MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

PROGRAMA DE RESIDÊNCIA EM TECNOLOGIA DA INFORMAÇÃO
TRIBUNAL DE CONTAS DO ESTADO DO RIO GRANDE DO NORTE

Aplicando BDD em Testes de REST API: Uma Experiência Prática

Washington Luiz da Silva Lima

Trabalho de Conclusão de Curso apresentado ao
Programa de Residência em Tecnologia da Informação
da Universidade Federal do Rio Grande do Norte
como requisito parcial para a obtenção do grau
de Residente em Tecnologia da Informação.

Orientador
Prof. Dr. Uirá Kulesza

Natal, RN
Junho de 2022

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Lima, Washington Luiz da Silva.
Aplicando BDD em testes de REST API: uma experiência prática / Washington Luiz da Silva Lima. - 2022.
14f.: il.

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte, Instituto Metrópole Digital, Residência em Tecnologia da Informação - TCE/RN, Natal, 2022.
Orientador: Dr. Uirá Kulesza.

1. Behavior-Driven Development (BDD) - Dissertação. 2. Teste de API - Dissertação. 3. Gherkin - Dissertação. 4. Cucumber - Dissertação. 5. Python Requests - Dissertação. I. Kulesza, Uirá. II. Título.

RN/UF/BCZM CDU 004

Elaborado por Raimundo Muniz de Oliveira - CRB-15/429

Aplicando BDD em Testes de REST API: Uma Experiência Prática

Washington Luiz da Silva Lima[1], Uirá Kulesza[2]

[1]Residente em TI – Instituto Metrópole Digital (IMD)
CIVT – UFRN – Av. Senador Salgado Filho, 3000
Lagoa Nova, CEP 59.078.970 – Natal – RN – Brasil

[2]Departamento de Informática e Matemática Aplicada
Universidade Federal do Rio Grande do Norte (UFRN)
Campus Universitário - Lagoa Nova, CEP 59072-970 – Natal – RN – Brasil

{lima.washii@gmail.com, uirakulesza@gmail.com}

Abstract. *Software development companies need to define software engineering processes, methods and techniques to support the development and quality improvement of their final product. Software testing is one of the development steps that can be used for identifying system defects in order to efficiently correct them. In this context, this work presents an experience of application of the Behavior-Driven Development (BDD) technique in the tests of REST applications. The work proposes the adoption of: (i) Behave technology to write test scenarios; (ii) Python programming language to make REST API requests and access the database; and (ii) Allure tool to generate reports of acceptance test runs. The work presents the results of the application of the BDD technique over a specific period of time.*

Resumo. *Empresas de desenvolvimento de software necessitam definir processos, métodos e técnicas de engenharia de software para apoiar o desenvolvimento e melhoria da qualidade do seu produto final. Testes de software é uma das etapas do desenvolvimento que podem ser usadas para defeitos em sistemas, de forma a corrigi-los de forma eficiente. Neste contexto, este trabalho apresenta uma experiência de aplicação da técnica de Behavior-Driven Development (BDD) nos testes de aplicações REST, no contexto do sistema SIAIObras do TCE-RN. O trabalho propõe a adoção da: (i) tecnologia Behave para escrever os cenários dos testes; (ii) linguagem de programação Python para fazer as requisições a REST API e acessar o banco de dados; e (ii) ferramenta Allure para gerar relatórios de execuções de testes de aceitação. O trabalho apresenta os resultados da aplicação da técnica de BDD ao longo de um período específico de tempo.*

1. Introdução

Atualmente o mercado de desenvolvimento de software têm se tornado mais competitivo e exigente em relação aos sistemas desenvolvidos, tornando necessário que as empresas de desenvolvimento criem seus produtos com qualidade e de forma ágil. Com o intuito de que isso aconteça é necessário contar com o apoio de uma equipe de qualidade que acompanhe o processo de desenvolvimento desde a sua especificação, e realize testes no software para garantir sua qualidade.

Segundo [Graham et al, 2008] o *"teste de software é um processo, ou uma série de processos, executado para validar se o código de computador faz o que foi projetado para fazer e não faz algo fora do que foi previamente planejado. Os testes podem ter diferentes objetivos como encontrar defeitos, aumentar o nível de confiança provendo informações do nível de qualidade do software e prevenir defeitos"*. De acordo com [Molinari, 2003] algumas das ações necessárias para atingir a qualidade no desenvolvimento de software são a validação e a verificação do produto que está sendo desenvolvido. Nessa conjuntura a realização de testes é um recurso estratégico para garantir a qualidade.

Devido a demanda de gerar respostas de software com qualidade e rapidez em ambientes empresariais, novas estratégias de testes que permitem agilidade na criação e execução dos testes têm sido propostas. Estas estratégias podem ser responsáveis tanto pelo suporte à equipe, quanto pela crítica ao produto. Inclusive ser voltado para a validação do negócio ou verificação da tecnologia. [Crispin e Gregory, 2009] discutem sobre as características dessas estratégias de testes, e apontam que tais estratégias foram criadas com o objetivo de ter uma compreensão comum para desenvolver as demandas dos usuários. Um desses processos é conhecido como *Behavior Driven Development* (BDD) cujo objetivo é criar cenários em linguagem natural que exprimem o comportamento de uma funcionalidade.

