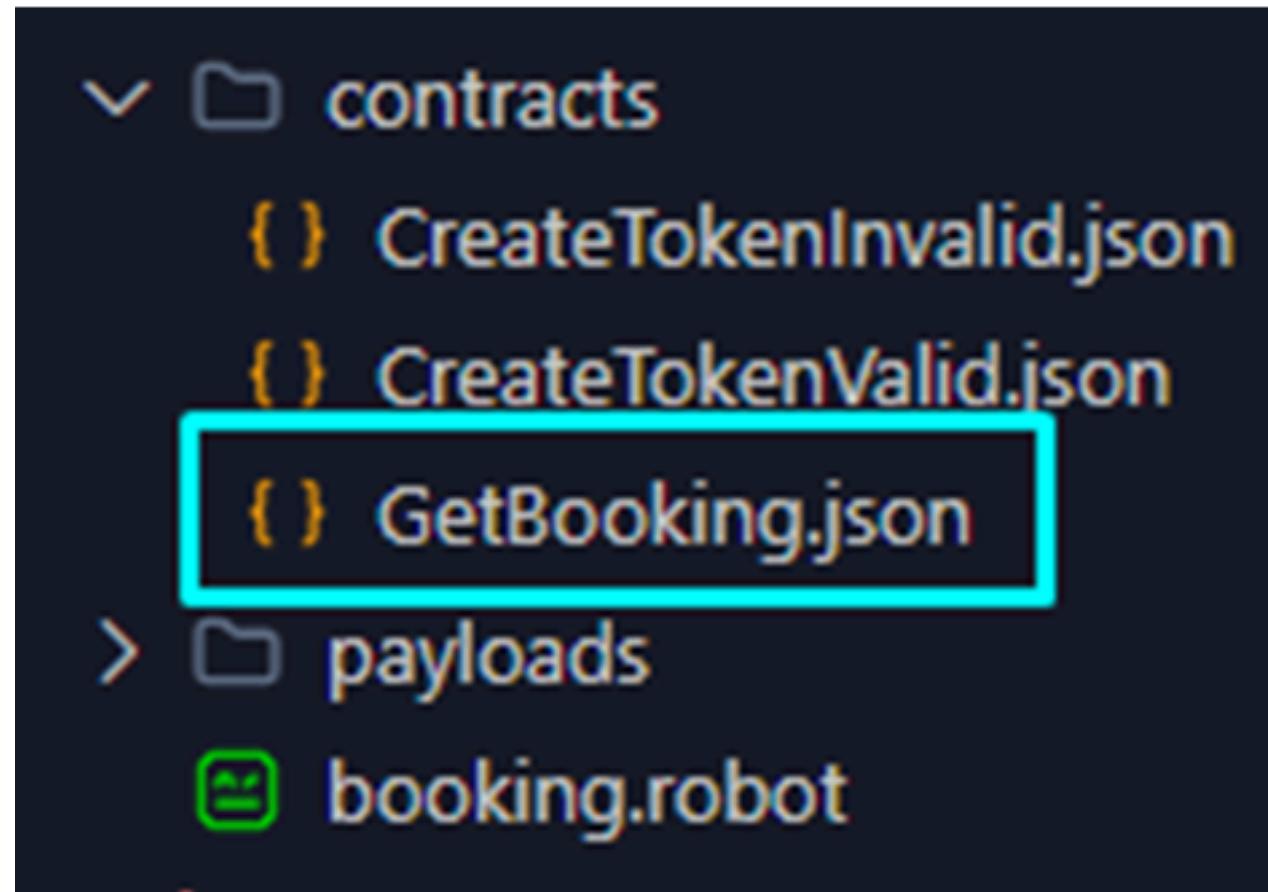
*** Test Cases ***
TC1: realizar autenticação com usuário válido
${response}    POST /auth    PostAuthValid.json    200
    Validate Json    ${response}    CreateTokenValid.json
TC2: realizar autenticação com usuário inválido
${response}    POST /auth    PostAuthInvalid.json    200
    Validate Json    ${response}    CreateTokenInvalid.json
```

DÚVIDAS?



# Payload

Crie o arquivo que conterá o JSON Schema do retorno da requisição GET para o endpoint /booking.



```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "bookingid": {
        "type": "number"
      }
    },
    "required": [
      "bookingid"
    ]
  }
}
```

# Test Case

GET On Session: usada para realizar a requisição GET utilizando uma sessão HTTP específica (no caso, iqs).

```
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas
    ${response}    GET On Session    alias=iqs    url=/booking    expected_status=200
    Validate Json    ${response}    GetBooking.json
```

# Refatorando

Criamos uma palavra-chave que irá abstrair a ação de requisição GET nos nossos testes.

```
> *** Settings ***
*** Keywords ***
GET /${endpoint}
[Arguments]    ${expected_status}
${response}    GET On Session    alias=iqs    url=${endpoint}    expected_status=${expected_status}
RETURN    ${response}

> POST /${endpoint} ...
> Validate Json ...
*** Test Cases ***
> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
TC3: obter reservas
    ${response}    GET On Session    alias=iqs    url=/booking    expected_status=200
    Validate Json    ${response}    GetBooking.json
```

# Refatorando

Agora podemos usar a nossa nova palavra-chave.

```
> *** Settings ***
*** Keywords ***
GET /${endpoint}
    [Arguments]    ${expected_status}
    ${response}    GET On Session    alias=iqs    url=${endpoint}    expected_status=${expected_status}
    RETURN    ${response}

> POST /${endpoint} ...
> Validate Json ...
*** Test Cases ***
> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
TC3: obter reservas
    ${response}    GET /booking    200
    Validate Json    ${response}    GetBooking.json
```

# Refatorando

Ainda podemos continuar refatorando...

```
*** Settings ***
Library <-- RequestsLibrary
Library <-- JSONLibrary

Suite Setup <-- Create Session <-- alias=iqs <-- url=https://restful-booker.herokuapp.com

*** Keywords ***
GET /${endpoint}
[Arguments] <-- ${expected_status}
${response} <-- GET On Session <-- alias=iqs <-- url=${endpoint} <-- expected_status=${expected_status}
RETURN <-- ${response}

POST /${endpoint}
[Arguments] <-- ${filename} <-- ${expected_status}
${payload} <-- Load Json From File <-- ${CURDIR}${/}payloads${/}${filename}
${response} <-- POST On Session <-- alias=iqs <-- url=${endpoint} <-- json=${payload} <-- expected_status=${expected_status}
RETURN <-- ${response}

Validate Json
[Arguments] <-- ${response} <-- ${filename}
Validate Json By Schema File <-- ${response.json()} <-- ${CURDIR}${/}contracts${/}${filename}

*** Test Cases ***
TC1: realizar autenticação com usuário válido...
TC2: realizar autenticação com usuário inválido...
TC3: obter reservas...
```

# Refatorando

Criamos a seção "Variables" e adicionamos a definição das nossas variáveis. Vamos usá-las no nosso código.

