

CENTRO UNIVERSITÁRIO UNIDADE DE ENSINO SUPERIOR DOM BOSCO
CURSO SISTEMAS DE INFORMAÇÃO

MURIEL MAGNO TELES DE CARVALHO

**APLICAÇÃO DE TESTE DE REGRESSÃO NA QUALIDADE DE SOFTWARE
USANDO POSTMAN E NEWMAN**

São Luís

2022

MURIEL MAGNO TELES DE CARVALHO

**APLICAÇÃO DE TESTE DE REGRESSÃO NA QUALIDADE DE SOFTWARE
USANDO POSTMAN E NEWMAN**

Monografia apresentada ao Curso de Sistema de Informação do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Giovanni Lucca França da Silva.

São Luís

2022

Dados Internacionais de Catalogação na Publicação (CIP)

Centro Universitário – UNDB / Biblioteca

Carvalho, Muriel Magno Teles de

A aplicação de teste de regressão na qualidade de software usando Postman e Newman. / Muriel Magno Teles de Carvalho. — São Luís, 2022.

55 f.

Orientador: Prof. Dr. Giovanni Lucca França da Silva.

Monografia (Graduação em Sistemas de Informação) - Curso de Sistemas de Informação - Centro Universitário Unidade de Ensino Superior Dom Bosco - UNDB, 2022.

1. Teste de software. 2. Regressão. 3. Qualidade – desenvolvimento. 4. Aplicação. I. Título.

CDU 004.415.53

APLICAÇÃO DE TESTE DE REGRESSÃO NA QUALIDADE DE SOFTWARE USANDO POSTMAN E NEWMAN

Monografia apresentada ao Curso de Sistema de Informação do Centro Universitário Unidade de Ensino Superior Dom Bosco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovada em: ____/____/____.

BANCA EXAMINADORA:

Prof. Dr. Giovanni Lucca França da Silva (Orientador)

Doutor em Engenharia Elétrica

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Prof. Me. Allisson Jorge Silva Almeida

Mestre em Engenharia Elétrica

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Prof. Francisco de Assis Silva Moura Junior

Especialista em Business Intelligence

Centro Universitário Unidade de Ensino Superior Dom Bosco (UNDB)

Dedico esse trabalho à
construção evolutiva da
humanidade.

AGRADECIMENTOS

Agradeço a todos os professores do Centro Universitário Dom Bosco, em especial meu orientador. Agradeço também a minha família pelo apoio e confiança, e a minha bela namorada por me ajudar em todos os momentos difíceis.

“Nossas virtudes e nossos defeitos são inseparáveis, assim como a força e a matéria. Quando se separam, o homem deixa de existir.”
(TESLA, Nikola).

RESUMO

Cada dia que passa, a tecnologia avança, esse avanço traz necessidades de desenvolver cada dia mais aplicações que sejam confiáveis e com qualidade garantida, onde o usuário não encontre erros ao utilizar o serviço ou aplicação de uma determinada empresa. A melhoria dos sistemas é relativa à atualização da tecnologia e as necessidades evolutivas dos processos digitais. Este trabalho tem como objetivo demonstrar a utilização do teste de regressão para auxiliar o time de desenvolvimento a garantir a qualidade desde o início do desenvolvimento ou atualização de um processo já existente. Evitando o retrabalho, o aumento de custo das correções em produção e melhorar as entregas do time.

Neste estudo foi concluído que a utilização do teste de regressão para validar fluxos antigos após uma alteração no código é importante para garantir a qualidade para o usuário final, evitando com que o erro vá para o ambiente de produção sem o conhecimento da equipe de desenvolvimento.

Palavras-chave: Teste de Software. Regressão. Aplicação. Qualidade. Desenvolvimento.

ABSTRACT

Every day, technology advances, this advancement brings the need to develop more and more applications that are reliable and with guaranteed quality, where the user does not find errors when using the service or application of a particular company. The improvement of systems is related to the updating of technology and the evolving needs of digital processes. This work aims to demonstrate the use of regression testing to help the development team to ensure quality from the beginning of the development or update of an existing process. Avoiding rework, increasing the cost of corrections in production and improving team deliveries.

In this study, it was concluded that the use of regression testing to validate old flows after a change in the code is important to guarantee the quality for the end user, preventing the error from going to the production environment without the knowledge of the development team.

Keywords: Software Test. Regression. Application. Quality. Development.

LISTA DE FIGURAS

Figura 1 – Pirâmide de testes.....	16
Figura 2 – Coleção Postman	27
Figura 3 – Ambientes no Postman.	28
Figura 4 – Requisição POST	29
Figura 5 – Pré-requisição	30
Figura 6 – Script de teste.....	31
Figura 7 – Criação de Coleção	32
Figura 8 – Criando nome da Coleção.....	33
Figura 9 - Validação do nome.....	33
Figura 10 – Criação do Ambiente	34
Figura 11 – Alteração do nome do Ambiente	34
Figura 12 – Adicionando link no Ambiente	35
Figura 13 – Preenchimento de Body	36
Figura 14 – Retorno da requisição 4Devs	37
Figura 15 – Body da criação de ônibus pela requisição	38
Figura 16 – Script pré-requisições para preenchimento das variáveis na criação do ônibus.....	38
Figura 17 – Script de teste para validar a criação do ônibus	39
Figura 18 – Resposta da requisição	40
Figura 19 - Resultados do teste.....	41
Figura 20 – New Runner Tab	42
Figura 21 – Runner	42
Figura 22 – Executando coleção com o Runner.....	43
Figura 23 – Executando os testes com Runner.....	44
Figura 24 – Exportando coleção para executar no Newman.....	45
Figura 25 – Executando exportação	46
Figura 26 – Salvando coleção em um diretório	47
Figura 27 - Exportando ambiente	48
Figura 28 – Exportando ambiente 2	48
Figura 29 - Seleccionando diretório.....	49
Figura 30 – Relatório do teste Newman	50

LISTA DE GRÁFICOS

Gráfico 1 – Custo por erros em cada cenário.....	17
---	----

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BNT	Associação Brasileira de Normas Técnicas
BUCBP	Biblioteca Universitária Consuelo Bello Pereira
CD	Continuous Delivery
CI	Continuous Integration
CRUD	Create, Read, Update, Delete
E2E	End-to-end
HTTP	Hypertext Transfer Protocol
TI	Tecnologia da Informação

SUMÁRIO

1 INTRODUÇÃO	14
1.1 Objetivos	15
2 FUNDAMENTAÇÃO TEORICA.....	16
2.1 Teste de software	16
2.1.1 Tipos de teste	17
2.1.2 Níveis de testes	18
2.1.2.1 Teste de unidade	18
2.1.2.2 Teste de integração	19
2.1.2.3 Teste de regressão	19
2.1.2.4 Teste de usabilidade	20
2.1.2.5 Teste de desempenho	20
2.1.2.5.1 Teste de carga.....	20
2.1.2.5.2 Teste de estresse	21
2.1.2.5.3 Teste de volume	21
2.1.2.5.4 Teste de Escalabilidade	21
2.1.2.6 Teste exploratório	22
2.1.3 Testes manuais	22
2.1.4 Testes automatizados	23
2.2 Conclusão da fundamentação	24
3 DESENVOLVIMENTO.....	25
3.1 Contexto do capítulo.....	25
3.2 Utilização do teste de regressão	25
3.2.1 Planejamento	25
3.2.2 Teste de regressão em API.....	26
3.2.3 Utilizando o Postman para teste de regressão	26
3.2.3.1 Coleções	27
3.2.3.2 Ambientes	28
3.2.3.3 Requisições.....	28
3.2.3.4 Pré-requisições.....	29
3.2.3.5 Teste	30
3.2.4 Utilizando o Newman.....	31
3.3 Executando o teste de regressão	31
3.3.1 Criando as coleções e requisições para o teste	31

3.3.2 Executando coleções de teste no Postman.....	41
3.3.3 Executando coleções de teste no Newman.....	44
3.4 Considerações finais	51
4 RESULTADO E DISCUSSÃO	52
4.1 Teste com o Postman.....	52
4.2 Teste com o Newman.....	52
5 CONCLUSÃO FINAL.....	53
5.1 Trabalhos futuros.....	53
REFERÊNCIAS	54

1 INTRODUÇÃO

Com o avanço da humanidade, as necessidades tecnológicas foram aumentando de forma exponencial, onde a maioria dos serviços que utilizamos hoje possui disponibilidade tecnológica para a utilização desses serviços. Para manter esses serviços funcionando de forma correta, precisamos entregar aplicações, modificações e correções com qualidade para os usuários, SOMMERVILLE (2010, p. 6) diz que os negócios e a sociedade estão mudando de maneira incrivelmente rápida, à medida que as economias emergentes se desenvolvem e as novas tecnologias se tornam disponíveis. Deve ser possível alterar seu software existente e desenvolver um novo software rapidamente. Muitas técnicas tradicionais de engenharia de software consomem tempo, e a entrega de novos sistemas frequentemente é mais demorada do que o planejado. É preciso evoluir para que o tempo requerido para o software dar retorno a seus clientes seja reduzido.

Estudos mostram que é muito mais caro corrigir um erro que está em produção, do que um erro encontrado ainda na fase de desenvolvimento, podendo causar grandes perdas para a empresa, de acordo com a pesquisa realizada pela Tricentis (2017) falhas de software causaram um prejuízo de quase 1,1 trilhão de dólares, e tiveram cerca de 4 bilhões de usuários afetados. Diversos fatores influenciam a qualidade de um software, onde podemos utilizar alguns atributos para mapeá-los e entendê-los de forma assertiva na hora de desenvolver, segundo a ISO (2011) 25010 que são: Funcionalidade, confiabilidade, compatibilidade, usabilidade, eficiência, manutenibilidade, portabilidade e segurança.

Esta qualidade nem sempre é garantida no dia a dia do desenvolvimento, é necessário garantir que foi desenvolvido não irá afetar nenhuma parte que já está funcionando. O profissional de qualidade de software tem como o objetivo garantir que a qualidade necessária seja entregue ao concluir o desenvolvimento da aplicação, evitando assim que um erro afete o funcionamento da aplicação em produção.

A ISO (2011) 25010 define a qualidade em uso é o grau em que um produto ou sistema pode ser usado por usuários específicos para atender às suas necessidades para atingir objetivos específicos com eficácia, eficiência, isenção de riscos e satisfação em contextos específicos de uso.

Para evitar que um erro no desenvolvimento afete outra parte da aplicação já desenvolvida, o teste de regressão automatizado é um dos caminhos utilizados na

testagem de software. A utilização do Postman já é conhecida pela equipe de TI, sua função é fazer uma ponte entre o usuário e a API, traduzindo os métodos HTTP que a API utiliza para se comunicar com outras aplicações.

1.1 Objetivos

O objetivo desse estudo é utilizar o Postman para efetuar o teste de regressão junto com o Newman, que é uma aplicação feita para executar as coleções criadas em linha de comando, podendo ser utilizado no CI e CD. Lembrando que o Postman pode ser utilizado para outros tipos de testes, os testes citados a seguir serão explicados no capítulo 2 deste documento, são:

- Teste unitário
- Teste de carga
- Teste de estresse
- Teste de volume

Quando é efetuado alguma melhoria ou correção em uma aplicação, o desenvolvedor não garante se alguma outra parte da aplicação quebrou ou gerou um bug desconhecido, por isso utilizar o teste de regressão automatizado é uma opção viável para aplicações com grande criticidade e com mudanças rotineiras, já que os testes nos serviços da aplicação é realizado quando uma nova correção é subida para qualquer ambiente, neste estudo o Postman será utilizado para demonstrar a elaboração e a utilização do teste de regressão, com todos os níveis necessários.

2 FUNDAMENTAÇÃO TEORICA

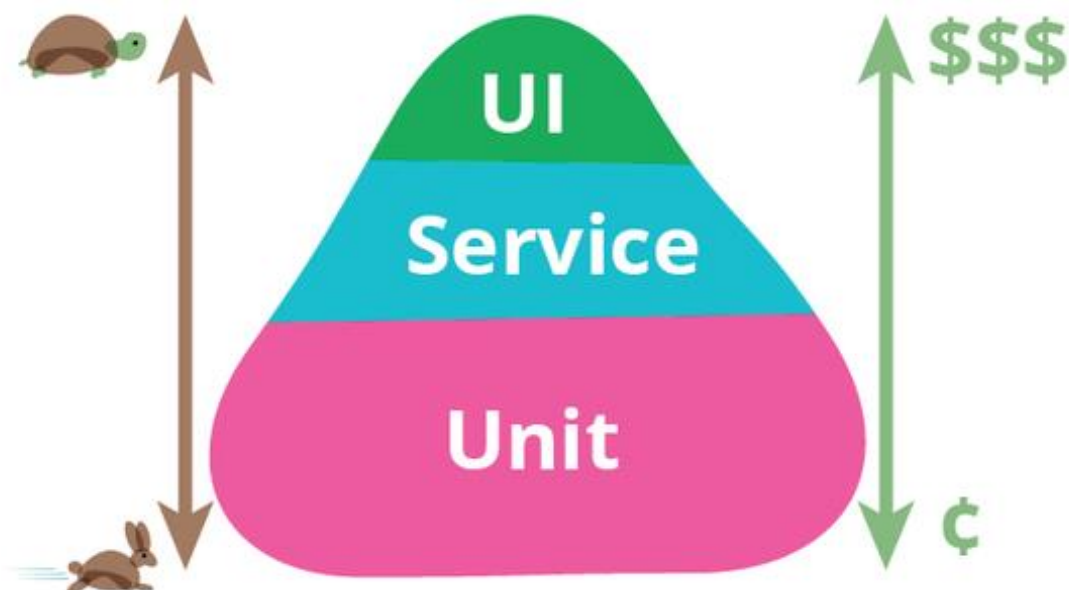
2.1 Teste de software

O teste de software faz parte do processo de validação da engenharia de software, onde o testador deve planejar qual tipo de teste e ou ferramenta que vai utilizar para avaliar a qualidade do software, diminuindo assim os riscos que a aplicação possa ter um problema e garanti que os requisitos solicitados estão funcionando de forma correta.

SOMMERVILLE (2011, p.144) o teste de software tem como o objetivo “mostrar que um programa faz o que é proposto a fazer e para descobrir os defeitos do programa antes do uso”. O conceito de teste de software pode mudar dependendo de cada autor, mas o objetivo é o mesmo, evitar que o usuário final tenha problemas na utilização do software, existem vários tipos de testes, e formas de executá-los, esse tópico tem como o objetivo entender como funciona cada um deles.