Neste contexto, este trabalho tem como objetivo principal projetar e implementar testes BDD para a API REST de um sistema do Tribunal de Contas do Estado do Rio Grande do Norte (TCE-RN), assim como mostrar as tecnologias usadas para gerar a automação desses testes. Ao longo dos últimos anos, o TCE-RN incorporou ao seu processo de desenvolvimento, técnicas e práticas de testes em seus projetos. O Instituto Metrópole Digital (IMD) da UFRN, através do seu programa de Residência em Tecnologia da Informação, vem contribuindo com tal iniciativa com a seleção de alunos para atuar na área de *Quality Assurance* (QA), durante essa parceria, apoiando assim a melhoria do processo de testes do TCE-RN.

Os sistemas do TCE-RN contam atualmente com testes funcionais automatizados na aplicação cliente (*front-end*) do sistema assim como na API REST que é a interface que a aplicação no lado do servidor (*back-end*) disponibiliza para ter acesso às suas funções. Os testes funcionais na API desenvolvidos neste trabalho seguem os conceitos do *Behavior Driven Development* (BDD), sendo utilizada a linguagem Python e o framework Cucumber para a escrita dos mesmos.

2. Fundamentação Teórica

Para realizar este trabalho, se faz necessário estudar alguns conceitos e técnicas como base utilizadas no desenvolvimento de testes do TCE-RN. Isso se faz necessário para nossa melhor compreensão sobre os termos usados ao longo deste trabalho.

2.1 Testes de Software

O primeiro conceito deste trabalho é o de teste de software. Segundo [Myers, Badgett e Sandler, 2012] teste de software é o processo de execução de um programa com a finalidade de encontrar erros, aumentando, assim, a confiabilidade do software.

Segundo [Sommerville, 2007] o processo de teste visa atingir dois objetivos distintos: demonstrar ao desenvolvedor e ao cliente que o sistema atende aos seus requisitos e demonstrar que o software não se comporta de maneira não correta, não desejável ou de forma diferente do que foi especificado. O principal objetivo do teste de software é revelar a presença de erros no produto [Myers, 1979]. O teste de software abarca o processo de execução do sistema, de modo controlado e com o objetivo de encontrar defeitos antes que o sistema esteja disponível para os clientes e analise se ele se comporta conforme foi especificado.

2.2 Linguagem Gherkin

Ao longo do desenvolvimento deste trabalho utilizamos a linguagem Gherkin para estruturação dos cenários de casos de testes automatizados da API SIAIObras. Gherkin é uma linguagem de domínio específica criada para a descrição de comportamentos. Isto lhe dá a habilidade de remover detalhes lógicos dos testes de comportamento. Gherkin é uma poderosa linguagem natural desenvolvida para que humanos possam a utilizar como forma de entendimento e compreensão acerca das especificações levantadas a partir da perspectiva do stakeholder. O Gherkin permite que o comportamento do software seja descrito sem detalhar como ele será implementado em qualquer um dos sessenta idiomas disponíveis.

De acordo com [Wynne e Hellesoy, 2012], Gherkin são as regras as quais Cucumber deve seguir de sintaxe básica ao qual cada cenário segue, onde cada cenário segue um conjunto de regras que suas especificações são lidas e executadas a partir dos arquivos de texto, escritos em linguagem natural. Cada cenário é uma lista de passos interpretados pelo Cucumber.

O Gherkin auxilia no processo de criação de documentações automatizáveis baseadas nos comportamentos esperados por um sistema, consequentemente sendo útil para a automatização de testes. Essa notação consiste em uma linguagem objetiva, simples e estruturada, que facilita o entendimento dos interessados na documentação do software.

2.3 Cucumber

No decorrer deste trabalho foi utilizado o framework Cucumber para a criação dos cenários de testes, esse framework utiliza a linguagem Gherkin como sua base de desenvolvimento.

Com a utilização do Cucumber, criamos arquivos chamados "*.features*", nessas features existem "*keywords*" (ou "palavras-chave"), que descrevem características da funcionalidade das features. É importante destacar que essas *keywords* são diferentes das *keywords* dos steps feitas com Python. Para isso é necessário entender que no Gherkin existem *keywords* que são utilizadas para especificar como cada *step* do arquivo *.feature* interage com o sistema. As principais *Keywords* são:

- Given (pt: Dado): Utilizada para especificar uma precondição, algo antes desse step deve ter acontecido para que esse step seja executado com

sucesso, um exemplo seria um *step*: “Dado que eu possua um token acesso” dentro desse *step* é feita a validação de uma condição antes de se prosseguir para os próximos passos a precondição seria ter um token de acesso. Por se tratar de uma precondição, normalmente vem escrito no passado;