```
*** Settings ***
Library  ... RequestsLibrary
Library  ... JSONLibrary

Suite Setup  Create Session  alias=${SESSION_NAME}  url=${BASE_URL}

*** Variables ***
${BASE_URL}  https://restful-booker.herokuapp.com
${SESSION_NAME}  iqs
${PAYLOADS_DIR}  ${CURDIR}${/}payloads
${CONTRACTS_DIR}  ${CURDIR}${/}contracts

*** Keywords ***
GET /${endpoint}
[Arguments]  ${expected_status}
${response}  GET On Session  alias=${SESSION_NAME}  url=${endpoint}  expected_status=${expected_status}
RETURN  ${response}

POST /${endpoint}
[Arguments]  ${filename}  ${expected_status}
${payload}  Load Json From File  ${PAYLOADS_DIR}${/}${filename}
${response}  POST On Session  alias=${SESSION_NAME}  url=${endpoint}  json=${payload}  expected_status=${expected_status}
RETURN  ${response}

Validate Json
[Arguments]  ${response}  ${filename}
${response}  Validate Json By Schema File  ${response.json()}  ${CONTRACTS_DIR}${/}${filename}

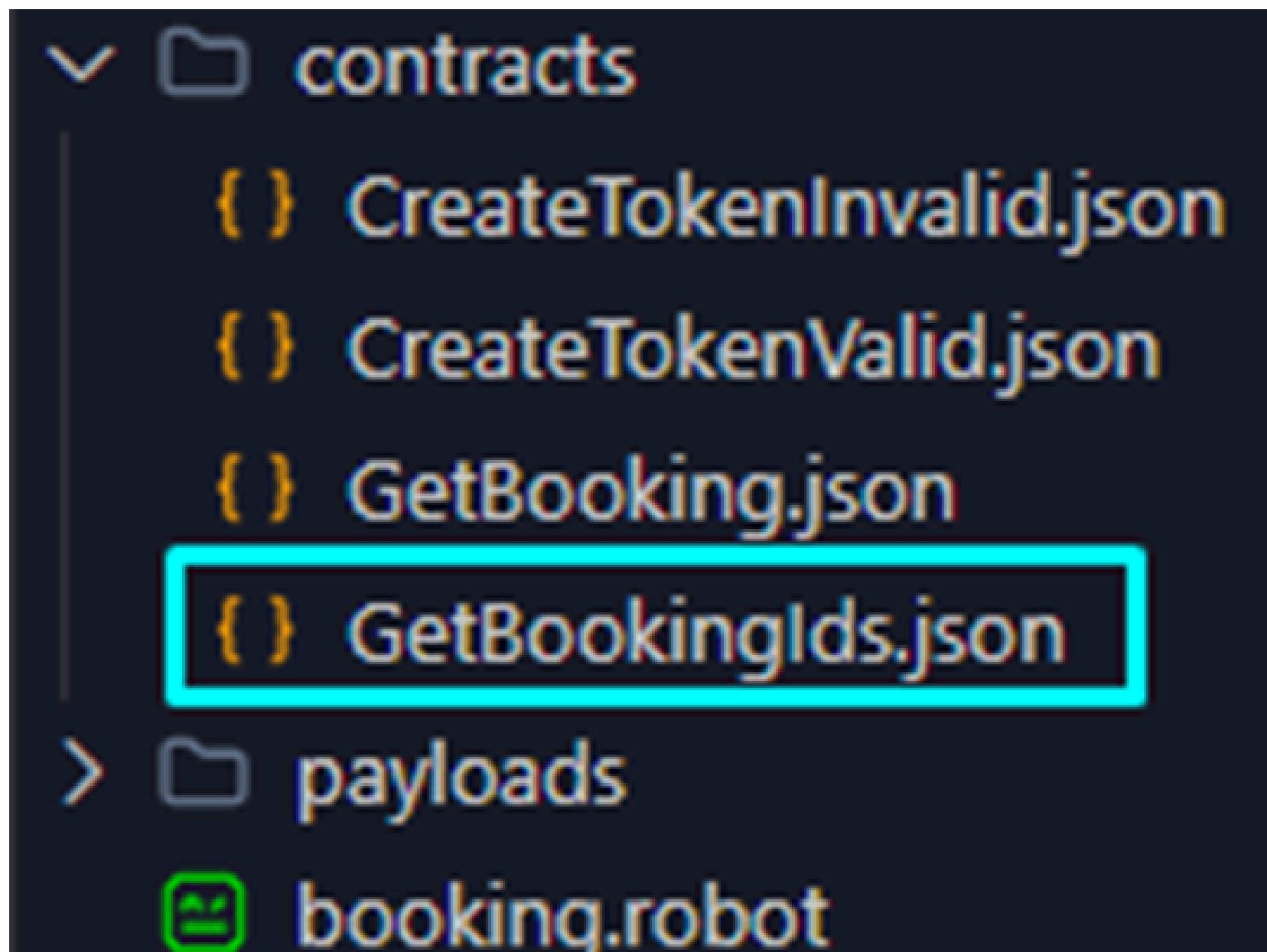
*** Test Cases ***
TC1: realizar autenticação com usuário válido...
TC2: realizar autenticação com usuário inválido...
TC3: obter reservas...
```

DÚVIDAS?



# JSON Schema

Crie o arquivo que conterá o JSON Schema do retorno da requisição GET para o endpoint /booking/:id



```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "firstname": {
      "type": "string"
    },
    "lastname": {
      "type": "string"
    },
    "totalprice": {
      "type": "number"
    },
    "depositpaid": {
      "type": "boolean"
    },
    "bookingdates": {
      "type": "object",
      "properties": {
        "checkin": {
          "type": "string"
        },
        "checkout": {
          "type": "string"
        }
      }
    },
    "required": [
      "checkin",
      "checkout"
    ]
  },
  "additionalneeds": {
    "type": "string"
  }
},
"required": [
  "firstname",
  "lastname",
  "totalprice",
  "depositpaid",
  "bookingdates"
]
```

# Test Case

**Evaluate:** usada para avaliar uma expressão Python e retornar seu resultado. Isso permite a execução de código Python diretamente dentro do script do nosso teste.

A expressão "`random.choice($response.json())`" escolhe aleatoriamente um item retornado de "`response.json()`". Desse retorno, obtemos o valor do campo "`bookingid`". Para usar a função "`choice`", precisamos importar o módulo `random`.