Para visualizar melhor as camadas e o esforço dos testes, é comum utilizarmos a pirâmide de teste, existem muitas representações, mas vou utilizar a do Martin Fowler, por ela podemos identificar os níveis de teste, o tempo de execução e o custo.

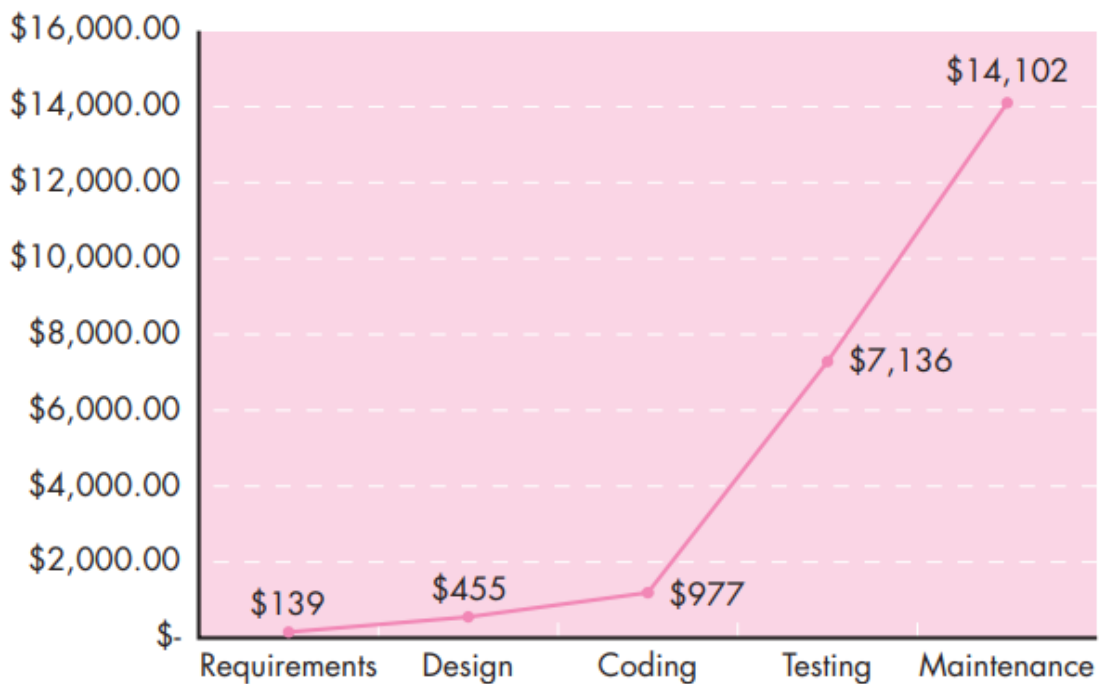
Figura 1 – Pirâmide de testes



Fonte: Próprio autor.

Quando é falado em qualidade de software e testes a serem utilizados, é idealizando uma aplicação que não possua erros em produção, diferente de uma aplicação que não passou por nenhum processo de teste antes, para exemplificar o custo de encontrar e solucionar um erro em cada ambiente, temos um gráfico baseado nos dados coletados por Boehm e Basili e com a ilustração de Cigital Inc.

Gráfico 1 – Custo por erros em cada cenário



Fonte: PRESSMAN (2010, p. 408 – 409).

2.1.1 Tipos de teste

Existem três tipos de testes, o teste de caixa branca, a caixa preta e por último o teste de caixa cinza. O teste caixa branca é uma técnica de teste usada na parte interna da aplicação, feita diretamente no código fonte, tem por objetivo garantir a qualidade na codificação feita pelo desenvolvedor, nesse tópico veremos alguns métodos para utilizar essa técnica de teste.

O teste de caixa preta é mais diferente do caixa branca, onde o testador não possui acesso ao código fonte da aplicação, fazendo assim teste funcionais ou não da aplicação, teste de aceitação, validar fluxos e demais utilizações da aplicação, podendo utilizar vários níveis de testes.

E temos nosso pouco falado teste de caixa cinza, podemos dizer que esse item o testador possui um conhecimento maior sobre estrutura de código e linguagem de programação, misturando vários níveis de testes para efetuar um bom processo nos testes, para explicar um pouco melhor essa técnica, vamos imaginar que o projeto que estamos testando é uma aplicação comum, que consome um *web service*, que funciona da seguinte forma, o testador inicia o modo debug da aplicação, onde faz um cadastro na tela da aplicação, e verifica se essa requisição chegou ao *web service*, e o processo funciona da forma inversa também.

2.1.2 Níveis de testes

Os níveis de testes são formas de efetuarmos os testes, onde cada nível tem busca um resultado diferente do outro, tendo maior qualidade quando é utilizado os níveis todos juntos ou os que fazem sentido em sua aplicação.

2.1.2.1 Teste de unidade

PRESSMANE (2010, p. 454) define o teste de unidade por uso intenso de técnicas de teste que exercitam caminhos específicos na estrutura de controle de um componente para garantir cobertura completa e detecção máxima de erros.

Essa técnica é muito usada nas empresas, onde para subir determinada melhoria ou correção o código deve ter uma cobertura de teste determinada pela empresa, o teste de unidade busca testar de forma separada funcionalidades da aplicação, por exemplo uma calculadora, ela possui várias operações, então divido essas funções e testo uma por uma, no caso do teste da soma $2 + 2 = 5$, depois a multiplicação $2 * 2 = 4$ e divisão $2 / 2 = 1$, tudo separado e diretamente no código, sempre esperando um resultado constante.

No dia a dia do desenvolvimento, existem ferramentas que podem auxiliar o desenvolvimento e execução desses testes, no caso da linguagem Java, temos o JUnit, um framework de teste muito utilizado na linguagem.

2.1.2.2 Teste de integração

GOUVEIA (2004, p. 30) Cita que o teste de integração é definido como sendo o modo que o teste é conduzido para integrar componentes em um sistema. Refere-se ao teste das interações entre unidades e módulos para garantir que eles possuam considerações consistentes e se comunicam corretamente.

O teste de integração é a validação que a integração entre diferentes partes de uma aplicação está acontecendo como o esperado, por exemplo, temos uma aplicação e nessa aplicação temos dois módulos, um modulo financeiro e outro modulo contábil, quando eu gero um lançamento financeiro, dependendo do que seja esse lançamento, ele deve afetar a contabilidade, gerando assim valores de impostos e demais taxas, se essa integração está funcionando como levantado no requisito, então o teste de integração fez sua parte.

2.1.2.3 Teste de regressão

SOMMERVILLE (2010, p. 156) define o teste de regressão envolve a execução de conjuntos de testes que tenham sido executados com sucesso, após as alterações serem feitas em um sistema. O teste de regressão verifica se essas mudanças não introduziram novos bugs no sistema e se o novo código interage com o código existente conforme o esperado.

O teste de regressão pode ser utilizado de forma manual ou utilizado via linguagem de programação, esse teste é muito utilizado atualmente, já que existe a necessidade de correção ou melhoria nas nossas aplicações, então o teste de regressão é utilizado, fazendo o teste de todas as funções já existentes e validando se algo está errado ou com falha.

Sempre que o desenvolvedor sobe ao repositório alguma melhoria ou algo do gênero, esse teste é feito para validar que o que tinha antes desse desenvolvimento continua funcionando do mesmo jeito que antes.

Esse tipo de teste exige bastante esforço da equipe desenvolvedora de testes, então é mais recomendado quando a aplicação sofre bastante alteração ou tem um nível de criticidade alta.

2.1.2.4 Teste de usabilidade

Hoje é comum ouvir falar da usabilidade, de como o usuário se sente ao usar determinada aplicação, o teste de usabilidade tem esse intuito, validar o fluxo de acesso do usuário, acessar links, realizar cadastros, validar retornos do sistema para o usuário, facilidade de achar os recursos, entre outros. É muito importante pensar no usuário quando uma aplicação é desenvolvida, já que a maioria das aplicações são feitas para a utilização humana.

NIELSEN citou que “Mesmo os melhores designers produzem produtos de sucesso apenas se seus projetos resolverem os problemas certos. Uma interface maravilhosa para os recursos errados falhará.”, isso significa que não importa o quão bonita a aplicação seja, se ela não for funcional, ela falhará.

RUBIN (2008, p. 21) define o teste de usabilidade como uma ferramenta de pesquisa, com raízes na metodologia experimental clássica. Por isso devemos usar o teste de usabilidade para confirmar todos os fluxos e que o usuário não fique preso na aplicação, é realmente experimentar a aplicação.

2.1.2.5 Teste de desempenho

SOMMERVILLE (2010, p. 159) define que os testes de desempenho precisam ser projetados para assegurar que o sistema possa processar a carga a que se destina. Isso normalmente envolve a execução de uma série de testes em que você aumenta a carga até que o desempenho do sistema se torne inaceitável.

Como citado, os testes de desempenho têm níveis, esses níveis são utilizados para testar o desempenho de uma aplicação, cada nível é um teste diferente, com resultados e funcionalidades diferentes.

2.1.2.5.1 Teste de carga

RIOS e MOREIRA (2006, p. 19) define que esse teste tem como objetivo avaliar a resposta de um software sob uma pesada carga de dados, repetição de certas ações de entrada de dados, entrada de valores numéricos grandes, grandes e complexas queries, consultas a um banco de dados.

Esse teste demonstra que as aplicações modernas possuem grande necessidade de recebimento de dados, de múltiplos usuários, e esse teste garante que a aplicação consegue operar em grande quantidade de cargas.

2.1.2.5.2 Teste de estresse

SOMMERVILLE (2010, p. 159) define que os testes de estresse são utilizados em sistemas distribuídos baseados em uma rede de processadores. Esses sistemas frequentemente apresentam degradação severa quando estão muito carregados. A rede fica inundada com dados de coordenação que os diferentes processos devem trocar.

Esse teste é bem confundido com o teste de carga, mas o objetivo do teste de estresse é assegurar a qualidade da aplicação mesmo com grande quantidade de dados para processamentos.

2.1.2.5.3 Teste de volume

BARTIÉ (2002, p. 115) define que este teste tem por objetivo determinar os limites de processamento e carga do software e de toda infraestrutura da solução. É operacionalizado através de incrementos sucessivos de volume das operações realizadas com o software, até que este atinja o limite máximo. O objetivo é conhecer os limites da solução e avaliar o quão próximo ou distante esses requisitos estão do limite.

Por exemplo o aplicativo de banco, existem algumas pessoas ou empresas que possuem uma grande quantidade de transações mensais, esse cliente precisa ter acesso ao seu extrato, então devemos ter certeza de que o aplicativo suporta a geração dessas transações e consultas.

2.1.2.5.4 Teste de Escalabilidade

SOMMERVILLE (2010, p. 334) define a escalabilidade de um sistema como sua capacidade de oferecer um serviço de alta qualidade, uma vez que aumenta a demanda de sistema.

Quantidade de acesso, tecnologia, e até servidores mudam, e nossa aplicação deve estar pronta para essas mudanças, esse teste tem como objetivo mapear a aplicação em caso de ampliação, quando falamos dessa ampliação, é importante saber que ela abrange todos os níveis da aplicação, desde o processamento de dados á requisições por segundo feita pelos usuários.

2.1.2.6 Teste exploratório

SWEBOK (2014, p. 89) define o teste exploratório como simultânea aprendizagem, projeto de teste e execução de teste, ou seja, os testes não são definidos antecipadamente em um plano de teste estabelecido, mas são dinamicamente projetados, executado e modificado. A eficácia dos testes exploratórios depende do conhecimento do engenheiro de software.

O teste exploratório tem como objetivo explorar determinada função ou processo de um sistema, esse teste tem que ter um proposito para ser feito, por exemplo, o método de login não possui documentação nenhuma, e precisamos efetuar uma melhoria nesse método, então a utilização do teste exploratório tem o propósito de testar todo o processo de login. O testador deve efetuar esse processo como um usuário comum do sistema, sempre pensando na usabilidade total da aplicação.

2.1.3 Testes manuais

Como o nome já diz, é a execução desses testes de forma manual por um humano, o testador, ele executa esses testes na camada de interface do usuário onde a interação humano e computador é mais vista. Nesse caso o roteiro de teste é feito pelo testador para validar as funções, fluxos e resultados da aplicação, onde ele pode usar alguns níveis de teste citados acima.

O teste manual tem seu lado positivo e negativo, alguns pontos positivos são:

- Garantir a qualidade do design da aplicação.
- Garantir que a usabilidade é voltada para o usuário.
- Efetuar o teste sem precisar subir em um ambiente de desenvolvimento ou homologação.

- Pouco investimento, já que não é necessário a utilização de uma ferramenta de automação ou habilidades técnicas para a execução.

E alguns pontos contrários:

- Custo por cada camada de teste.
- Lentidão na execução.
- Possível falha humana não detectada.
- Alguns níveis de testes não são possíveis efetuar de forma manual.

O teste manual sempre irá acontecer, ele não está excluído do mundo da engenharia de software, muito pelo contrário, o papel desse teste é fundamental para casos com mudança continua na regra.

2.1.4 Testes automatizados

Testes desenvolvidos para serem executados de forma automática, utilizando linguagem de programação ou ferramentas para elaborar esses testes, contemplando os níveis de teste. Para um teste automatizado ser bem realizado, a regra não deve efetuar constante mudança, pois se isso acontecer, será necessário mudar a lógica do teste, alterando o código ou script.

Como o teste manual, o teste automatizado também possui pontos positivos e pontos negativos, onde podemos entender melhor o contexto de sua utilização no dia a dia, pontos positivos:

- Pouca chance de falha humana.
- Testes com alta velocidade de execução.
- Existem testes que só podem ser feitos utilizando a automação, como o teste de carga.

E alguns pontos negativos:

- Alto custo de desenvolvimento.
- Limitação de ferramenta ou habilidade do testador.
- Tempo de desenvolvimento do teste.
- Alta dificuldade de obter falhas de usabilidade e design da aplicação.

É perceptível que o teste automatizado completa o teste manual, e a utilização dos dois fazem a garantia da qualidade do software.

2.2 Conclusão da fundamentação

Este capítulo trouxe uma visão detalhada sobre os testes de softwares, seus tipos, técnicas e níveis de teste. Um conjunto de técnicas de teste podem ser utilizadas juntas, sem afetar o objetivo da qualidade de software, a utilização de muitas técnicas, tipos e níveis fazem com que a garantia da qualidade seja entregue.