- When (pt: Quando): Utilizado quando será executada uma ação de que se espera uma reação vinda do sistema, que será validada no *step* “Then”. Este passo vem escrito no presente;
- Then (pt: Então): Valida a ação caso o esperado aconteça. Segue sempre um passo do tipo “Quando”, pois aqui é validada a reação da ação recebida. Por se tratar do resultado esperado, normalmente vem escrito na forma de futuro próximo;
- And (pt: E): Caso seja necessário mais uma interação com o sistema para complementar um fluxo, mas que não necessariamente se trata de uma ação ou reação, se utiliza “And”;
- But (pt: Mas): No geral serve a mesma funcionalidade do “And”, porém é normalmente utilizado após uma validação negativa depois do “Then”;

A Figura 1 apresenta um exemplo de um arquivo ".feature" dos testes de API utilizando o Cucumber como ferramenta para seu desenvolvimento. Na figura é apresentado os passos ao quais o cenário deve executar para escrever esse arquivo bem mostra a utilização de algumas das keywords acima citadas nesse texto

features > TC > TC0002_Documento > TC0002_04_POST_Documento.feature

```
4
5   Funcionalidade: Realizar a requisição POST  /v1/Documento
6
7       @TC0002_04_01
8       Cenario: POST Documento: envio dos dados de documento da obra
9       Dado que possua um token de acesso
10      Quando acesso arquivo usando a request POST Documento
11      """
12      {
13          'descricao': 'teste automatizado 06-05-2022 com mime type4',
14          'idObraServico': 17,
15          'idTipoDocumento': 1,
16          'idOrgao': 422,
17          'emEdicao': True
18      }
19      """
20      Entao verifico que o código de resposta da requisição é 200
21      E verifico que o documento foi cadastrado
22      E excluo o documento
```

Figura 1. Endpoints de ObraServico da API SIAIObras.

2.4. REST API

A API SIAIObras é uma API baseada em um estilo arquitetural híbrido, sendo assim uma REST API. De acordo com a [Red Hat, 2020], REST não é um protocolo ou

padrão, mas sim um conjunto de restrições de arquitetura. Os desenvolvedores de API podem implementar a arquitetura REST de maneiras variadas. Quando um cliente faz uma solicitação usando uma API RESTful, essa API transfere uma representação do estado do recurso ao solicitante ou endpoint. Essa informação (ou representação) é entregue via HTTP utilizando um dos vários formatos possíveis: *Javascript Object Notation* (JSON), HTML, XLT, Python, PHP ou texto sem formatação.

Segundo [Rodriguez, 2008], a arquitetura REST, modelo utilizado na estruturação de serviços Web, consiste em um conjunto de princípios e conceitos que guiam os desenvolvedores durante a construção de um software para a Web. Essa arquitetura é orientada aos recursos presentes no sistema em construção, definindo seus possíveis estados, como devem ser endereçados, e como devem ser compartilhados por meio do protocolo HTTP (Hypertext Transfer Protocol).

2.5 Python Requests

No projeto de teste foi utilizado a linguagem de programação Python para realizar os testes na api, para gerir as requisições a API, foi utilizada uma biblioteca do python, a requests. A biblioteca Requests é uma simples e elegante biblioteca HTTP para Python, que auxilia muito no acesso às informações que usam o protocolo HTTP, foi empregado também no processo de testes a biblioteca pyodbc para gerir as requisições feitas ao banco de dados do SIAIObras com o intuito de verificação de steps dos testes.

2.6 Behavior-Driven Development (BDD)

Como forma de facilitar a definição e a manutenção de uma documentação viva, ou seja, uma documentação dinâmica e que acompanha o processo de desenvolvimento desde o levantamento de requisitos e de cenários funcionais até a implementação, surgem algumas técnicas, como a abordagem *Behavior Driven Development* (BDD).

De acordo com [North 2006], o BDD é uma técnica de desenvolvimento ágil. Tendo essa técnica como objetivo integrar linguagem de programação e regras de negócio. Além de focar o comportamento do software, o BDD também possibilita uma compreensão mais ampla das regras de negócio,em vez de focar em detalhes técnicos.

A abordagem BDD é uma técnica, que se baseia na definição de cenários de uso, usado para registrar o comportamento esperado do software, automatizando cenários de uso. Comumente, as especificação de cenários BDD são realizadas com apoio da linguagem Gherkin, possuindo as palavras-chave necessárias para definição de uma feature, seus cenários e seus respectivos passos. Sendo assim, os cenários de uso, na abordagem BDD, são vistos como os requisitos e, ao mesmo tempo, os casos de teste que validam o funcionamento destes requisitos, mesclando ambos artefatos em apenas um.

2.7 Allure Report

O Allure é uma ferramenta de relatório de testes que funciona em múltiplas linguagens, sendo leve e flexível. Com a ajuda dos gráficos do Allure, as equipes de QA e de desenvolvimento conseguem acompanhar o fluxo de informações a respeito dos testes executados.

Da perspectiva do ciclo de desenvolvimento como um todo, os relatórios Allure encurtam o ciclo de vida de defeitos comuns no software, com as as falhas de teste

sendo divididas em *bugs* e testes quebrados. Dessa forma, os desenvolvedores e testadores responsáveis têm informações valiosas em mãos, que podem contribuir para corrigir *bugs* encontrados no código do software ou dos testes mais rapidamente.

