**Pré-requisitos:** Possuir conhecimentos da ferramenta Robot Framework e da linguagem de programação Python.

Dicas de estudos em Robot Framework:

* Artigo com explicação resumida sobre o funcionamento do Robot Framework: <https://imasters.com.br/back-end/testes-web-com-robot-framework-para-python-introducao>
* Vídeo de 3h de duração, com resumo e demonstração de funcionamento do Robot Framework: <https://youtu.be/r-Ju-O_miv0>
* Curso Básico de Robot Framework,na Udemy: <https://www.udemy.com/course/automacao-de-testes-com-robot-framework-basico/> (Não é gratuito)
* Curso de Robot Framework, gratuito na QANinja: <https://qaninja.academy/curso/robot-beginner/>
* Manual do usuário robot: <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html>
* Site oficial: <https://robotframework.org/>
* [Vídeo interno, mostrando como iniciar um teste de API com o Robot Framework:](https://robot-framework.readthedocs.io/en/latest/autodoc/robot.api.html" \l "module-robot.api.deco)

Links das documentações das bibliotecas utilizadas:

Robot:

* RequestsLibrary: <http://marketsquare.github.io/robotframework-requests/doc/RequestsLibrary.html>
* Collections: <https://robotframework.org/robotframework/2.1.2/libraries/Collections.html>
* FakerLibrary: <https://guykisel.github.io/robotframework-faker/>

Python:

* Json: <https://docs.python.org/pt-br/3/library/json.html>#
* Jsonschema: <https://python-jsonschema.readthedocs.io/en/stable/>
* Requests: <https://requests.readthedocs.io/en/latest/>
* Robot.api.deco: [https://robot-framework.readthedocs.io/en/latest/autodoc/robot.api.html#module-robot.api.deco](https://robot-framework.readthedocs.io/en/latest/autodoc/robot.api.html" \l "module-robot.api.deco)

**Objetivos do Projeto:**

* Fornecer scripts genéricos e adaptáveis a qualquer endpoint da API v4.
* Garantir a saúde da API, contendo testes capazes de enviar um conjunto de requisições definidas e analisar cada resposta retornada, comparando-se com o comportamento esperado, registrado em documentação.

**Escopo:**

Para definir o escopo do projeto, foi utilizada a heurística VADER, que é a sigla para as seguintes palavras: Verbs (Verbos), Authorization (Autorização), Data (Dados), Errors (Erros) e Responsiveness (Capacidade de resposta). Para saber mais informações, a leitura deste artigo é recomendada: <https://medium.com/revista-tspi/chama-o-darth-vader-para-te-ajudar-a-testar-as-apis-4fafc5846b32>

A heurística nos auxilia a definir, de uma forma padronizada, o que deve ser testado em uma API. Testando todos os pontos indicados podemos verificar que a API está minimamente saudável e pronta para ser utilizada.

Para aplicação específica na API v4, chegou-se ao seguinte mapa mental:



Pensando em atender esses requisitos, foi definido que seriam feitos quatro templates de testes, baseados no Verbo, ou Método HTTP (GET, PUT, POST e DELETE), definindo os testes que devem ser feitos em cada endpoint compatíveis com o método.

Dentro de cada template específico para o método, há um conjunto de testes que consideram os demais pontos definidos pela heurística VADER.

**VADER na Prática**

Como a Heurística VADER é aplicada na prática nos testes da API v4?

**V (Verbos):** Abrangidos através de todos os testes de valores válidos, pois verificam se o endpoint é capaz de receber a requisição no método a que se propõe, e que é capaz de retornar os valores esperados.

**A (Autorização/Autenticação):** Abrangidos nos testes que esperam acesso negado em caso de autenticação inválida, e também nos testes básicos de envio de requisição válida, pois verificam o acesso permitido com a autenticação configurada corretamente.

**D (Dados):** Abrangidos através da comparação com o Json Schema documentado na análise do body de resposta, e também nos testes de envio de requisição com formatos json inválidos.

**E (Erros):** Abrangidos através de todos os testes com valores inválidos e que esperam que a requisição seja negada, pois estes analisam se a resposta contém os valores de status code e body informando corretamente o motivo do erro. Alguns exemplos são: testes de ausência de campos obrigatórios, testes e envio de campos com valores inválidos, etc.

**R (Responsividade):** Abrangida de forma implícita, através da configuração do timeout no envio de cada requisição. Não há teste específico para responsividade, mas caso a requisição demore mais que o configurado, os testes falharão informando timeout excedido.