A forma de organização dos testes, utilização e automação é determinada de formas diferentes em cada contexto, já que no mundo da tecnologia tudo depende do contexto, mas o teste de regressão pode ser utilizado em qualquer processo da aplicação fazendo e é de suma importância para garantir a qualidade quando é feita alguma manutenção ou melhoria.

3 DESENVOLVIMENTO

3.1 Contexto do capítulo

O objetivo deste capítulo é abordar a utilização do teste de regressão, o planejamento desse tipo de teste, além de identificar os melhores conceitos para sua utilização, em seguida falaremos da utilização do teste de regressão em API e para concluir, a utilização das ferramentas Postman e Newman.

3.2 Utilização do teste de regressão

Foi citado um pouco sobre os testes no segundo capítulo deste documento, mas agora o objetivo é aprofundar na utilização deste teste, e seu planejamento. O teste de regressão pode ser utilizado em vários contextos, em várias metodologias e ferramentas, a equipe de desenvolvimento está sempre fazendo melhorias e correções nas aplicações, essas correções e melhorias são divididas por feature ou release dependendo de cada empresa.

O teste de regressão busca validar a aplicação por completo, não só a parte que foi arrumada ou melhorada, mas sim todas as funções já existentes, e essa validação pode ser feita de forma manual ou automatizada, então o planejamento desse teste é de suma importância para garantir a qualidade da aplicação.

3.2.1 Planejamento

O planejamento dos testes é sempre complicado para a equipe de qualidade de software, e o teste de regressão não seria diferente, é importante entender o que será automatizado e o que será feito de forma manual, podem existir funções e atividades de uma aplicação que a regra de negócio está sempre sendo alterada, então a utilização da automação não é uma boa saída nesse cenário.

O analista de qualidade de software realiza a criação de um documento chamado Plano de Testes, onde levanta todas as necessidades de validação passada pelo analista de negócios ou dono do produto, esse documento pode ser produzido em linguagem natural ou em uma escrita muito utilizada no dia a dia do QA, o BDD. Exemplo de um cenário de teste escrito em BDD:

Funcionalidade: Acesso ao sistema

Cenário: CT0001 – Efetuar o login com sucesso no sistema

Dado Que eu preciso acessar o sistema

E possuo cadastro ativo na aplicação

Quando aceso a tela de login

E informo meu usuário valido

E informo minha senha correta

Então o sistema deve confirmar meu login

E me redirecionar para a tela principal da aplicação

Pode parecer simples, mas ele é bem utilizado na hora de efetuar o planejamento dos testes, dessa forma, entendemos as regras a serem testada e validas.

3.2.2 Teste de regressão em API

O teste de regressão em APIS, são validações que serão realizadas nos serviços através de rotas disponibilizadas para testes e utilização das mesmas, caso a API não possua um teste de regressão automatizado, devemos fazer a criação, caso exista, quando for desenvolvido algo novo vamos incluir essa validação no roteiro já existente, ou se for uma correção na aplicação, devemos validar se algo no fluxo foi alterado, caso tenha sido alterado, basta alterar no roteiro também, ou só executa-lo caso a aplicação não tenha nenhuma alteração.

3.2.3 Utilizando o Postman para teste de regressão

O Postman é uma ferramenta open source, onde serão realizadas as homologações das APIS de forma simples e com a possibilidade de utilizar várias linguagens para o teste, tanto Java, Javascript, Python e outras.

Essa ferramenta possui vários recursos de alto nível na hora do teste, transformando a utilização mais fácil e com baixa curva de aprendizado, diferente de outras ferramentas que o analista de qualidade de software deve ter um maior conhecimento em desenvolvimento

Para iniciar o processo de teste, é necessário entender um pouco mais sobre a utilização do Postman, entender as coleções, ambientes, requisições, pré-requisições e teste.

3.2.3.1 Coleções

Quando é efetuado testes em API, se faz necessário testar todo o CRUD, utilizando os métodos HTTP. Tipos de métodos que utilizaremos nos testes:

POST- adicionar novos dados

PUT— substituir dados existentes

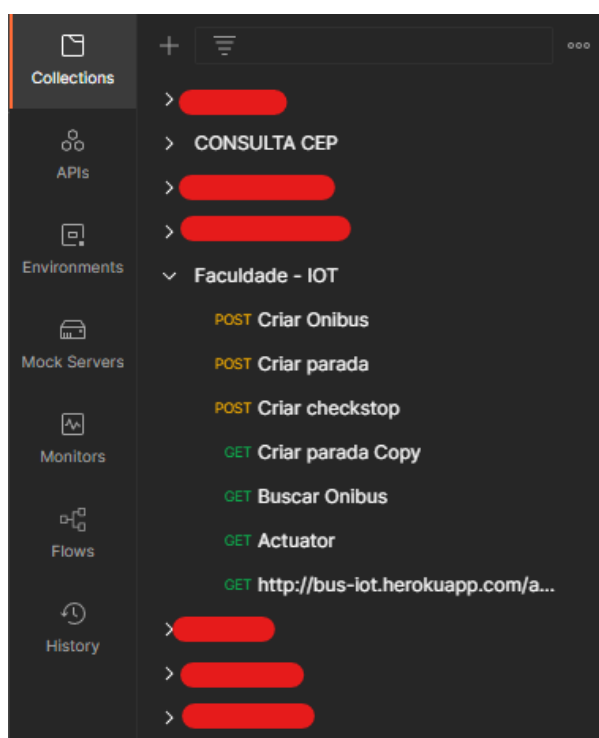
PATCH— atualizar alguns campos de dados existentes

DELETE— excluir dados existentes

Para cada método, temos um teste diferente, e com isso, já podemos imaginar quantos arquivos teremos em um teste complexo em uma aplicação.

Para agrupar esses arquivos de requisições o Postman criou as coleções, onde organizamos todas essas requisições em um único lugar, essas coleções é o que utilizamos para fazer o teste automatizado no Newman, exportando a mesma para execução.

Figura 2 – Coleção Postman



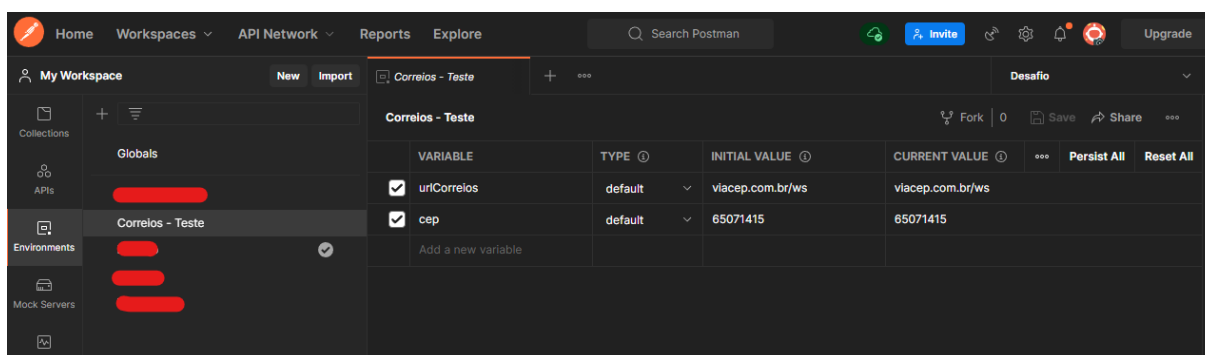
Fonte: Próprio autor.

3.2.3.2 Ambientes

Ambientes são conjuntos de variáveis que utilizamos em nossas execuções dos testes, essas variáveis podem ser alteradas, criadas e utilizadas a qualquer momento nas requisições, basta ser selecionada no momento da execução dos testes.

Utilizamos os ambientes para evitar deixar estático os valores no teste, já que os valores podem ser alterados em cada tipo de requisição ou resposta. Segue exemplo prático na Figura 3:

Figura 3 – Ambientes no Postman

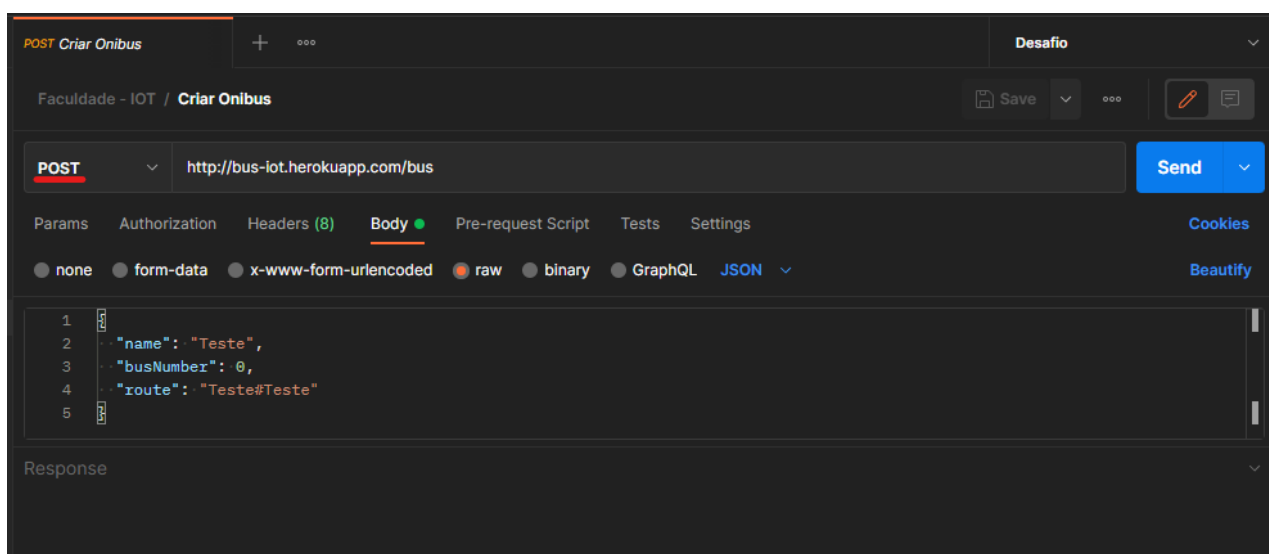


Fonte: Próprio autor.

3.2.3.3 Requisições

As requisições é a forma que utilizamos para fazer a conexão com a aplicação, com os métodos HTTP, como falamos no tópico das coleções, fazemos todas as operações como se fossemos usuários reais da aplicação, essas requisições podem ser diferentes, de acordo com a aplicação que estamos nos conectando, já que existem parâmetros diferentes em cada momento.

Por exemplo, se a aplicação for um cadastro de ônibus, temos uma requisição do tipo POST, onde solicitamos a aplicação para criar um determinado ônibus com os dados que informamos, claro que esses dados podem ser diferentes de cada tipo de aplicação, segue um exemplo na Figura 4:

Figura 4 – Requisição POST

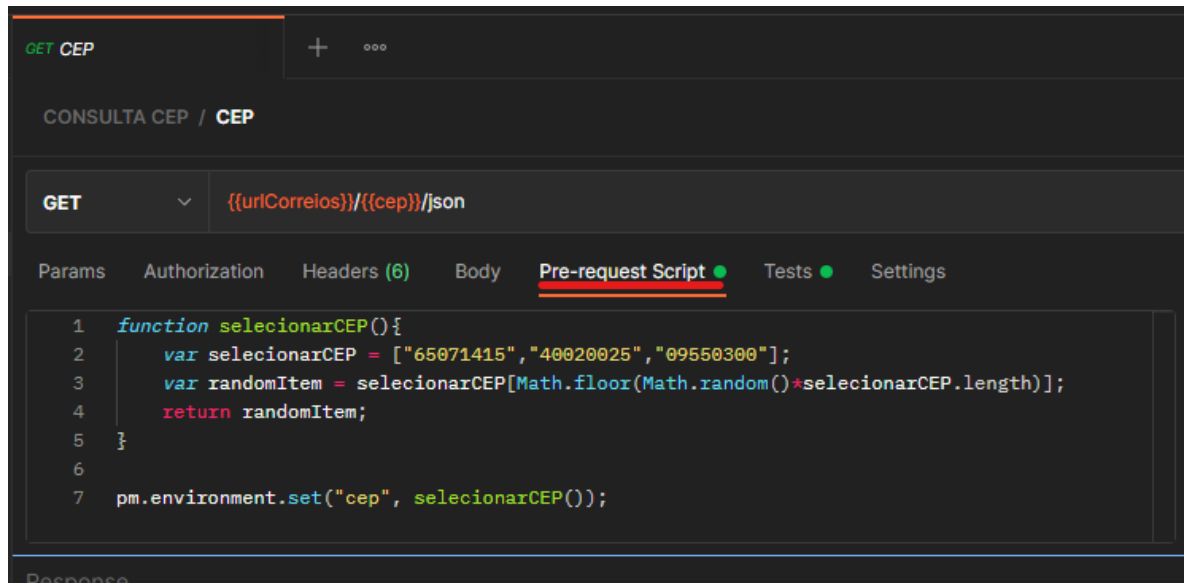
Fonte: Próprio autor.

Como falamos, o tipo é POST, e nessa requisição, informamos os dados no corpo da requisição, onde passamos o nome do ônibus, o seu número e a rota, quando fazemos o envio do mesmo a aplicação pode informar que a criação ocorreu com sucesso ou não. Veremos mais sobre as requisições na execução dos testes no capítulo 4 deste documento.

3.2.3.4 Pré-requisições

As pré-requisições são scripts que podemos executar antes de uma requisição, por exemplo, queremos consultar de forma aleatória um CEP, não queremos deixar estático na requisição um único CEP, então fazemos um script em JavaScript para alterar valores de variáveis, parâmetros e dados da requisição.

Exemplo de pré-requisição na Figura 5:

Figura 5 – Pré-requisição

Fonte: Próprio autor.