3. Metodologia de Desenvolvimento do Trabalho

3.1 Visão Geral

Nosso trabalho propõe a condução de testes de API usando a técnica de *Behavior-Driven Development* (BDD). A metodologia conduzida para o trabalho envolveu a realização das seguintes atividades:

1. Escolha do sistema a ser testado - o sistema escolhido foi o SIAIObras do TCE-RN. Os motivos para sua escolha foram:
    - O sistema é de suma importância para o TCE-RN, o sistema SIAIObras é responsável por manter os dados de obra cadastradas, iniciadas, pausadas e encerradas por todos os órgãos jurisdicionados ao TCE;
    - Com mudanças na leis estaduais, os órgãos passaram a ter um sistema no qual eles tenham as obras e serviços de engenharia cadastrados de desenvolvimento próprio, para que os órgão não precisem fazer todo o sistema, o TCE-RN decidiu abrir a API para os jurisdicionados para que assim eles possam aproveitar o que já foi feito;
    - Sendo assim foi necessário refazer algumas partes da API, pois algumas verificações nos processos de cadastros de obras eram realizados apenas no cliente, sendo necessário ter sua verificação validada também na API, para garantir que todas as regras de negócios sejam atendidas na API, o processo de construção dos teste automatizados da API foram iniciados com o intuito de garantir a qualidade dela.
2. Estudo das tecnologia que foram implementadas no projeto de teste - para esse projeto, o Python foi escolhido como linguagem de programação, da qual utilizamos: (i) a biblioteca *pyodcb* para manipular o banco de dados do sistema; (ii) a biblioteca *requests* para tratar as requisições feitas a API; (iii) o Cucumber para a criação dos cenários de testes; e (iv) o Allure como ferramenta para gerar relatórios sobre os testes executados;
3. Projeto e implementação dos testes - atividade que envolveu o projeto e implementação dos testes para o sistema SIAIObras usando as tecnologias mencionadas acima. Tal atividade é detalhada na próxima subseção 3.2;
4. Avaliação dos resultados dos testes - esta atividade é detalhada na seção 4 e envolveu a apresentação e análise dos testes realizados.

3.2 Projeto e Implementação dos Testes

Os testes de API no SIAIObras devem seguir as regras de negócios para cada *endpoint*. Para elaboração dos testes são criados cenários para cada Endpoint do sistema, onde são testados cenários de sucesso e falhas. A Figura 2 ilustra os *Endpoints* do sistema representados de acordo com a documentação do Swagger.

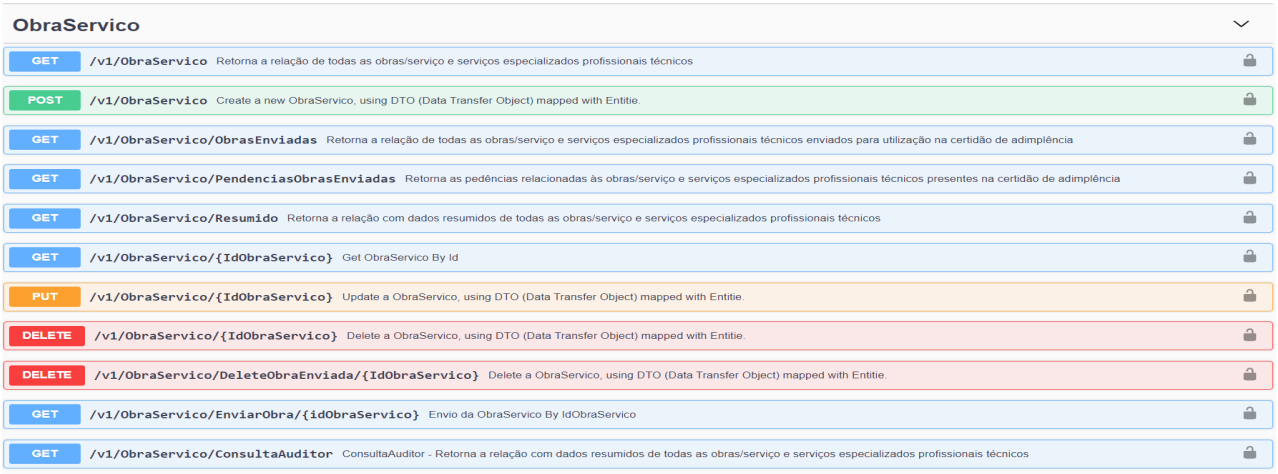

Figura 2. Endpoints de ObraServico da API SIAIObras.

A Figura 2 mostra a documentação do SWAGGER do SIAIObras onde são listadas as rotas e as requisições disponíveis.Para fazer essas requisições foi utilizada a biblioteca *Requests* do Python, sendo que no swagger são listadas 4 tipos de requisições, sendo elas:

- POST: criar dados no servidor;
- GET: leitura de dados no servidor;
- DELETE: excluir as informações no servidor;
- PUT: atualizações de registros no servidor.