```
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
> TC4: obter reserva por ID
    ${response} ... GET /booking ... 200
    ${bookingid} ... Evaluate ... random.choice($response.json()['bookingid']) ... modules=random

    ${response} ... GET On Session ... alias=${SESSION_NAME} ... url=/booking/${bookingid} ... expected_status=200
    Validate Json ... ${response} ... GetBookingIds.json
```

# Refatorando

Criamos uma palavra-chave que irá abstrair a ação de requisição GET por ID nos nossos testes.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***

> GET /${endpoint}...
GET /${endpoint}/${id}
[Arguments]    ${expected_status}
${response}    GET On Session    alias=${SESSION_NAME}    url=${endpoint}/${id}    expected_status=${expected_status}
RETURN    ${response}

> POST /${endpoint}...
> Validate Json ...
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
TC4: obter reserva por ID
${response}    GET /booking    200
${bookingid}    Evaluate    random.choice(${response.json()}['bookingid'])    modules=random

${response}    GET On Session    alias ${SESSION_NAME}    url /booking/${bookingid}    expected_status=200
Validate Json    ${response}    GetBookingIds.json
```

# Refatorando

Agora podemos usar a nossa nova palavra-chave.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***

> GET /${endpoint} ...
    GET /${endpoint}/${id}
    ... [Arguments] ... ${expected_status}
    ... ${response} ... GET On Session ... alias=${SESSION_NAME} ... url=${endpoint}/${id} ... expected_status=${expected_status}
    ... RETURN ... ${response}

> POST /${endpoint} ...
> Validate Json ...
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
TC4: obter reserva por ID
    ... ${response} ... GET /booking ... 200
    ... ${bookingid} ... Evaluate ... random.choice(${response.json()}['bookingid']) ... modules=random

    ... ${response} ... GET /booking/${bookingid} ... 200
    ... Validate Json ... ${response} ... GetBookingIds.json
```

# Refatorando

Também podemos abstrair a mesma lógica da seleção aleatória do bookingid.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***
> GET /${endpoint} ...
> GET /${endpoint}/${id} ...
> POST /${endpoint} ...
> Validate Json ...
Select Random BookingId From Booking List
    ${response}    GET /booking    200
    ${bookingid}   Evaluate    random.choice(${response.json()}['bookingid'])    modules=random
    RETURN    ${bookingid}

*** Test Cases ***
> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
    TC4: obter reserva por ID
        ${response}    GET /booking    200
        ${bookingid}   Evaluate    random.choice(${response.json()}['bookingid'])    modules=random
        ${response}    GET /booking/${bookingid}    200
        Validate Json    ${response}    GetBookingIds.json
```

# Refatorando

Agora podemos usar a nossa nova palavra-chave.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***
> GET /${endpoint} ...
> GET /${endpoint}/${id} ...
> POST /${endpoint} ...
> Validate Json ...
    Select Random BookingId From Booking List
        ${response}    GET /booking    200
        ${bookingid}   Evaluate    random.choice(${response.json()}['bookingid'])    modules=random
        RETURN    ${bookingid}

*** Test Cases ***
> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
    TC4: obter reserva por ID
        ${bookingid}    Select Random BookingId From Booking List
            ${response}    GET /booking/${bookingid}    200
            Validate Json    ${response}    GetBookingIds.json
```

DÚVIDAS?



# Payload

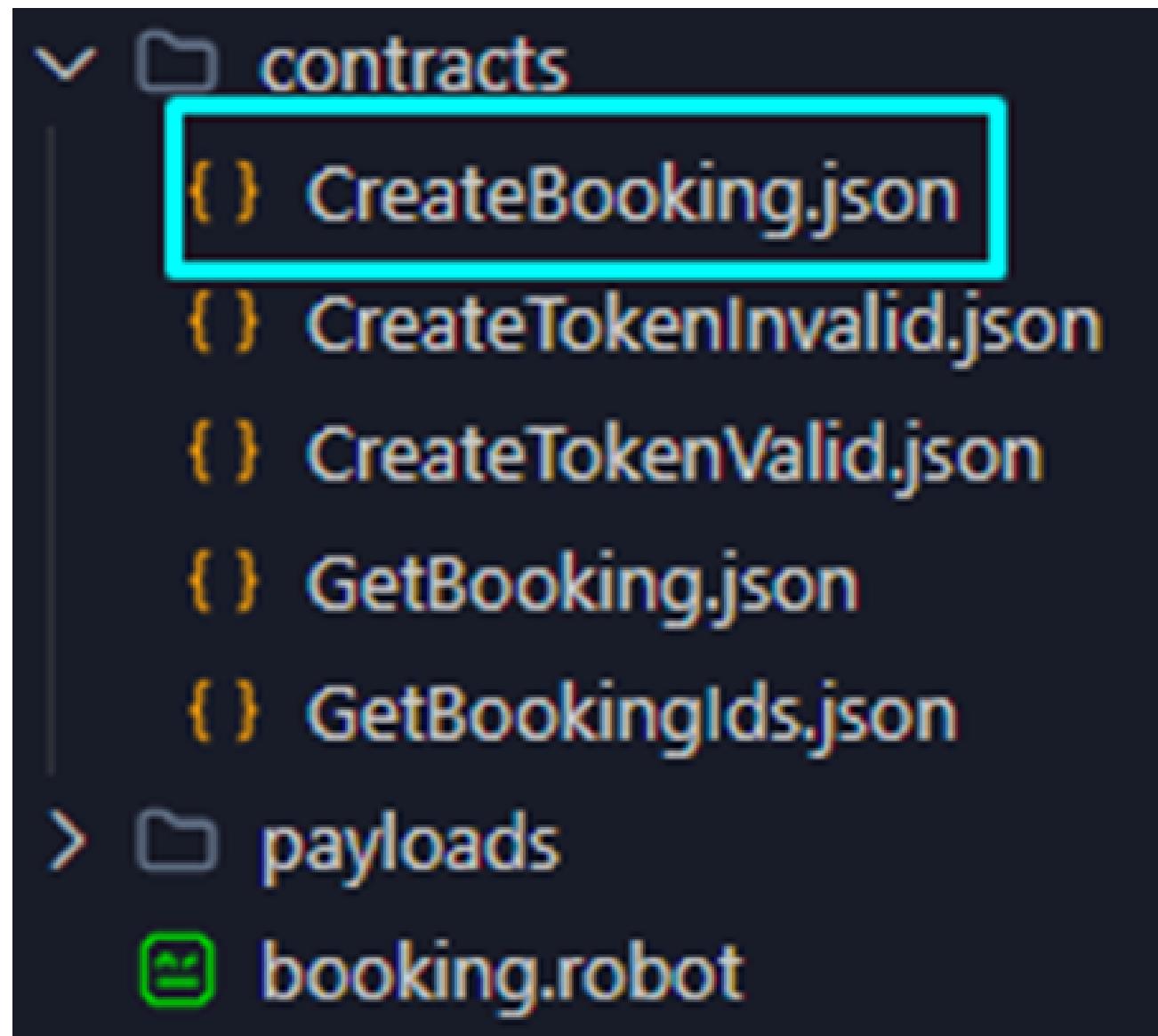
Crie o arquivo contendo o payload que será utilizado na requisição POST para o nosso teste.