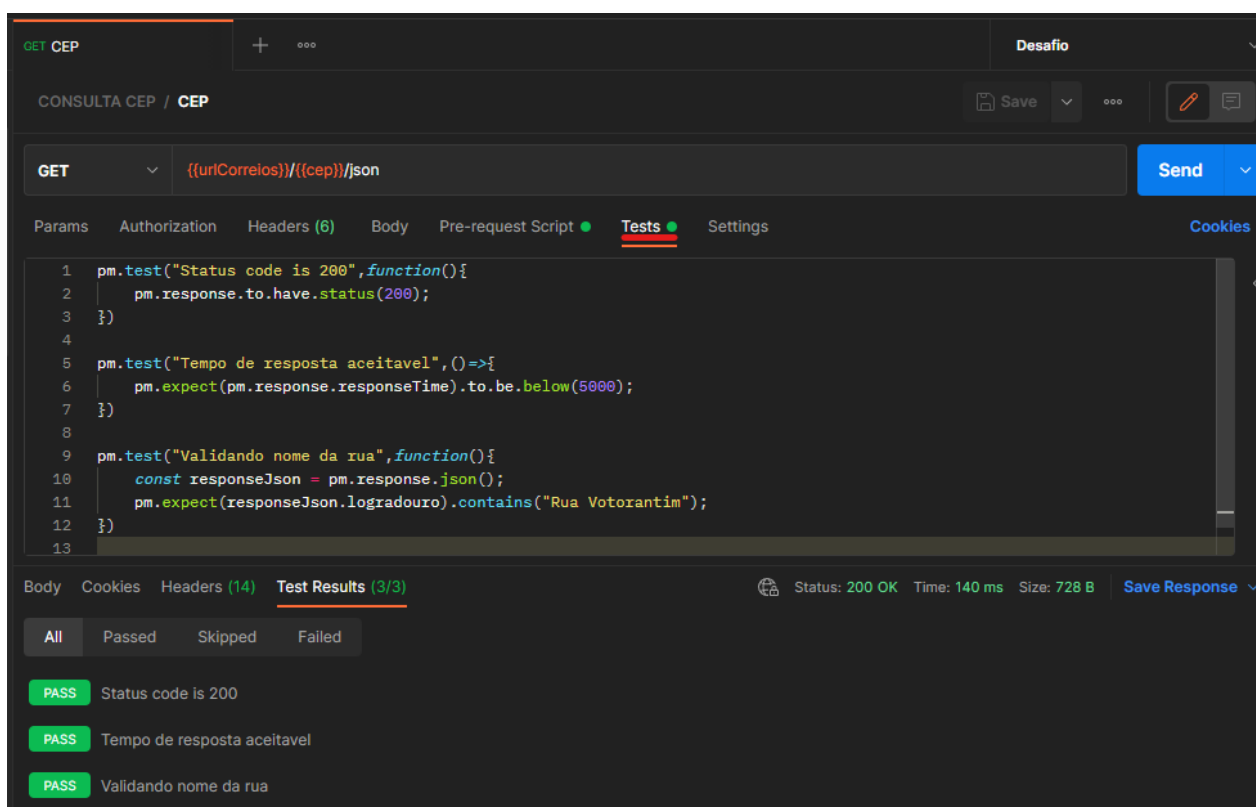
3.2.3.5 Teste

Este é o ponto mais importante na utilização do Postman, os scripts de teste, nessa aba, fazemos os scripts para validar o que estamos testando, por exemplo, o código de retorno, o tempo de resposta, os dados retornados e outras informações que sejam uteis para nosso teste.

Esses testes são realizados em tempo de execução, utilizando o Node.js que é uma biblioteca da linguagem JavaScript, então esses testes são feitos em JavaScript. Para realizar esses testes, o Postman segue uma frequência logica, essa frequência é a pré-requisição, requisição, a resposta dessa requisição e por último o teste que escrevemos.

Podemos ver a utilização dos testes na Figura 6:

Figura 6 – Script de teste



Fonte: Próprio autor.

3.2.4 Utilizando o Newman

O Newman é uma aplicação criada para executar as coleções do Postman por linha de comando, utilizando o JavaScript para essa tarefa, dessa forma ele executa teste de forma rápida e fácil, sem precisar de esforço do usuário diretamente pela linha de comando, o desenvolvimento dessa aplicação, tem o intuito de facilitar a integração entre servidores de compilação e integração contínua, nesse documento, iremos demonstrar seu uso no Azure DevOps.

3.3 Executando o teste de regressão

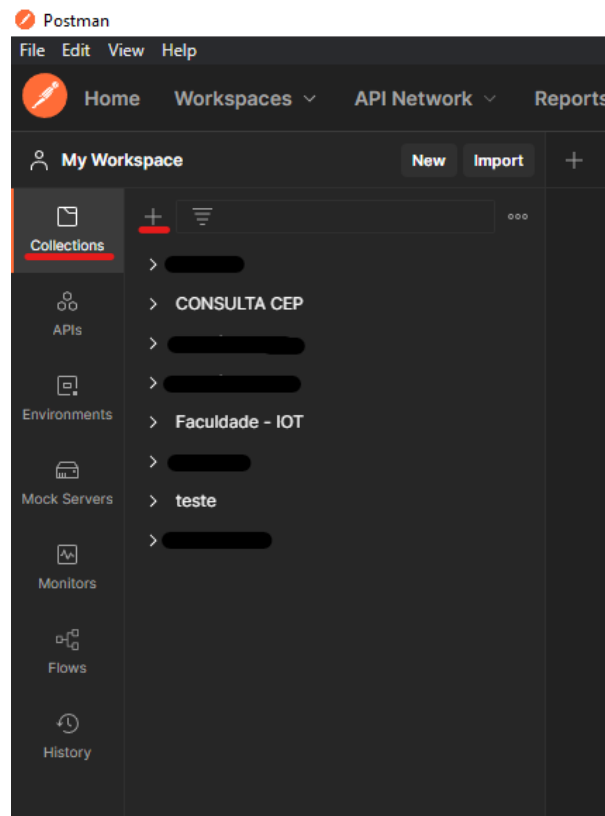
Vimos nos capítulos anteriores como utilizar as ferramentas necessárias para efetuar o teste de regressão, vamos desenvolver um teste de regressão em uma API básica de cadastro.

3.3.1 Criando as coleções e requisições para o teste

O primeiro passo para efetuarmos o teste de regressão com o Postman, é efetuar o cadastro das coleções para organizarmos as requisições criadas para o teste, esse passo é mais ilustrativo, onde iremos demonstrar o passo a passo para depois demonstrar os resultados desse teste.

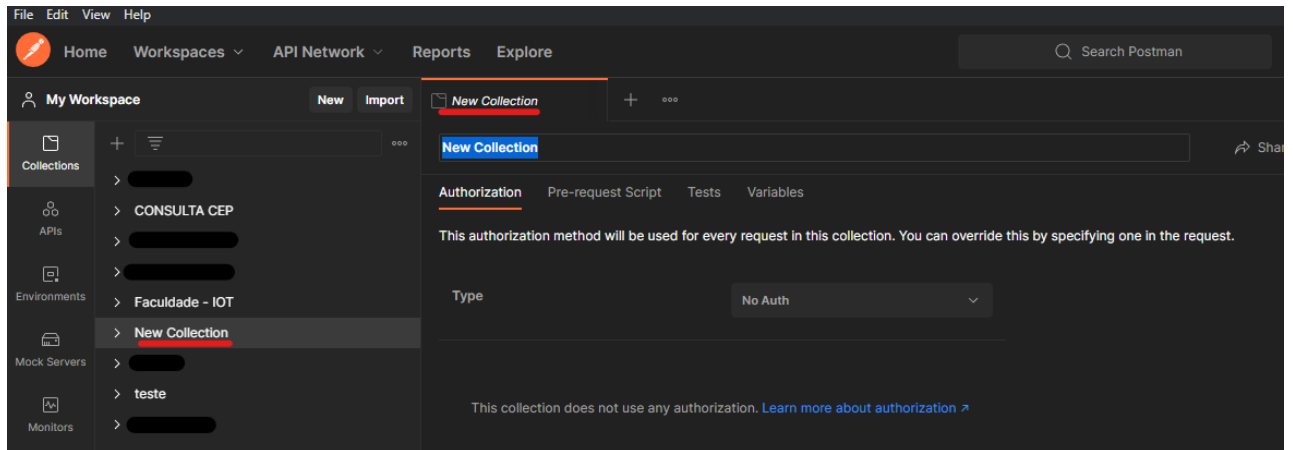
O primeiro passo para criarmos a coleção é selecionar nosso workspace, clicar em *Collections* e depois no símbolo de mais (+), como demonstrado na Figura 7:

Figura 7 – Criação de Coleção

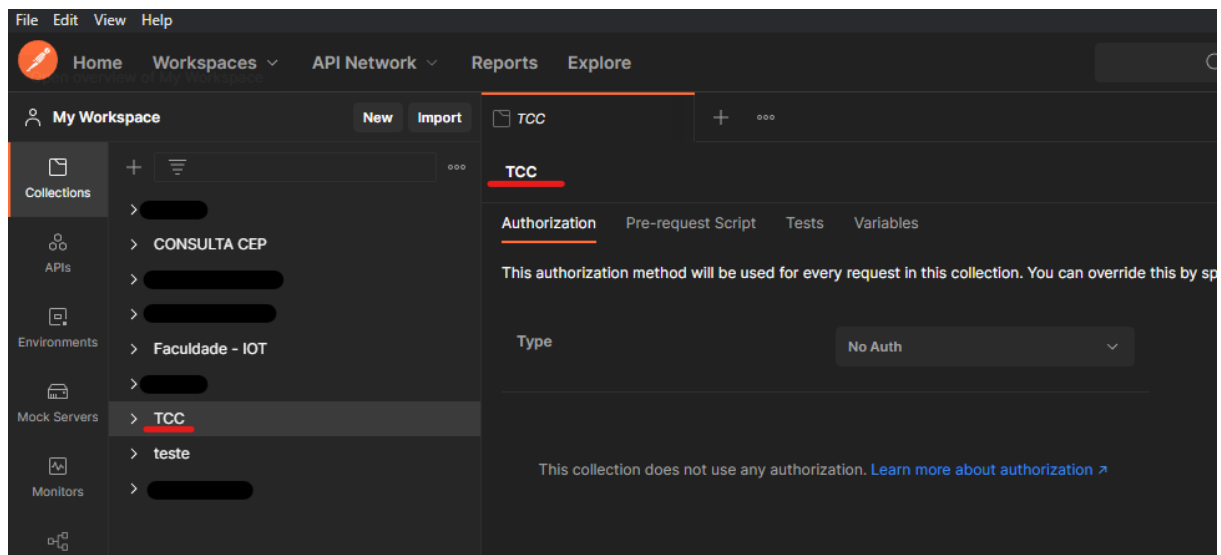


Fonte: Próprio autor.

Após efetuarmos esse processo, irá aparecer uma nova coleção com o nome New Collection, e podemos alterar esse nome, no caso, vou colocar como TCC. Os passos estão demonstrados na Figura 8 e Figura 9:

Figura 8 – Criando nome da Coleção

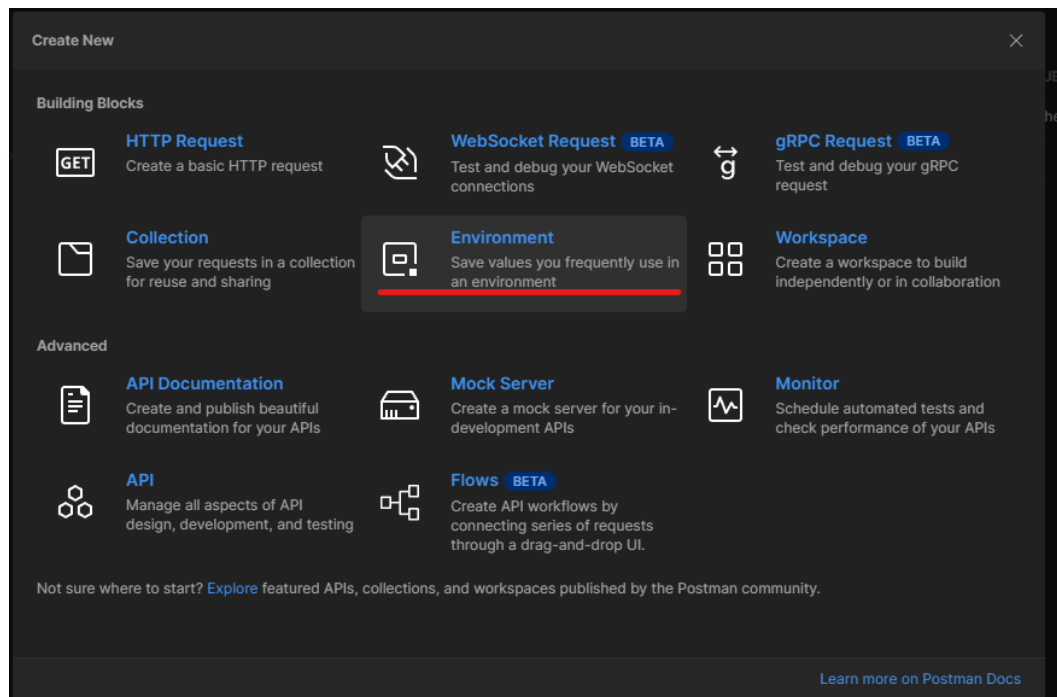
Fonte: Próprio autor.

Figura 9 - Validação do nome

Fonte: Próprio autor.

