UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TAYNARA LUANA CAETANO DOS SANTOS

ADERÊNCIA DE TÉCNICAS DE TESTES AUTOMATIZADOS EM UMA API REST: ABORDAGEM EM UMA APLICAÇÃO DE HUB INTEGRADOR DE MARKETPLACES

CAMPO MOURÃO

2023

TAYNARA LUANA CAETANO DOS SANTOS

ADERÊNCIA DE TÉCNICAS DE TESTES AUTOMATIZADOS EM UMA API REST: ABORDAGEM EM UMA APLICAÇÃO DE HUB INTEGRADOR DE MARKETPLACES

Adhrence of Automated Testing Tecniques in a Rest API: Approach in a Marketplaces Integator HUB Application

Dissertação apresentada como requisito para obtenção do título de Mestre em Inovações Tecnológicas da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador(a): Wyrllen Everson de Souza

CAMPO MOURÃO

2023

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Campo Mourão

TAYNARA LUANA CAETANO DOS SANTOS

ADERÊNCIA DE TÉCNICAS DE TESTES AUTOMATIZADOS EM UMA API REST: ABORDAGEM EM UMA APLICAÇÃO DE HUB INTEGRADOR DE MARKETPLACES

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestra Em Inovações Tecnológicas da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Inovações Tecnológicas.

Data de aprovação: 24 de Fevereiro de 2023

Dr. Wyrllen Everson De Souza, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Alexandre Rasi Aoki, Doutorado - Universidade Federal do Paraná (Ufpr)

Dr. Diogo Heron Macowski, Doutorado - Universidade Tecnológica Federal do Paraná

Dra. Magda Cardoso, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 24/02/2023.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, por me conceder saúde, paciência, entendimento e sabedoria para seguir em frente e conseguir dar o meu melhor, por ser minha força e fortaleza a cada dia e em todos os momentos.

A minha amada e preciosa avó, Arenita, por me apoiar e incentivar todos os dias não só nessa fase importante, mas sim em todos os momentos. Por acreditar em mim e sempre me motivar. Por não medir esforços para me apoiar na realização de mais um sonho. Sem você, nada disso que estou vivendo agora seria possível, amo você de todo meu coração!

Aos meus primos que sempre estiveram comigo e principalmente no desenvolvimento deste trabalho. Juliana, obrigada por corrigir este trabalho, por me apoiar e me ajudar nas decisões mais difíceis. Charles, você foi meu apoio todos esses anos, me ajudou a caminhar nessa área técnica, acreditou no meu potencial, e esteve comigo durante toda a minha vida acadêmica. Amo vocês!

Aos meus amigos e colegas de trabalho, anjos que Deus colocou em minha vida. Narayane, foram os seus conselhos que me ajudaram a chegar aonde eu cheguei em minha carreira. Fagner Melo, obrigada por acreditar em meu potencial e sempre me motivar a fazer com que esse projeto acontecesse, me concedendo o espaço e apoiando. André Thomazini, por ser meu apoio nas horas mais difíceis e sempre me dar os melhores direcionamentos em meu trabalho. Everton Trindade e José Eduardo, os dois desenvolvedores e mentores que tenho até hoje, agradeço por cada tempo destinado a me ensinarem, esse projeto hoje está sendo entregue pois vocês me ensinaram a desenvolver, me direcionaram. Rodrigo Lisboa, você me motivou a continuar, e não desistir do meu objetivo. Obrigado a todos, vocês moram em meu coração!

Eu não poderia deixar de reservar este espaço para uma das minhas maiores inspirações, meu mentor Julio de Lima, que com todo seu conhecimento me possibilitou aprender tudo que eu precisava sobre testes de API, para que este trabalho fosse possível de acontecer. Obrigada mestre, por ter sido meu professor, mentor e a minha referência técnica.

Ao meu companheiro de vida Juliano, por toda paciência e entendimento que teve comigo durante essa etapa final de meu trabalho, obrigada por sempre se fazer presente em minha vida e por estar ao meu lado nesse momento tão importante.

Ao meu professor e orientador Wyrllen, nós dois sabemos o quanto difícil foi essa jornada, mas conseguimos chegar até aqui. Agradeço a todo apoio, tempo dedicado, pela paciência que você teve comigo durante todos esses 3 anos. Agradeço por ter acreditado em meu trabalho e potencial, e ter sido o meu orientador nesta etapa importante em minha vida, muito obrigada por tudo!

RESUMO

Com o crescente avanço da tecnologia inúmeras plataformas de Software vêm surgindo em diversos ramos de atividades, principalmente em aplicações WEB (World Wide Web). Essas aplicações podem utilizar diferentes tipos de arquiteturas, sendo uma delas a REST (Representational State Transfer). Essa arquitetura é baseada em uma interface de programação de aplicações (API – Application Programming Interface) que utilizam microsserviços que facilitam o processo de criação de novos componentes. Esses microsserviços precisam passar por uma etapa de testes, onde suas funcionalidades serão avaliadas. É de suma importância que os processos de garantia da qualidade dos produtos de Software, sejam aplicados durante o ciclo de desenvolvimento através dos testes de Software. Porém o processo de teste, pode se tornar oneroso por executar diversas vezes, ações específicas de uma determinada funcionalidade de forma manual. Diante das alternativas, destaca-se o teste automatizado, por ser capaz de executar rotinas automáticas e evitar o esforço manual constante. Este trabalho visa avaliar a redução do esforço manual de testes de uma API REST, com a utilização de testes automatizados, que será responsável por executar rotinas de testes automatizadas através de scripts que serão desenvolvidos com a utilização dos frameworks (mocha e chai), (JavaScript) como linguagem de programação principal e (Node.js) como interpretador de JavaScript para que o código possa ser executado fora do ambiente de um navegador WEB. Esses scripts serão responsáveis por validar cenários de testes da API que atualmente são executados de forma manual e de forma repetitiva. Pretende-se também avaliar os mesmos cenários de testes com a ferramenta Postman. No primeiro experimento do teste utilizando de frameworks, são avaliados os protocolos de inserção de registros em uma aplicação de cliente x servidor, executando 37 cenários de testes simultâneos. Com o segundo experimento utilizando a ferramenta Postman, também foram avaliados os mesmos 37 cenários de testes. Por fim, realizou-se um experimento envolvendo dois analistas de testes para executar estes 37 cenários de forma manual. A avaliação do processo de teste automatizado utilizando dos frameworks resultou em uma execução de sete segundos, enquanto o teste executado no Postman, nos apresenta um trabalho mais custoso por ter que inserir várias chamadas de forma separada, enquanto na automação pode-se abordar as chamadas de forma unificada e centralizada. Já a execução dos 37 cenários de testes executados de forma manual, resultou em duas horas de trabalho sem pausas. Com isso, avalia-se a utilização da técnica de testes automatizada utilizando a junção dos frameworks como alternativa de redução do processo de teste em uma API REST.

Palavras-chave: Teste de Software; Técnicas de Testes; API REST; Framework; Teste Automatizado;

ABSTRACT

With the increasing advancement of technology, numerous software platforms have emerged in various fields of activities, especially in WEB applications (World Wide Web). These applications can use different types of architectures, one of which is REST (Representational State Transfer). This architecture is based on an application programming interface (API - Application Programming Interface)) that uses microservices that facilitate the process of creating new components. These microservices need to go through a testing stage, where their functionality will be evaluated. It is of paramount importance that the quality assurance processes of Software products are applied during the development cycle through Software testing. However, the testing process can become onerous by performing specific actions of a particular functionality manually several times. In view of the alternatives, automated testing stands out, as it is capable of executing automatic routines and avoiding constant manual effort. This work aims to evaluate the reduction of the manual effort of testing a REST API, with the use of automated tests, which will be responsible for executing automated test routines through scripts that will be developed with the use of frameworks (mocha and chai), (JavaScript) as the main programming language and (Node.js) as a JavaScript interpreter so that the code can be executed outside of a WEB browser environment. These scripts will be responsible for validating API test scenarios that are currently performed manually and repetitively. It is also intended to evaluate the same test scenarios with the Postman tool. In the first test experiment using frameworks, the protocols for inserting records in a client x server application are evaluated, executing 37 simultaneous test scenarios. With the second experiment using the Postman tool, the same 37 test scenarios were also evaluated. Finally, an experiment was carried out involving two test analysts to execute these 37 scenarios manually. The evaluation of the automated test process using the frameworks resulted in an execution of seven seconds, while the test executed in Postman, presents us with a more expensive work by having to insert several calls separately, while in automation, the calls can be addressed in a unified and centralized way. The execution of the 37 test scenarios performed manually resulted in two hours of work without breaks. With this, the use of the automated testing technique is evaluated using the joining of frameworks as an alternative to reduce the testing process in a REST API.

Keywords: Software Testing; Testing Techniques; REST API; Framework; Automated Test;

LISTA DE ILUSTRAÇÕES

Figura 1 - Representação da funcionalidade de uma API REST, representando a comunicação entre cliente e servidor. 19
Figura 2 - Representação do código de status que pode ser retornado em operações CRUD. 20
Figura 3 - Representação das requisições cliente/servidor através do protocolo HTTP: POST, GET, PUT.....20
Figura 4 - Fases em que o teste é empregado de acordo com o Modelo V. 23
Figura 5 - Representação da aplicação do Teste Unitário. 25
Figura 6 - Página da Documentação da API REST, desenvolvida com SWAGGER UI. 30
Figura 7 - Representação do processo de testes atual. 38
Figura 8 - Proposta de teste automatizada apresentada. 39
Figura 9 - Swagger: Cadastro de Marca. 40
Figura 10 - Representação do indicador de tempo da execução do teste: Power BI. 41
Figura 11 - Representação do indicador de tempo da execução do teste: Power BI. 42
Figura 12 - Padronização das pastas do projeto de automação da API REST. .46
Figura 13 - Representação da estrutura do projeto. 47
Figura 14 - Representação do arquivo JSON - Dados de acesso a aplicação da API. 48
Figura 15 - Representação do arquivo JSON - Cadastro de uma categoria para API. 48
Figura 16 - Representação do arquivo JSON - Cadastro de uma marca para API. 49
Figura 17 - Representação do arquivo JSON - Cadastro de uma variação para API. 49
Figura 18 - Representação do arquivo JSON - Cadastro de um produto para API. 50
Figura 19 - Representação da construção da classe de métodos. 50
Figura 20 - Classe com funções randômicas. 51
Figura 21 - Classe de serviços com métodos de requisições. 52
Figura 22 - Representação da Classe de Testes. 53
Figura 23 - Representação da Classe de Testes: Cenários de Produtos Cadastrados. 53
Figura 24 - Representação da execução do teste no terminal VSCODE. 54
Figura 25 - Configuração do repositório do projeto de automação no Jenkins. 55
Figura 26 - Configuração da execução do teste automatizado no Jenkins. 55
Figura 27 - Execução do teste automatizado. 56
Figura 28 - Representação do Fluxograma do teste em funcionamento. 57
Figura 29 - Fluxo de nomenclatura dos processos de testes automatizados. ..58
Figura 30 - Fluxo descritivo do nome das classes de testes do projeto automatizado. 58
Figura 30 - Resultado da pesquisa - participante 01. 60
Figura 30 - Resultado da pesquisa - participante 02. 60
Figura 30 - Resultado da pesquisa - participante 03. 61

Figura 30 - Resultado da pesquisa - participante 04. 61
Figura 30 - Apresentação do tempo de execução do teste automatizado após a implementação. 62
Figura 30 - Quantidade de bugs encontrados deste o período de início deste trabalho até a sua conclusão. 68

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface – Interface de Programação de Aplicativos.
BDD	Behaviour Driven Development – Especificação por exemplo, técnica de desenvolvimento ágil.
CHAI	Bibliotecas de asserções.
CSV	Formato de arquivo: comma-separated-values – valores separados por vírgula.
Gherkin	Elemento que facilita a padronização da documentação de testes automatizados.
HTTP	Hyper Text Transfer Protocol – Protocolo de Transferência de Hipertexto.
JSON	JavaScript Objetc Notation – Notação de Objeto JavaScript.
JMETER	*Software* de Código aberto para testes de desempenho.
MOCHA	Estrutura de testes JavaScript.
POC	Proof of Concept – Prova de Conceito.
POSTMAN	Plataforma de construção e testes de API.
REST	Representational State Transfer – Transferência Representacional de Estado.
SOAP	Simple Object Access Protocol – Protocolo Simples de Acesso a Objetos.
TDD	Test Driven Development – Desenvolvimento Orientado a Testes.
URI	Uniform Resource Identifier – Identificador Uniforme de Recursos.
XML	Extensible Markup Language: Linguagem de marcação.
WEB	World Wide Web – Sistemas de Documentos em Hipermídia.

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Caracterização do problema 15
1.2 Objetivos da pesquisa 16
1.3 Questões da pesquisa 16
1.4 Organização da dissertação 17
2 REVISÃO DE LITERATURA 18
2.1 API REST 19
2.2 Terminologia e conceitos de teste de *Software* 21
2.2.1 Teste de *Software* 22
2.2.2 Tipos de Teste de *Software* 24
2.2.3 Técnicas de teste de *Software* 26
2.2.4 Teste automatizado de API 29
2.3 Ferramentas e frameworks para testes automatizados de API 32
2.4 Considerações finais do capítulo 34
3 MATERIAL E MÉTODO 36
3.1 Redução do processo de teste de API com a utilização do teste automatizado 36
3.1.1 Contextualização do projeto 36
3.2 Técnica de teste proposta 37
3.3 Solução de testes automatizadas proposta 41
3.4 Mapeamento das ferramentas e frameworks escolhidos 42
3.4.1 Node.js 43
3.4.2 GitLab 43
3.4.3 Jenkins 44
3.4.4 JavaScript 44
3.4.5 Mocha 44
3.4.6 Chai 45
3.5 Implementação e desenvolvimento do projeto e automação proposto 45
3.6 Integração contínua 54
3.7 Metodologia de desenvolvimento 56
4 RESULTADOS E DISCUSSÕES 59
4.1 Limitações da Pesquisa 59

4.2 Resultados 59
4.2.1 Redução do tempo da execução de testes 62
4.2.2 Maior cobertura de cenários de testes da API.....63
4.2.3 Facilidade na execuão dos testes automatizados 65
4.2.4 Facilidade na escrita de novos cenários de testes no projeto de automação.....65
5 CONCLUSÃO 67
REFERÊNCIAS.....70

1 INTRODUÇÃO

Com o crescente avanço do mercado digital, inúmeras plataformas de Software vêm surgindo em diferentes ramos de atividades, principalmente em aplicações WEB (World Wide Web). Essas aplicações corporativas são divididas em componentes de serviços WEB e são conhecidas como arquitetura de microsserviços (ARCURI, 2019). O desenvolvimento de um serviço WEB, é construído de forma que suporte à interoperabilidade entre diferentes ambientes em uma rede na qual é baseada em padrões e protocolos WEB, podendo ser implementado em dois tipos distintos de arquitetura, sendo: arquiteturas baseadas em REST (Representational State Transfer) e também como arquiteturas baseadas em SOAP (Simple Object Access Protocol) (MARQUES, 2018).

A transferência de estado representacional é baseada nos recursos da arquitetura, suportando diversos formatos de como: JSON, XML, CSV entre outros (MESHRAM, 2021). A utilização de Software que utilizam arquitetura de microsserviços é uma prática comum entre grandes organizações como: NETFLIX, TWITTER, AMAZON e UBER (ARCURI, 2019). Uma aplicação que utiliza a arquitetura REST, é baseada em protocolos cliente/servidor, que processam as solicitações inseridas pelos usuários e retornam as respostas devidamente apropriadas, onde essas solicitações são enviadas através de URLs, que são utilizadas conjuntamente com um protocolo HTTP, resultando em um serviço REST (ALAM et al., 2020).

Na maioria das vezes, realizar testes entre a comunicação dessas aplicações se torna oneroso, pois em um ambiente competitivo, a necessidade de acertar e corrigir rápido um determinado erro na funcionalidade do Software, tem sido um fator de relevância para as empresas. A atenção ao processo pela qualidade de Software tem sido um requisito demandado pelos usuários que buscam como resultado das funcionalidades inseridas na aplicação, seu devido funcionamento e eficiência (RODRIGUES, 2018). Um Software de qualidade, identifica-se como aquele que atende as necessidades dos usuários (RODRIGUES; FARINA, 2019). Na medida em que um sistema cresce, novas funcionalidades são inseridas e essas estão sujeitas a falhas em seu comportamento esperado (TRINDADE, 2021). O processo do desenvolvimento de um Software é uma atividade na qual requer concentração, pois

uma vez que uma falha é inserida na lógica implementada no produto, o custo do Software tende a ser elevado (SPIRLANDELI, 2019).

Quanto mais cedo as falhas de um Software forem mapeadas, menor será o seu custo de correção (TRINDADE, 2021). Existem diversas técnicas de testes que podem ser aplicadas nesse processo de validação para o mapeamento de erros, sendo: manuais e automatizadas, e para cada uma delas existem ferramentas e frameworks capazes de auxiliar nesse processo de teste. No entanto, a prática pela técnica do teste manual, tende a resultar no atraso da entrega do Software, pois a realização dos testes funcionais tende a ter um tempo mais elevado e com uma escalabilidade de performance menor, consequentemente o custo do teste é maior (CURTI; DALLILO, 2022).

A prática da utilização de ferramentas de testes automatizados é adotada para evitar que testes repetitivos sejam executados de forma manual e também é apresentada como solução para a redução do tempo e do custo com o teste (TRINDADE, 2021). A aderência do teste automatizado é uma das atividades que vem ganhando espaço no meio corporativo, por parte de testadores de Software (RODRIGUES; FARINA, 2019). O processo de automação de teste pode ser definido como uma substituição parcial ou total da intervenção humana, através de testes baseado em ferramentas autônomas, nos quais são construídos por humanos e executados pela máquina através de simples comandos (RODRIGUES, 2018).

Embora a realização de testes mais robustos e automatizados para esse tipo de arquitetura de microsserviços seja importante, as abordagens sobre o teste automatizado de APIs RESTful são limitadas (SAHIN; AKAY, 2021). A realização de testes em uma API REST apresenta desafios, pois diversas requisições e respostas de entradas e saídas são realizadas em um servidor remoto, o que facilita a possibilidade de erros nas funcionalidades por existir uma vasta quantidade de registros e processos a serem testados, (ARCURI, 2019), (SAHIN; AKAY, 2021).

A importância deste trabalho está relacionada principalmente com a redução dos testes manuais realizados por analistas de testes de Software, com o uso da automação de testes. Emprega-se neste trabalho ferramentas Open Source (código aberto), que permitem a partir de scripts, melhorar a performance da execução da atividade em um ambiente produtivo, contribuindo assim com o processo de garantia da qualidade do Software e consequentemente reduzindo os custos com o teste de Software. O processo de automação de testes vem ganhando espaço no setor da

qualidade de Software, e este trabalho poderá contribuir tanto para a literatura quanto para a construção de projetos simplificados com a junção das ferramentas e frameworks escolhidos.