```
> └── contracts
    └── payloads
        └── {} PostAuthInvalid.json
        └── {} PostAuthValid.json
        └── {} PostBooking.json
            └── booking.robot
```

```
{
  "firstname": "Felipe",
  "lastname": "Sutter",
  "totalprice": 100,
  "depositpaid": true,
  "bookingdates": {
    "checkin": "2024-03-01",
    "checkout": "2024-03-02"
  },
  "additionalneeds": "Sem exercícios"
}
```

# JSON Schema

Crie o arquivo que conterá o JSON Schema do retorno da requisição POST para o endpoint /booking



```
{ "$schema": "http://json-schema.org/draft-07/schema#", "type": "object", "properties": { "bookingId": { "type": "number" }, "booking": { "type": "object", "properties": { "firstname": { "type": "string" }, "lastname": { "type": "string" }, "totalprice": { "type": "number" }, "depositpaid": { "type": "boolean" }, "bookingdates": { "type": "object", "properties": { "checkin": { "type": "string" }, "checkout": { "type": "string" } } }, "required": [ "checkin", "checkout" ] }, "additionalneeds": { "type": "string" } }, "required": [ "firstname", "lastname", "totalprice", "depositpaid", "bookingdates" ] }, "required": [ "bookingId", "booking" ] }
```

# Test Case

**Set Variable:** utilizada para atribuir um valor a uma variável.

**Dictionaries Should Be Equal:** verifica se dois dicionários são iguais.

```
*** Settings ***
Library    Collections
Library    RequestsLibrary
Library    JSONLibrary

Suite Setup    Create Session    alias=${SESSION_NAME}    url=${BASE_URL}

> *** Variables ***
> *** Keywords ***
> *** Test Cases ***

> TC1: realizar autenticação com usuário válido...
> TC2: realizar autenticação com usuário inválido...
> TC3: obter reservas...
> TC4: obter reserva por ID...
> TC5: criar reserva
    ${response}    POST /booking    PostBooking.json    200
    Validate Json    ${response}    CreateBooking.json

    ${bookingid}    Set Variable    ${response.json()}[bookingid]
    ${booking}    Set Variable    ${response.json()}[booking]

    ${response}    GET /booking/${bookingid}    200
    Dictionaries Should Be Equal    ${booking}    ${response.json()}