Já criamos a coleção, agora vamos criar o ambiente, para adicionarmos as variáveis de nosso teste, no caso, a primeira variável que teremos em nosso ambiente é o link da API que vamos utilizar para efetuar os testes, essa API é uma aplicação feita para um outro projeto de estudos, é uma aplicação de cadastro de ônibus e parada de ônibus. A criação do ambiente é bem parecida com a criação da coleção, como demonstrado na Figura 10:

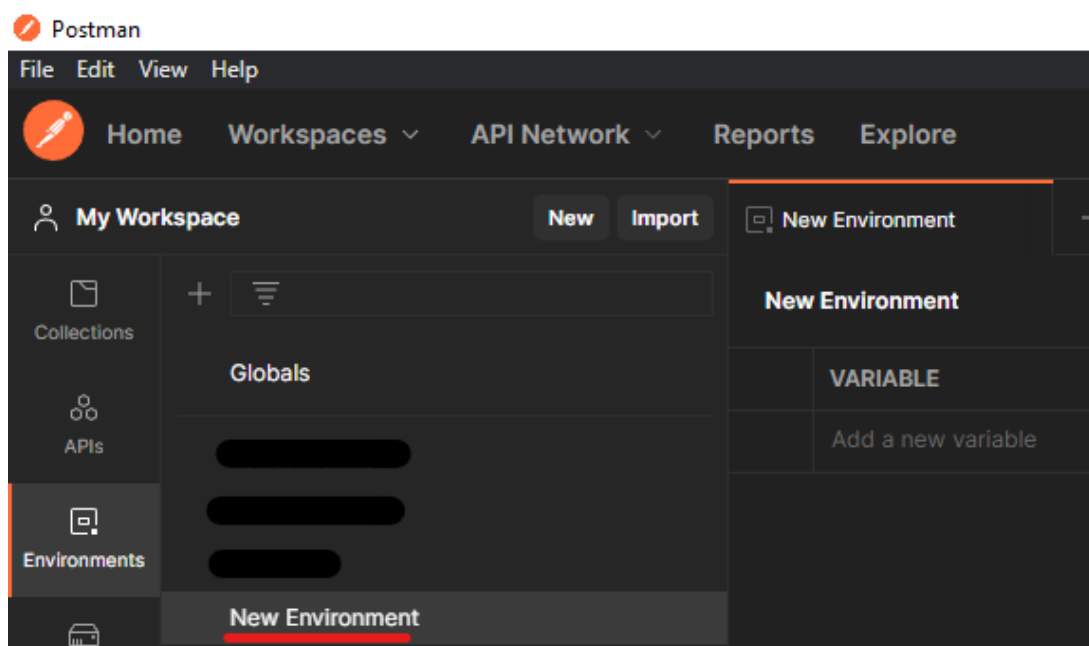
Figura 10 – Criação do Ambiente



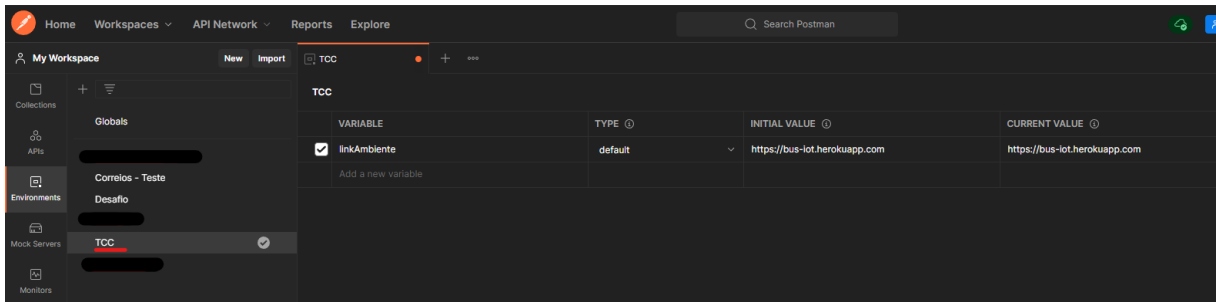
Fonte: Próprio autor.

Ao clicarmos, teremos um novo ambiente, com o nome padrão de New Environment que podemos alterar, como fizemos na coleção, e utilizarei o nome de TCC:

Figura 11 – Alteração do nome do Ambiente



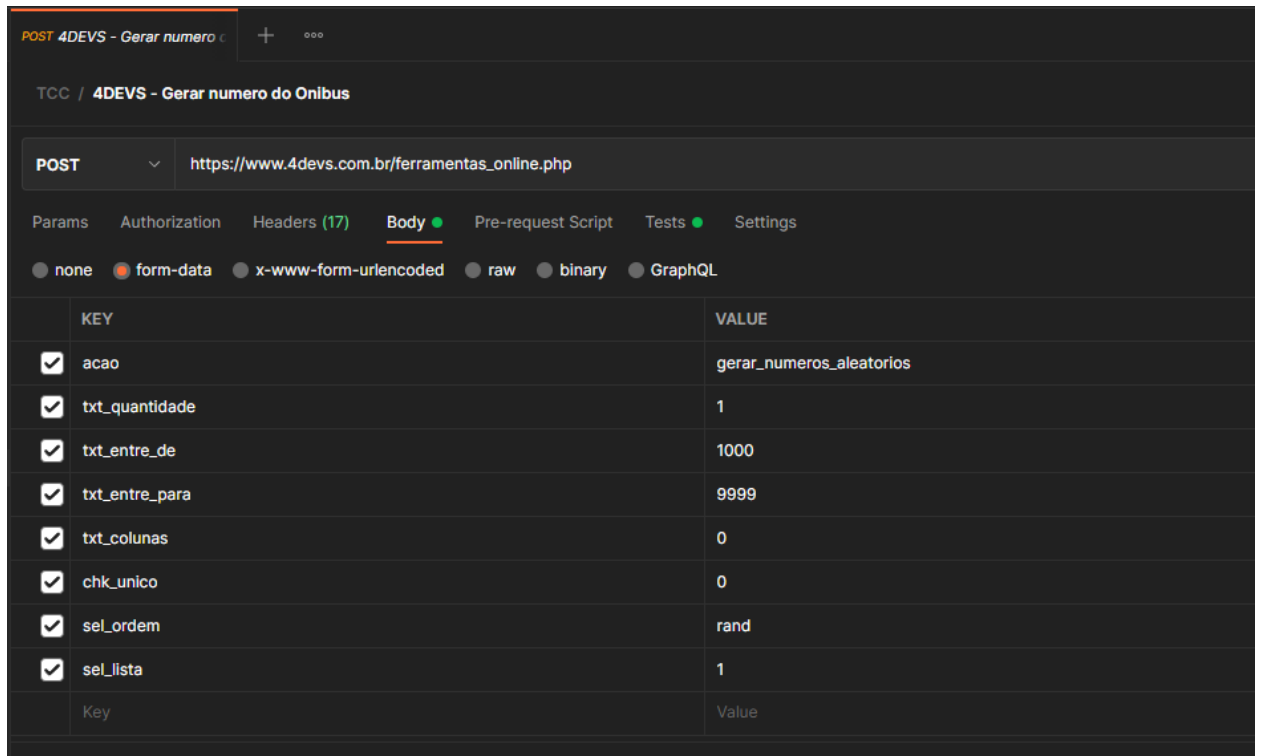
Fonte: Próprio autor.

Figura 12 – Adicionando link no Ambiente

Fonte: Próprio autor.

Agora que já fizemos a configuração inicial para utilizar o Postman, podemos começarmos as nossas requisições, onde faremos todo o fluxo necessário para o teste, sempre importante lembramos dos métodos HTTP que vamos utilizar, a necessidade dos scripts e pré-requisições. Vamos começar com um método POST, para a criação de um ônibus na aplicação.

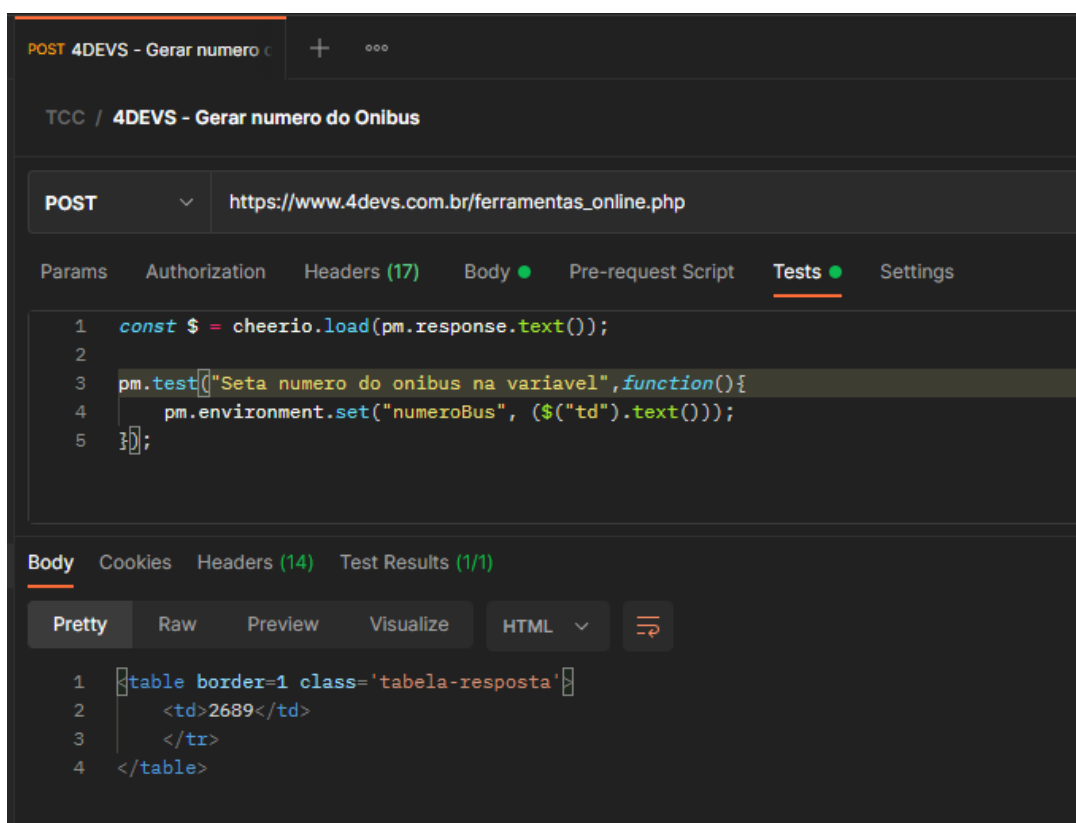
No Postman, podemos utilizar várias aplicações e fazemos uma integração entre elas, nesse caso de cadastro de ônibus, fiz uma comunicação com um serviço bastante conhecido pelo time de desenvolvimento que é o 4Devs, onde estou gerando um número aleatório para o cadastro. Informamos os dados necessários para a requisição, nesse serviço é utilizado no corpo da requisição um formulário com os dados preenchidos.

Figura 13 – Preenchimento de Body

Fonte: Próprio autor.

Após esse processo, quando enviamos a requisição, a aplicação nos responde um número aleatório:

Figura 14 – Retorno da requisição 4Devs



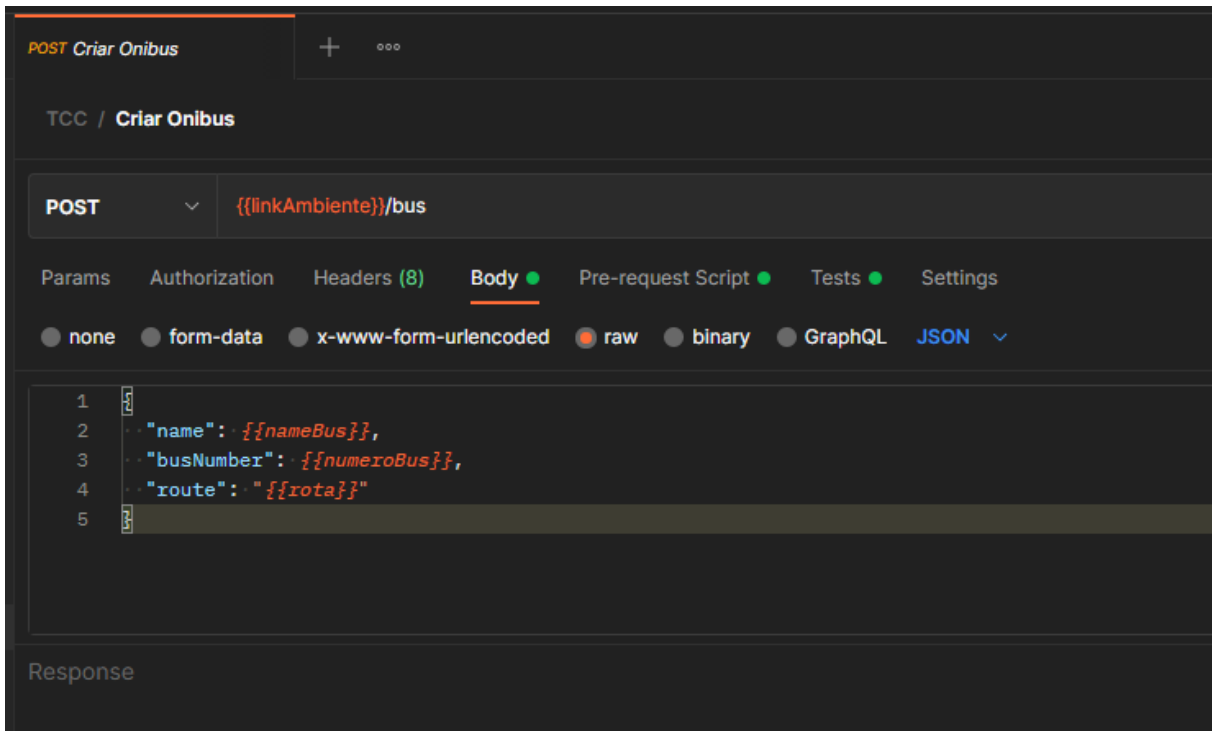
```
POST 4DEV5 - Gerar numero c + ...  
TCC / 4DEV5 - Gerar numero do Onibus  
POST https://www.4devs.com.br/ferramentas_online.php  
Params Authorization Headers (17) Body ● Pre-request Script Tests ● Settings  
1 const $ = cheerio.load(pm.response.text());  
2  
3 pm.test("Seta numero do onibus na variavel", function(){  
4   pm.environment.set("numeroBus", $("td").text());  
5 });  
Body Cookies Headers (14) Test Results (1/1)  
Pretty Raw Preview Visualize HTML ↕  
1 <table border=1 class='tabela-resposta'>  
2   <td>2689</td>  
3 </tr>  
4 </table>
```

Fonte: Próprio autor.

Nesse ponto já concluímos nossa primeira requisição, agora vamos utilizar essa variável criada em uma outra requisição, mostrando assim a integração entre as requisições e variáveis do ambiente no Postman.