**Não escopo:**

Os testes do presente projeto não abrangem o funcionamento do produto final, bem como suas características e comportamentos específicos, se limitando a verificar o comportamento geral da API. Porém, os testes podem servir de auxílio aos testadores do produto final, caso estes façam as devidas adaptações.

**Estrutura do Projeto de Testes da API v4**

**Consulta automática da documentação:**

A abordagem de testes escolhida utiliza a consulta automática da documentação para definir os valores esperados em cada endpoint. Esta opção nos traz os seguintes benefícios:

* Permite a utilização dos templates de testes em variados endpoints que contenham variados comportamentos, com necessidade miníma de adaptações.
* Retira a necessidade de armazenar os valores esperados em variáveis dentro do scripts de testes.
* É necessário apenas que a documentação seja alterada em casos de mudança de comportamento da API, pois o teste se adaptará dinamicamente.

As seguintes informações provenientes da documentação são utilizadas:

* Status code esperado
* Headers esperados na requisição
* Headers esperados na resposta
* Body válido de exemplo
* JSON Schema da requisição
* JSON Schema da resposta

Então, é necessário que a documentação tenha essas informações registradas para que os testes funcionem corretamente. A documentação consultada se encontra no seguinte link: <https://jsapi.apiary.io/apis/apiintelbrasv4>, sendo que seus valores equivalem ao registrado a API Blueprint, que se encontra aqui: <https://git.intelbras.com.br/zeus-api/api-blueprint/-/tree/api_v4>.

Caso o teste falhe devido a não encontrar esses valores da documentação, converse com os desenvolvedores envolvidos na API v4 sobre a inclusão das informações na API Blueprint.

O JSON Schema é a mais complexa das informações necessárias para o teste, porém, ele é essencial para saber as regras que os dados enviados e recebidos devem seguir. Para saber mais sobre JSON Schema, consulte a documentação: <https://json-schema.org/understanding-json-schema/>

**Os Testes**

**Testes Requisição GET:**

**GET is Succeed:** Este teste envia uma requisição GET para o endpoint de teste, configurando a autenticação de forma correta. Então analisa o status code, os headers e o body retornado, comparando com o comportamento esperado, registrado na documentação.

**GET Without Authentication:** Este teste envia uma requisição GET para o endpoint de teste, mas não configura a autenticação nos headers da requisição. Então espera que a requisição seja recusada por acesso não autorizado.

**GET With Empty Authentication:** Este teste envia uma requisição GET para o endpoint de teste, configura a autenticação nos headers da requisição, porém com seu valor vazio. Então espera que a requisição seja recusada por acesso não autorizado.

**GET With Wrong Authentication:** Este teste envia uma requisição GET para o endpoint de teste, configura a autenticação nos headers da requisição, porém com um token inválido. Então espera que a requisição seja recusada por acesso não autorizado.

**Testes Requisição PUT:**

**PUT is Succeed:** Este teste envia uma requisição PUT para o endpoint de teste, configurando a autenticação de forma correta, usando um body válido na requisição, que é o body de exemplo registrado na documentação. Então analisa o status code e os headers retornados, comparando com o comportamento esperado, registrado na documentação. Também espera que o body de retorno seja vazio, pois não é previsto que as requisições PUT tenham body de retorno na API v4.

**PUT Without Authentication:** Este teste envia uma requisição PUT para o endpoint de teste, usando um body válido na requisição, que é o body de exemplo registrado na documentação, mas não configura a autenticação nos headers da requisição. Então espera que a requisição seja recusada por acesso não autorizado.

**PUT With Empty Authentication:** Este teste envia uma requisição PUT para o endpoint de teste, usando um body válido na requisição, que é o body de exemplo registrado na documentação, e configura a autenticação nos headers da requisição, porém com seu valor vazio. Então espera que a requisição seja recusada por acesso não autorizado.

**PUT With Wrong Authentication:** Este teste envia uma requisição PUT para o endpoint de teste, usando um body válido na requisição, que é o body de exemplo registrado na documentação, e configura a autenticação nos headers da requisição, porém com um token inválido. Então espera que a requisição seja recusada por acesso não autorizado.

**PUT Without Required Keys:** Teste separa em uma lista quais são os campos obrigatórios, indicados como “required” no Json Schema documentado, a seguir inicia um loop onde cada campo obrigatório é retirado do body de exemplo extraído da documentação, e a requisição é enviada. Então, espera que a requisição seja recusada e retorne erro devido a ausência de campo obrigatório.