Para cada *Endpoint* foram criados cenários de sucessos e falhas, onde são criadas *features e steps*, que vão testar as regras de negócios. Os *endpoints* testados da API SIAIObras foram os *endpoints* do módulo de cadastro de Obras/Serviços de Engenharia. Este módulo é responsável pela funcionalidade de cadastro de obras e serviços de engenharias dos jurisdicionados do TCE-RN. Os *endpoints* testados foram: ObraServico, Fiscal, Documento, Geolocalização e Observação Geral. Para cada endpoint foram criados cenários de testes para testar as suas respectivas ações as quais podiam desempenhar, sendo elas, os métodos POST, PUT, GET E DELETE. O projeto de testes automatizados da API SIAIObras não existia antes do início de Janeiro de 2022.

A Figura 3 apresenta um exemplo de teste de API utilizando o Cucumber. Na figura é apresentado o arquivo ".feature", onde temos um cenário de teste de envio de dados de um fiscal de uma obra do tipo direta. Ele representa um cenário de teste onde o caso de teste deve passar com sucesso. Na linha 11 da Figura 3 acionamos o *step* que irá solicitar um *token* de acesso a API SIAIObras para ter acesso ao *endpoint* ao qual se vai testar. Na linha 12 é acionado o *step* fazemos acesso ao *endpoint* e nele enviamos os campos necessários para se enviar um fiscal para uma obra. Na linha 23 dessa mesma figura, o cenário faz a verificação via código de resposta retornado pelo servidor. Na linha 24 o fiscal inserido nesse cenário de caso de teste é inativado para que assim não seja gerado dados desnecessários no banco de dados, para isso é utilizado a biblioteca pyodc para fazer o acesso ao banco de dados e fazer um update na tabela no campo que precisa ser inativado. As linhas 26 a 28 são linhas as quais detêm o código da obra ao qual o fiscal deve ser inativado, a seguir na Figura 3 temos um exemplo.

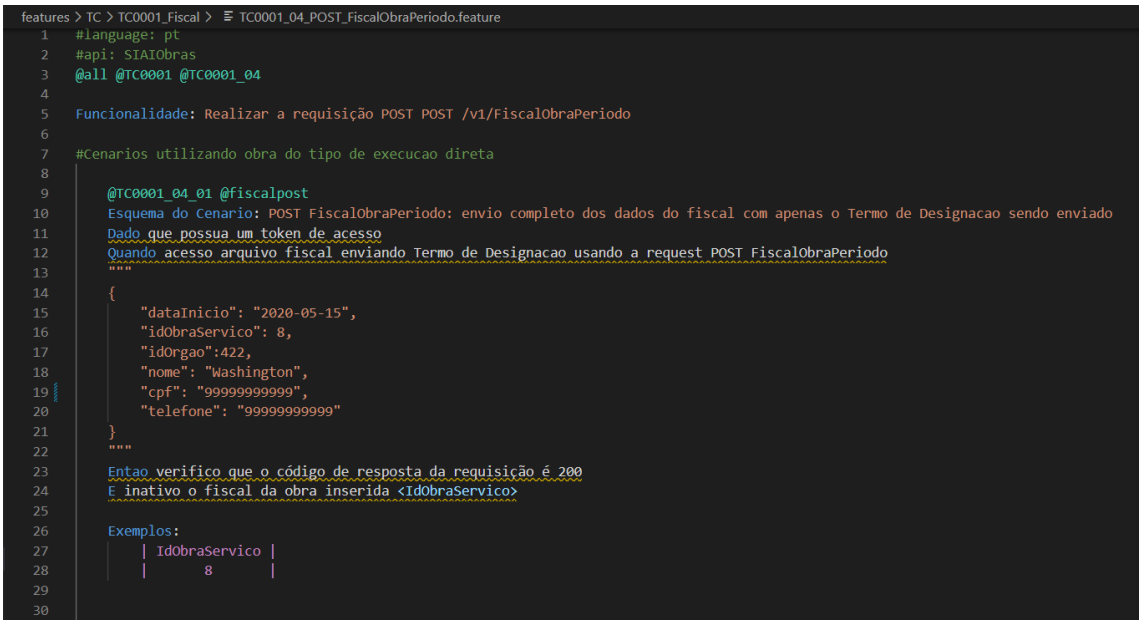

Figura 3. Exemplo de um arquivo .feature criado utilizando cucumber.

Para que cada linha da Figura 3 seja executada é necessário se fazer uma step utilizando a biblioteca requests do Python, como é mostrado na Figura 4 a seguir. Na Figura 4 é criado o *step* de verificação do código de resposta do servidor, onde o *step* vai verificar se o código recebido pela resposta da requisição feita a api é igual ao que foi informado no cenário de teste, sendo igual ao cenário de teste, continuará executando os próximos passos, se o código informado deferir do código recebido da API o teste apresentará o status de falha.