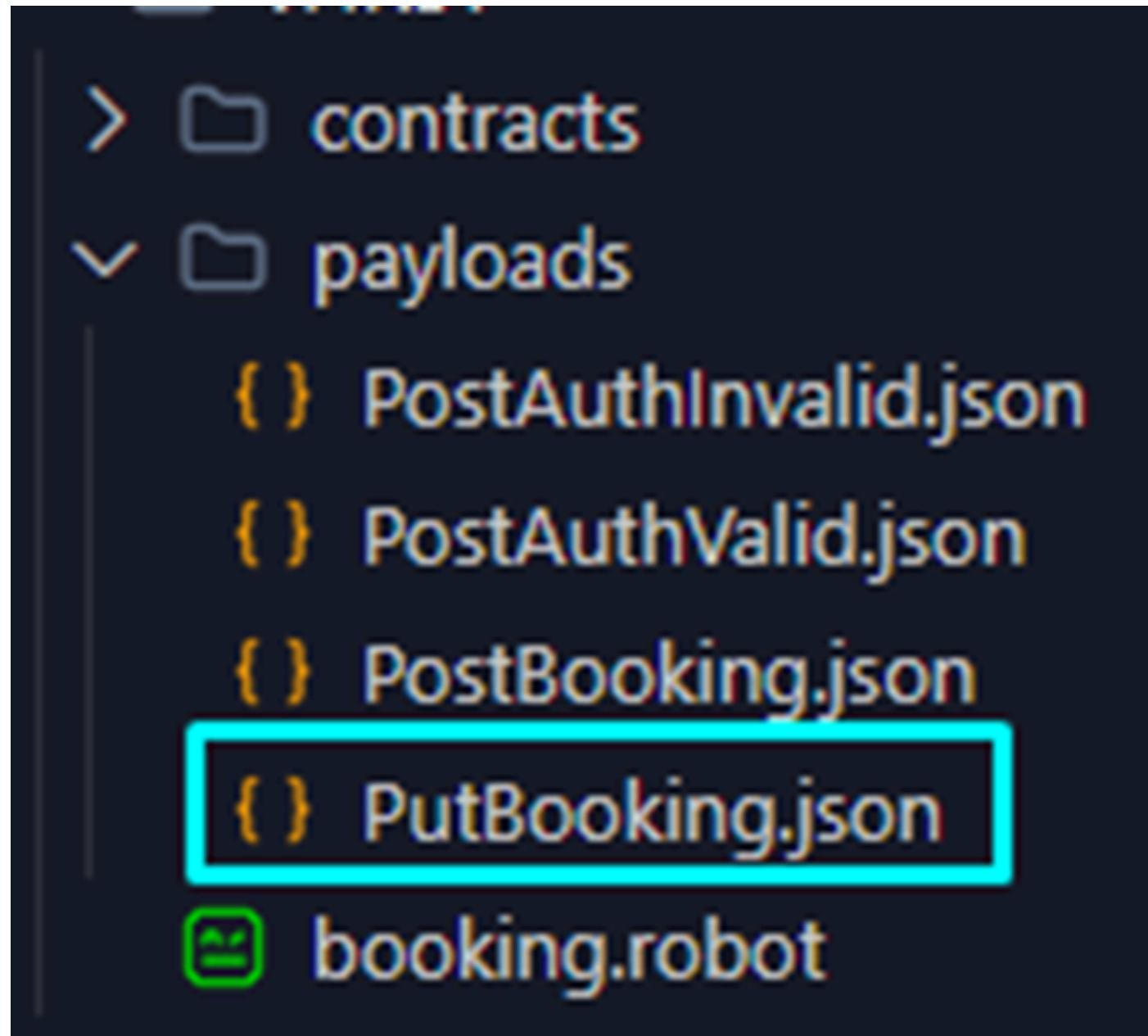

```

DÚVIDAS?



# JSON Schema

Crie o arquivo contendo o payload que será utilizado na requisição PUT para o nosso teste.

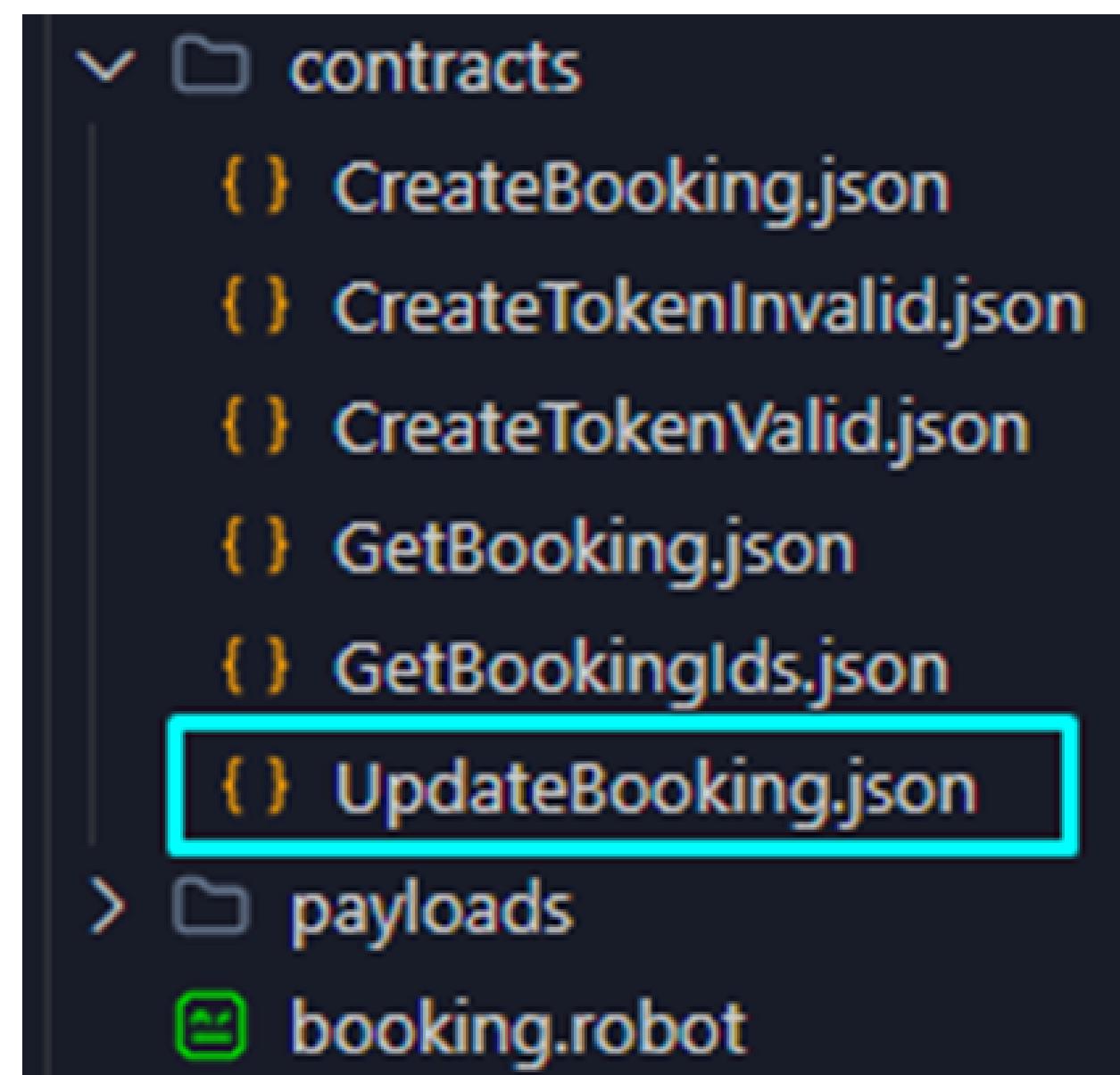


```
> contracts
payloads
  PostAuthInvalid.json
  PostAuthValid.json
  PostBooking.json
  PutBooking.json
  booking.robot
```

```
{
  "firstname": "Fulano",
  "lastname": "Testando",
  "totalprice": 1,
  "depositpaid": false,
  "bookingdates": {
    "checkin": "2024-02-08",
    "checkout": "2024-02-09"
  },
  "additionalneeds": "Nada"
}
```

# JSON Schema

Crie o arquivo que conterá o JSON Schema do retorno da requisição PUT para o endpoint /booking/:id



```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "firstname": {
      "type": "string"
    },
    "lastname": {
      "type": "string"
    },
    "totalprice": {
      "type": "number"
    },
    "depositpaid": {
      "type": "boolean"
    },
    "bookingdates": {
      "type": "object",
      "properties": {
        "checkin": {
          "type": "string"
        },
        "checkout": {
          "type": "string"
        }
      },
      "required": [
        "checkin",
        "checkout"
      ]
    },
    "additionalneeds": {
      "type": "string"
    }
  },
  "required": [
    "firstname",
    "lastname",
    "totalprice",
    "depositpaid",
    "bookingdates"
  ]
}
```

# Test Case

Criaremos um dicionário de cabeçalhos HTTP onde o campo Cookie é definido para incluir um token de autenticação.

**PUT On Session:** realiza uma requisição HTTP PUT utilizando uma sessão específica.