**PUT Without Optional Keys:** Teste separa em uma lista quais são os campos opcionais, que não foram indicados como “required” no Json Schema documentado, a seguir inicia um loop onde cada campo opcional é retirado do body de exemplo extraído da documentação, e a requisição é enviada. Então, espera que a requisição seja aceita e retorne o status esperado em uma requisição bem sucedida.

**PUT With Wrong Type Values:** Teste passa por todos os campos previstos no endpoint, substituindo os valores válidos do body de exemplo por valores com tipos inválidos e enviando as requisições. Então, espera que a requisição seja recusada e retorne erro devido ao recebimento de tipo inválido.

**PUT With Valid Boolean Values:** Teste identifica quais campos são do tipo boolean e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, primeiramente enviando o body de requisição contendo o valor oposto ao da configuração atual (ex: se atualmente campo está definido como “true”, teste enviará o valor “false”, e vice-versa) e depois enviando o outro valor booleano restante. Em ambos os casos o teste espera que a requisição seja aceita e retorne o status esperado em uma requisição bem sucedida. Caso não hajam campos boolean no endpoint, a execução “pula” este teste.

**PUT With Valid Enum Values:** Teste identifica quais campos são do tipo enum e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo cada valor válido do range enum, um a um. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos enum no endpoint, a execução “pula” este teste.

**PUT With Valid Minimum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “minLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres mínima permitida para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “minLength” no endpoint, a execução “pula” este teste.

**PUT With Valid Maximum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “maxLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres máxima permitida para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “maxLength” no endpoint, a execução “pula” este teste.

**PUT With Invalid Less Than Minimum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “minLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres mínima permitida para este campo menos um, ou seja com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de número de caracteres inferior ao permitido. Caso não hajam campos com o parâmetro “minLength” no endpoint, a execução “pula” este teste. Como não existem quantidades de caracteres negativas, teste também “pula” campos com “minLength” iguais a zero.

**PUT With Invalid Greater Than Maximum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “maxLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres máxima permitida para este campo mais um, ou seja, com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de número de caracteres superior ao permitido. Caso não hajam campos com o parâmetro “maxLength” no endpoint, a execução “pula” este teste.

**PUT With Valid Minimum Integer Values:** Teste identifica quais campos contém o parâmetro “minimum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro mínimo permitido para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “minimum” no endpoint, a execução “pula” este teste.

**PUT With Valid Maximum Integer Values:** Teste identifica quais campos contém o parâmetro “maximum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro máximo permitido para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “maximum” no endpoint, a execução “pula” este teste.

**PUT With Invalid Less Than Minimum Integer Values:** Teste identifica quais campos contém o parâmetro “minimum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro mínimo permitido para este campo menos um, ou seja com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de valor numérico inferior ao permitido. Caso não hajam campos com o parâmetro “minimum” no endpoint, a execução “pula” este teste.

**PUT With Invalid Greater Than Maximum Integer Values:** Teste identifica quais campos contém o parâmetro “maximum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro máximo permitido para este campo mais um, ou seja com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de valor numérico superior ao permitido. Caso não hajam campos com o parâmetro “maximum” no endpoint, a execução “pula” este teste.

**PUT With Invalid Json Data:** Teste desconfigura a estrutura json correta do body de exemplo e envia requisições com o formato json inválido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de json com formato inválido.

**Testes requisição POST:**

**POST is Succeed:** Este teste envia uma requisição POST para o endpoint de teste, configurando a autenticação de forma correta, usando um body válido na requisição, que é o body de exemplo registrado na documentação. Então analisa o status code e os headers retornados, comparando com o comportamento esperado, registrado na documentação. Em alguns endpoints é esperado que o body de retorno seja vazio, em outros que retorne o “id” do recurso criado ou alguma outra informação, de acordo com o definido na documentação.

**POST Without Authentication:** Este teste envia uma requisição POST para o endpoint de teste, usando um body válido na requisição, que é o body de exemplo registrado na documentação, mas não configura a autenticação nos headers da requisição. Então espera que a requisição seja recusada por acesso não autorizado.

**POST With Empty Authentication:** Este teste envia uma requisição POST para o endpoint de teste, usando um body válido na requisição, que é o body de exemplo registrado na documentação, e configura a autenticação nos headers da requisição, porém com seu valor vazio. Então espera que a requisição seja recusada por acesso não autorizado.