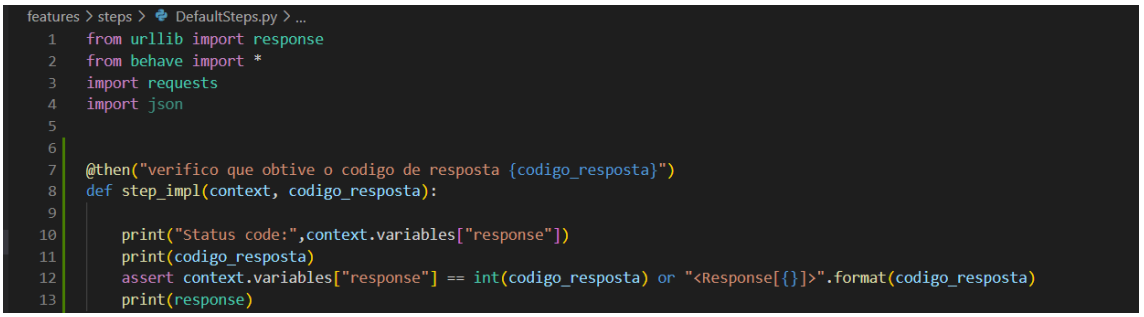

Figura 4. Exemplo de um step criado com a biblioteca requests.

4. Avaliação

Esta seção apresenta os resultados dos testes que foram gerados ao longo de todo o desenvolvimento dos testes. Esses dados foram coletados através dos testes automatizados dos relatórios gerados pela ferramenta Allure.

A Tabela 1 mostra que foram criados 22 *features* para o módulo de cadastro de Obra/Servico de Engenharia, com um total de 165 cenários de testes elaborados, onde

155 cenários foram executados com sucesso e apresentaram status de *passed* no Allure equivalendo a 93,44% dos testes executados, sendo assim 10 desses cenários apresentaram *status failed* equivalente a 6,06% do total de testes executados, tendo um total de 494 steps executados. Para obtenção desses dados foi utilizada a ferramenta Allure para gerar gráficos sobre os testes otimizados. A Figura 5 mostra o gráfico de execução dos cenários de testes, mostrando os cenários aos quais foram passados com sucesso, e os cenários aos quais tiveram o resultado falho.

Tabela 1. Dados coletados durante execução dos testes das funcionalidades de cadastro Obra/Servico de Engenharia da API SIAIObras

Configurações	Testes automatizados concluídos
Quantidade de features	22
Cenários executados	165
Cenários com falhas	10
Quantidade de steps executadas	499
Quantidades de testes executados com sucesso	155
Quantidades de testes executados com falha	10

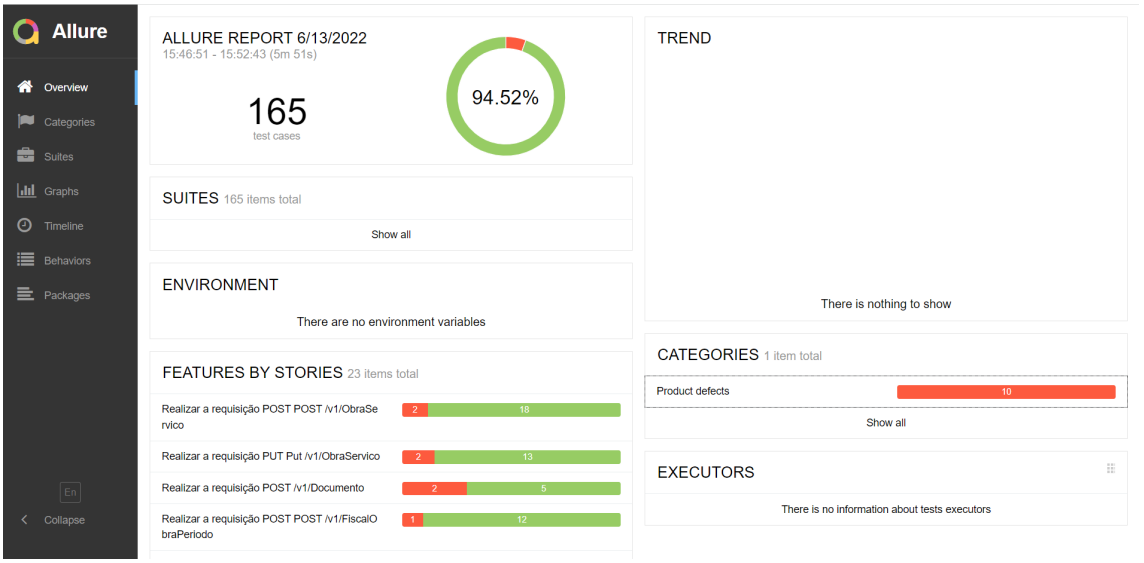

Figura 5. Relatório gerado pela ferramenta Allure

Tabela 2. Cenários de teste por funcionalidades, Dados coletados durante execução dos testes das funcionalidades de cadastro Obra/Servico de Engenharia da API SIAIObras.

Funcionalidades	Número de Cenários
ObraServico	57
Fiscal	41
Documento	27
Localização	24
Observação Geral	26
Total	47

A Tabela 2 mostra como os cenários de teste estão distribuídos dentre os *endpoints* testados, os 165 cenários foram distribuídos entre as funcionalidade de ObraServico. Fiscal, Documento, Localização, Observação Geral, sendo que

ObraServico conta com 57 cenários, Fiscal 41, Documento 27, Localização 24 e Observação Geral 26.