1.1 Caracterização do problema

Considerando o crescimento exponencial do mercado digital e suas tecnologias, as aplicações de Software, estão cada vez mais sujeitas a erros e vulnerabilidades em seu processo de qualidade (RODRIGUES, 2018). Software que utilizam a arquitetura de microsserviços, precisam ser devidamente testados de forma que todos os pontos de comunicação entre as aplicações, sejam validados e apresentem o resultado esperado (ARCURI, 2019). Embora alguns estudos abordem ferramentas distintas que auxiliam analistas de testes a validarem esse tipo de atividade, a falta de conhecimento técnico é um fator de desafio no momento de implementar seus conhecimentos na escolha da tecnologia a ser empregada no contexto a ser testado (RODRIGUES, 2018).

Validar a funcionalidade de uma API REST é um fator imprescindível, pois a qualidade e o desempenho dos serviços WEB precisam estar aderentes ao que foi especificado (BANIAȘ et al., 2021). Uma das formas de se avaliar a integração entre diferentes microsserviços é através de testes baseados em: end2end, testes de integração, testes de regressão e testes funcionais. A realização dos testes de integração e regressão em uma API REST, determina se a lógica da API corresponde adequadamente às expectativas da escalabilidade, testabilidade, funcionalidade e confiabilidade do Software (LY, 2018).

Escolher o framework ou a ferramenta adequada para cada contexto de testes, requer maturidade no conhecimento técnico do assunto. O problema enfrentado neste estudo, é como auxiliar os analistas de testes de uma organização X, a otimizarem seu tempo de testes e melhorarem o processo de qualidade no Software através do teste automatizado.

.

1.2 Objetivos da pesquisa

Este trabalho tem como objetivo principal apresentar uma técnica de testes automatizada em uma API REST, na qual reduza o tempo e o custo dos testes de Software, sendo eficiente no processo de execução e validação dos testes, consequentemente reduzindo o risco a falhas no Software e possibilitando que o critério de aceite dos valores enviados pela API estejam 100% aderentes a documentação, com a utilização de um framework de testes automatizados. A redução do custo do teste será avaliada através de indicadores de qualidade de testes empregados de acordo com o contexto da organização.

A fim de atingir o objetivo geral, designaram-se os seguintes objetivos específicos:

- Mapear as principais requisições utilizadas pela API REST;
- Desenvolver as abordagens dos cenários de testes a serem automatizados;
- Desenvolver scripts de testes automatizados baseados no processo de validação funcional dos cenários levantados para a automação;
- Testar a técnica aplicada para a execução do teste desenvolvido, calculando assim o tempo de teste realizado a partir de um indicador de tempo e comparando com o teste manual, podendo calcular e estimar o tempo gasto com cada processo;
- Apresentar um relatório comparativo do tempo de testes em execução através das ferramentas escolhidas e também do processo manual.

1.3 Questões da pesquisa

Este trabalho tem como foco a construção de um processo de testes automatizados para uma API REST, como forma de avaliar a otimização do tempo demandado pela equipe de testadores, avaliando a redução do esforço manual.

- Por que a técnica de testes automatizada tende a reduzir o esforço manual e consequentemente aumentar o nível da qualidade do produto?

- Quais são as principais ferramentas e frameworks que são utilizadas para validar o processo de automação de testes em sistemas WEB, no qual utiliza a arquitetura REST?
- Quais as tecnologias e frameworks utilizadas para o desenvolvimento dos testes automatizados da API REST?
- Como pode-se avaliar qual ferramenta, framework ou técnica pode ser aplicada no contexto de testes automatizados de API REST?

1.4 Organização da dissertação

O primeiro capítulo introduziu o contexto geral da definição do problema e as devidas questões da pesquisa, bem como uma abordagem geral dos objetivos e a metodologia de pesquisa proposta.

No segundo capítulo será apresentado o referencial teórico da pesquisa, abordando os principais conceitos de API REST e testes de Software. Também, serão apresentados os principais conceitos relacionados aos testes automatizados em sistemas WEB, nos quais utilizam a arquitetura REST, bem como uma abordagem sobre as tecnologias que podem ser utilizadas para tal processo, os desafios, benefícios, pontos de atenção na aderência ao seu uso, assim como trabalhos relacionados ao presente contexto desta pesquisa.

No terceiro capítulo será apresentada a estratégia de testes proposta e as tecnologias empregadas no desenvolvimento deste projeto, bem como a metodologia empregada para alcançar os resultados esperados.

No quarto capítulo serão apresentados os resultados obtidos através da aderência das tecnologias propostas resultantes do estudo realizado no capítulo quatro.

Por fim, o quinto capítulo apresentará as conclusões deste trabalho e a abertura a sugestões para trabalhos futuros.

2 REVISÃO DE LITERATURA

No desenvolvimento de Software que utilizam a arquitetura de microsserviços, a aderência pelo desenvolvimento utilizando a interface de programação (API), tem se expandido. API pode ser descrita como mediadora entre cliente e servidor, que possibilita a comunicação entre serviços (ARCURI, 2019). Uma vez que Software são desenvolvidos com o uso de APIs, deve-se manter um padrão de qualidade elevado, para que este possibilite a comunicação entre as integrações desenvolvidas de forma segura (BANIAȘ et al., 2021).

O teste de Software é uma técnica que possibilita o processo de melhoria na garantia da qualidade (TRINDADE, 2021). Diversas técnicas e frameworks de testes podem ser empregadas para a realização de testes de APIs, porém precisa-se levar em consideração que o teste depende do contexto no qual será avaliado. Aplicações que utilizam API REST e suas arquiteturas, necessitam ser testadas com agilidade e também com a garantia de que todos os fluxos possíveis de comunicação sejam efetivamente validados. A validação destes processos pode ser realizada tanto de forma manual como de forma automatizada (RODRIGUES, 2018). A diferença que é empregada para cada uma delas, é o custo e o tempo no qual são levados em consideração para sua execução. Escolher a melhor técnica se torna um processo complexo, visto que existem inúmeras tecnologias de mercado que podem ser exploradas e utilizadas no contexto.

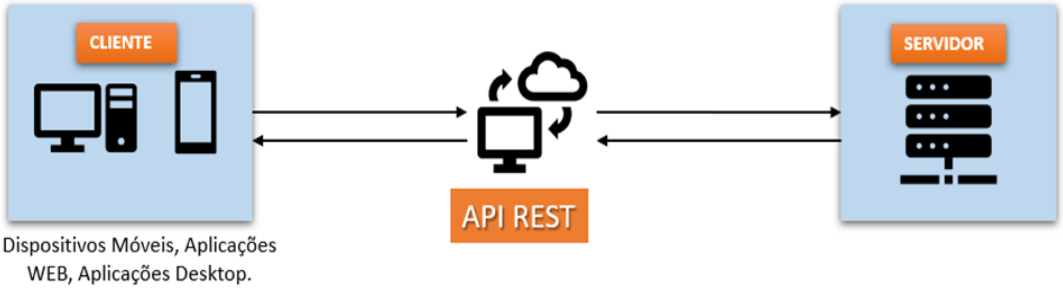
Este capítulo apresenta a revisão da literatura desta pesquisa, inicialmente descrevendo os principais conceitos relacionados a API REST e suas arquiteturas. A segunda abordagem deste capítulo está relacionada às terminologias e conceitos de testes de Software, no qual conceitua os processos de testes, aplicações, tipos e técnicas de testes utilizados diante de cada contexto a ser executado. Ao final do capítulo, também será apresentado um embasamento teórico sobre ferramentas e frameworks de testes automatizados, nos quais auxiliam no processo da qualidade de Software, e como elas proporcionam uma redução no custo e no tempo do teste levando em consideração sistemas que utilizam arquitetura de microsserviços.

2.1 API REST

APIs REST nada mais é do que uma interface de programação de aplicações que são desenvolvidas principalmente para aplicativos que utilizam arquitetura de microsserviços e de protocolos HTTP, que realizam a comunicação de informações entre estes (ARCURI, 2019). Serviços REST, utilizam combinações de padrões HTTP, JSON e XML para descrever suas diretrizes de desenvolvimento de serviços HTTP (BANIAȘ et al., 2021).

A figura 1, apresenta a comunicação entre o Cliente e o Servidor que é realizada através de uma API REST, sendo os mais diversos dispositivos realizando solicitações a essa API. Ao realizar a solicitação o servidor retorna à informação através da API REST para o cliente. A solicitação é realizada através de uma URL do endpoint da API REST, que inclui as operações de CRUD sendo: (GET, POST, PUT, DELETE) e também seus devidos parâmetros, onde a resposta a essa solicitação é dada através do formato JSON ou XML (BANIAȘ et al., 2021).

Figura 1 - Representação da funcionalidade de uma API REST, representando a comunicação entre cliente e servidor.



Fonte: Adaptado de Banias et al.,(2021 p. 4).

Além das operações CRUD que são disponibilizadas pela API, também existem os códigos de status que são utilizados para retornar o status da operação realizada (SUZANTI et al., 2020). O código de status é um número contendo três dígitos, ex: 200, 201, 404, 422 e 503 (BANIAȘ et al., 2021). Existem inúmeros códigos que retornam valores resultantes de operações em uma API REST, como representado na figura 2.

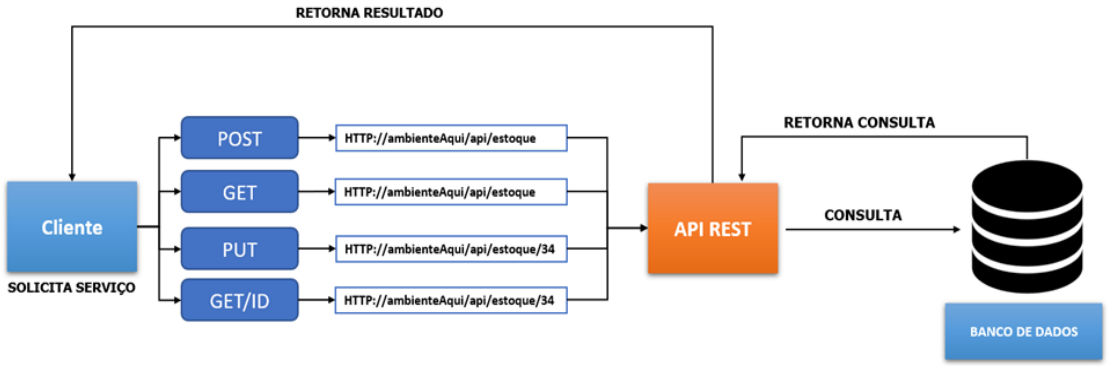
Figura 2 - Representação do código de status que pode ser retornado em operações CRUD.

REQUISIÇÕES HTTP	OPERAÇÃO	STATUS CODE - HTTP
POST	Create/Criação	Sucesso: • 201 – create • 200 – ok Erro: • 400 (Bad Request) • 404 (Not Found) • 409 (Conflict) • 422 (Unprocessable Entity)
GET	Read / Consulta	Sucesso: • 200 – ok Erro: • 400 (Bad Request) • 404 (Not Found)
PUT	Update/Replace/Alteração	Sucesso: • 200 – OK • 204 – (No Content) Erro: • 404 (Not Found)

Fonte: Adaptado de (MARQUES, 2018).

A Figura 3, apresenta o processo simplificado de algumas solicitações que são realizadas através da comunicação cliente / servidor. Essa comunicação é realizada pelo cliente que envia sua solicitação a API REST através de operações HTTP e suas devidas URIs, na qual a API REST realiza a consulta no banco de dados da aplicação e retorna essa consulta para o cliente. Uma URI é definida com base na estrutura dos métodos solicitados e é utilizada para identificar um recurso tornando-o endereçável (ARCURI, 2021). Considera-se então na Figura 2, um exemplo de um serviço REST, que permite gerenciar estoques de determinados produtos através de inserções, alterações e consultas.

Figura 3 - Representação das requisições cliente/servidor através do protocolo HTTP: POST, GET, PUT.



Fonte: Adaptado de Suzanti et al,(2020 p.12).

A utilização da arquitetura REST, dispõe de uma gama de orientações necessárias para desenvolver serviços coesos que sejam escaláveis e com uma performance com alto desempenho (MARQUES, 2018). O padrão de arquitetura REST indica aos desenvolvedores que estes utilizem métodos HTTP de maneira consistente de acordo com seus protocolos devidamente estruturados, onde pode-se ter operações de criação, atualização, leitura e remoção de dados através destes métodos (ARCURI, 2021), (MARQUES, 2018). Ainda de acordo com os autores, essas operações indicam que para que um novo recurso possa ser criado através da utilização de um serviço WEB, o método POST deve é utilizado.

- Para que um recurso já existente possa ser recuperado, o método GET é utilizado;
- Para que um recurso já existente possa ser alterado, o método PUT é utilizado;
- Para que um recurso já existente possa ser removido, o método DELETE é utilizado.

Diante do contexto de desenvolvimento baseado em API REST utilizando a arquitetura de microsserviços, deve-se garantir que as solicitações e respostas a essas, estejam devidamente aderentes ao processo, pois um retorno de status ou uma informação vinda de forma errônea pode acarretar a inúmeras falhas em operação do Software. Com isso a aderência pelo teste de Software deve ser levada em consideração para os processos de validação destes serviços, geralmente os tipos de testes mais utilizados neste contexto são: testes unitários, testes de integração, testes de componentes e os testes end-to-end (LIMA et al., 2021). O teste de uma API REST, apresenta desafios, pois possuem em resposta a suas requisições HTTP, diversas entradas e saídas (ARCURI, 2019).

2.2 Terminologia e conceitos de teste de *Software*

O processo de Teste de Software contribui de forma efetiva para o processo de melhoria da qualidade do produto (TRINDADE, 2021). O teste depende do contexto em que o mesmo é solicitado, sendo assim existem diferentes tipos, técnicas e ferramentas específicas para cada contexto.

Este tópico tem como objetivo apresentar os conceitos e terminologias sobre os testes de Software, que são apresentados e conceituados pela literatura, abordando suas principais atividades dentro de um contexto de desenvolvimento de Software. Será abordado também, as principais técnicas de testes e o uso de tecnologias relacionadas ao contexto de testes automatizados.

2.2.1 Teste de *Software*

O Teste de Software é uma das atividades mais adotadas que são utilizadas para avaliar o processo de qualidade de um Software, pois permite verificar se a aplicação funciona da forma que é especificada, ou seja, para identificar antecipadamente todo e qualquer erro, defeito ou falha que venha surgir durante sua operação (PINHEIRO, 2014). O teste fornece o último elemento a partir do qual a qualidade pode ser estimada e especificamente os erros podem ser descobertos antes mesmo da sua implantação (PRESSMAN et al., 2016).

Segundo (RODRIGUES, 2018) o processo de teste de Software, envolve uma vasta quantidade de entradas e os mais diversos caminhos possíveis de processos a serem executados e devidamente testados. Ainda de acordo com o autor, na medida em que o Software cresce a complexidade dessas entradas e caminhos podem possuir incontáveis possibilidades a serem testadas, o que torna um tempo ainda maior na execução da verificação de um Software.

Um dos objetivos do teste de Software é encontrar erros e defeitos nos quais estão inseridos nas funcionalidades que estão contidas dentro de um conjunto de cenários a serem validados (TRINDADE, 2021).

De acordo com Pereira (2019), o teste de software durante o desenvolvimento é um fator de alto custo. Quanto antes os defeitos forem encontrados, menor será o seu custo de resolução, pois atualmente há necessidade de validar os sistemas de forma mais rápida e segura para que o quanto antes o mesmo possa ser disponibilizado em um ambiente de produção (TRINDADE, 2021).

As técnicas de validação e verificação de um software, são compostas por diversas atividades para verificar se a aplicação e sua construção, seguem de acordo com a especificação do que foi solicitado, este processo de verificação e validação,

estão inseridos dentre as atividades propostas no processo de Garantia da Qualidade de Software (PINHEIRO, 2014).

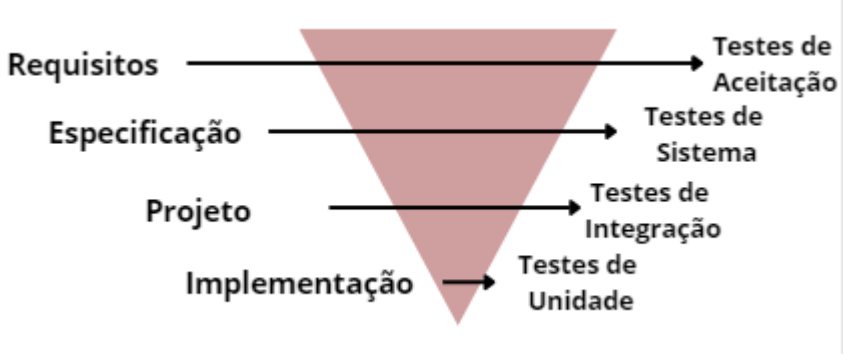
Os testes podem ser fundamentados como uma das principais técnicas de redução de manutenção e escalabilidade na melhoria da qualidade do Software (ARCURI, 2019). A confiabilidade do Software, requer um comportamento adequado de suas funcionalidades (RODRIGUES, 2018). Realizar testes de acordo com cada contexto, é uma prática que quando adotada corretamente dentro do processo, tende a alcançar o nível de qualidade esperado para o produto.

Dentro do processo de testes de Software, existem as fases em que diferentes tipos de testes podem ser empregados. As fases de testes nada mais são do que processos de testes executados em determinados momentos do projeto (RODRIGUES, 2018).

Um dos processos implementados para a verificação e validação do Software em seus diferentes estágios de desenvolvimento, é o “modelo v”. O modelo V, se trata de um modelo sequencial que representa o ciclo de vida de um Software em desenvolvimento, descreve a relação de cada fase com seu respectivo processo e o que pode ser empregado em cada estágio, abordando as fases de testes desde o teste do Software até o teste de componente ISTQB Glossary.

A figura 4, apresenta o exemplo da utilização dos testes em diferentes camadas do desenvolvimento, apresentando seus tipos e em que fase estes podem ser utilizados.

Figura 4 - Fases em que o teste é empregado de acordo com o Modelo V.



Fonte: Adaptado de Rodrigues, (2018, p 45).

Os testes podem ser fundamentados como uma das principais técnicas de redução e manutenção da escalabilidade do processo de melhoria na garantia da qualidade do Software.