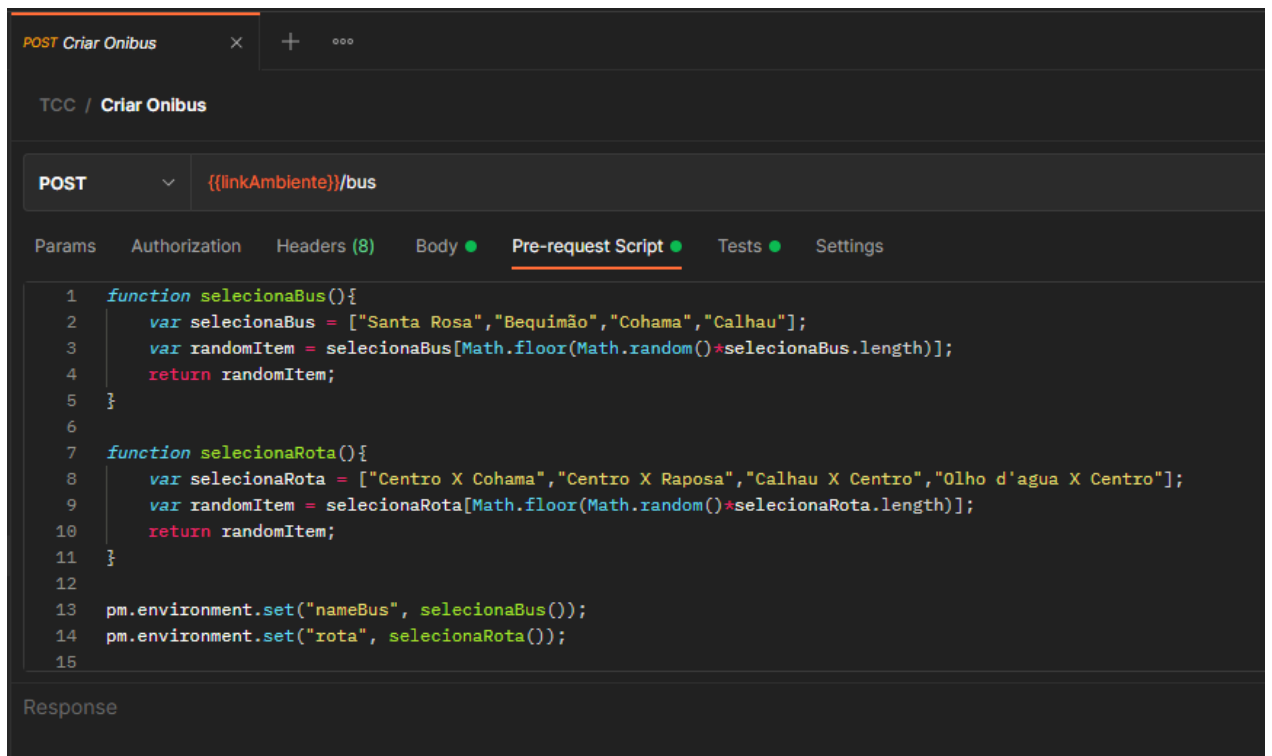
Para isso, vamos fazer a criação do ônibus em nossa aplicação de teste, podemos verificar na imagem abaixo que temos o atributo *numberBus*, onde foi adicionada a variável gerada pela requisição que citamos acima.

Figura 15 – Body da criação de ônibus pela requisição



Fonte: Próprio autor.

Figura 16 – Script pré-requisições para preenchimento das variáveis na criação do ônibus

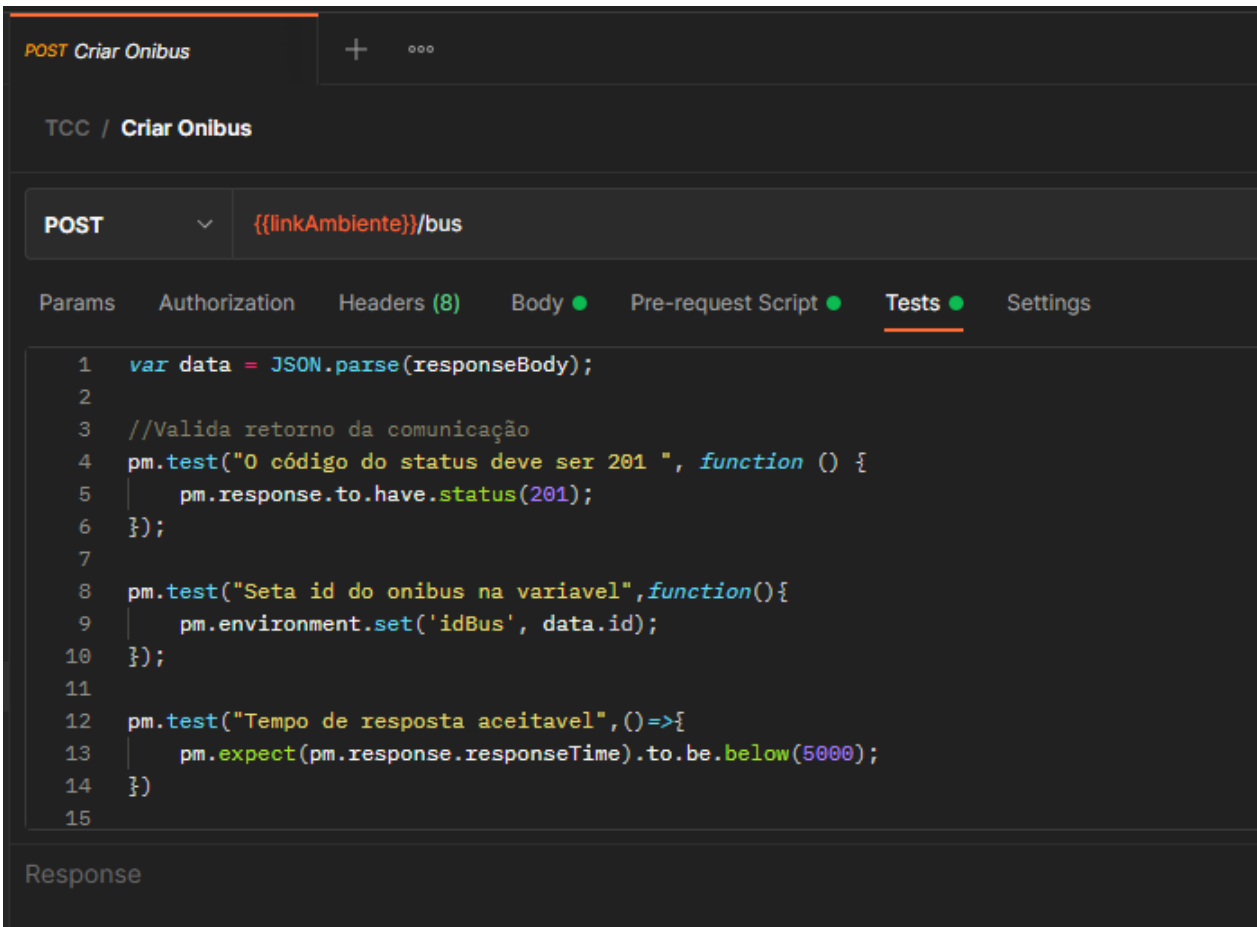


Fonte: Próprio autor.

Nessa requisição, temos também scripts de pré-requisição, que o Postman executa utilizando a linguagem JavaScript, o script será disponibilizado no anexo desse documento.

E como falamos antes, temos a aba mais importante para nosso projeto, que são os testes, fazemos as afirmativas que nosso teste deve ter, utilizando linguagem de programação, como demonstrado na Figura 17:

Figura 17 – Script de teste para validar a criação do ônibus



The screenshot shows the Postman interface for a POST request named "Criar Onibus". The URL is set to "{{linkAmbiente}}/bus". The "Tests" tab is selected, displaying a JavaScript script for testing the response. The script includes comments in Portuguese and uses Postman's test functions to validate the status code, set environment variables, and check the response time.

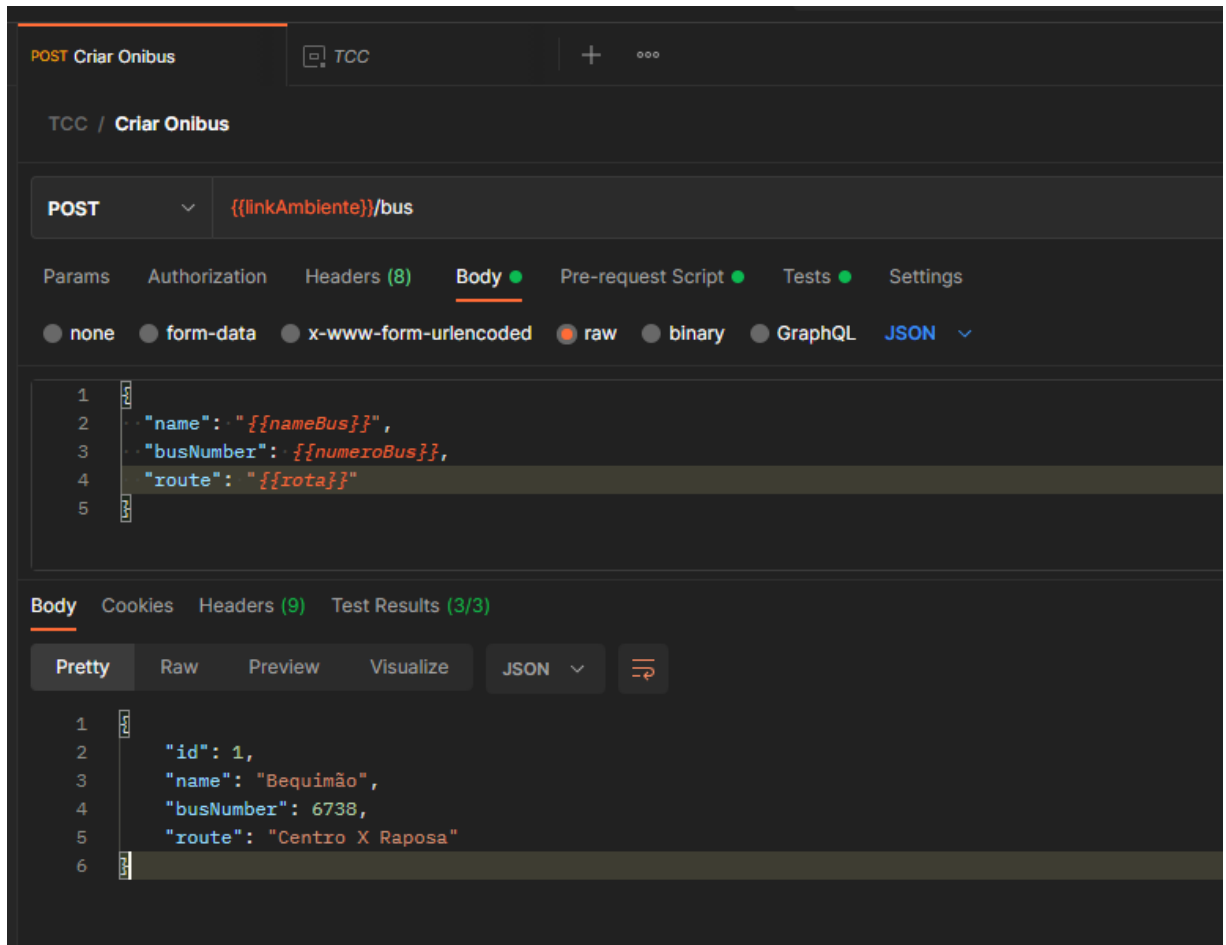
```
1  var data = JSON.parse(responseBody);
2
3  //Valida retorno da comunicação
4  pm.test("O código do status deve ser 201 ", function () {
5    pm.response.to.have.status(201);
6  });
7
8  pm.test("Seta id do onibus na variavel",function(){
9    pm.environment.set('idBus', data.id);
10 });
11
12 pm.test("Tempo de resposta aceitavel",()=>{
13   pm.expect(pm.response.responseTime).to.be.below(5000);
14 })
15
```

Response

Fonte: Próprio autor.

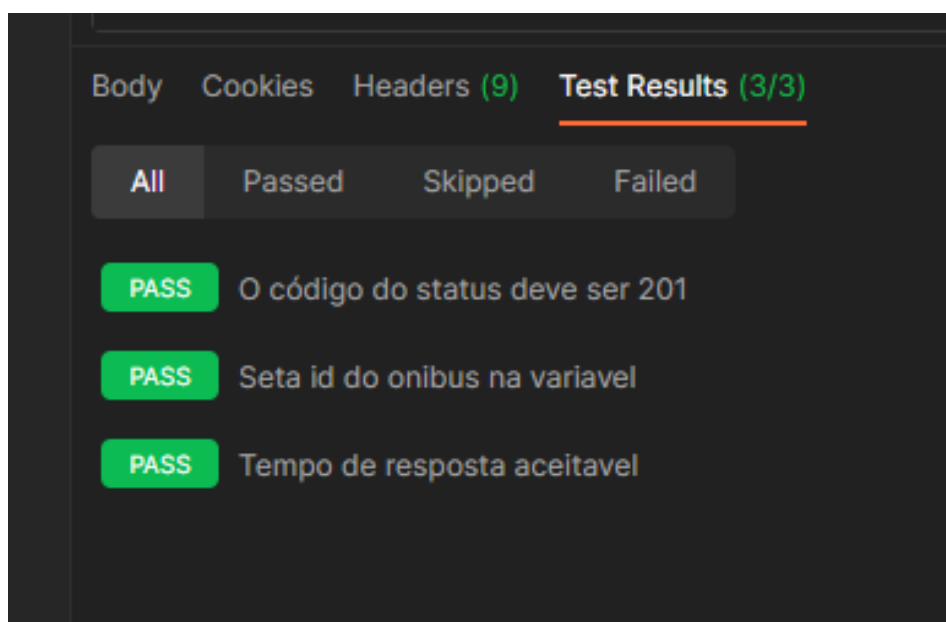
Com isso já temos tudo para executar nossa requisição de criação do ônibus, onde iremos fazer a comunicação com a aplicação, informando que desejamos criar esse objeto e a aplicação deve nos responder como o esperado.

Figura 18 – Resposta da requisição



Fonte: Próprio autor.

Efetuamos nossa requisição de criação de ônibus e a aplicação nos respondeu o resultado, podemos verificar os resultados dos testes nessa requisição, na imagem temos (3/3), isso quer dizer que temos três testes, e os três foram executados com sucesso, como demonstrado na seguinte imagem:

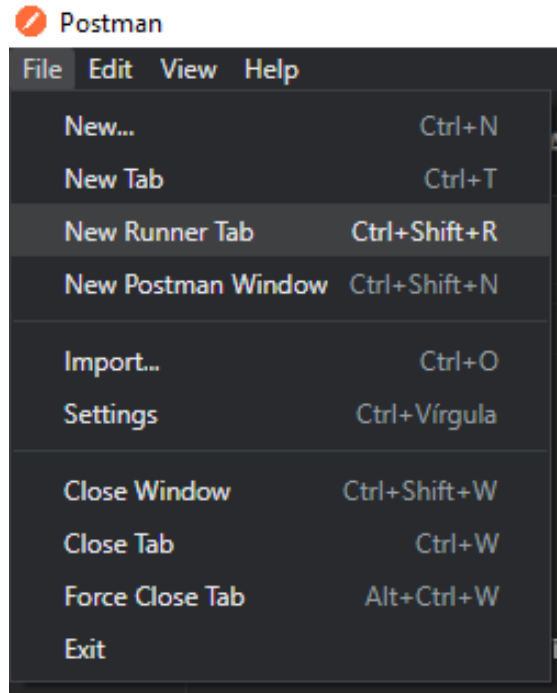
Figura 19 - Resultados do teste

Fonte: Próprio autor.