A elaboração desses testes foi guiada principalmente pelas regras de negócios do sistema, pois queríamos garantir que nenhuma regra de negócio ficasse de fora das verificações feitas na API, garantindo assim que todos os requisitos fossem testados. O objetivo central era trazer mais segurança no uso dessa API, pois como citado, a API irá se tornar pública para todos os jurisdicionados. Além dos cenários de teste das regras de negócios, foram criados casos de testes que englobam outras situações, tais como, o uso de partição de equivalência proposta por (Myers, 1979), a qual busca reduzir o número total de casos de teste necessários, particionando as condições de entrada em um número finito de classes de equivalência, sendo elas classificadas em dois tipos: o conjunto de entradas válidas e inválidas para o sistema.

A Figura 6 apresenta um exemplo de caso de teste onde é testado uma das regras de negócios do sistema que está incluída na classe de equivalência do tipo inválida, no exemplo da imagem temos um cenário de teste onde um campo obrigatório para o sucesso do cenário não é informado, nesse caso é verificado se o retorno da API é 400, quando testada a api retorna 400 o que garante que o sistema não vai aceitar que o usuário deixe de passar campos obrigatórios ao informar os dados.

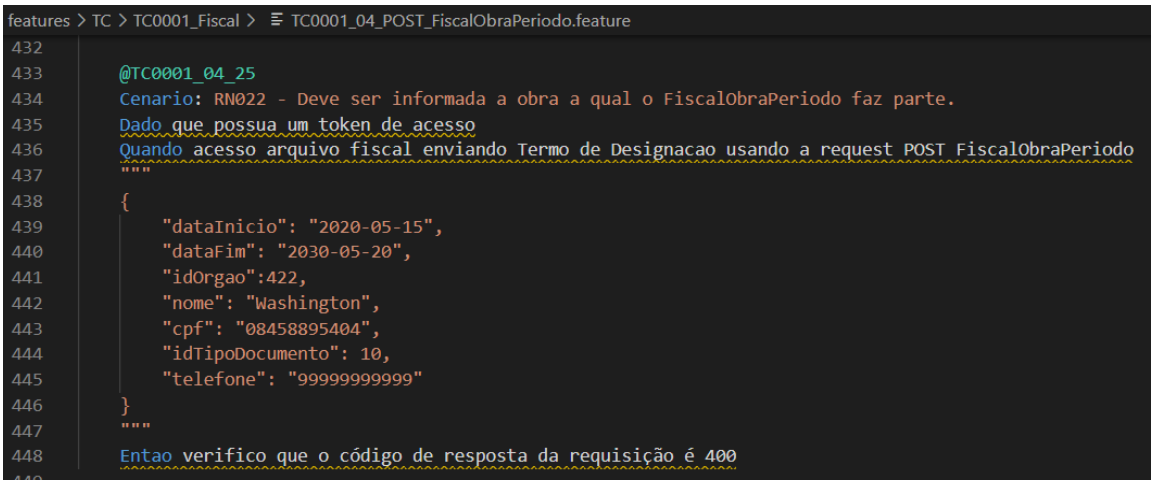

Figura 6. Cenário de teste pertencente a classe de equivalência do tipo inválida

A Figura 7 um exemplo de caso de teste onde se é testado um entrada de dados que pertence a classe de equivalência do tipo válida, no exemplo da imagem temos um cenário de teste onde é enviado 2 observações juntas numa mesma requisição para o sucesso do cenário, nesse caso é verificado se o retorno da API é 200, quando testada a api retorna 200 o que garante que o sistema aceitou o envio dos dados.