2.2.2 Tipos de Teste de *Software*

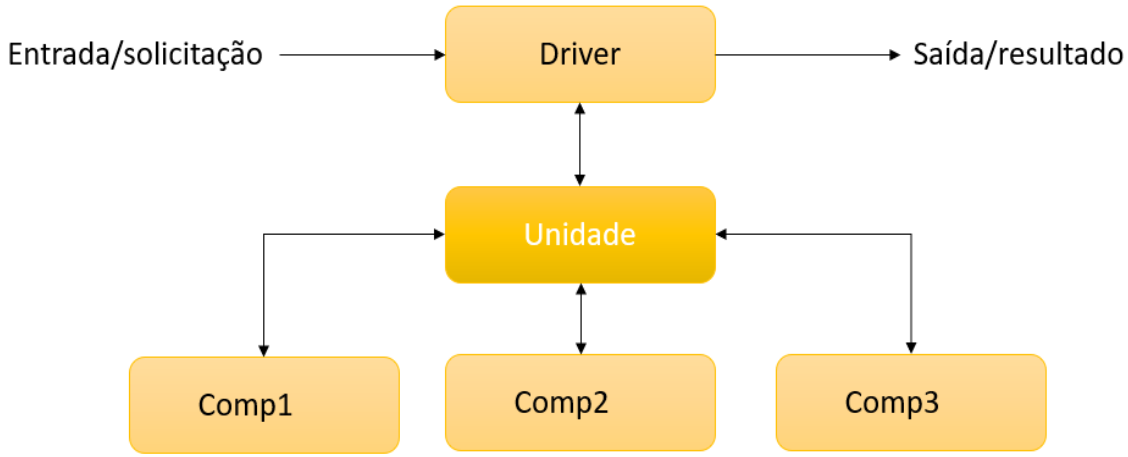
A fase do teste de Software é essencial para auxiliar no processo da garantia da qualidade, onde testar um Software se baseia em executar o programa total ou partes deste, a fim de avaliar se o mesmo está funcionando conforme o esperado (ARAUJO, 2021). Diversos são os tipos de testes de software que podem ser realizados por inúmeras pessoas em distintas etapas de um projeto, com objetivos diferentes. A literatura aborda inúmeros tipos de testes de Software.

Pinheiro, (2014) em seu trabalho, aborda que as atividades de testes de software podem ser divididas em três etapas, tendo cada um seu objetivo específico diante do contexto, sendo: teste unitário, teste de integração e o teste de sistema. Já (PRESSMAN, 2011) aborda diferentes tipos de testes: Teste de Caixa Preta, Teste de Caixa Branca, Teste Unitário, Teste de Regressão e Teste de Integração.

O teste unitário é uma prática que geralmente é realizada pelos desenvolvedores, em unidades da aplicação de forma individual, esse teste verifica se as partes do Software estão funcionando adequadamente de forma isolada das demais (BERNARDO, 2011), (PRESSMAN, 2011). A unidade a ser testada, consiste em uma parte menor do Software que pode ser testada de forma separada e independente das demais (PINHEIRO, 2014), (RODRIGUES, 2018).

A Figura 5, representa a forma como os processos de testes de unidade são executados, pode-se observar que existe entradas e saídas de solicitações que são responsáveis por validar os dados válidos e inválidos via entradas e saída. (PRESSMAN, 2011) descreve a solicitação de uma chamada ao driver que sequencialmente submete-se a uma chamada ao componente do teste, que fornece as entradas e monitora as execuções solicitadas, gerando assim as saídas resultantes do teste. Os componentes abaixo da unidade, simulam o comportamento dos módulos em que o mesmo em fase de teste depende

Figura 5 - Representação da aplicação do Teste Unitário.



Fonte: adaptado de Pinheiro (2014, p 35).

Após a execução dos testes de unidade, a fase seguinte é a execução dos testes de integração, que tem como objetivo principal realizar a validação da comunicação entre todos os componentes. Trindade (2021) descreve que o teste de unidade é o teste de camada mais baixa que serve como primeira linha de defesa, que permite a detecção de bugs e previne que estes não voltem a ocorrer.

O teste de integração tem como objetivo, a execução do sistema por inteiro, onde nessa etapa todos os componentes no Software são testados de forma conjunta, (PINHEIRO, 2014) e (TRINDADE, 2021), apresentam o teste de integração como uma camada aplicacional que tem como objetivo a validação das regras de negócio do sistema, incluindo testes End-to-End como forma de validação da interação entre os componentes e a interface do Software. É um teste executado em todos os módulos do software validando se estes interagem de forma correta e estável conforme o esperado (RODRIGUES, 2018).

Este teste inclui desde validações da integração entre pequenas unidades do Software até a integração de unidades dependentes como: servidores e banco de dados. (BERNARDO, 2011) e (PRESSMAN, 2011), apresentam um exemplo disso sendo um fluxo de atividade de uma parte do Software para outra, validando se o comportamento se encontra como o esperado.

Pinheiro (2014), descreve o teste de caixa preta como um teste baseado na especificação do sistema, onde avalia-se os requisitos funcionais do Software, o mesmo independe da linguagem de programação. Este teste é realizado sem

referência a sua estrutura interna, ou seja, não necessita que o código fonte esteja aberto para a execução dele (TRINDADE, 2021). O objetivo é verificar se o Software está produzindo as saídas esperadas de acordo com as entradas inseridas no teste, (PRADO, 2018). Nesse caso, tudo aquilo que está relacionado com a implementação do Software ficará em segundo plano, para que a funcionalidade do Software ganhe um maior destaque e seja validada, (ARAUJO, 2021).

O teste de caixa branca envolve tudo aquilo que está relacionado com a implementação do Software e fica em primeiro plano, para que assim os casos de testes possam ser derivados. (ARAUJO, 2021), (TRINDADE, 2021) e (RODRIGUES, 2018), descrevem o teste de caixa branca como um procedimento executado com base na estrutura interna do Software, ou seja, necessita que o código fonte esteja visível para a execução do teste. O objetivo é verificar se a lógica interna do Software tem a sua devida cobertura de testes, (PRADO, 2018).

O teste de regressão é realizado em um sistema ou componente no qual já foi testado anteriormente, porém sofreu uma modificação, neste caso o teste de regressão é executado a fim de garantir que defeitos não tenham sido introduzidos em áreas nas quais o software não sofreu modificações, (TRINDADE, 2021).

Além dos principais tipos de testes mapeados pela literatura, aborda-se o teste E2E (end-to-end), que é descrito como teste de ponta a ponta. Trindade (2021), descreve o teste End-to-End como processos executados seguindo um passo a passo, para reproduzir um comportamento funcional do sistema, este tipo de teste foca no comportamento do sistema, descrevendo a funcionalidade de uma forma onde a sua compreensão seja acessível. Os testes E2E, são conhecidos como testes de aceitação e de funcionalidade, estes testes normalmente podem levar um tempo para serem executados, pois eles testam toda a lógica e todas as funções do sistema (ERONEN, 2019).

2.2.3 Técnicas de teste de *Software*

Existem algumas técnicas de testes de Software que são retratadas pela literatura, porém duas delas são as abordagens estratégicas mais utilizadas: Teste Manual e Teste Automatizado. De acordo com o (PRESSMAN, 2011) as técnicas de testes têm como objetivo identificar as condições, casos e dados. Escolher qual a

técnica na qual será utilizada, depende de alguns fatores, podendo destacar entre eles:

- Nível de complexidade do componente ou sistema a ser testado;
- Requisitos contratuais ou requisitos solicitados pelo cliente;
- Níveis e tipos de riscos nos quais tendem a ser influências diretas na qualidade do Software;
- Documentação disponível para teste;
- Conhecimento e habilidade do analista de testes;
- Ferramentas disponíveis;
- Tempo e custo;
- Modelo do ciclo de vida utilizado no desenvolvimento.

A técnica de teste manual, tende a resultar em um processo de atraso na entrega do Software. Realizar testes funcionais em determinados pontos do sistema, demanda um tempo e consequentemente um custo elevado (CURTI; DALLILO, 2022).

O teste manual é considerado uma prática de testes dependentes de uma intervenção humana, onde após a implementação da funcionalidade é realizado um processo manual para verificar se tudo que foi desenvolvido está funcionando conforme o esperado (BERNARDO, 2011) e (TRINDADE, 2021).

A execução de um teste manual pode ser rápida e efetiva, porém a repetição manual de uma quantidade relativa de conjunto de testes é uma tarefa exaustiva e cansativa, pois envolve muito esforço e tempo dos profissionais nos quais estão inseridos no processo (RODRIGUES, 2018).

De acordo com Bernardo (2011), erros no Software trazem grandes prejuízos para a organização como um todo e também demandam um tempo para identificar e corrigir os erros, muitas vezes se torna um processo demorado, o que ocasiona atrasos nos prazos combinados e na entrega do Software com a qualidade comprometida.

Com isso, a necessidade de melhorar a performance, o ganho de tempo na execução dos testes e a redução dos custos, a automação é levada em consideração para os processos de melhoria na qualidade do produto.

Ao contrário do teste manual, o teste automatizado é considerado uma prática independente de uma intervenção humana, seu objetivo é melhorar a qualidade do Software através da validação e verificação do produto (BERNARDO, 2011). Automatizar um teste consiste basicamente na conversão das atividades que são

executadas de forma manual em uma sequência de passos pré-definidos, executados através de scripts (RODRIGUES, 2018), (TRINDADE, 2021).

Apesar do teste automatizado ser efetivo, ele não pode substituir totalmente o teste manual, pois os desenvolvedores definem com propriedade dados de entrada de alta complexidade e por este fato, na maioria das vezes a definição de casos de testes específicos não é levada totalmente em consideração diante do contexto.

Com a aderência do teste automatizado de Software, o tempo de sua execução é reduzido de forma significativa (FONSECA, 2021). Portanto, aderir ao teste automatizado, favorece o aumento da performance da equipe, auxilia na melhoria da qualidade do Software, contribui para a escalabilidade dos testes manuais, possibilita a não execução de testes exaustáveis que demandam um alto custo e um tempo elevado para sua execução (RODRIGUES; FARINA, 2019).

Diversas são as técnicas que podem-se utilizar para realização dos testes de acordo com cada contexto. Lima et al., (2021), propõe em seu trabalho a técnica de BDD (Behaviour-Driven Development), para a construção de cenários de testes que serão escritos por usuários não técnicos e consequentemente o desenvolvimento dos testes automatizados em uma camada de serviço de uma API REST. Laranjeiro et al, (2021) em seu trabalho, aborda a ferramenta de bBOXRT que utiliza um documento de descrição de serviço como entrada para gerar um conjunto de entradas inválidas para a realização dos testes automatizados da API REST.

Outra técnica que pode ser utilizada para a geração de casos de testes automatizados é RESTTESTGEN que é apresentada por Viglianisi et al, (2020) essa técnica se baseia em uma documentação (Swagger), que inclui a lista de operações disponíveis nos formatos de dados de entrada e saídas das solicitações realizadas na API. Em contrapartida, Banias et al., (2021) propõe outro método de validação de qualquer API REST, através da especificação de sua OpenAPI. Nessa abordagem diversos critérios de cobertura e métricas de desempenho são inseridos para criar uma estatística escalável do teste da API.

A avaliação de qual ferramenta, técnica ou framework a ser utilizado no contexto a ser testado, precisa ser estudada e mapeada de forma que a escolha resulte na aderência correta para o processo.

2.2.4 Teste automatizado de API

À medida em que as APIs REST da Web evoluem e se tornam progressivamente base de integração de Software, a sua validação tem se tornado cada vez mais crítica (SEGURA et al., 2018). Santos, (2018) retrata que a medida em que uma API evolui, componentes nos quais estão relacionados com a API tendem a sofrer alterações. Quando uma nova modificação é inserida na funcionalidade de uma API, não é totalmente possível avaliar todos os impactos que podem acontecer nas diversas partes do sistema, quando utiliza-se apenas os testes manuais como verificação dessa inserção (FILHO, 2021).

Uma API REST precisa estar em excelente funcionalidade, para isso os testes de Software são empregados neste processo, para auxiliar na garantia da qualidade dos serviços REST. A realização dos testes de API se torna desafiadora principalmente devido à dificuldade de avaliar se as saídas de uma chamada estão corretas (SEGURA et al., 2018). Os testes de API são cruciais para auxiliar na garantia da funcionalidade, confiabilidade, qualidade entre outros fatores, nos quais evitam que a API apresente um mau funcionamento ou que a mesma tenha processos ineficazes (BANIAȘ et al., 2021).

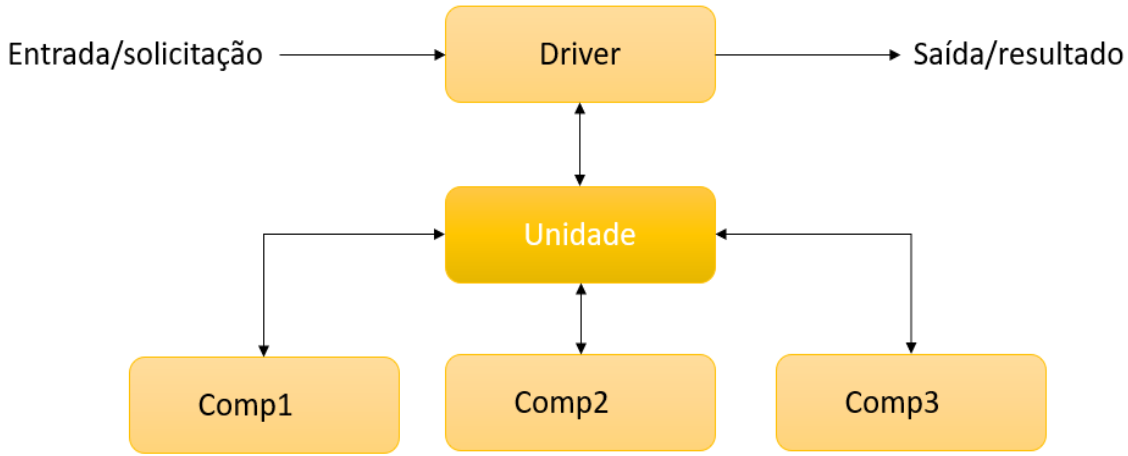
Realizar testes de API é uma fase importante no processo de validação da qualidade, pois essa camada de testes está associada a uma proximidade maior do usuário, e também está conectada ao cliente e servidor (BANIAȘ et al., 2021). No entanto, realizar testes em uma API REST, é um desafio se tratando principalmente da dificuldade em avaliar se a saída de uma determinada chamada da API, está de acordo com o resultado esperado (SEGURA et al., 2018).

Quando uma API é construída faz-se necessário o desenvolvimento de sua documentação de contrato, pois é através dessa abordagem que caso ocorra uma evolução na API, os dados da documentação possam ser validados e atualizados de forma dinâmica, (SANTOS, 2018). Ainda segundo o autor, testes de aceitação podem ser utilizados para a verificação da documentação da API, validando se o retorno esperado no contrato está de acordo com o resultado do teste executado.

Processos de testes automatizados de Software, têm ganhado importância no meio coorporativo, por empresas que buscam pela qualidade de seus produtos (RODRIGUES; FARINA, 2019).

Na figura 6, pode-se observar um exemplo de uma documentação desenvolvida para uma API que tem por finalidade o gerenciamento de marcas, contendo algumas requisições sendo: GET, POST, DELETE, GET por ID e PUT. De acordo com (FILHO, 2021), os testes de API podem ser divididos em duas partes distintas, sendo: [i] testes funcionais, que validam a funcionalidade que a API propõe a executar, nesse processo de teste muitas vezes se faz necessário a execução de mais de uma requisição para validar se as informações realmente estão corretas por meio dos métodos POST, PUT e GET. [ii] Testes técnicos, cujo seu objetivo é validar se a API está funcionando de acordo com o que está especificado na documentação de apoio da API, este teste é baseado em contrato, ou seja, válida se o que foi implementado está de acordo com o que está estabelecido no contrato.

Figura 6 - Página da Documentação da API REST, desenvolvida com SWAGGER UI.



Fonte: adaptado de Santos (2018 p, 38).

Realizar um teste de API de forma manual, demanda um tempo significativo e relativamente o seu custo de execução é maior, utilizando de um exemplo de uma API na qual possui em funcionalidade cerca de 25 tipos de serviços, testar a integração entre estes serviços, seus retornos, cenários de sucesso e erro, cenários de regra de negócio, aplicação de testes de integração, funcional e E2E, podem demandar um esforço não escalável.

Uma das grandes vantagens de se aplicar a automação dos testes segundo (BERNARDO, 2011), é a abordagem de que a inserção de casos pode ser escalável em sua execução, podendo repetir a qualquer momento do processo os cenários

descritos, reduzindo o tempo de teste e os esforços manuais empregados. Sequenciar chamadas, obter e comparar respostas imprevisíveis, lidar com casos de testes não planejados que resultam em falhas, parametrizar rotas para testes, são os principais desafios enfrentados durante o processo de teste de uma API REST (LIMA et al., 2021).

Algumas perguntas precisam ser levadas em consideração, no momento em que se pretende aplicar testes em uma API.

1. Quais são as chamadas existentes na API?
2. Quais os status de retorno esperados?
3. Quais são os requisitos definidos para cada dado enviado na requisição dos métodos solicitados?

É necessário também levar em consideração algumas validações cruciais de testes que são necessárias serem validadas no momento em que se iniciam os testes em uma API REST, sendo:

- Status Code: (código de status retornado), sendo estes retornos de sucesso e erro, é necessário validar estes pontos para garantir que a API esteja funcionando conforme o esperado.
- Body do Response: é necessário validar os campos que são retornados no arquivo JSON por exemplo, independente de qual técnica e ferramenta de teste automatizado está sendo utilizada.
- BODY do envio de um arquivo seja JSON ou XML: deve-se validar o envio correto das informações bem como a estrutura incorreta, a fim de verificar o comportamento da API.
- Header: é necessário à validação do header como forma de garantir a segurança do acesso à aplicação, pois é neste momento que são informadas as credenciais de acesso a parametrização do endPoint que está sendo utilizado.
- Validar os retornos esperados de acordo com o que está presente na documentação da API.

Estes pontos devem ser levados em consideração devido ao custo da criação de testes automatizados que normalmente é maior do que os manuais, pois o tempo em que se utiliza implementando o teste, torna-o maior do que uma execução de forma manual (BERNARDO, 2011).

À medida em que as APIs REST ganham impulso, o mesmo acontece com os testes que precisam ser executados nelas, neste cenário é necessário medir e realizar

uma comparação automática da eficácia de um bom teste. Antes de iniciar um processo de automatização de testes de API, faz-se necessário ter uma ferramenta de automação na qual seja específica para testar os serviços da API (PEREIRA, 2019). Escolher a ferramenta ideal consiste em entender através do contexto empregado, qual teste é necessário de ser executado. Diversas são as ferramentas e frameworks de automação de testes que são disponibilizadas para esse tipo de processo (SPIRLANDELI, 2019).

A execução do teste manual de API é uma prática que é comum nos processos de testes, pois pode-se contar com a utilização de ferramentas que possibilitam a visualização e execução dos mesmos, como: Postman, SoapUI e Apache JMeter.

Portanto, realizar os testes de API se torna uma parte importante na fase de testes de Software, pois é essa camada que está mais próxima do usuário e também realiza o processo de conexão entre o cliente e servidor (BANIAȘ et al., 2021). No entanto, quando os temas são relacionados a testes automatizados para Software que utilizam arquitetura de microsserviços, poucos são os resultados obtidos, tornando a pesquisa limitada (SAHIN; AKAY, 2021).

2.3 Ferramentas e frameworks para testes automatizados de API

A realização dos testes automatizados, conta com ferramentas especializadas que otimizam a escrita, escalabilidade, execução e manutenção dos testes (BERNARDO, 2011). As ferramentas de automação de testes estão sendo cada vez mais utilizadas pelas equipes de testes de software. Existem diversos tipos de ferramentas nas quais são utilizadas para o processo de automação de testes, onde estas podem serem utilizadas em diferentes contextos ou áreas (RODRIGUES, 2018).