**POST With Wrong Authentication:** Este teste envia uma requisição POST para o endpoint de teste, usando um body válido na requisição, que é o body de exemplo registrado na documentação, e configura a autenticação nos headers da requisição, porém com um token inválido. Então espera que a requisição seja recusada por acesso não autorizado.

**POST Without Required Keys:** Teste separa em uma lista quais são os campos obrigatórios, indicados como “required” no Json Schema documentado, a seguir inicia um loop onde cada campo obrigatório é retirado do body de exemplo extraído da documentação, e a requisição é enviada. Então, espera que a requisição seja recusada e retorne erro devido a ausência de campo obrigatório.

**POST Without Optional Keys:** Teste separa em uma lista quais são os campos opcionais, que não foram indicados como “required” no Json Schema documentado, a seguir inicia um loop onde cada campo opcional é retirado do body de exemplo extraído da documentação, e a requisição é enviada. Então, espera que a requisição seja aceita e retorne o status esperado em uma requisição bem sucedida. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With Wrong Type Values:** Teste passa por todos os campos previstos no endpoint, substituindo os valores válidos do body de exemplo por valores com tipos inválidos e enviando as requisições. Então, espera que a requisição seja recusada e retorne erro devido ao recebimento de tipo inválido.

**POST With True Value in Boolean Fields:** Teste identifica quais campos são do tipo boolean e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor “true” no campo testado. Então, espera que a requisição seja aceita e retorne o status esperado em uma requisição bem sucedida. Caso não hajam campos boolean no endpoint, a execução “pula” este teste. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With False Value in Boolean Fields:** Teste identifica quais campos são do tipo boolean e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor “false” no campo testado. Então, espera que a requisição seja aceita e retorne o status esperado em uma requisição bem sucedida. Caso não hajam campos boolean no endpoint, a execução “pula” este teste. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With Valid Enum Values:** Teste identifica quais campos são do tipo enum e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo cada valor válido do range enum, um a um. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos enum no endpoint, a execução “pula” este teste. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With Valid Minimum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “minLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres mínima permitida para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “minLength” no endpoint, a execução “pula” este teste. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With Valid Maximum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “maxLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres máxima permitida para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “maxLength” no endpoint, a execução “pula” este teste. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With Invalid Less Than Minimum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “minLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres mínima permitida para este campo menos um, ou seja com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de número de caracteres inferior ao permitido. Caso não hajam campos com o parâmetro “minLength” no endpoint, a execução “pula” este teste. Como não existem quantidades de caracteres negativas, teste também “pula” campos com “minLength” iguais a zero.

**POST With Invalid Greater Than Maximum Quantity of Characters:** Teste identifica quais campos contém o parâmetro “maxLength” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo um valor com a quantidade de caracteres máxima permitida para este campo mais um, ou seja, com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de número de caracteres superior ao permitido. Caso não hajam campos com o parâmetro “maxLength” no endpoint, a execução “pula” este teste.

**POST With Valid Minimum Integer Values:** Teste identifica quais campos contém o parâmetro “minimum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro mínimo permitido para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “minimum” no endpoint, a execução “pula” este teste. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With Valid Maximum Integer Values:** Teste identifica quais campos contém o parâmetro “maximum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro máximo permitido para este campo. Então, o teste espera que as requisições sejam aceitas e retornem o status esperado em uma requisição bem sucedida. Caso não hajam campos com o parâmetro “maximum” no endpoint, a execução “pula” este teste. Em cada loop o recurso criado é deletado, para que não interfira nos próximos testes.

**POST With Invalid Less Than Minimum Integer Values:** Teste identifica quais campos contém o parâmetro “minimum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro mínimo permitido para este campo menos um, ou seja com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de valor numérico inferior ao permitido. Caso não hajam campos com o parâmetro “minimum” no endpoint, a execução “pula” este teste.

**POST With Invalid Greater Than Maximum Integer Values:** Teste identifica quais campos contém o parâmetro “maximum” e os separa em uma lista, a seguir, teste passa por todos os campos desta lista, enviando o body de requisição contendo o valor inteiro máximo permitido para este campo mais um, ou seja com o valor inválido mais próximo ao limite válido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de valor numérico superior ao permitido. Caso não hajam campos com o parâmetro “maximum” no endpoint, a execução “pula” este teste.