```
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
> TC4: obter reserva por ID ...
> TC5: criar reserva ...
> TC6: substituir reserva
    ${bookingid}.....Select Random BookingId From Booking List
    ${response}.....POST /auth.....PostAuthValid.json.....200
    ${token}.....Set Variable.....${response.json()}[token]

    ${payload}.....Load Json From File.....${PAYLOADS_DIR}${/}PutBooking.json
    ${headers}.....Create Dictionary.....Cookie=token\=${token}
    ${response}.....PUT On Session.....alias=${SESSION_NAME}.....url=/booking/${bookingid}.....json=${payload}.....headers=${headers}.....expected_status=200

    Validate Json.....${response}.....UpdateBooking.json

    ${newResponse}.....GET /booking/${bookingid}.....200
    Dictionaries Should Be Equal.....${newResponse.json()}.....${response.json()}


```

# Refatorando

Podemos abstrair a lógica para obter o token.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***

> GET /${endpoint}...
> GET /${endpoint}/${id}...
> POST /${endpoint}...
Get Token
    ${response}    POST /auth    PostAuthValid.json    200
    ${token}    Set Variable    ${response.json()[token]}
    RETURN    ${token}

> Validate Json...
> Select Random BookingId From Booking List...
*** Test Cases ***
TC1: realizar autenticação com usuário válido...
TC2: realizar autenticação com usuário inválido...
TC3: obter reservas...
TC4: obter reserva por ID...
TC5: criar reserva...
TC6: substituir reserva
    ${bookingid}    Select Random BookingId From Booking List

    ${response}    POST /auth    PostAuthValid.json    200
    ${token}    Set Variable    ${response.json()[token]}

    ${payload}    Load Json From File    ${PAYLOADS_DIR}${/}PutBooking.json
    ${headers}    Create Dictionary    Cookie=token\=${token}
    ${response}    PUT On Session    alias=${SESSION_NAME}    url=/booking/${bookingid}    json=${payload}    headers=${headers}    expected_status=200

    Validate Json    ${response}    UpdateBooking.json

    ${newResponse}    GET /booking/${bookingid}    200
    Dictionaries Should Be Equal    ${newResponse.json()}    ${response.json()}


```

# Refatorando

Criamos uma palavra-chave que irá abstrair a ação de requisição PUT nos nossos testes.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***

> GET /${endpoint}...
> GET /${endpoint}/${id}...
> POST /${endpoint}...
PUT /${endpoint}/${id}
[Arguments]  ${filename}  ${expected_status}
${token}  Get Token
${payload}  Load Json From File  ${PAYLOADS_DIR}${/}${filename}
${headers}  Create Dictionary  Cookie=token\-${token}
${response}  PUT On Session  alias=${SESSION_NAME}  url=${endpoint}/${id}  json=${payload}  headers=${headers}  expected_status=${expected_status}
RETURN  ${response}

Get Token
${response}  POST /auth  PostAuthValid.json  200
${token}  Set Variable  ${response.json()}[token]
RETURN  ${token}

> Validate Json...
> Select Random BookingId From Booking List...
*** Test Cases ***

> TC1: realizar autenticação com usuário válido...
> TC2: realizar autenticação com usuário inválido...
> TC3: obter reservas...
> TC4: obter reserva por ID...
> TC5: criar reserva...
TC6: substituir reserva
${bookingid}  Select Random BookingId From Booking List

${payload}  Load Json From File  ${PAYLOADS_DIR}${/}putBooking.json
${headers}  Create Dictionary  Cookie=token\-${token}
${response}  PUT On Session  alias=${SESSION_NAME}  url=/booking/${bookingid}  json=${payload}  headers=${headers}  expected_status=200

Validate Json  ${response}  UpdateBooking.json

${newResponse}  GET /booking/${bookingid}  200
Dictionaries Should Be Equal  ${newResponse.json()}  ${response.json()}


```

# Refatorando

Agora podemos usar a nossa nova palavra-chave.

```
> *** Settings ***
> *** Variables ***
*** Keywords **

> GET /${endpoint} ...
> GET /${endpoint}/${id} ...
> POST /${endpoint} ...
PUT /${endpoint}/${id}
    [Arguments]  ${filename}  ${expected_status}
    ${token}  Get Token
    ${payload}  Load Json From File  ${PAYLOADS_DIR}${/}${filename}
    ${headers}  Create Dictionary  Cookie=token\=${token}
    ${response}  PUT On Session  alias=${SESSION_NAME}  url=${endpoint}/${id}  json=${payload}  headers=${headers}  expected_status=${expected_status}
    RETURN  ${response}

Get Token
    ${response}  POST /auth  PostAuthValid.json  200
    ${token}  Set Variable  ${response.json()}[token]
    RETURN  ${token}

> Validate Json ...
> Select Random BookingId From Booking List ...
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
> TC4: obter reserva por ID ...
> TC5: criar reserva ...
TC6: substituir reserva
    ${bookingid}  Select Random BookingId From Booking List
    ${response}  PUT /booking/${bookingid}  PutBooking.json  200
    Validate Json  ${response}  UpdateBooking.json
    ${newResponse}  GET /booking/${bookingid}  200
    Dictionaries Should Be Equal  ${newResponse.json()}  ${response.json()}


```

DÚVIDAS?



# Payload

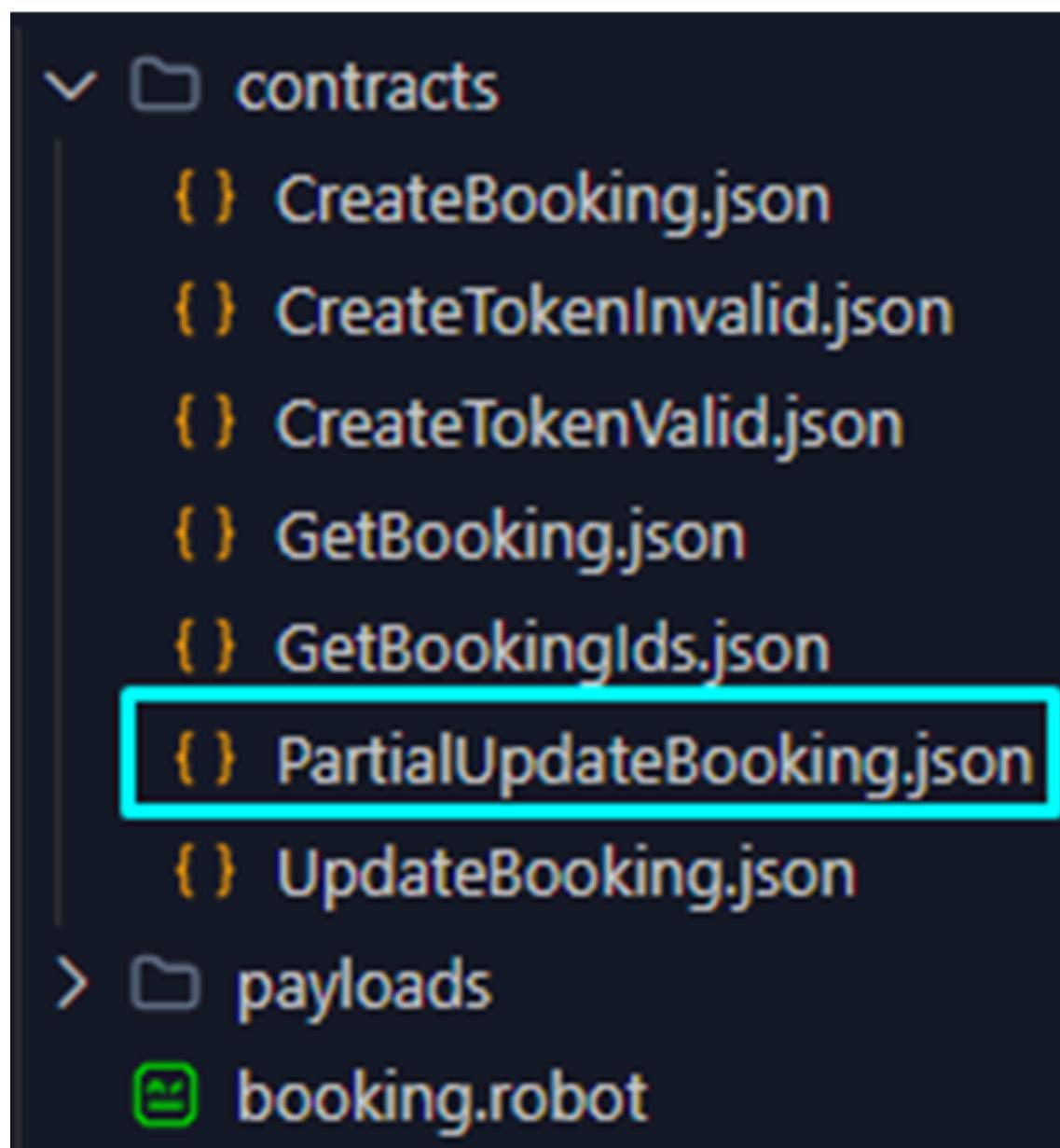
Crie o arquivo contendo o payload que será utilizado na requisição PATCH para o nosso teste.