Escolher a ferramenta adequada para o processo de automação de testes, é ideal para que obtenha o resultado e também possibilita que os testes possam ser executados de forma frequente e escalável com um curto espaço de tempo, que vai ao oposto dos testes manuais nos quais necessitam de uma grande quantidade de testes sendo executados em grande escala de repetições demandando um alto custo e tempo do teste (BANIAȘ et al., 2021).Cada ferramenta de testes automatizados tem suas devidas particularidades, sendo assim é fundamental antes de iniciar o processo

de automatização do teste, avaliar as vantagens e desvantagens de cada ferramenta escolhida, e relacionar a escolha ao contexto do teste a ser empregado.

Arcuri (2019) em seu trabalho realiza uma abordagem de geração automatizada de testes de integração para serviços RESTful, utilizando uma técnica de validação RESTful de forma isolada, para assim abordar o teste de caixa branca e utilizar o algoritmo MIO (Muitos Objetivos Independentes), para a geração de casos de testes. Embora essa técnica seja aderente ao processo em execução, existem outras abordagens que podem ser utilizadas, como o framework Jest.Js que é apresentado por Spirlandeli, (2019). O framework proporciona a melhoria no processo de desenvolvimento de Software e possibilita que este se torne mais intuitivo em um contexto de testes automatizados para Back-end. JestJs é um framework de testes automatizados, desenvolvido por JavaScript e seu foco é baseado na simplicidade da sua utilização, e também pode ser utilizado para testes de API.

O cucumber é outro framework de testes automatizados, no qual pode ser utilizado para a realização de testes de API REST, porém deve-se levar em consideração a junção de outras tecnologias a esse framework, que possibilitam o desenvolvimento de testes de acordo com o contexto da API a ser validada. Curti; Dallilo (2022) em seu trabalho, avaliam o cucumber como uma ferramenta de testes automatizados que auxilia no processo de redução do tempo de testes executado por um analista de testes comparado com o processo manual.

Se tratando de testes automatizados que podem ser utilizados para os testes de API, o Roobot framework tem sido apresentado pela literatura como uma ferramenta promissora a esse processo, porém ainda é pouco explorada. Fonseca (2021) apresenta o Roobot framework como ferramenta de automação de testes de back-end, o resultado aborda sua execução, o tempo e o custo como fatores escaláveis na redução do teste comparada ao teste manual.

Algumas ferramentas também são mapeadas pela literatura como forma de executar testes de API REST, Corradini et al., (2021) propõe em seu trabalho, a ferramenta Restats, para a execução de testes automatizados de APIs REST e também como forma de medir a cobertura de testes de acordo com a funcionalidade da API, como forma de avaliar qual parte do processo requer mais testes. Além de ferramentas utilizadas para a execução de testes automatizados, também existem ferramentas que possibilitam a geração automática de testes automatizados, com isso Atlidakis et al., (2019) em seu estudo, abordam o RESTler como uma ferramenta

capaz de analisar a especificação de uma API REST, gerando sequências de solicitações capazes de testar automaticamente os serviços por meio de sua API REST.

A literatura apresenta diversas abordagens sobre ferramentas para automação de testes bem como os frameworks que podem ser utilizados, portanto a escassez dos temas relacionados a testes automatizados de API, foi um desafio para a descrição deste trabalho. Diversas foram as abordagens consideradas em blogs, sites e comunidades relacionadas a tecnologia, porém para essa dissertação apenas trabalhos acadêmicos foram selecionados.

2.4 Considerações finais do capítulo

No presente capítulo os conceitos básicos relacionados aos contextos de testes necessários do projeto de dissertação, foram apresentados de forma breve em três partes: principais conceitos relacionados a API REST e suas arquiteturas, terminologias e conceitos de testes de software e ferramentas e frameworks para testes automatizados de API.

Na primeira parte foram apresentados os principais conceitos relacionados a uma API REST e suas devidas arquiteturas, abordando desde as terminologias de uma API REST até a sua construção arquitetural. Estes contextos são importantes, visto que, para realizar testes sejam manuais ou automatizados de uma API REST, deve-se primeiramente conhecer sua arquitetura e funcionalidade, para que esta possa ser testada.

Por sua vez, a segunda parte aborda as terminologias e conceitos de testes de software, abordando suas aplicações. Além disso, foram apresentados os diferentes tipos de testes de software e diferentes técnicas que podem ser aplicadas em seus devidos contextos. Por fim, ainda na segunda parte é apresentado o contexto e conceito do teste automatizado com foco em API, que proporciona o entendimento dos processos de testes automatizados de software.

A terceira parte apresenta as ferramentas e frameworks que são utilizadas para testes de API REST. É importante conhecer sobre as ferramentas e frameworks a serem utilizados, pois no momento da escolha de qual será aderida, espera-se que

sejam apresentados resultados satisfatórios e qualitativos da sua aplicação para o contexto do teste a ser realizado.

Diante dos tipos de testes, técnicas e frameworks apresentados, destaca-se a técnica de testes automatizados por possibilitar o teste de regressão, integração e funcional de uma API de forma automatizada com um tempo de execução menor, quando comparado ao teste realizado de forma manual. Diversos são os frameworks e ferramentas disponíveis para testes automatizados de API estão disponíveis na literatura, porém esta pesquisa aborda a junção dos frameworks mocha e chai, para criação de scripts de testes automatizados, que atendam as especificações da API REST a ser testada. O mocha e o chai ainda são pouco explorados pela literatura, o que motiva a escrita deste presente trabalho.

Assim sendo, foi aderido o uso desta técnica e da junção dos frameworks para a melhoria no processo de qualidade e otimização dos testes de software a serem aplicados em uma API de Hub Integrador de Marketplaces ao longo deste documento.

3 MATERIAL E MÉTODO

3.1 Redução do processo de teste de API com a utilização do teste automatizado

Para que se alcance um nível de qualidade de qualquer processo de Software no qual utiliza arquitetura de microsserviços, requer-se uma estratégia bem estruturada e desenvolvida por parte das pessoas responsáveis pelo teste de Software. (TRINDADE, 2021), defende que uma estratégia de testes de Software precisa ser definida desde o início do desenvolvimento do Software, sendo abordada inicialmente na fase de análise de requisitos. Quando um processo é estruturado e segue uma estratégia bem definida, o mesmo tende a ajudar a equipe de testes a executar um trabalho mais assertivo e sem comprometer o nível da qualidade. No capítulo anterior, buscou-se abordar sobre a importância da antecipação dos testes em um conceito de tempo e custo. Ou seja, quanto mais cedo um erro é encontrado e corrigido, menor é o seu custo (SPIRLANDELI, 2019).

Mesmo seguindo um bom processo de estruturação de testes, faz-se necessário entender e mapear quais são os desafios enfrentados pelo mau funcionamento de uma API REST, ou seja, o que ocasiona o mau comportamento das funcionalidades de um processo de integração entre componentes de um sistema que utiliza o padrão REST (ARCURI, 2019).

3.1.1 Contextualização do projeto

O projeto estudado se trata de uma aplicação de Hub Integrador de MARKETPLACES (canais de venda). Um Hub Integrador de Marketplace é um centralizador de operações no qual permite gerenciar em um único canal todos os pedidos, mensagens de SAC, controle e gerenciamento de estoque e anúncios de produtos em diferentes MARKETPLACES (MKT, 2022).

O Software permite que novos canais de venda se integrem de forma ágil e fácil em seus serviços. Porém durante a integração desses novos canais de venda alguns testes de validações de diferentes envios de produtos, precisam ser gerenciados a fim de garantir que todos os fluxos possíveis de testes sejam

executados e aplicados, para que o produto a ser enviado ao MARKETPLACE, esteja 100% aderente as validações necessárias.

Com isso, muitas vezes, a equipe de testes leva em torno de 8 horas de trabalho para realizar essas validações específicas de envio de produto para estes canais de venda, considerando que atualmente existem cerca de 45 tipos de testes de produtos a serem validados. A necessidade de integrar o MARKETPLACE, testar os processos básicos para a sua liberação de forma ágil, muitas vezes acaba excedendo o tempo previsto e as datas de entregas combinadas, pelo fato de o processo chegar para a equipe de testes sem as validações iniciais do processo.

Com isso, identifica-se a necessidade de melhorar esse tempo fazendo uso de um processo de automação. O sistema de Hub Integrador é desenvolvido utilizando uma arquitetura de microsserviços, na qual atualmente trabalha com serviços REST através de sua API. O que possibilita a automação dos processos de cadastros necessários via API.

3.2 Técnica de teste proposta

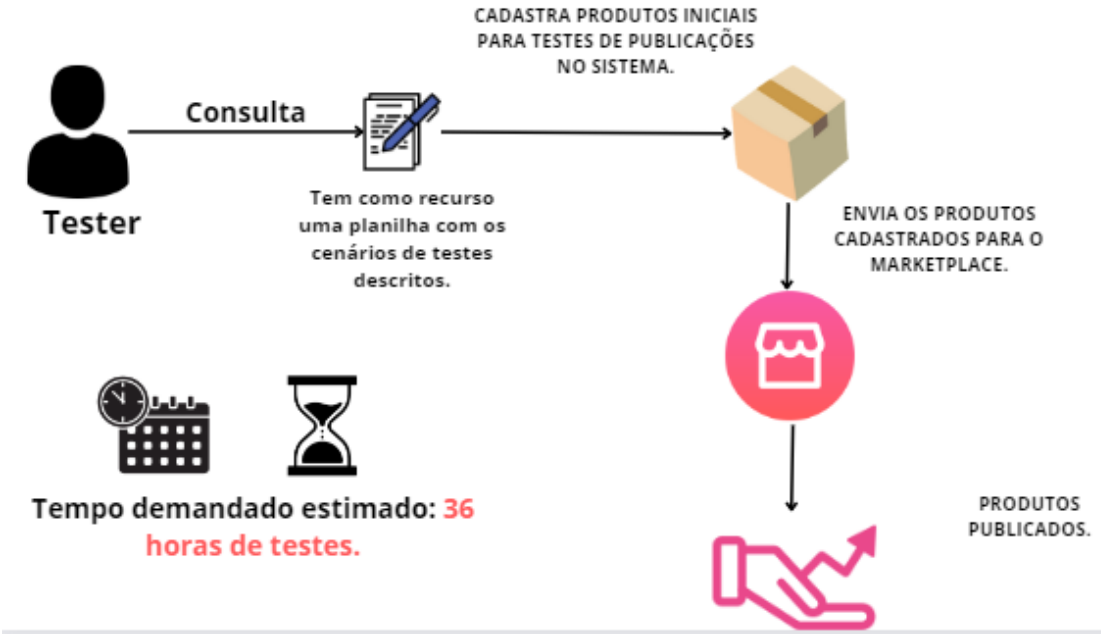
Um dos principais desafios enfrentados durante o teste automatizado de API, é a validação do sequenciamento de chamadas que a API contém e a comparação entre suas respostas, que podem ser imprevisíveis (BANIAȘ et al., 2021). Executar vários testes em paralelo, manusear, apresentar resultados de tempo de execução satisfatórios, também é um desafio no qual precisa ser gerenciado e estruturado (TRINDADE, 2021).

Em nossa primeira abordagem, foi realizado um estudo em uma API REST de um sistema de HUB Integrador de MARKETPLACES, cujo a sua API interna realiza integração com diversas plataformas e MARKETPLACES mundiais ex: Mercado Livre, Amazon, NETSHOES, entre outros.

Identificou-se que um dos processos mais morosos do teste é a publicação de produtos para os canais de venda, visto que para publicar um determinado produto e garantir que todo fluxo de comunicação esteja devidamente funcional entre as integrações, é preciso validar diversos cenários, não apenas um simples cadastro, mas sim a alteração deste, a consulta e também seus atributos específicos de envio.

A figura 7, representa o cenário atual do processo de testes dessas novas integrações. Observa-se que o “Tester” possui como recurso uma planilha que contém todos os cenários de testes que devem ser realizados na validação de uma nova integração. Esses, incluem os cadastros básicos de produtos, que serão cadastrados para serem enviados aos canais de vendas. Com isso, após o cadastro concluído o testador tem a função de publicar item a item, ou enviar para o MARKETPLACE as transmissões em lote de produtos. Porém, todo esse processo atualmente é feito de forma manual, estimando um tempo de testes manuais em 36 horas. Considerando o cenário onde existe a necessidade de entregas mais ágeis em um curto espaço de tempo, esse processo de teste acaba consumindo cerca de 60% de todo esforço na entrega da atividade.

Figura 7 - Representação do processo de testes atual.



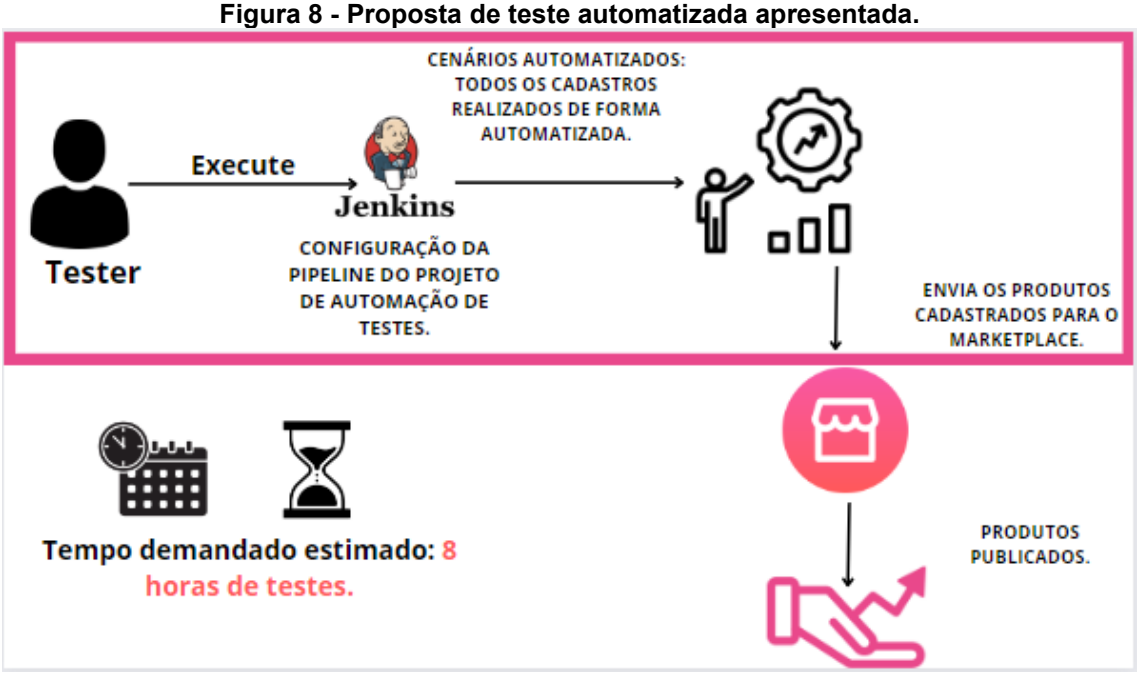
Fonte: Autoria própria (2023).

Após a abordagem do problema enfrentado pela equipe de testadores, propomos uma estratégia de testes automatizados para o contexto a ser validado, considerando a otimização do tempo e custo. Como resultado da abordagem realizada inicialmente foram mapeados diversos fluxos de cadastros de produtos via API REST, estes produtos são cadastrados no Software, para fins de publicação em canais de venda, e assim são gerenciados pela aplicação.

A figura 8, representa a abordagem de teste automatizado proposta nessa dissertação. Ao invés do Tester gerenciar seus testes via planilha, agora contará com uma configuração de integração contínua, que estará configurada com o projeto de automação de testes desenvolvido. Um dos processos da integração contínua do projeto é a criação de uma pipeline, que consiste em uma série de etapas realizadas a fim de disponibilizar uma nova versão de um software. Ao executar a pipeline do projeto no ambiente de testes utilizando a ferramenta Jenkins, aplicam-se a realização de todos os cadastros de produtos necessários a fim de publicá-los nos canais de venda configurados para os testes de forma automatizada, tudo isso sem a necessidade de testes exaustivos. O único processo no qual não será automatizado neste trabalho, é a geração de pedidos para testes de controle de estoque dos produtos. Para gerar pedidos é necessário que tenha-se o acesso total da aplicação do MARKETPLACE. E para a execução efetiva dos testes válida-se apenas cenários que se tornam onerosos de serem executados de forma manual.

Com isso apresenta-se a proposta de redução da execução dos testes que demandaria 36 horas estimadas, passando a demandar uma estimativa de 8 horas.

Figura 8 - Proposta de teste automatizada apresentada.



Fonte: Autoria própria (2023).

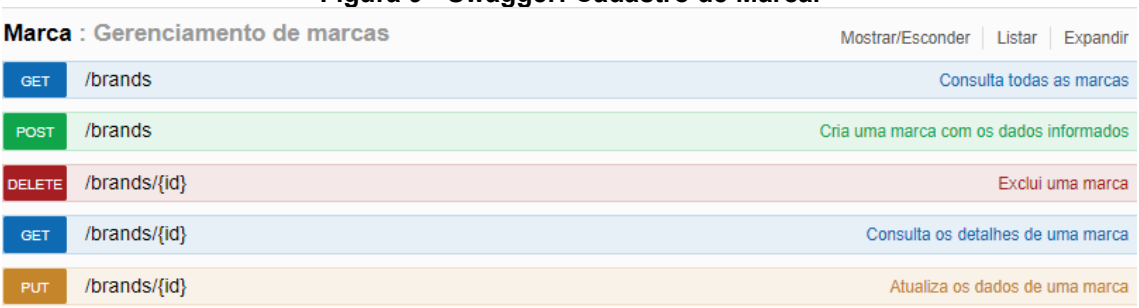
A API REST na qual está sendo utilizada para validação de cadastro, possui como critérios de aceitação para a composição de um produto, o cadastro de uma marca e categoria. O cadastro de uma marca é uma rota diferente do cadastro de

categoria e consequentemente do produto. Sendo assim, apresentam-se duas rotas a serem validadas.

Na figura 9, é apresentado um exemplo de documentação do cadastro de uma marca que é composto por:

- Consulta de marca (GET);
- Cadastro de marca (POST);
- Remoção de marca (DELETE);
- Consulta de marca por código (GET/ID);
- Alteração de marca (PUT/ID).

Figura 9 - Swagger: Cadastro de Marca.



Fonte: Anymarket (2022)

Para os testes automatizados da API REST propostos, mapeia-se alguns cenários de cadastro e para cada um terá sua devida documentação (Swagger), especificado. Nesse projeto utiliza-se os cadastros de (Marca - brands) como representado na figura 9, cadastro de categoria, cadastro de variações (tamanho do produto, cor, voltagem), cadastro do produto e publicação deste produto para o MARKETPLACE.

Após o processo de análise e prototipagem proposto, foi realizada a escolha das ferramentas. Todo fluxo do teste realizado é um processo de teste de integração entre os componentes e este teste que precisa ser realizado de forma E2E (end-to-end). A técnica de testes automatizada pode ser empregada para otimizar o esforço de custo e tempo no qual é utilizado para realização desse fluxo (BERNARDO, 2011).