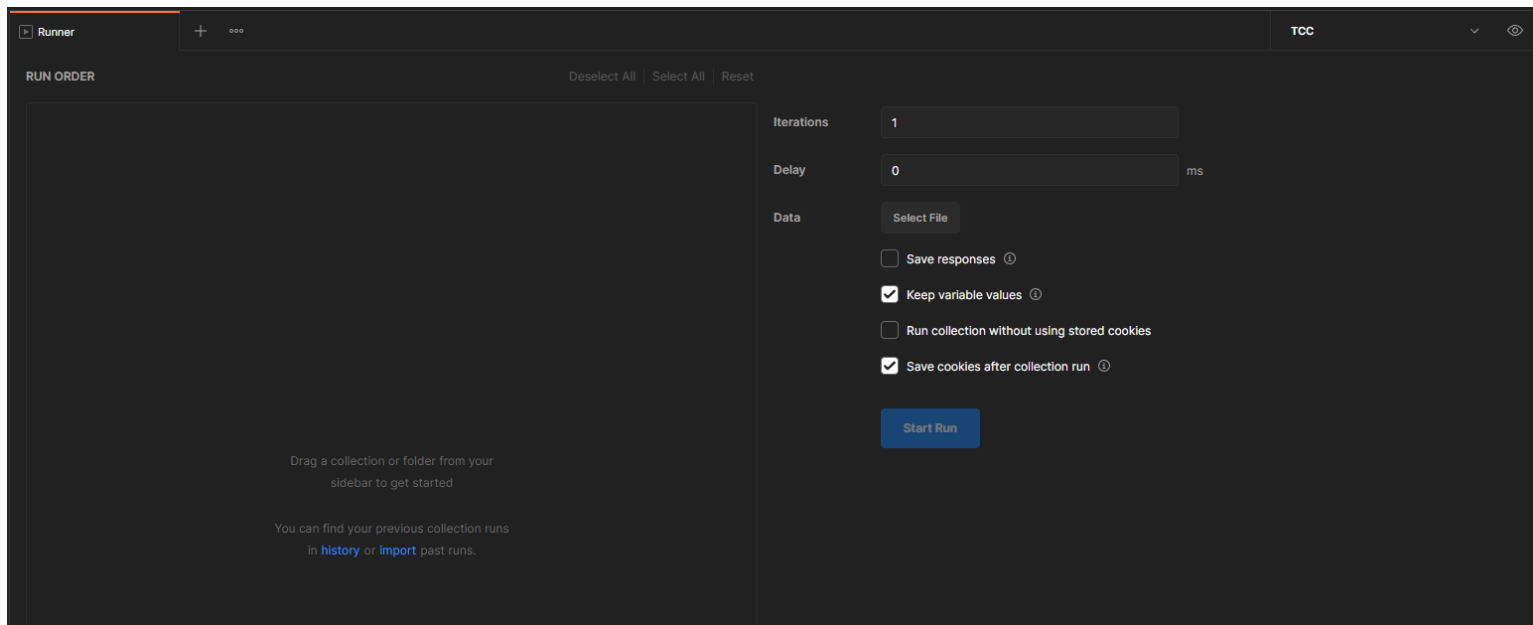
3.3.2 Executando coleções de teste no Postman

Criamos nossos cenários de teste, nossa coleção, nossas requisições com seus scripts e testes, e agora devemos efetuar a execução dessa coleção de teste no Postman, para depois executarmos no Newman.

Para efetuar essa execução no postman é bem simples, basta clicarmos na opção, *File* e depois em *New Runner Tab*:

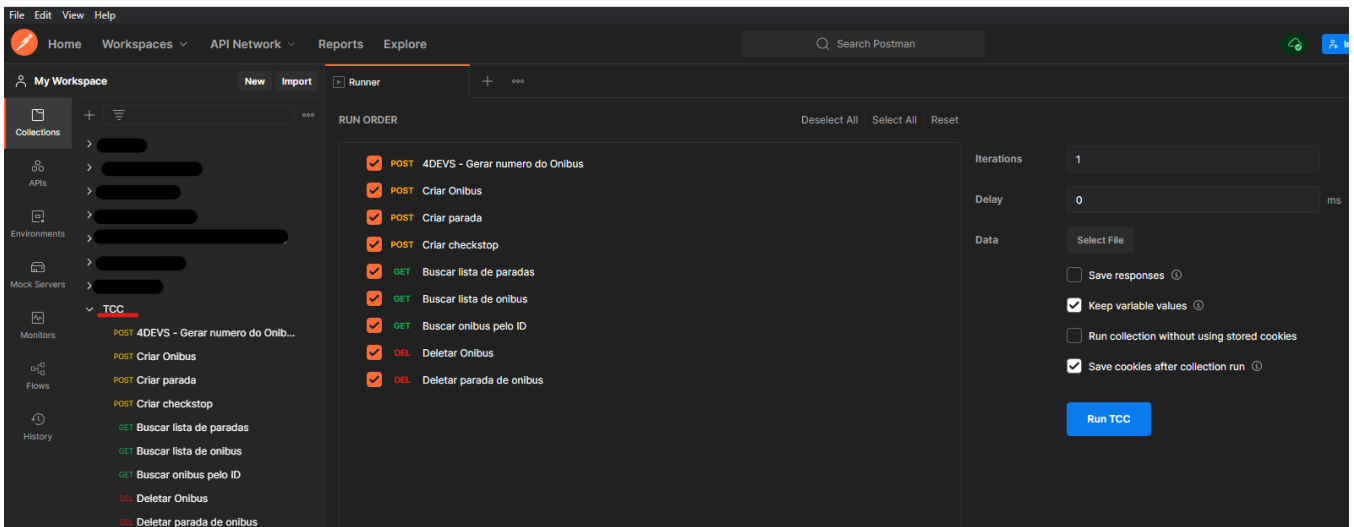
Figura 20 – New Runner Tab

Fonte: Próprio autor.

Figura 21 – Runner

Fonte: Próprio autor.

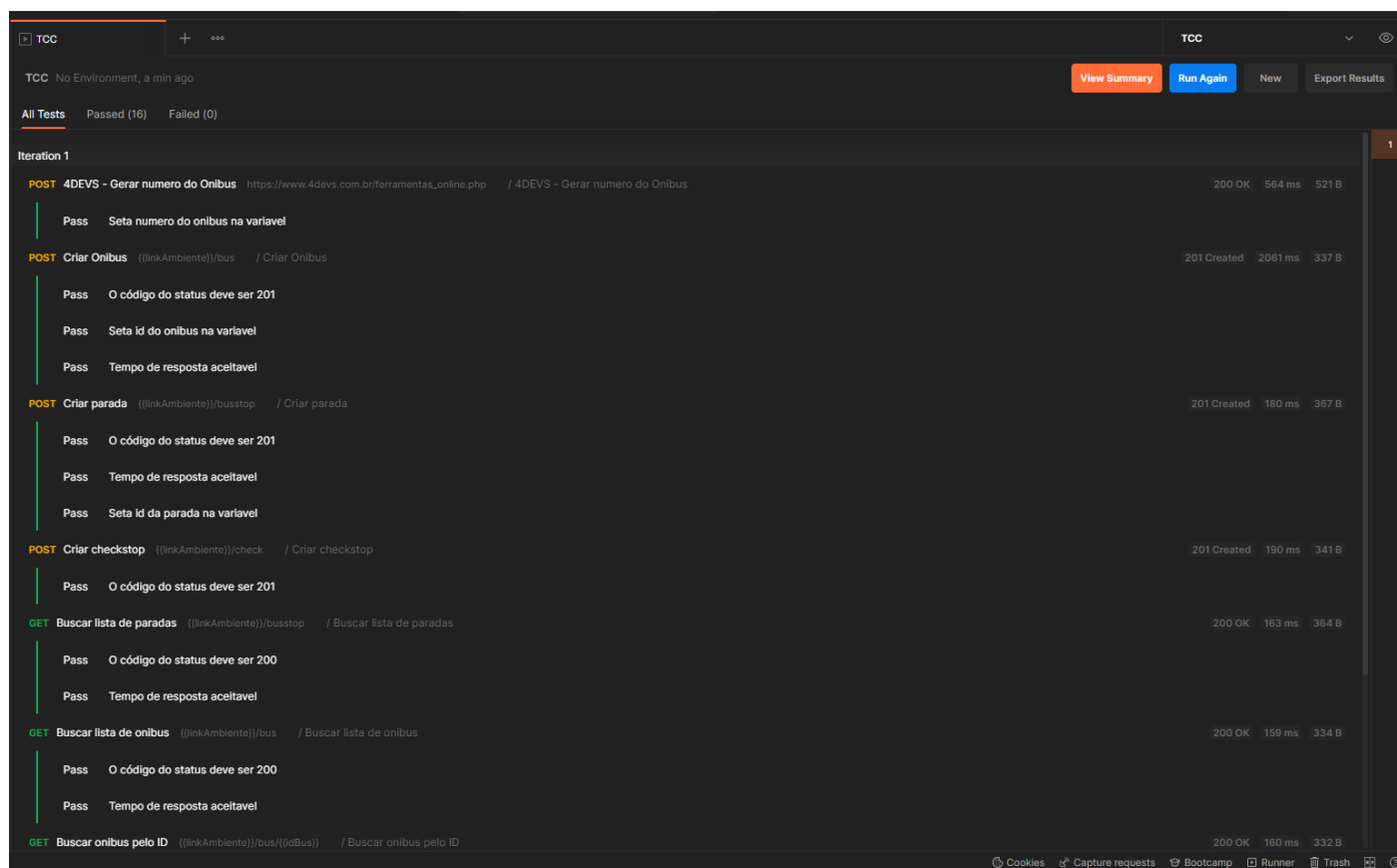
O executor de coleções do Postman é bem intuitivo, para selecionar qual coleção vamos executar, basta clicar na coleção e arrastar para a tela do executor, que ficará como na Figura 22:

Figura 22 – Executando coleção com o Runner

Fonte: Próprio autor.

Existem algumas opções nessa tela que é bastante importante, dependendo de qual tipo de teste estamos fazendo, temos a quantidade de iterações e o atraso, como nosso objetivo é fazer um teste de regressão em nossa aplicação, então irei deixar como está o padrão, demonstrado na imagem acima, e para iniciar a execução dos testes dessa coleção, basta clicar em *Run*.

Figura 23 – Executando os testes com Runner



Fonte: Próprio autor.

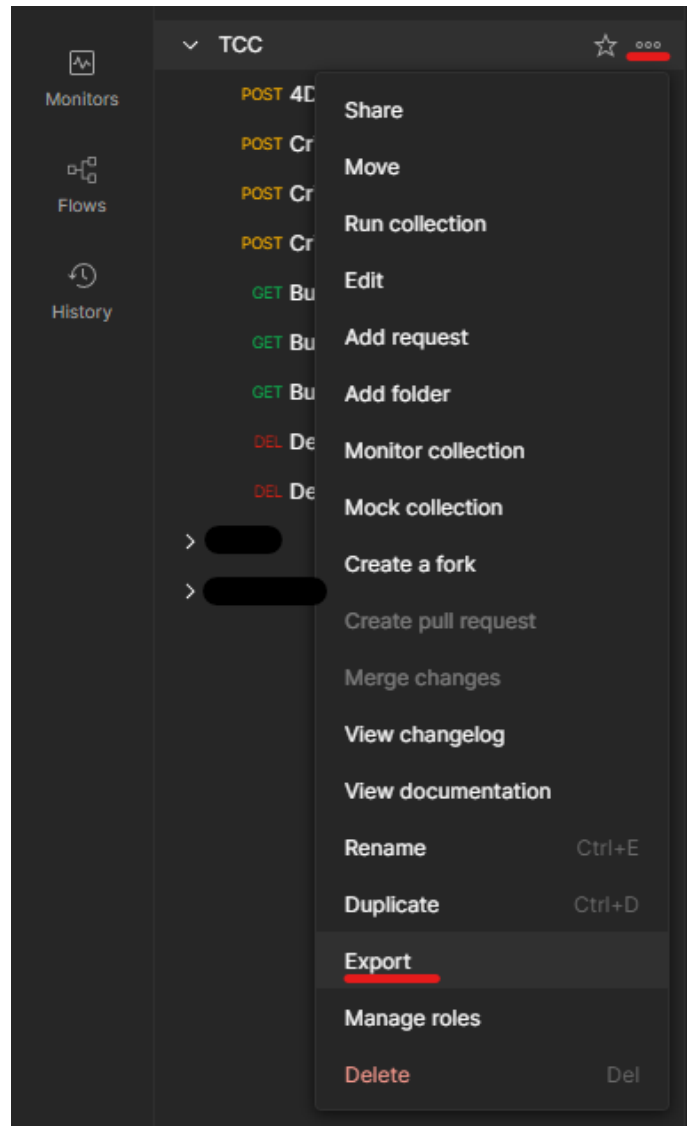
3.3.3 Executando coleções de teste no Newman

Com nossa coleção criada, com todos os dados, requisições e testes, vamos seguir o processo utilizando a ferramenta Newman, que é de grande importância para executarmos os testes criados no Postman em integração continua.

Considerando que já estamos com o Newman instalado, vamos fazer o processo de exportar a nossa coleção e o nosso ambiente para uma pasta e lá executarmos o Newman com a plataforma Visual Studio Code, como demonstrado nas seguintes imagens.