**POST Create New Resources Until Limit Value:** Teste identifica se endpoint suporta receber múltiplos recursos, caso sim, envia requisições POST até que o valor máximo de recursos seja atingido. Na criação de cada um deles, o teste espera que o status code, headers e body retornados sejam os esperados e registrados na documentação, não havendo erro em nenhuma das requisições. Ao final do teste, todos os recursos são deletados, para que não interfiram nos próximos testes. Caso endpoint não trabalhe com múltiplos recursos, a execução “pula” este teste.

**POST Create New Resources Exceeding Limit Value:** Teste identifica se endpoint suporta receber múltiplos recursos, caso sim, envia requisições POST até que o valor máximo de recursos seja excedido. Na última requisição, teste espera que essa seja recusada e retorne erro devido ao número limite de recursos excedido. Ao final do teste, todos os recursos são deletados, para que não interfiram nos próximos testes. Caso endpoint não trabalhe com múltiplos recursos, a execução “pula” este teste.

**POST With Invalid Json Data:** Teste desconfigura a estrutura json correta do body de exemplo e envia requisições com o formato json inválido. Então, o teste espera que a requisição seja recusada e retorne erro devido ao recebimento de json com formato inválido.

**Testes Requisição DELETE:**

**DELETE Multiple Resources:** Teste identifica se objetivo do delete para este endpoint seria excluir apenas um ou múltiplos recursos, caso seja deletar multiplos recursos, o teste cria estes recursos através de requisição POST, e depois envia a requisição DELETE para o endpoint. Então, teste confere se Status Code e Headers retornados são os esperados, confere também se todos os recursos do endpoint foram excluídos, exceto os recursos default. Caso endpoint não trabalhe com múltiplos recursos, a execução “pula” este teste.

**DELETE Created Resource:** Teste identifica se objetivo do delete para este endpoint seria excluir apenas um ou múltiplos recursos, caso seja apenas um, ou seja, um recurso específico, o teste cria este único recurso através de requisição POST, e depois envia a requisição DELETE para o endpoint. Então, teste confere se Status Code e Headers retornados são os esperados, confere também se o recurso do endpoint foi excluído. Caso endpoint não trabalhe com um único recurso específico, a execução “pula” este teste.

**DELETE Default Resource:** Teste identifica se além do objetivo do delete para este endpoint ser excluir apenas um recursos, se este endpoint contém algum recurso nativo, ou seja, definido como default e que não pode ser deletado. O teste envia a requisição DELETE para o endpoint que contém o recurso default, então, espera que a requisição seja recusada e retorne erro devido a tentativa de excluir um recurso default. Caso endpoint não trabalhe com um único recurso específico ou não contenha recurso default, a execução “pula” este teste.

**Resources:**

O projeto contém em cinco recursos Robot e três recursos Python, seus objetivos são:

* **common\_resource.resource (Robot):** Recurso destinado a agrupar os demais recursos e bibliotecas necessários no projeto, assim cada teste precisa apenas importar o common\_resource, sem a necessidade de importar cada recurso separadamente. Também armazena as variáveis comuns a todos os testes, como endereço da API e as credenciais de acesso ao DUT.
* **documentation\_analysis.resource (Robot):** Recurso destinado a análise da documentação, definindo os valores esperados em cada teste e identificando características do endpoint testado. Trabalha em auxílio com a find\_documentation\_info.py.
* **configure\_request\_data.resource (Robot):** Recurso destinado a manipular os dados antes de enviá-los a API, configurando os headers, o body de requisição e/ou fazendo mais configurações desejadas. Temos um body válido de exemplo proveniente da documentação, porém, geralmente é necessário alterá-lo para fazer os variados testes, fazendo com que as keywords contidas nesse recurso sejam necessárias.
* **send\_requests\_to\_api.resource (Robot):** Este recurso contém keywords que enviam requisições a API, sejam do método GET, PUT, POST ou DELETE. A maioria das keywords utilizam recursos da configure\_request\_data.resource, pois é necessário que os dados sejam manipulados antes do envio.
* **api\_response\_analysis.resource (Robot):** Contém keywords que analisam as respostas das requisições e comparam com os valores esperados. A maioria delas depende das variáveis que foram definidas em documentation\_analysis.resource, para os resultados esperados, e da variável que armazena a resposta da API, definida através das keyword de send\_requests\_to\_api.resource.
* find\_documentation\_info.py (Python)
* api\_service.py (Python)
* api\_variables.py (Python)