```
> ⚒ contracts
  ✓ payloads
    { } PatchBooking.json
    { } PostAuthInvalid.json
    { } PostAuthValid.json
    { } PostBooking.json
    { } PutBooking.json
    📄 booking.robot
```

```
{
  "depositpaid": false
}
```

# JSON Schema

Crie o arquivo que conterá o JSON Schema do retorno da requisição PATCH para o endpoint /booking/:id



```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "firstname": {
      "type": "string"
    },
    "lastname": {
      "type": "string"
    },
    "totalprice": {
      "type": "number"
    },
    "depositpaid": {
      "type": "boolean"
    },
    "bookingdates": {
      "type": "object",
      "properties": {
        "checkin": {
          "type": "string"
        },
        "checkout": {
          "type": "string"
        }
      }
    },
    "required": [
      "checkin",
      "checkout"
    ]
  },
  "additionalProperties": {
    "type": "string"
  },
  "required": [
    "firstname",
    "lastname",
    "totalprice",
    "depositpaid",
    "bookingdates"
  ]
}
```

# Test Case

**PATCH On Session:** usada para realizar a requisição PATCH utilizando uma sessão HTTP específica (no caso, iqs).

**Dictionary Should Contain Sub Dictionary:** usado para verificar se um dicionário contém todos os itens de outro dicionário como um subconjunto.

```
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
> TC4: obter reserva por ID ...
> TC5: criar reserva ...
> TC6: substituir reserva ...
TC7: editar reserva
    ${bookingid} ..... Select Random BookingId From Booking List

    ${token} ..... Get Token
    ${payload} ..... Load Json From File ... ${PAYLOADS_DIR}${/}PatchBooking.json
    ${headers} ..... Create Dictionary ... Cookie=token\=${token}
    ${response} ..... PATCH On Session ... alias=${SESSION_NAME} ... url=booking/${bookingid} ... json=${payload} ... headers=${headers} ... expected_status=200

    Validate Json ... ${response} ... PartialUpdateBooking.json

    ${newResponse} ..... GET /booking/${bookingid} ... 200
    Dictionary Should Contain Sub Dictionary ... ${newResponse.json()} ... ${response.json()}
```

# Refatorando

Criamos uma palavra-chave que irá abstrair a ação de requisição PATCH nos nossos testes.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***

> GET /${endpoint}...
> GET /${endpoint}/${id}...
> POST /${endpoint}...
> PUT /${endpoint}/${id}...
PATCH /${endpoint}/${id}
    [Arguments]  ${filename}      ${expected_status}
    ${token}      Get Token
    ${payload}    Load Json From File  ${PAYLOADS_DIR}(/)${filename}
    ${headers}   Create Dictionary  Cookie=token\=${token}
    ${response}  PATCH On Session  alias=${SESSION_NAME}  url=${endpoint}/${id}  json=${payload}  headers=${headers}  expected_status=${expected_status}
    RETURN      ${response}

> Get Token...
> Validate Json...
> Select Random BookingId From Booking List...
*** Test Cases ***
> TC1: realizar autenticação com usuário válido...
> TC2: realizar autenticação com usuário inválido...
> TC3: obter reservas...
> TC4: obter reserva por ID...
> TC5: criar reserva...
> TC6: substituir reserva...
TC7: editar reserva
    ${bookingid}  Select Random BookingId From Booking List

    ${token}      Get Token
    ${payload}    Load Json From File  ${PAYLOADS_DIR}(/)PatchBooking.json
    ${headers}   Create Dictionary  Cookie=token\=${token}
    ${response}  PATCH On Session  alias=${SESSION_NAME}  url=booking/${bookingid}  json=${payload}  headers=${headers}  expected_status=200
    Validate Json  ${response}  PartialUpdateBooking.json

    ${newResponse}  GET /booking/${bookingid}  200
    Dictionary Should Contain Sub Dictionary  ${newResponse.json()}  ${response.json()}
```