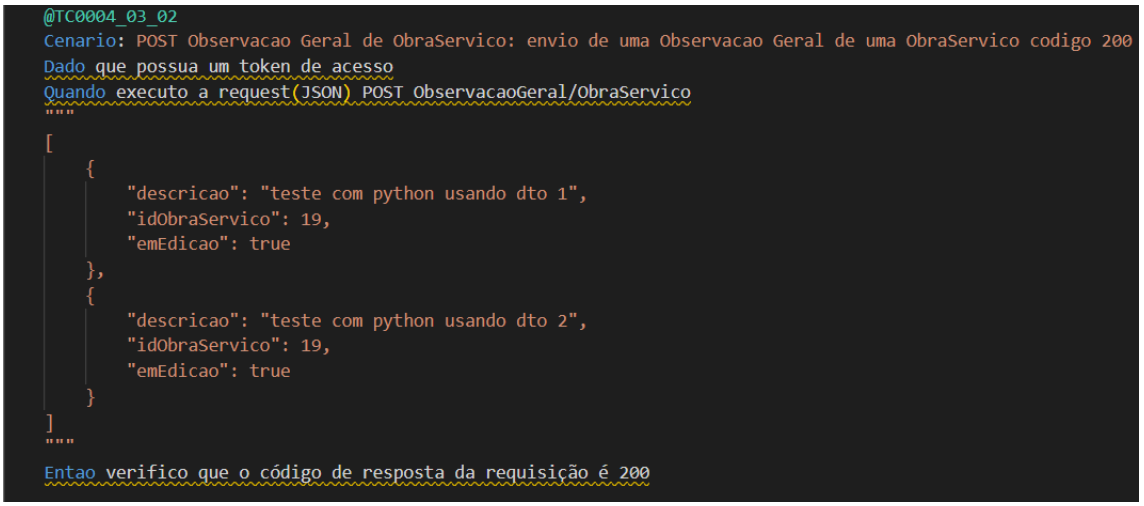

Figura 7.Cenário de teste pertencente a classe de equivalência do tipo válida

5.Conclusões

Os testes automatizados da API SIAIObras são relativamente novos, tendo sido iniciada a sua implementação em Janeiro de 2022, ainda assim é possível notar uma maturidade em relação a bugs e erros encontrados durante a sua execução. A Implementação de testes usando a técnica de BDD para o teste de uma API REST auxiliou para o desenvolvimento de testes mais sucintos, organizados e que venham a servir como documentação para o próprio sistema. Já que os cenários de testes são descritos como casos de usos de software, o que vem a exemplificar como se deve usar o sistema de forma adequada e quais são as entradas não válidas para eles.

Além disso, com os testes automatizados é possível detectar falhas muito mais rápido, principalmente utilizando como auxílio gráfico o Allure Report, software que foi utilizado para gerar os gráficos das execuções dos testes. E com isso os testes que antes só existiam na interface gráfica do SIAIObras passaram também a ocorrer na API. Isso garantiu que as verificações necessárias para o uso do sistema são satisfeitas, já que anteriormente algumas verificações eram feitas apenas no cliente e com a abertura da APÌ para os jurisdicionados se mostrou necessário que essas verificações também acontecessem na API.

Ao longo deste trabalho um fator que gerou dificuldades foi integrar as tecnologias usadas já citadas neste trabalho, como o projeto de teste de API no TCE é relativamente recente, não se têm uma documentação ou exemplos aos quais se possa seguir e sanar dúvidas que podem surgir ao longo do desenvolvimento do projeto, pois é mais comum o uso de frameworks de teste para o teste de API, como o PyTest. O PyTest é um framework de uso geral, mas é especialmente usado para testes funcionais e de API, no TCE os testes de API são feitos com a linguagem Python em si.

Existem ainda muitas oportunidades de melhoria para o projeto, os próprios casos de testes podem ainda serem melhorados, pois nem todas as entradas das classes de equivalências inválidas foram testadas, com isso o projeto de testes pode se tornar ainda mais robusto.

6. Referências

Crispin, L. e Gregory, J. (2009), Agile Testig: a pratical guide for testers e agile teams, 1ªEdição

Graham, Dorothy; Veenendaal, Erik van; Evans, Isabel; Black, Rex. Foundations of Software Testing: ISTQB Certification. Cengage Learning Business Press., 2008

Molinari, Leonardo. Testes de software: Produzindo sistemas melhores e mais confiáveis. São Paulo. Érica, 2003.

Myers, G.J. The Art of Software Testing. John Wiley & Sons, New York, 1979.

Myers, Glenford, J.; Badgett, Tom; Sandler, Corey. The Art of Software Testing, 3. ed. New Jersey: Wiley Publishing, 2012.

North, D. (2006). Introducing behaviour driven development. Better Software Magazine.

Rodriguez, Alex. Restful web services: The basics. IBM Developer Works, v. 33, p. 18, 2008.

Red Hat. (08 de Maio de 2020). O que é API REST?. https://www.redhat.com/pt-br/topics/api/ what-is-a-rest-api, day = 10, month = 06, year = 2020, note = (Accessed on 06/10/2020).

Sommerville, I. Engenharia de software. 8. ed. São Paulo: Pearson AddisonWesley, 2007

Wynne, M., Hellesoy, A. (2012) “The Cucumber Book: Behaviour-Driven Development for Testers and Developers.” Editora: The Pragmatic Programmers.

Anexo I

RESIDÊNCIA EM TECNOLOGIA DA INFORMAÇÃO

Aluno	Washington Luiz da Silva Lima
Matrícula	20212001142
Data da Defesa	27/Junho/2022
Hora da Defesa	16h
Local da Defesa	Google Meet
Título do Trabalho	Aplicando BDD em Testes de REST API: Uma Experiência Prática

Examinador (Nome e Afiliação)	**Nota**	**Rubrica**
ORIENTADOR: Uirá Kulesza (UFRN)	7,5	
Jadson José dos Santos (UFRN)	7,0	
Lindemberg Silva Pereira (TCE-RN)	8,0	
MÉDIA FINAL	7,5	

FAIXA DE MÉDIA FINAL	**CONCEITO FINAL**	**STATUS**
0,0 a 2,9	**E**	**REPROVADO**
3,0 a 4,9	**D**	**REPROVADO**
5,0 a 6,9	**C**	**APROVADO**
7,0 a 8,9	**B**	**APROVADO**
9,0 a 10	**A**	**APROVADO**

TERMO DE APROVAÇÃO DE
VERSÃO FINAL DO TRABALHO DE CONCLUSÃO DE CURSO

Atesto que o referido Trabalho de Conclusão de Curso foi considerado aprovado pela Banca Examinadora e que todas as solicitações de correção feitas por esta Banca foram atendidas satisfatoriamente dentro do prazo, ficando o aluno, portanto, com **CONCEITO FINAL** B na atividade.

Natal, 27 de Junho de 2022.

ORIENTADOR