Utilizou-se uma abordagem referente aos estudos realizados por outros pesquisadores, mapeando os prós e contras da técnica escolhida que estão descritos no início deste capítulo.

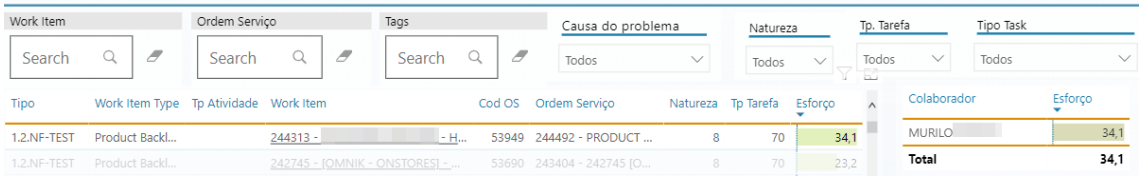
3.3 Solução de testes automatizadas proposta

Inicialmente a abordagem realizada, foi a identificação da necessidade de redução do tempo de testes demandado para a entrega da validação dos novos canais de venda. A abordagem proposta utilizou algumas métricas de testes de Software que podem ser utilizadas para obter informações relacionadas a qualidade da entrega do produto para o cliente, considerando os fatores de tempo e custo do Software, (ALMEIDA et al., 2018). O objetivo dessa abordagem é a identificação do tempo gasto pela equipe de testes para a liberação do processo a ser entregue para o cliente.

Uma outra métrica utilizada nesta pesquisa, foi baseada em tempo de entrega. O tempo de entrega é definido por Almeida, et. al. (2018) como a habilidade de entregar uma atividade, sem exceder o prazo combinado, podendo contribuir com o gerenciamento e o escopo do projeto.

A figura 10, representa uma atividade de apontamento de horas demandadas para a execução dos testes de um canal de venda por completo, antes da automação ser proposta. Observa-se que existe um valor estimado em 34,1 horas de testes executados. O apontamento representado abaixo, foi coletado com um analista de testes responsável por essa liberação.

Figura 10 - Representação do indicador de tempo da execução do teste: Power BI.



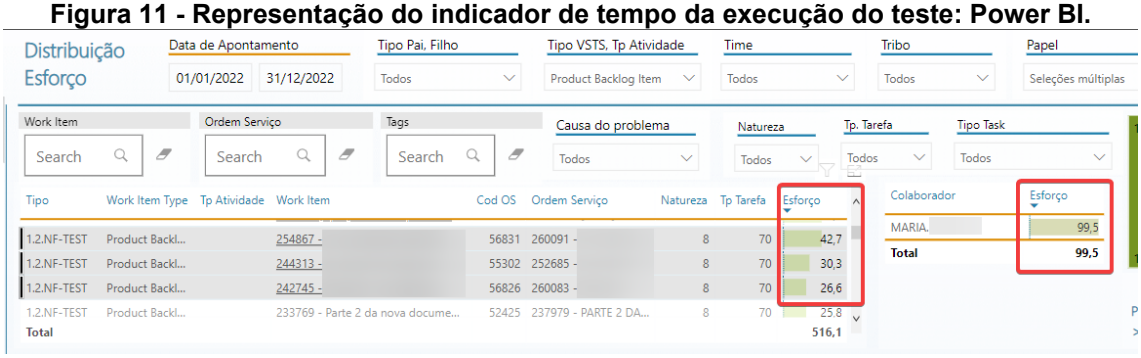
Fonte: Autoria própria (2023).

Realizar a estimativa do tempo na execução de um teste, é importante para estabelecer prazos. No entanto, nem sempre as estimativas de um testador podem ser comparadas com a de outro, pois a técnica utilizada pode ser comum entre as partes, mas a concepção, objetividade e nível de experiência com o teste podem ser diferentes.

A figura 11, representa um outro comparativo do esforço empregado por outra analista de testes, ela executou exatamente o mesmo processo do analista representado na figura 10. Comparando outras 3 novas atividades de testes de validação de novos canais de venda. Observa-se que o primeiro teste resultou em 42

horas e 7 minutos, o segundo teste em 30 horas e 3 minutos e por fim o terceiro teste resultou em 26 horas e 6 minutos, totalizando um esforço de 99 horas e 5 minutos da testadora. Existe uma oscilação de tempo de testes da mesma pessoa, isso se dá aos fatores: percepção, conhecimento e experiência empregados durante o teste.

Figura 11 - Representação do indicador de tempo da execução do teste: Power BI.



Fonte: Autoria própria (2023).

Medir o processo de testes é importante dentro do desenvolvimento de Software, pois através disso é possível mensurar o tempo que será gasto em cada atividade.

Após a identificação do tempo gasto, propõe-se uma análise em quais os processos de testes que mais demandam tempo para sua execução, em seguida identificam-se os cadastros dos mais variados tipos de produtos no sistema, atualmente cada canal de venda tem seu nome e suas particularidades, então todos os possíveis cenários de testes precisam ser validados a fim de encontrar o maior número possível de bugs, antes mesmo que estes cheguem para o cliente final.

Por fim, todos os cenários de testes nos quais se tornam onerosos de serem executados, foram devidamente mapeados, descritos e inseridos dentro de um contexto para o desenvolvimento do processo de testes automatizados.

3.4 Mapeamento das ferramentas e frameworks escolhidos

O projeto de automação foi desenvolvido na linguagem de programação Javascript, com o uso do Node.js e também com o apoio dos frameworks mocha e chai para a escrita e a execução dos cenários de testes. As tecnologias escolhidas, foram definidas através de experimentos realizados para avaliar qual eram as ferramentas e frameworks que mais se adequam a necessidade do projeto a ser

desenvolvido. As tecnologias utilizadas para o desenvolvimento deste projeto de automação, são apresentadas nesta seção.

3.4.1 Node.js

O Node.js de acordo com sua definição, é um conjunto de bibliotecas responsáveis pelo tempo de execução de uma aplicação que foi desenvolvida utilizando JavaScript, ou seja, toda interpretação do código é realizada pelo Node.js fora do ambiente de navegação Web.

Para sua utilização, faz-se necessário a instalação da aplicação na máquina na qual será utilizada. O Node.js pode ser instalado tanto em versões Windows quanto Linux. Em nosso trabalho utilizamos a versão do sistema operacional Windows.

Utiliza-se o Node.js neste trabalho, para que o JavaScript possa ser interpretado, e que algumas das funcionalidades nas quais serão desenvolvidas, utilizam- se das bibliotecas fornecidas pelo Node.js.

3.4.2 GitLab

Todo desenvolvimento de código-fonte de um determinado projeto quando iniciado, deve ser armazenado em um repositório que irá fornecer recursos necessários para manter o código hospedado e seguro. O GitLab, nada mais é do que uma plataforma de hospedagem de código-fonte na qual permite que profissionais de desenvolvimento de software se hospedem e contribuam em projetos tanto privados quanto abertos. Com a utilização do GitLab, é possível trabalhar com projetos que utilizem ferramentas de DevOps e também proporciona de forma nativa, ferramentas de integração e entrega contínua (CI/CD) (“The One DevOps Platform | GitLab”, 2022)

O projeto de automação de testes desenvolvido, utilizou como ferramenta de hospedagem de código o GitLab, pois se tratando de segurança e código armazenado de forma privada, somente integrantes liberados no projeto, poderão contribuir com o desenvolvimento e visualizar o código-fonte bem como as demais configurações existentes.

3.4.3 Jenkins

Além do GitLab que fornece ferramentas para processos de CI/CD (CI – Continuous Integration, CD – Continuous Delivery), o Jenkins disponibiliza esses recursos e também é utilizado para construir e testar o software de forma contínua e automatizada. A automação de teste realizada através do Jenkins, possibilita que as equipes entreguem soluções mais rápidas e com um custo menor.

Utilizou-se o Jenkins no projeto de testes automatizados desenvolvido neste trabalho, para que os testes desenvolvidos possam ser executados em um build (processo de construção de um artefato) de integração contínua que direciona o teste empregado a um ambiente de testes configurado.

3.4.4 JavaScript

Linguagem de programação leve, baseada e interpretada em objetos e funções, também é conhecido como uma linguagem de script voltada para páginas WEB, que também pode ser utilizada em diversos outros ambientes que não sejam navegadores, por exemplo: Node.js

Utiliza-se neste projeto o JavaScript por questões das ferramentas e linguagens de programação, que já são utilizadas na construção do Software a ser testado. Como a aplicação na qual será utilizada para testes, se trata de uma aplicação Web. A escolha pela utilização do JavaScript, foi aderida por ser uma linguagem já própria para desenvolvimento Web, e também por ser uma linguagem leve.

3.4.5 Mocha

O Mocha, é uma estrutura de testes do JavaScript, que possui diversos recursos que são executados com a utilização do Node.js e também em navegadores, possibilitando que os testes sejam executados em série e a geração de relatórios precisos durante o mapeamento de exceções que não são capturadas para cenários de testes corretos

Em nosso projeto utilizamos o framework mocha, por este permitir que as suítes de testes sejam executadas de forma rápida e fácil na nossa aplicação JavaScript. A figura 15, apresenta o processo de instalação do framework mocha. Observa-se que o comando executado é npm install -g mocha. O comando npm é

uma ferramenta do Node.js que permite o gerenciamento de pacotes no qual são possíveis instalar e desinstalar dependências na aplicação que está sendo desenvolvido (“npm - a JavaScript package manager”, 2021). Quando o comando -g é informado, quer dizer que essa dependência, pacote, ou framework informado, está sendo instalada como global no projeto.

3.4.6 Chai

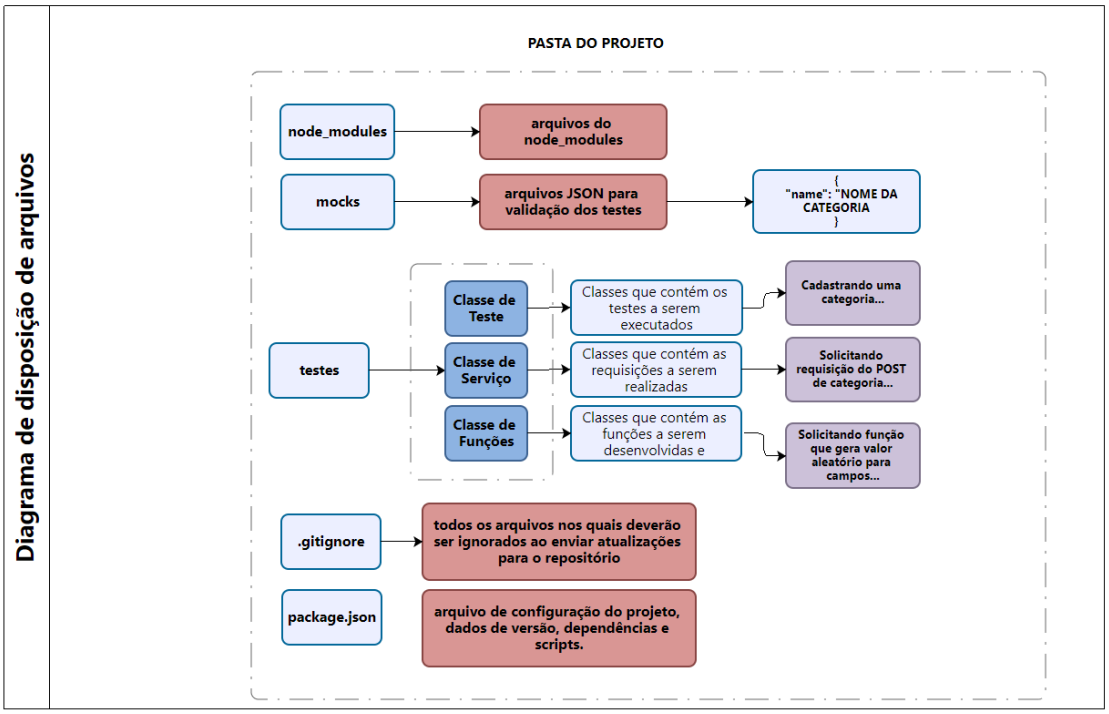
O Chai é descrito como uma biblioteca baseada em BDD/TDD que é executada em Node.js, ela também pode ser realizada em navegadores Web que utilizam aplicações JavaScript. Ainda é definido como uma biblioteca de asserções, que torna o teste fácil de ser compreendido, fornecendo diversas asserções que podem ser executadas no código de testes proposto.

Utiliza-se a biblioteca chai neste projeto de automação de testes, pois o mesmo fornece as asserções no formado BDD/TDD para o node que pode ser utilizado juntamente com o framework de testes Mocha que é integrado com o JavaScript.

3.5 Implementação e desenvolvimento do projeto e automação proposto

Para a implementação do projeto, uma vez definida a estratégia de testes e as ferramentas propostas instaladas, foi realizado o processo de diagramação da estrutura de pastas para a construção do projeto de automação, que está representado na figura 12.

Figura 12 - Padronização das pastas do projeto de automação da API REST.



Fonte: Autoria própria (2023).

O objetivo da estrutura de disposição de arquivos representado na figura 19, é explicar quais pastas e quais arquivos serão utilizados neste projeto, no entanto essa construção precisa seguir um processo passo a passo:

- A pasta node_modules irá representar o conjunto de bibliotecas responsáveis pela execução da aplicação de teste automatizado, que foi desenvolvida utilizando o JavaScript;
- Mocks irá representar a pasta contendo nossos arquivos de testes no formato JSON;
- Testes: apresenta as funções que serão utilizadas no projeto proposto, o arquivo de serviço que irá representar as nossas requisições HTTP, e por fim o arquivo descritivo com os cenários de testes propostos;
- O arquivo de .gitIgnore serão todos aqueles que intencionalmente serão ignorados pelo Git;
- Por fim, apresenta-se o arquivo package.json que nele irá conter os dados do projeto tais como: versão, dependências, scripts entre outros dados de configuração do projeto.

A figura 13, representa a estrutura já criada para o desenvolvimento do teste automatizado, na representação 1 desta figura, é apresentada a hierarquia de pastas

e arquivos. Já na representação 2, o arquivo de configuração package.json é apresentado contendo as configurações necessárias para o desenvolvimento deste projeto. Na representação 2 da figura 13 especifica-se os seguintes dados:

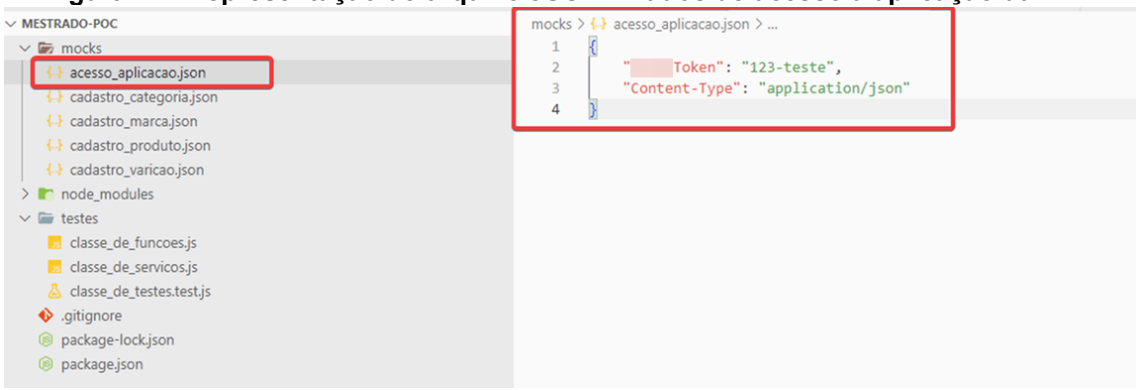
- Name: representa o nome do projeto, pacote ou aplicação a ser criado;
- Version: indica qual é a versão atual do projeto, visto que a cada modificação este campo deve ser alterado;
- Description: corresponde a descrição da aplicação, projeto ou pacote;
- Main: irá definir o processo de inicialização da aplicação, dado que se houver uma requisição em nosso projeto;
- Scripts: é um conjunto de scripts que serão executados em Node.js;
- devDependencies: irá definir a lista de pacotes e bibliotecas do Node.js, que estão instaladas como dependência de desenvolvimento.

Figura 13 - Representação da estrutura do projeto.

Fonte: Autoria própria (2023).

Se tratando de um projeto desenvolvido em um sistema WEB, este já se encontra em utilização pelos clientes, é necessário realizar a configuração do arquivo, sendo esse próprio para autenticação. Na figura 14, observa-se que foi criado um arquivo com extensão. json, com os dados de autenticação da aplicação a ser testada. Conforme as leis de proteção de dados, não pode deixar público os dados da empresa, então foi informado apenas os dados de testes local. O campo token, representa o token da aplicação a ser acessada. O content-type, indica o cabeçalho utilizado para indicar o tipo de arquivo a ser utilizado do recurso Content-Type - HTTP | MDN.

Figura 14 - Representação do arquivo JSON - Dados de acesso a aplicação da API.



Fonte: Autoria própria (2023).

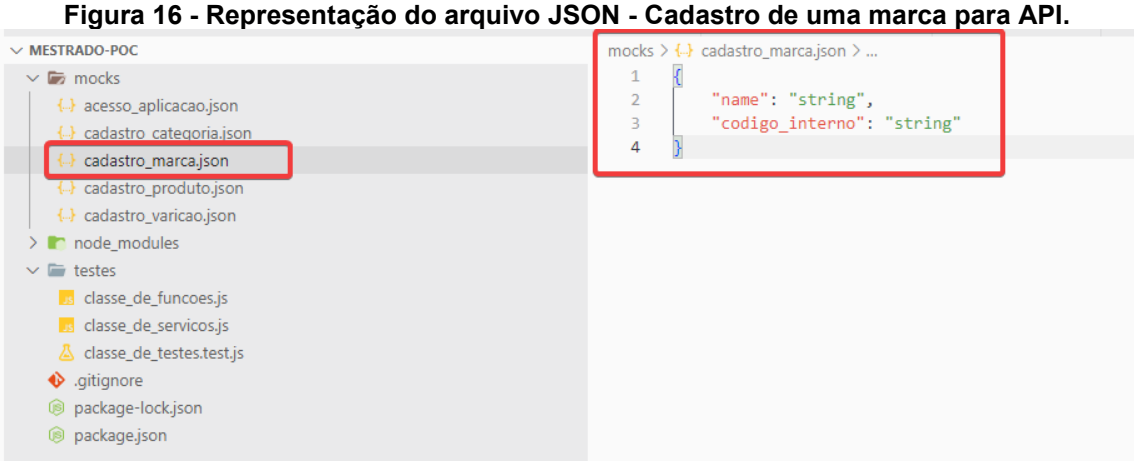
Para que um produto seja cadastrado no sistema, faz-se necessário o cadastro de uma categoria. A figura 15 representa o arquivo .json com os campos de cadastro.

Figura 15 - Representação do arquivo JSON - Cadastro de uma categoria para API.

Fonte: Autoria própria (2023).

Após o cadastro de uma categoria, uma marca deve ser cadastrada, pois o produto é inserido no sistema, após a informação de marca. Sendo assim, a figura 16, representa o arquivo .json, no qual registra o cadastro de uma marca via API.

Figura 16 - Representação do arquivo JSON - Cadastro de uma marca para API.