# Refatorando

Agora podemos usar a nossa nova palavra-chave.

```
> *** Settings ***
> *** Variables ***
*** Keywords ***

> GET /${endpoint}...
> GET /${endpoint}/${id}...
> POST /${endpoint}...
> PUT /${endpoint}/${id}...
PATCH /${endpoint}/${id}
    [Arguments]  ${filename}..... ${expected_status}
    ${token}..... Get Token
    ${payload}..... Load Json From File  ${PAYLOADS_DIR}${/}${filename}
    ${headers}..... Create Dictionary  Cookie=token\=${token}
    ${response}..... PATCH On Session  alias=${SESSION_NAME}  url=${endpoint}/${id}  json=${payload}  headers=${headers}  expected_status=${expected_status}
    RETURN  ${response}

> Get Token ...
> Validate Json ...
> Select Random BookingId From Booking List ...
*** Test Cases ***

> TC1: realizar autenticação com usuário válido...
> TC2: realizar autenticação com usuário inválido...
> TC3: obter reservas...
> TC4: obter reserva por ID...
> TC5: criar reserva...
> TC6: substituir reserva...
TC7: editar reserva
    ${bookingid}  Select Random BookingId From Booking List
    ${response}..... PATCH /booking/${bookingid}  PatchBooking.json  200
    Validate Json  ${response}  PartialUpdateBooking.json

    ${newResponse}..... GET /booking/${bookingid}  200
    Dictionary Should Contain Sub Dictionary  ${newResponse.json()}  ${response.json()}


```

DÚVIDAS?



# Test Case

**DELETE On Session:** usada para realizar a requisição DELETE utilizando uma sessão HTTP específica (no caso, iqs).

```
*** Test Cases ***

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
> TC4: obter reserva por ID ...
> TC5: criar reserva ...
> TC6: substituir reserva ...
> TC7: editar reserva ...
TC8: deletar reserva
    ${bookingid} ..... Select Random BookingId From Booking List

    ${token} ..... Get Token
    &{headers} ..... Create Dictionary .. Cookie=token\=${token}
    ${response} ..... DELETE On Session .. alias=${SESSION_NAME} .. url=/booking/${bookingid} .. headers=${headers} .. expected_status=201 .. #! RFC9110

    GET /booking/${bookingid} .. 404

    ${response} ..... GET /booking .. 200
    Should Not Have Value In Json .. ${response.json()} .. json_path=$[?(@.bookingid == ${bookingid})].bookingid
```

# JSON Path

O JSON Path é uma linguagem de consulta para documentos JSON, similar ao XPath. Permite navegar através de um documento JSON para selecionar e extrair informação. Link: <https://jsonpath.com/>

Should Not Have Value In Json: verifica se um valor não está presente em um JSON Path especificado.

```
> *** Settings ***
> *** Variables ***
> *** Keywords ***
> *** Test Cases ***

> TC1: realizar autenticação com usuário válido...
> TC2: realizar autenticação com usuário inválido...
> TC3: obter reservas...
> TC4: obter reserva por ID...
> TC5: criar reserva...
> TC6: substituir reserva...
> TC7: editar reserva...
> TC8: deletar reserva
    ${bookingid}.....Select Random BookingId From Booking List

    ${token}.....Get Token
    &{headers}.....Create Dictionary.....Cookie=token\-${token}
    ${response}.....DELETE On Session.....alias=${SESSION_NAME}.....url=/booking/${bookingid}.....headers=${headers}.....expected_status=201.....#! RFC9110
    GET /booking/${bookingid}.....404

    ${response}.....GET /booking.....200
    Should Not Have Value In Json.....${response.json()}.....json_path=${[?(@.bookingid == ${bookingid})].bookingid}
```

# Refatorando

Criamos uma palavra-chave que irá abstrair a ação de requisição DELETE nos nossos testes.

```
> *** Settings ***
> *** Variables ***
*** Keywords **

> GET /${endpoint}...
> GET /${endpoint}/${id}...
> POST /${endpoint}...
> PUT /${endpoint}/${id}...
> PATCH /${endpoint}/${id}...
DELETE /${endpoint}/${id}
[Arguments]  ${expected_status}
${token}  Get Token
&{headers}  Create Dictionary  Cookie=token\-${token}
${response}  DELETE On Session  alias=${SESSION_NAME}  url=${endpoint}/${id}  headers=${headers}  expected_status=${expected_status}
RETURN  ${response}

> Get Token ...
> Validate Json ...
> Select Random BookingId From Booking List ...
*** Test Cases **

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
> TC4: obter reserva por ID ...
> TC5: criar reserva ...
> TC6: substituir reserva ...
> TC7: editar reserva ...
TC8: deletar reserva
${bookingid}  Select Random BookingId From Booking List

${token}  Get Token
&{headers}  Create Dictionary  Cookie=token\-${token}
${response}  DELETE On Session  alias=${SESSION_NAME}  url=/booking/${bookingid}  headers=${headers}  expected_status=201  #| RFC9110

GET /booking/${bookingid}  404

${response}  GET /booking  200
Should Not Have Value In Json  ${response.json()}  json_path=${?(@.bookingid == ${bookingid})}.bookingid
```

# Refatorando

Agora podemos usar a nossa nova palavra-chave.

```
> *** Settings ***
> *** Variables ***
*** Keywords **

> GET /${endpoint}...
> GET /${endpoint}/${id}...
> POST /${endpoint}...
> PUT /${endpoint}/${id}...
> PATCH /${endpoint}/${id}...
DELETE /${endpoint}/${id}
    [Arguments]  ${expected_status}
    ${token}      Get Token
    &{headers}   Create Dictionary  Cookie=token\=${token}
    ${response}   DELETE On Session  alias=${SESSION_NAME}  url=${endpoint}/${id}  headers=${headers}  expected_status=${expected_status}
    RETURN  ${response}

> Get Token ...
> Validate Json ...
> Select Random BookingId From Booking List ...
*** Test Cases **

> TC1: realizar autenticação com usuário válido ...
> TC2: realizar autenticação com usuário inválido ...
> TC3: obter reservas ...
> TC4: obter reserva por ID ...
> TC5: criar reserva ...
> TC6: substituir reserva ...
> TC7: editar reserva ...
TC8: deletar reserva
    ${bookingid}  Select Random BookingId From Booking List
    DELETE /booking/${bookingid}  201  #! RFC9110
    GET /booking/${bookingid}  404
    ${response}  GET /booking  200
    Should Not Have Value In Json  ${response.json()}  json_path=${?(@.bookingid == ${bookingid}).bookingid}
```

DÚVIDAS?



# DOCUMENTAÇÃO

## Robot Framework

<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html>

## RequestLibrary

<https://marketsquare.github.io/robotframework-requests/doc/RequestsLibrary.html>

## JSONLibrary

<https://robotframework-thailand.github.io/robotframework-jsonlibrary/JSONLibrary.html>

# OBRIGADO!