Fonte: Autoria própria (2023).

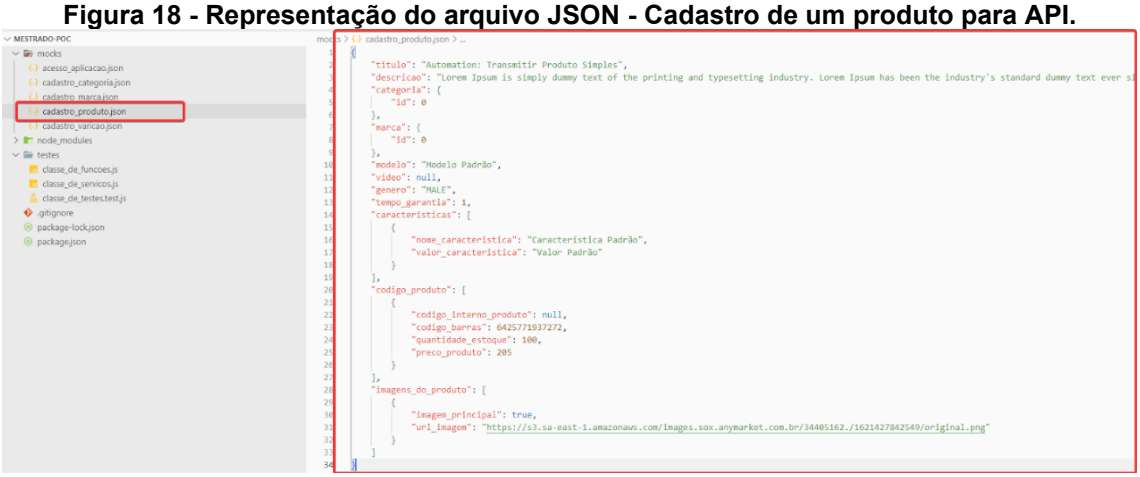
No sistema testado, existe a possibilidade de realizar cadastros de produtos com a informação de variação, por exemplo: cor: amarelo, verde, azul e também pode corresponder a outros tipos de variações como: tamanho: p, m, g, entre outras variações, essa representação pode ser observada na figura 17.

Figura 17 - Representação do arquivo JSON - Cadastro de uma variação para API.

Fonte: Autoria própria (2023).

Ao realizar o cadastro de todos os componentes que contemplam um produto, pode-se observar então este cadastro sendo apresentado na figura 18.

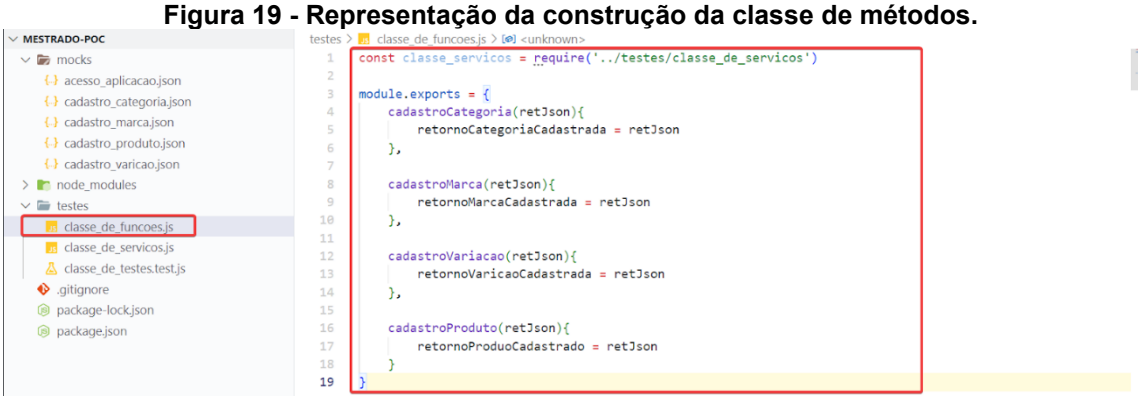
Figura 18 - Representação do arquivo JSON - Cadastro de um produto para API.



Fonte: Autoria própria (2023).

Após a construção dos arquivos de mocks (testes .json), é necessário construir a classe que estarão contidas as funções. As funções são desenvolvidas utilizando JavaScript e também por bibliotecas do Node.Js. A figura 19, representa os métodos que são utilizados para receber o retorno dos cadastros realizados. Ainda na representação da figura 16, é apresentada a classe de serviço sendo instanciada. A aplicação realiza a instância de outras classes através da biblioteca require, que está contida no Node.Js. Porém, para que as classes possam ser exportadas, seus métodos precisam estar dentro do “module.exports”, que permite que todas as funções, métodos ou validações que estão contidas nele, sejam instanciadas por outras classes.

Figura 19 - Representação da construção da classe de métodos.

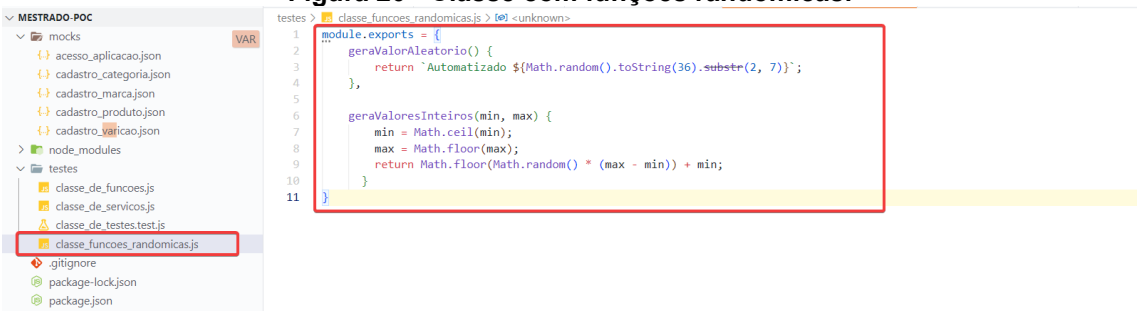


Fonte: Autoria própria (2023).

Neste projeto, alguns campos não podem ter seus valores repetidos, por isso foram inseridos alguns métodos que geram valores aleatórios, para que possa ser possível executar o teste automatizado quantas vezes forem necessárias, sem que

haja problemas de integridade de dados duplicados. Com isso, a figura 20 representa os métodos que geram valores aleatórios tanto para formatos de texto (string), quanto formatos numéricos.

Figura 20 - Classe com funções randômicas.



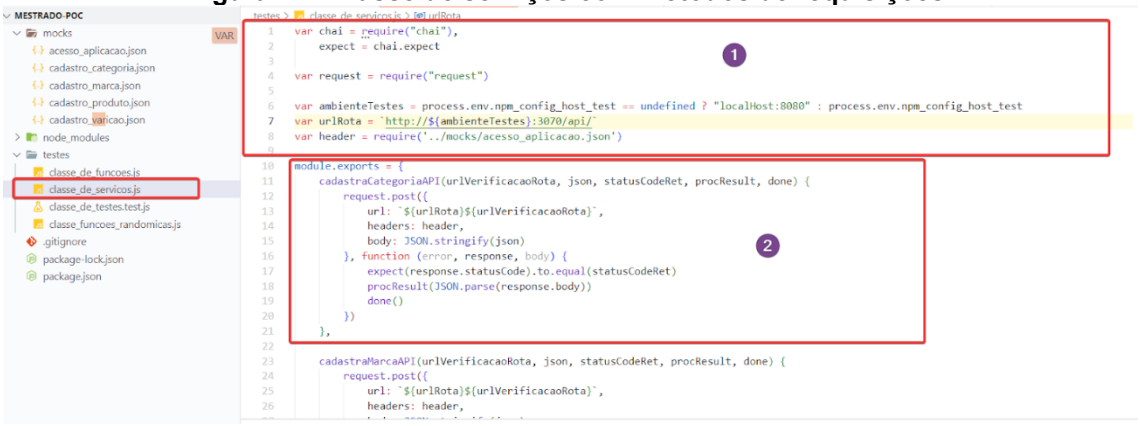
Fonte: Autoria própria (2023).

A classe de serviço é que contém as requisições: GET, POST, DELETE, UPDADE e assim por diante. A figura 21, representa as configurações da classe de serviço, visto que o primeiro bloco representado como “1”, mapeia o ambiente de testes que direciona a chamada como “ambienteTestes”. Na sequência é inserida a urlRota que irá direcionar para qual endpoint será testado. Depois, será instanciado o nosso header, que contém o arquivo de configuração e acesso ao ambiente. Por fim, o método que será exportado dentro do modules.exports, sendo o método de requisição da chamada a API mapeado como “2”. Esse está construído da seguinte forma:

- request.post (cadastro do registro);
- url: mapeamento da URL a ser testada (endPoint);
- header: configuração e acesso ao ambiente;

Ainda na figura 21, são apresentados os parâmetros informados no método sendo eles: urlVerificaRota, json, statusCodeRet, procResult, done. Esses parâmetros são mapeados para que sejam utilizados na chamada do teste.

Figura 21 - Classe de serviços com métodos de requisições.



Fonte: Autoria própria (2023).

Após todos os arquivos necessários para o desenvolvimento dos testes propostos neste trabalho serem devidamente construídos, os testes em questão serão mapeados. A figura 22, representa a construção da classe de testes, inicialmente construindo os testes de cadastros de componentes para os produtos. Nessa classe apresenta-se alguns pontos a serem observados:

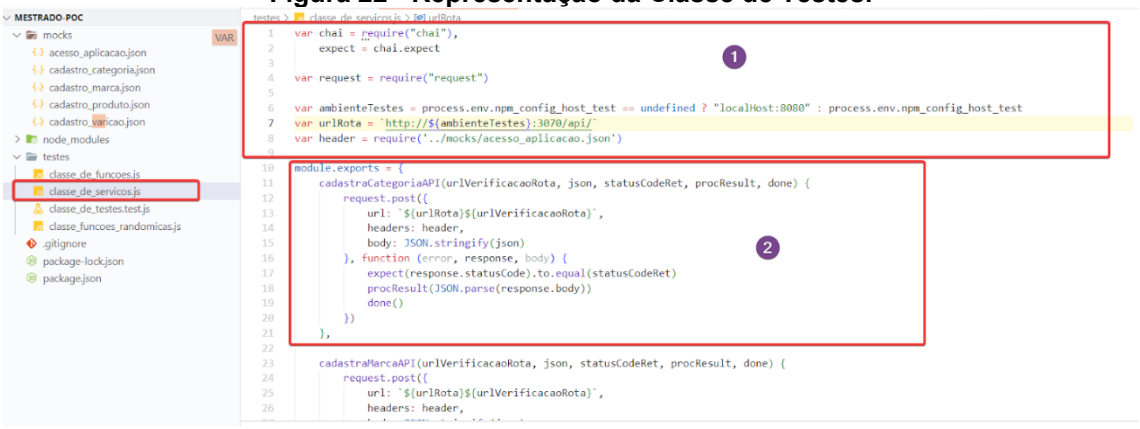
- As devidas bibliotecas utilizadas estão sendo carregadas ao iniciar o arquivo;
- [1] As instâncias das classes que são necessárias para serem mapeadas e utilizadas na construção do teste automatizado;
- [2] Utiliza-se o describe para descrever o caso de teste no qual está sendo desenvolvido. O objetivo do describe é fornecer uma descrição do caso de testes (DEODATO, 2018).
- [3] O context apresentado, que irá apresentar o contexto do teste em execução. Permite-se que os contextos de testes possam ser descritos inúmeras vezes dentro de um describe.
- [4] Utiliza-se o it que é a biblioteca do chai responsável por descrever o nosso teste no formato BDD, que será testado de fato no endpoint.

Ainda de acordo com a imagem 22, dentro do nosso caso de testes IT, apresenta-se a classe de serviço sendo chamada e já instanciando o método de cadastro (de acordo com cada it a ser representado), após a chamada ao método, são apresentados os parâmetros sendo informados, que são representados da seguinte maneira:

- Brands: representa a url de verificação da rota, sendo a marca;
- Cadastro_marca: representa o arquivo .json a ser testado;

- 200: representa o statusCode retornado na aplicação de acordo com a documentação proposta;
- Classe_de_funcoes.cadastroMarca: representa a função de retorno do cadastro de marcas sendo chamada para receber o retorno;
- Done(): indica que o teste em questão terminou;

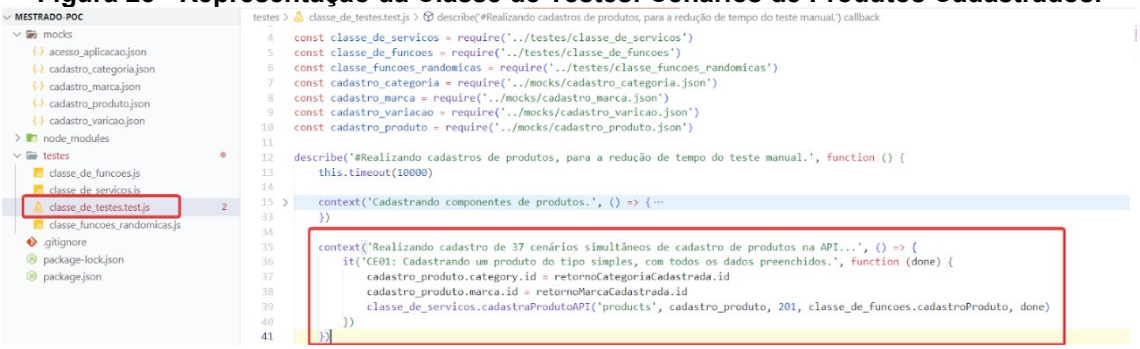
Figura 22 - Representação da Classe de Testes.



Fonte: Autoria própria (2023).

Como objetivo deste trabalho, apresenta-se o último processo da construção da classe, os cenários de cadastros de produtos que serão inseridos no ambiente de testes para otimizar o tempo dos testadores com relação ao cadastro de novos produtos com dados específicos para cada novo canal de venda a ser testado. A figura 23, representa um dos cenários de testes de inserção de produtos, seguindo o princípio do teste de componentes. A diferença do cadastro de produto para os demais, é que utiliza-se apenas um arquivo para todos os cenários, basta apenas alterar o campo do arquivo .json dentro do teste mencionado no IT.

Figura 23 - Representação da Classe de Testes: Cenários de Produtos Cadastrados.



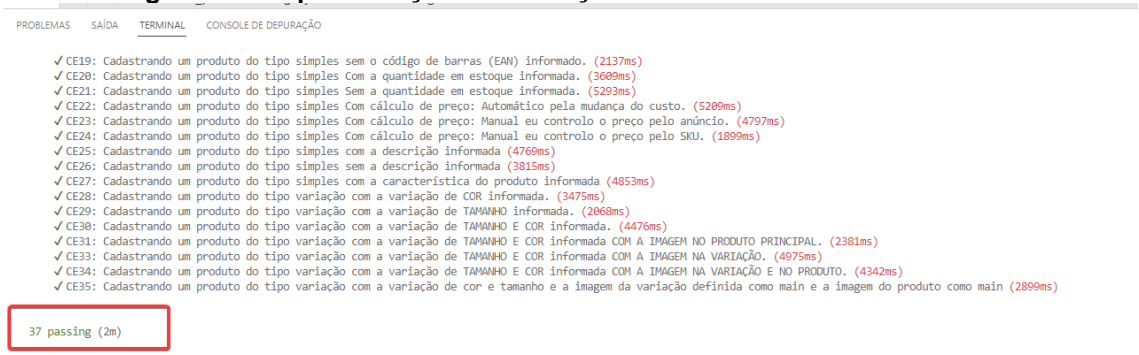
Fonte: Autoria própria (2023).

Após o desenvolvimento dos 37 cenários de testes propostos neste trabalho, é realizado o teste local no próprio terminal contido no VSCODE utilizado para desenvolvimento do projeto. Para a execução do arquivo de testes, basta acessar a pasta do .test.js e executar o seguinte comando:

“mocha .\classe_de_testes.test.js”

O comando acima irá realizar a chamada do teste construído, como representado na figura 24. Ainda nessa mesma figura, observa-se a execução finalizada, resultando em 37 testes passando em exatos 2 minutos. O tempo de todo teste de API será executado conforme o tempo de resposta da API para o processo.

Figura 24 - Representação da execução do teste no terminal VSCODE.



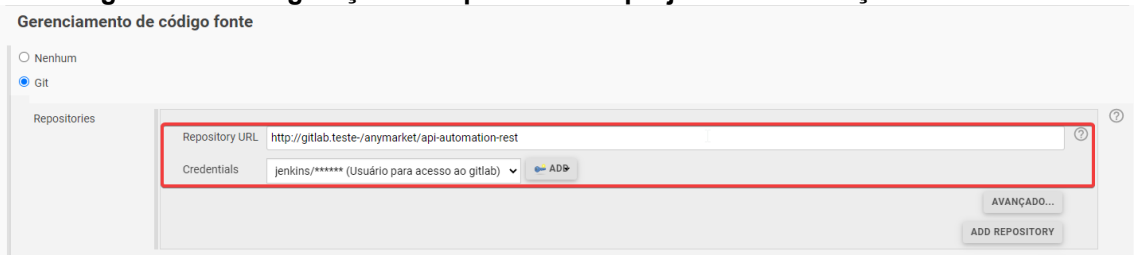
Fonte: Autoria própria (2023).

3.6 Integração contínua

A integração contínua é um processo inserido no desenvolvimento incremental das entregas frequentes a clientes. Seu objetivo é integrar rotineiramente o Software e suas dependências para validar que nenhuma modificação tenha danificado ou prejudicado o sistema, podendo ser alterações em código fonte, configurações, dependências, testes ou entre outros fatores que tendem a serem utilizados (BERNARDO, 2011).

Neste projeto, utiliza-se o Jenkins como ferramenta de integração contínua, este facilita a execução automática dos casos de testes, permitindo a configuração nos ambientes utilizados para a integração contínua. A configuração da ferramenta utilizada, está representada na figura 25, onde realiza-se o mapeamento do projeto no qual está contido nosso código fonte.

Figura 25 - Configuração do repositório do projeto de automação no Jenkins.

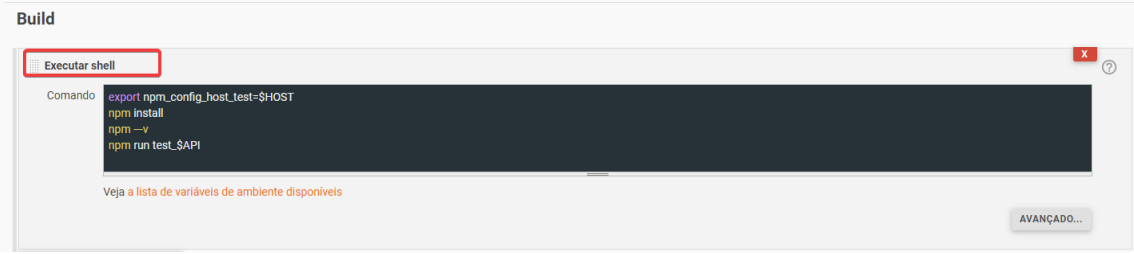


Fonte: Autoria própria (2023).

Após a configuração do repositório onde está contido o código, é realizada a configuração da execução do projeto. Como representado na figura 26, utiliza-se os seguintes comandos:

- Exports = npm_config_host_test=$HOST: sua funcionalidade é pegar qual é o direcionamento do ambiente de testes a ser executado o projeto, visto que a empresa estudada, possui diversos ambientes para testes, então propõe-se uma configuração dinâmica para que o teste seja independente do ambiente;
- Npm install: para instalar as bibliotecas e dependências do Node.Js;
- Npm –v: apresenta a versão do Node.Js instalada;
- Npm run_test$API: executa o teste a partir do comando que foi inserido no package.json.

Figura 26 - Configuração da execução do teste automatizado no Jenkins.

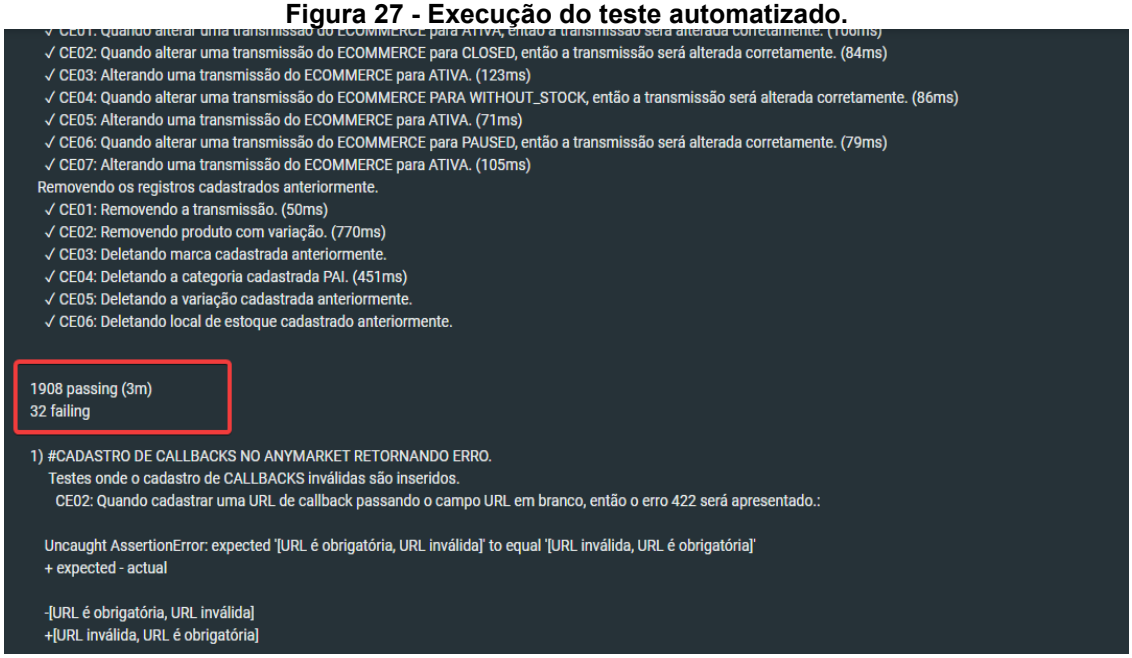


Fonte: Autoria própria (2023).

Por fim, a execução do teste pode ser feita através de um build devidamente configurado no jenkins. Assim, ao informar a branch na qual está sendo desenvolvida, pode-se ter diretamente através do jenkins, um relatório de execução do teste, contendo os descritivos, contextos e cenários, como representado na figura 27. Ainda nesta figura, observa-se que foram realizados 1908 testes executados em 3 minutos, enquanto outros 32 testes falharam. Aqueles que falharam são exemplos de

mudanças de código fonte realizadas na API, cujo ao executar o teste E2E a validação via automação apresenta o resultado positivo da sua funcionalidade e desenvolvimento.

Figura 27 - Execução do teste automatizado.



Fonte: Autoria própria (2023).

3.7 Metodologia de desenvolvimento

A metodologia de trabalho utilizada nesta pesquisa, se trata de um método experimental com base nos resultados obtidos ao aplicar os testes automatizados em um contexto de testes de API.

Inicialmente foi realizado o processo de apresentar a revisão bibliográfica mapeando a utilização do teste automatizado em contextos de API REST, sua aderência e os resultados que este pode oferecer em um contexto que aborda o fator de custo e tempo, como principais fatores de resultado esperado para a qualidade e entrega ágil de um Software. Em sequência a este processo foi realizada uma pesquisa amplificando quais as ferramentas, frameworks e técnicas de testes estão sendo utilizadas em um contexto de testes automatizados para APIs.

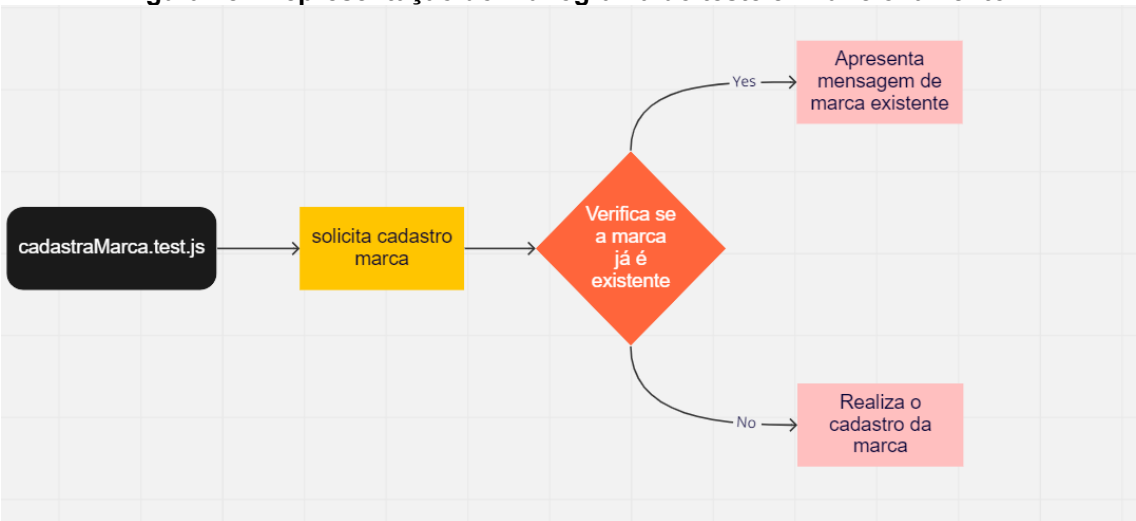
Após a revisão bibliográfica dos temas propostos neste trabalho, é proposto uma POC (prova de conceito) com uma abordagem ainda não apresentada pela literatura, que é a utilização dos frameworks Mocha e chai, para desenvolvimento de um projeto de automação de testes para uma API REST.

As ferramentas e frameworks escolhidos para o desenvolvimento deste trabalho, permitem que o tempo demandado para a realização de testes seja gradualmente reduzido e também apresenta como um resultado não planejado, a quantidade de erros que foram mapeados através desta implementação. As ferramentas e frameworks escolhidos são:

- Node.js: Como interpretador de JavaScript e gerenciador de pacotes e bibliotecas bem como frameworks a serem utilizados no projeto;
- Visual Studio Code: Como ferramenta de escrita do nosso código-fonte a ser desenvolvido no projeto de teste automatizado;
- GitLab: Como repositório de armazenamento do projeto de automação;
- Jenkins: Como ferramenta de testes automatizada de integração contínua;
- JavaScript: Como linguagem de desenvolvimento padrão para o projeto de automação a ser desenvolvido;
- Mocha: Como framework de execução de suítes de testes automatizados;
- Chai: Como biblioteca de asserções aos testes automatizados a serem desenvolvidos no projeto.

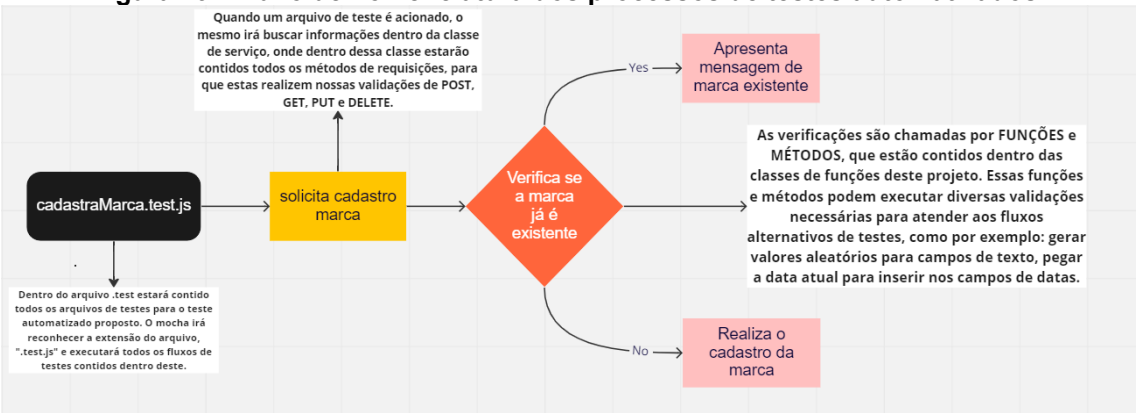
A figura 28 representa a funcionalidade de um teste final para um cadastro de marca, apresentando apenas o processo de um cadastro de uma marca sendo executado o seguinte fluxo:

Figura 28 - Representação do Fluxograma do teste em funcionamento.



Fonte: Autoria própria (2023).

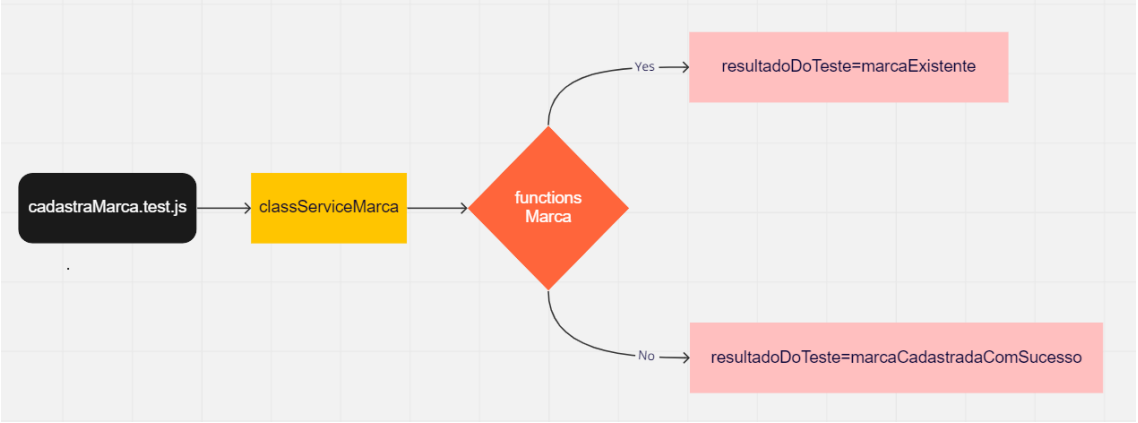
Figura 29 - Fluxo de nomenclatura dos processos de testes automatizados.



Fonte: Autoria própria (2023).

A figura 30 representa o escopo final de uma funcionalidade completa por trás das suas nomenclaturas, pode-se notar que os fluxos agora estão identificados com suas devidas funcionalidades, que foram anteriormente mencionadas na arquitetura e construção deste projeto:

Figura 30 - Fluxo descritivo do nome das classes de testes do projeto automatizado.



Fonte: Autoria própria (2023).

4 RESULTADOS E DISCUSSÕES

Essa seção descreve os resultados e discussões da aderência pelo teste automatizado em um contexto de API REST com o uso dos frameworks mocha e chai.

4.1 Limitações da Pesquisa

Diversas são as ferramentas e frameworks de testes automatizados que se encontram disponibilizadas na literatura, porém existe uma particularidade e uma escassez de artigos, dissertações e teses que se enquadram em testes automatizados de API REST. Portanto foi realizada uma pesquisa nas seguintes bases:

- CAPES;
- IEEE;
- Scielo.

Poucos são os trabalhos que tem relação com a técnica de testes proposta, devido aos frameworks escolhidos, que ainda não tiveram trabalhos publicados a seu respeito até a data do desenvolvimento deste presente trabalho.

4.2 Resultados

Abordagens, técnicas, metodologias, ferramentas e frameworks de apoio ao teste automatizado são propostos para considerar sua utilização em um contexto de testes de API REST. Para a avaliação do experimento realizado, utilizou-se como parâmetros as seguintes perguntas:

1) A solução proposta do projeto de testes automatizado para o contexto de uma API REST, resultou na redução do tempo de testes?

2) A cobertura dos testes automatizados propostos, agregou valor na qualidade do Software?

3) O projeto de automação de testes de API REST, proporciona a redução do custo?

Foi realizada uma pesquisa com 4 (quatro) analistas de testes do projeto proposto, por meio de um formulário eletrônico. Entre as quatro avaliações obteve-se as seguintes respostas:

Figura 31 - Resultado da pesquisa - participante 01.

PARTICIPANTE 01 - RESPOSTA	PERGUNTA (PARÂMETROS)
Sim, pois executa uma quantidade de casos de teste que levaria muito tempo ao serem executados manualmente.	**A solução proposta do projeto de testes automatizado para o contexto de uma API REST, resultou na redução do tempo de testes?**
Sim, pois da um feedback muito mais rápido sobre os impactos das mudanças no Software ao ser executado continuamente.	**A cobertura dos testes automatizados propostos, agregou valor na qualidade do Software?**
Sim, pois ao reduzir o tempo necessário para a execução dos testes, consequentemente também diminuo o custo da execução dos mesmos.	**O projeto de automação de testes de API REST, proporciona a redução do custo do teste?**

Fonte: Autoria própria (2023).

Figura 32 - Resultado da pesquisa - participante 02.

PARTICIPANTE 02 - RESPOSTA	PERGUNTA (PARÂMETROS)
Sim, reduziu drasticamente o tempo dos testes no contexto de API, que normalmente são feitos manualmente.	**A solução proposta do projeto de testes automatizado para o contexto de uma API REST, resultou na redução do tempo de testes?**
Sim, reduziu de forma significante, devido a todos membros do time possuírem acesso aos testes e executar os mesmos de forma prática e fácil.	**A cobertura dos testes automatizados propostos, agregou valor na qualidade do Software?**
Sim, apresentou redução de custos operacionais de forma expressiva, visto que são executados dezenas de testes por minutos de forma automática não exigindo de forma direta o conhecimento das regras de negócio do sistema.	**O projeto de automação de testes de API REST, proporciona a redução do custo do teste?**

Fonte: Autoria própria (2023).

Figura 33 - Resultado da pesquisa - participante 03.

PARTICIPANTE 03 - RESPOSTA	PERGUNTA (PARÂMETROS)
Sim, adquirimos agilidade e redução de teste na execução dos testes devido a realização da automação, pois há uma grande quantidade de cenários de testes cobertos para serem executados.	**A solução proposta do projeto de testes automatizado para o contexto de uma API REST, resultou na redução do tempo de testes?**
Sim, pois com a automação temos uma visão dos impactos de forma rápida comparando a execução manual.	**A cobertura dos testes automatizados propostos, agregou valor na qualidade do Software?**
Sim, com a automação de teste foram encontrados bugs durante a execução dos testes automatizados, com isso o custo do teste e resolução do bug é de custo baixo.	**O projeto de automação de testes de API REST, proporciona a redução do custo do teste?**

Fonte: Autoria própria (2023).

Figura 34 - Resultado da pesquisa - participante 04.

PARTICIPANTE 04 - RESPOSTA	PERGUNTA (PARÂMETROS)
Sim, com o projeto de testes automatizado reduzimos o tempo de testes que uma vez teriam que ser manuais e agora são executados em um curto período de tempo, reduzindo o esforço do time e cobrindo uma grande parcela de testes.	**A solução proposta do projeto de testes automatizado para o contexto de uma API REST, resultou na redução do tempo de testes?**
Sim, após a execução do projeto de automação temos o resultado de forma rápida e com isso conseguimos observar se houve alguma quebra dos casos de testes executados e já aplicar a devida correção para a garantia da qualidade do Software que está sendo entregue.	**A cobertura dos testes automatizados propostos, agregou valor na qualidade do Software?**
Sim, com uma cobertura ampla de testes automatizados é reduzido o tempo de teste do analista e consequentemente ao observar alguma inconsistência no resultado é possível tomar ações assim reduzindo o custo.	**O projeto de automação de testes de API REST, proporciona a redução do custo do teste?**

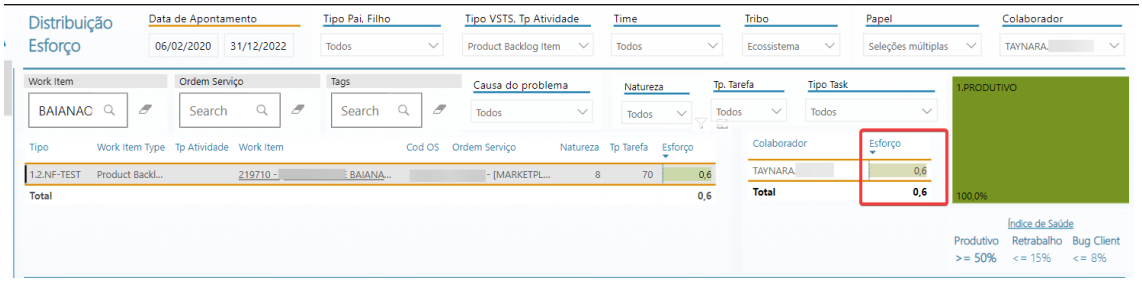
Fonte: Autoria própria (2023).

Como apresentado nas figuras 31, 32, 33 e 34 contém as respostas coletadas através do preenchimento do formulário eletrônico. Essas respostas foram preenchidas por quatro analistas que atuam diretamente com os testes nos quais foram automatizados no contexto. Com isso, foram coletadas as avaliações do processo empregado mediante as pessoas diretamente envolvidas, tendo como resultado a redução de 24 horas de testes.

4.2.1 Redução do tempo da execução de testes

Considerando que o tempo médio de testes era de 36 horas, agora com o desenvolvimento automatizado dos cenários propostos neste trabalho, apresenta-se a redução gradual de 24 horas, podendo entregar o processo de validação de uma nova integração em até seis horas de testes seguidos. Tendo em vista que erros e incidentes podem acontecer durante os testes, foram identificados através deste processo os bugs iniciais, com isso estes foram mapeados e enviados para correção, não sendo necessário refazer todo o teste de forma manual, apenas rodar o build da automação de testes proposto, para verificar se o problema persiste ou se este já foi resolvido. A figura 35 representa o cenário descrito acima.

Figura 35 - Apresentação do tempo de execução do teste automatizado após a implementação.



Fonte: Autoria própria (2023).

4.2.2 Maior cobertura de cenários de testes da API

Durante a realização deste trabalho no processo de avaliação dos frameworks propostos, foram apresentadas diferentes abordagens de processos de testes automatizados que visavam ser empregados no contexto de API REST. Contudo, nenhum dos trabalhos selecionados abordou um teste por completo utilizando cenários de sucesso e erro, validações de dados de entrada e suas saídas, consultas e inserções simultâneas entre outros processos E2E que podem ser realizados em APIs REST.

Algumas abordagens realizadas neste trabalho por outros autores, apresentaram o POSTMAN como uma ferramenta promissora para testes E2E em API REST. Mesmo se tratando de um teste de API, o POSTMAN não foi utilizado neste trabalho pelo fato da necessidade em realizar alguns testes E2E (end2End), estes precisam começar e terminar na própria classe, validando todos os cenários possíveis, e também foi realizado o processo de integrar os testes em uma ferramenta da integração contínua, que utilizou-se no projeto o GitLab e o Jenkins. Com a utilização do POSTMAN, não seria possível realizar esse tipo de integração, visto que algumas funcionalidades do POSTMAN não são gratuitas para sua utilização. Um outro processo que foi levado em consideração em não escolher o POSTMAN como ferramenta para os testes, se dá ao fato de ter que criar diversas runners de execução para cada cenário, já que via mocha e chai, foram criadas as suítes de testes de forma rápida, compreensível e de fácil manutenção, tudo em uma só classe.

Um dos frameworks que vem sendo adotados por equipe de testadores para realização de testes de FRONT END é o Cypress, que também pode ser utilizado para testes de API REST. Contudo neste trabalho o Cypress não foi aderido pelo fato da necessidade em ter que adquirir uma licença a mais para geração de relatórios dos testes que estavam sendo buscados. Considerando que o Cypress supriria a necessidade do projeto, avaliou-se a ferramenta na qual possibilitaria apenas alguns recursos de relatórios visuais do processo, e diante da quantidade de vezes que houvesse a necessidade de executar o teste por dia, não se tornaria viável por se tratar da necessidade de adquirir uma licença.

Em um outro contexto de testes de API REST, pode-se utilizar uma ferramenta para avaliação dos endpoint sendo o JMETER. Mesmo se tratando de uma ferramenta openSource, esta não foi aderida, pelo fato de ter a sua execução parecida com o

POSTMAN, já que os cenários de testes a serem executados precisam estar construídos de forma isolada das classes, e executados runners a runners, visto que o JMeter é uma ferramenta própria para testes de carga e performance, para o nosso contexto apenas seria viável para popular uma base de dados com registros minimamente parecidos.

Este trabalho buscou abordar a técnica de testes de integração e E2E para realização de testes de serviços RESTful em várias entradas e saídas, testando componentes tanto de forma isolada como integrada, podendo validar serviços sequenciais e gerenciados.

Após a validação das três ferramentas mencionadas acima, levando em consideração os prós e contras ao seu uso neste projeto, avaliou-se uma terceira maneira de automatizar o fluxo de testes proposto, sendo: utilização da junção do framework de testes mocha e da biblioteca de asserções chai, juntamente com o JavaScript como linguagem de programação e o Node.js como interpretador de JavaScript. A escolha da ferramenta foi dada através da análise de viabilidade de seu uso, trazendo alguns benefícios ao projeto:

- Sua execução de testes apresenta os devidos relatórios necessários para validação dos dados, sem custos adicionais;
- A maneira em que os testes são escritos e gerenciados é de fácil entendimento;
- O mocha e chai são openSource;
- De acordo com a necessidade do projeto a utilização do chai para a escrita dos testes abrange contextos próprios para o cenário proposto;
- Utilizando o node existe a possibilidade de instalar diversas bibliotecas para a construção de processos internos em nosso código, sem a necessidade de pagar pelo seu uso;
- Os testes podem ser executados de forma paralela, e também de forma independente;
- Os testes podem ser mapeados em diferentes contextos em uma mesma classe;
- Os testes podem ser enviados a um repositório;
- Os testes podem ser executados em pipelines com o uso de ferramentas de integração contínuas e gerenciados diretamente em seus respectivos ambientes de testes.

4.2.3 Facilidade na execuão dos testes automatizados

Com o projeto todo integrado a uma ferramenta de integração contínua e o repositório onde este se encontra armazenado, sua execução fica totalmente automatizada, pois foi construída uma pipeline que possibilita a execução de branchs específicas para cada cenário de testes automatizados do projeto, que se deseja testar. Em alguns trabalhos utilizados para este estudo, identificou-se algumas técnicas utilizadas por exemplo, no trabalho de Arcuri (2019), que utilizou-se de técnicas de geração de testes de integração para serviços RESTfull, porém a proposta é realizada para testes de forma isolada. Neste trabalho, utilizou-se da técnica de testes de integração com sua execução de forma paralela, não sequencial e não dependente. Ou seja, nenhum teste depende do outro para ser executado, cada um trabalha de forma independente, podendo assim executar diversas branchs a qualquer momento do teste sem impactar algum serviço em execução.

A ferramenta escolhida foi o Jenkins e com isso obtém-se relatórios da execução de cada teste, possibilitando ao testador ou até mesmo alguém que apenas clique no build para executar, o entendimento do teste em que está sendo realizado, pois o teste automatizado possui toda a descrição da sua execução.

4.2.4 Facilidade na escrita de novos cenários de testes no projeto de automação

O formato utilizado para o desenvolvimento dos testes automatizados, foi baseado no BDD, mas neste trabalho foi padronizado a escrita e proposto uma escrita global para o projeto. Viglianisi et al, (2020), em seu trabalho aborda a técnica de geração de casos de testes automatizados através da ferramenta RESTTESTGEN, essa técnica descreve os cenários baseados em uma documentação, porém, no trabalho desenvolvido para essa dissertação, buscou-se avaliar além da documentação da API, os principais cenários de testes que de fato fizessem sentido de serem testados com o nível de criticidade adotado e que fossem fatores para a minimização do esforço empregado nos testes. Com essa possibilidade de escrever os testes no padrão:

“Quando realizar o cadastro de uma marca e essa marca não estiver duplicada, então a marca deverá ser cadastrada corretamente.”

Com isso, apresentando a proposta de escrever no teste aquilo que realmente quer que seja executado, proporcionou aos analistas de testes e a pesquisadora deste trabalho, uma facilidade em escrever os cenários para o projeto.

5 CONCLUSÃO

Executar os testes manuais é uma atividade que se torna cara e está sujeita a muitos erros e enganos, devido a inúmeras funcionalidades que uma API pode fornecer. A técnica de testes automatizada reduz o esforço manual, pois ações que são executadas de forma repetitivas tendem a ser substituídas por scripts de testes automatizados que executam as funcionalidades da API. Além de ser uma prática efetiva no quesito qualidade de Software, a automação de testes apresenta segurança em momentos de alterações no código, manutenção, refatoração e até mesmo em momentos de inserções de novas funcionalidades (BANIAȘ et al., 2021). Com o desenvolvimento de um script de testes automatizados é possível executar rotinas de testes inúmeras vezes sem o esforço manual a ser empregado. O fator de custo do Software tende a ser reduzido, pois com a automação de testes a possibilidade de encontrar erros durante o processo é maior do que quando executado o teste de forma manual. O tempo de execução do teste também é reduzido levando em consideração que em um cenário de testes de API, existem inúmeras rotas e endpoints a serem testados, quando existe um processo automatizado, foi possível garantir que a maior parte do fluxo da regra de negócio da API, foi devidamente testada perante o teste desenvolvido.

No contexto de testes automatizados para APIs REST, diversas são as ferramentas e frameworks que estão disponíveis para serem utilizadas. No entanto, escolher a principal ferramenta para ser validado o teste de uma API, requer o conhecimento do contexto do teste a ser empregado. Realizou-se um processo de desenvolvimento de uma POC, abordando algumas ferramentas e frameworks de testes automatizados para serem empregados na API REST a ser testada na empresa X.

Foi realizada a avaliação de um framework de testes sendo o mocha e a junção com a biblioteca chai, nossa aplicação foi desenvolvida utilizando a linguagem de programação JavaScript e as bibliotecas do NodeJs. A seleção das tecnologias utilizadas foi avaliada após um experimento realizado com uma equipe de 4 testadores dentro de um contexto de API REST. Ao permitir a execução dos testes automatizados na redução de tempo demandado em um processo de avaliação de uma determinada atividade de testes, obteve-se um resultado escalável do tempo demandado bem como também foi possível mapear uma vasta quantidade de erros que foram

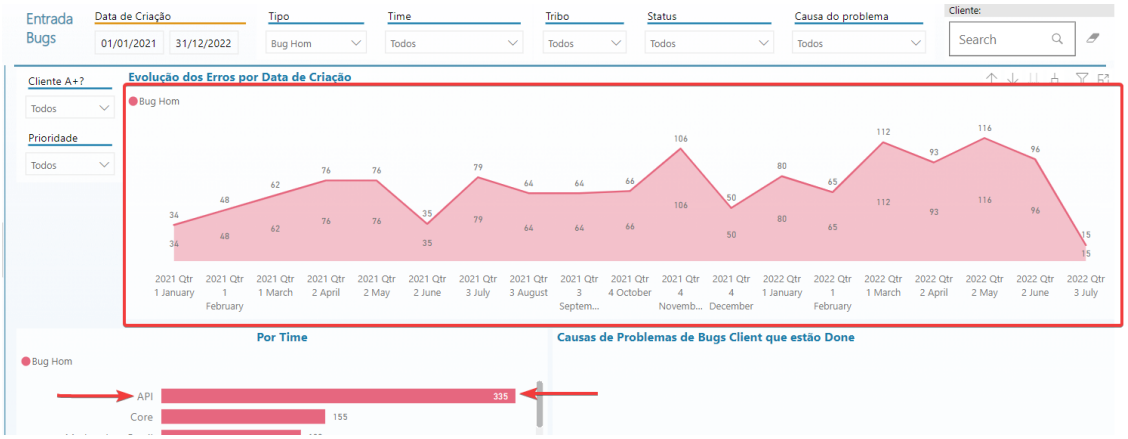
detectados perante o desenvolvimento e execução dos testes automatizados propostos.

Essa pesquisa abrangeu a importância da aderência do teste automatizado em um contexto de API REST. Foi proposto a realização dos testes funcionais validando cenários de sucesso e erro, baseando-se na documentação da API REST a ser testada. Porém neste contexto, foi realizada a abordagem dos cenários de erro cujo a API retorna um status de validação, baseando-se na funcionalidade da aplicação.

O objetivo dessa pesquisa era apenas fornecer uma ferramenta capaz de gerenciar os testes, executá-los e reduzir o esforço manual no momento de testar uma nova integração com Marketplaces. Contudo durante o desenvolvimento dos scripts e dos resultados que esses testes estavam proporcionando dentro da equipe de testadores, foi decidido avaliar a possibilidade de automatizar novas rotas e novas funcionalidades da API REST, abordando diversos outros cenários, o que resultaram em uma vasta quantidade de falhas encontradas na API antes mesmo de serem mapeadas por um cliente em operação.

Na figura 36, observa-se o mapeamento resultante do filtro dos bugs encontrados dentro do período da iniciação deste projeto de automação até a data corrente dessa entrega. Pode-se observar a quantidade de bugs na API encontradas é de 335 bugs, esse filtro foi possível graças a uma filtragem por tags em cada atividade, ou seja, foram padronizadas todas as tarefas de erros abertas com uma tag chamadas de “automação”, com isso foi possível ao final do processo filtrar a quantidade de bugs encontrados no período.

Figura 36 - Quantidade de bugs encontrados deste o período de início deste trabalho até a sua conclusão.



Fonte: Autoria própria (2023).

Por fim, para avaliar-se qual é a ferramenta, framework ou técnica de testes pode ser aplicada no contexto de testes automatizados de API REST, é necessário primeiramente conhecer o produto que será testado, entender da regra de negócio, separar os processos que mais se tornam onerosos e repetitivos a serem testados, realizar um mapeamento sistemático de todos os *endpoints* e empregar o teste automatizado naqueles em que o nível de risco se torna maior, caso uma modificação seja realizada e este venha ser afetado.

Como proposta de trabalhos futuros, deseja-se realizar um estudo sobre a possibilidade de automatizar fluxos de RPA em robôs que são utilizados para executar pequenas ações rotineiras. Ações essas que possibilitam a captura de informações de pequenas solicitações de usuários em plataformas digitais, com o objetivo de mapear os principais fluxos de ações e realizar testes automatizados assertivos com a finalidade de resultar em uma melhor experiência do usuário.

REFERÊNCIAS

ALAM, M. S. et al. A REST and HTTP-based Service Architecture for Industrial Facilities. *In* IEEE Conference on Industrial Cyberphysical Systems (ICPS), 2020. **Anais.[...]**, Tampere, Finland. p. 398 – 401.

ALMEIDA, W.; FURTADO, F.; MONTEIRO, L. Pesquisa em Métricas para melhoria, medição e avaliação de software. Encontro Unificado de Computação do Piaui, 2020 **Anais.[...]**, Piaui. p. 129 – 134.

ARAUJO, F. S. **Avaliação de geradores automáticos de dados de teste com ênfase no teste de mutação**. Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de São Carlos, São Carlos, 2021. Disponível em:http://www. https://repositorio.ufscar.br/handle/ufscar/14107. Acesso em: 15 set. 2022.

ARCURI, A. RESTful API Automated Test Case Generation with EvoMaster. **ACM Transactions on Software Engineering and Methodology,** v28, n 03, p 1-37, jan. 2019. DOI https://dl.acm.org/doi/10.1145/3293455 Acesso em 20 de maio de 2022.

ATLIDAKIS, V.; GODEFROID, P.; POLISHCHUK, M. RESTler: stateful REST API fuzzing Proceedings of the 41st International Conference on Software Engineering. Quebec, 2019 **Anais.[...]**, Quebec. p. 748-758

BANIAȘ, O. et al. Automated Specification-Based Testing of REST APIs. **Sensors,.,** Romania v21, n 03, p 1-37, Ago. 2021. DOI https://doi.org/10.3390/s21165375 Acesso em 30 de agosto de 2022.

BERNARDO, P. C. **Padrões de testes automatizados**. Dissertação (Mestrado em Ciências da Computação) – Universidade de São Paulo, São Paulo, 2011. Disponível em: https://www.teses.usp.br/teses/disponiveis/45/45134/tde-02042012-120707/publico/TestesAutomatizados_PauloCheque_Dissertacao.pdf. Acesso em: 15 set. 2022.

CORRADINI, D. et al. Empirical Comparison of Black-box Test Case Generation Tools for RESTful APIsarXiv, IEEE International Workshop on Source Code Analysis and Manipulation Luxembourg, 2021 **Anais.[...]**, Luxembourg. p. 226-236

CURTI, F. M.; DALLILO, F. D. Automação de testes utilizando a ferramenta cucumber **RECIMA21 - Revista Científica Multidisciplinar,.,** Rio de Janeiro v03, n 02, p. e321133–e321133, fev. 2022. DOI https://doi.org/10.47820/recima21.v3i2.1133 Acesso em 30 de agosto de 2022.

ERONEN, V. **Automatiska tester med Postman i IBM Cloud**. Tese (Doutorado em Tecnologia da Informação) – Universidade Svenska, Svenska, 2019. Disponível em: http://www.theseus.fi/handle/10024/172489. Acesso 16 junho. 2022.

FONSECA, M. A. N. **Desenvolvimento de testes automatizados para backend**. Dissertação (Mestrado em Engenharia Informática) – Faculdade de Ciências e Tecnologia, Lisboa, 2021. Disponível em: http://hdl.handle.net/10362/120492. Acesso 06 agosto. 2022.

LARANJEIRO, N.; AGNELO, J.; BERNARDINO, J. A Black Box Tool for Robustness Testing of REST Services. **IEEE Access**, Canada. v. 9, p. 24738–24754, Fev 2021.

LY, M. **Creating API Test Automation of a Service for Company X** Tese (Doutorado em Programme in Business Information Technology) – Laurea University of Applied Sciences, Finlândia, 2018. Disponível em: https://www.theseus.fi/bitstream/handle/10024/150706. Acesso em: 02 dez. 2022.

MARQUES, A. I. A**. Desenvolvimento de API para aplicação cloud**. Dissertação (Mestrado em Engenharia Informática) – Instituto Politécnico de Leiria, Leiria, 2018. Disponível em:http://www. https://iconline.ipleiria.pt/handle/10400.8/3263. Acesso em: 20 set. 2022.

MESHRAM, S. U. Evolution of Modern Web Services – REST API with its Architecture and Design. **International Journal of Research in Engineering, Science and Management**, v. 4, n. 7, p. 83–86, 9 jul. 2021.DOI: http://dx.doi.org/10.7124/bc.000027 Disponível em: https://journal.ijresm.com/index.php/ijresm/article/view/970 Acesso em: 20 maio 2022

PEREIRA, H. F. M. **Automatização de testes para plataformas Oracle - Xstore.** Dissertação (Mestrado em Engenharia Electrotécnica e de Computadores) – Faculdade de Engenharia do Porto, Porto, 201. Disponível em: https://hdl.handle.net/10216/121218. Acesso em: 25 out. 2022.

PINHEIRO, P. V. P. **Teste baseado em modelos para serviços RESTful** Tese (Mestrado em Ciências da Computação) – Universidade Federal de Goiás, Goiânia, 2018. Disponível em: http://repositorio.bc.ufg.br/tede/handle/tede/8333. Acesso em: 16 mai. 2028.

PRESSMAN, R. S. **Engenharia de Software - Uma Abordagem Profissional** - 7 ed. São Paulo PdfCoffe, 2011.

RODRIGUES, A. C. B. **Um arcabouço conceitual para diagnóstico organizacional a respeito da utilização da automação de testes de software**. Tese (Doutorado em Informática) – Universidade Federal do Amazonas, Manaus, 2018. Disponível em: https://tede.ufam.edu.br/handle/tede/6501. Acesso 27 fev. 2018.

SAHIN, O.; AKAY, B. A Discrete Dynamic Artificial Bee Colony with Hyper-Scout for RESTful web service API test suite generation. **Applied soft computing**, v. 104, p. 107246-, 2021. DOI: https://doi.org/10.1016/j.asoc.2021.107246 Acesso em 18 de outubro de 2022.

SEGURA, S. et al. Metamorphic Testing of RESTful Web APIs. **IEEE Transactions on Software Engineering**, v. 44, n. 11, p. 1083–1099, nov. 2018. DOI:10.1109/TSE.2017.2764464 Acesso em 23 de janeiro de 2021.

SPIRLANDELI, C. A utilização de testes automatizados no desenvolvimento de software. **Educação, tecnologia e gestão**. **,** v2, n 02, p.1-24, dez. 2019. DOI https://revistaedufatec.fatecfranca.edu.br/wp-content/uploads/2020/03/edufatec-n02v2a01.pdf Acesso em 14 de maio de 2021.

2.ed. 24, 2019.

SUZANTI, I. O. et al: REST API Implementation on Android Based Monitoring Application.*In* Journal of Physics: Conference Series, 2020, **Anais.[...]**.Surabaya: 2020. p. 022088-022098.

VIGLIANISI, E.; DALLAGO, M.; CECCATO, M. RESTTESTGEN: Automated Black-Box Testing of RESTful APIs. *In* IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), 2020, **Anais.[...]**.Porto: 2020. p. 142-152.