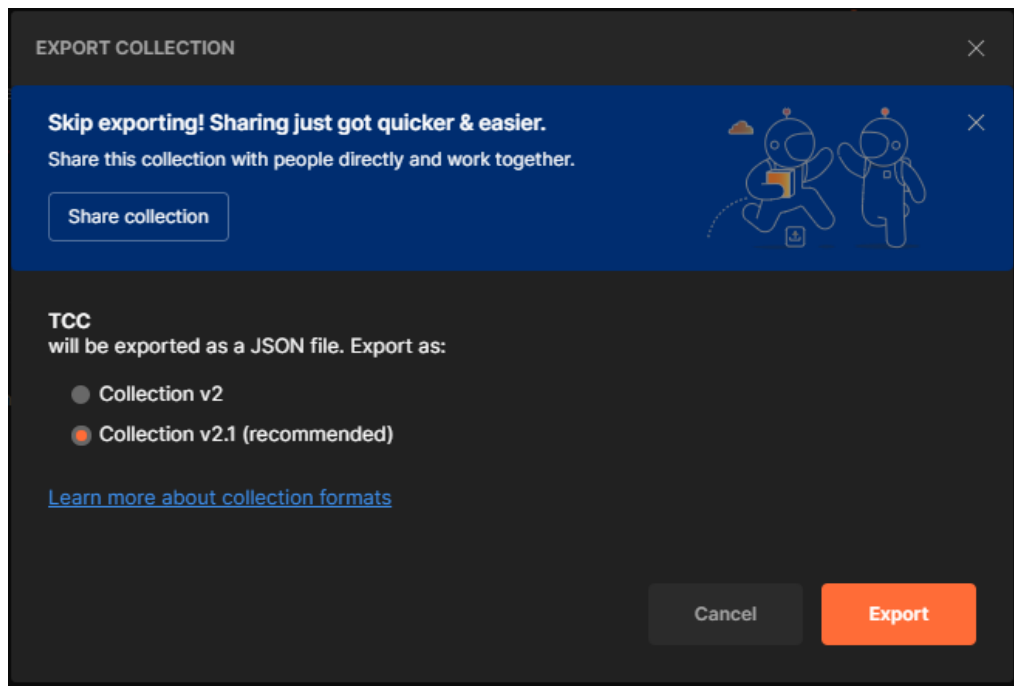
Primeiro devemos exportar a coleção, clicamos na coleção e em seguida nos três pontos, como marcado em vermelho na Figura 24, e depois na opção exportar:

Figura 24 – Exportando coleção para executar no Newman



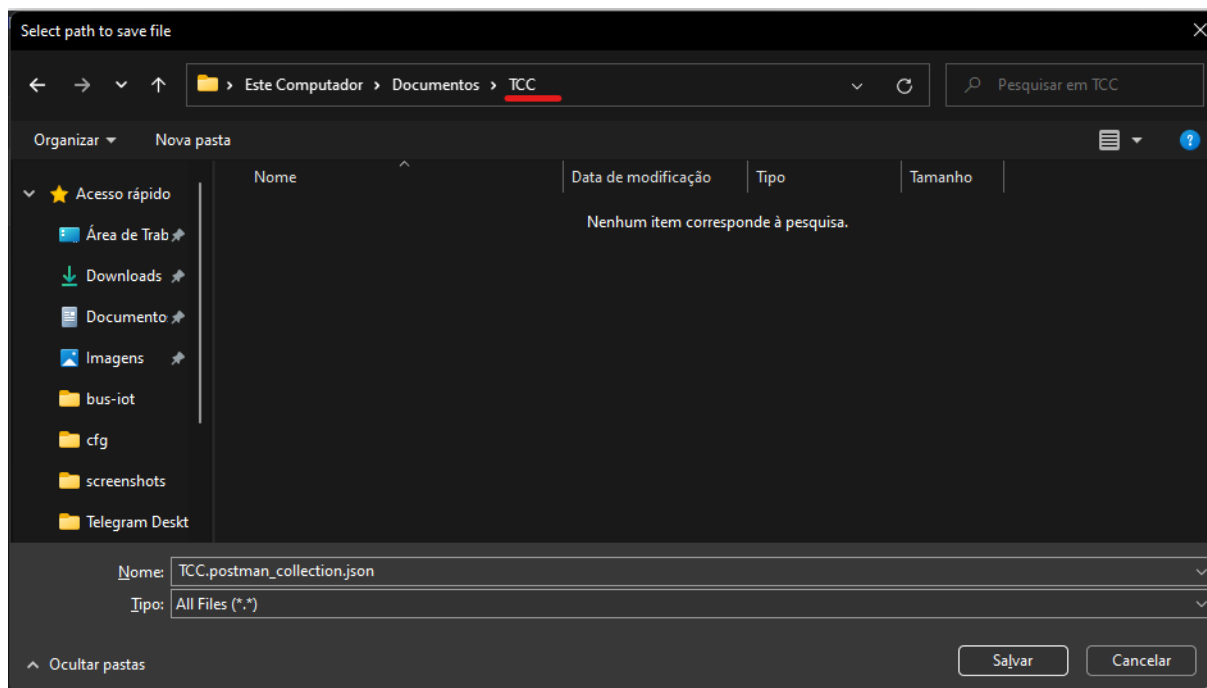
Fonte: Próprio autor.

Ao clicarmos em exportar, temos a seguinte tela, deixamos a opção recomendada e clicamos e exportar novamente:

Figura 25 – Executando exportação

Fonte: Próprio autor.

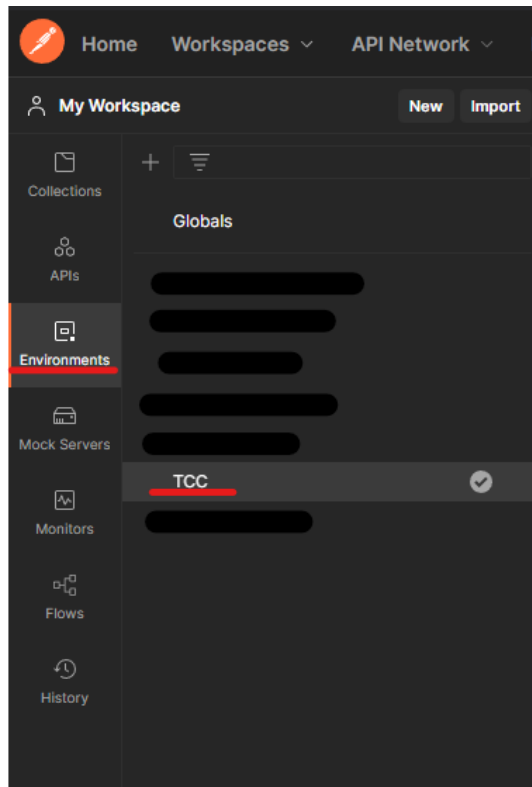
Então o Postman irá abrir a tela de diretórios do sistema operacional, nesse meu exemplo é o Windows, e cliço em salvar.

Figura 26 – Salvando coleção em um diretório

Fonte: Próprio autor.

Salvamos a coleção, agora devemos fazer o mesmo com o ambiente, precisaremos dele para a execução dos testes no Newman. A exportação do ambiente é diferente da coleção, devemos seguir esses passos:

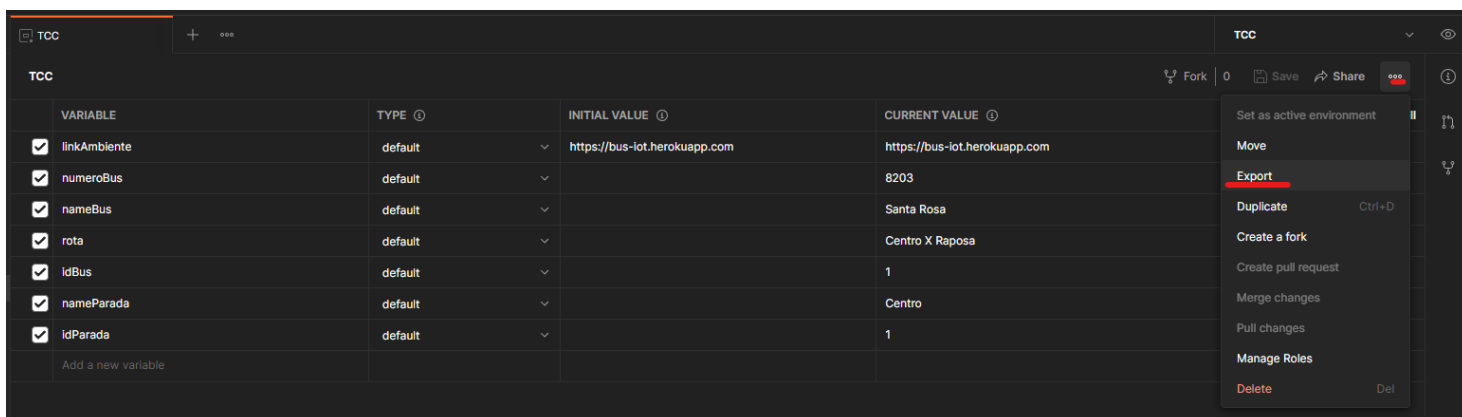
Figura 27 - Exportando ambiente



Fonte: Próprio autor.

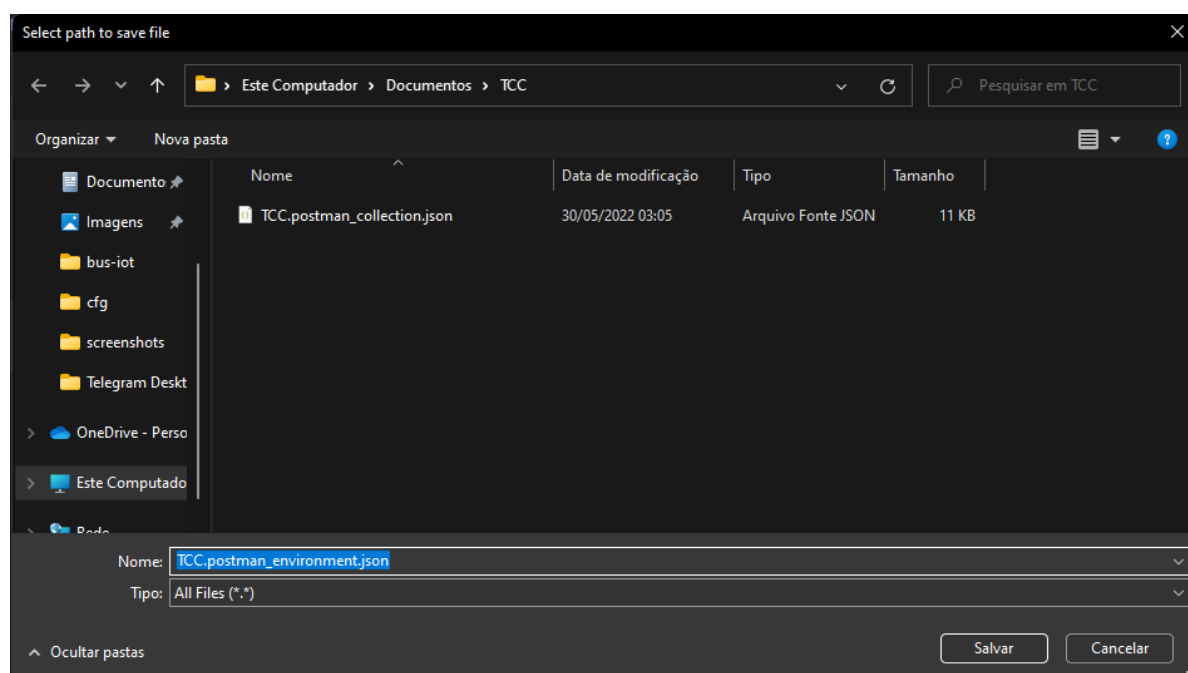
Depois irá abrir a tela de ambientes, e clicamos nos três pontos e depois em exportar:

Figura 28 – Exportando ambiente 2



Fonte: Próprio autor.

Selecionamos a pasta e salvamos:

Figura 29 - Seleccionando diretório

Fonte: Próprio autor.

Com isso concluímos a primeira parte da execução da coleção de teste no Newman, agora vem a parte legal, executar essa coleção em linha de comando, utilizando o terminal, nesse nosso exemplo vou utilizar o Visual Studio Code, quando executamos o processo de teste no Newman, ele já faz um relatório padrão no terminal, como demonstrado na Figura 30:

Figura 30 – Relatório do teste Newman

```

PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL

PS C:\Users\Muriel Magno\Documents\TCC> newman run TCC.postman_collection.json -e TCC.postman_environment.json
newman

TCC

→ 4DEVS - Gerar numero do Onibus
  POST https://www.4devs.com.br/ferramentas_online.php [200 OK, 521B, 713ms]
  ✓ Seta numero do onibus na variavel

→ Criar Onibus
  POST https://bus-iot.herokuapp.com/bus [201 Created, 338B, 800ms]
  ✓ O código do status deve ser 201
  ✓ Seta id do onibus na variavel
  ✓ Tempo de resposta aceitavel

→ Criar parada
  POST https://bus-iot.herokuapp.com/busstop [201 Created, 367B, 163ms]
  ✓ Seta id da parada na variavel

→ Criar checkstop
  POST https://bus-iot.herokuapp.com/check [201 Created, 341B, 161ms]
  ✓ O código do status deve ser 201

→ Buscar lista de paradas
  GET https://bus-iot.herokuapp.com/busstop [200 OK, 364B, 152ms]
  ✓ O código do status deve ser 200
  ✓ Tempo de resposta aceitavel

→ Buscar lista de onibus
  GET https://bus-iot.herokuapp.com/bus [200 OK, 335B, 152ms]
  ✓ O código do status deve ser 200
  ✓ Tempo de resposta aceitavel

→ Buscar onibus pelo ID
  GET https://bus-iot.herokuapp.com/bus/5 [200 OK, 333B, 156ms]
  ✓ O código do status deve ser 200
  ✓ Tempo de resposta aceitavel

→ Deletar Onibus
  DELETE https://bus-iot.herokuapp.com/bus/5 [204 No Content, 228B, 157ms]
  ✓ O código do status deve ser 204

→ Deletar parada de onibus
  DELETE https://bus-iot.herokuapp.com/busstop/5 [204 No Content, 228B, 153ms]
  ✓ O código do status deve ser 204



|                                                                    | executed | failed |
|--------------------------------------------------------------------|----------|--------|
| iterations                                                         | 1        | 0      |
| requests                                                           | 9        | 0      |
| test-scripts                                                       | 9        | 0      |
| prerequest-scripts                                                 | 2        | 0      |
| assertions                                                         | 16       | 0      |
| total run duration: 3.3s                                           |          |        |
| total data received: 575B (approx)                                 |          |        |
| average response time: 289ms [min: 152ms, max: 800ms, s.d.: 250ms] |          |        |


```

Fonte: Próprio autor.

3.4 Considerações finais

A execução do teste de regressão na aplicação foi realizada utilizando os métodos HTTP de GET, POST e DELETE. Nessa aplicação não temos o PUT ou PATH, mas caso exista na aplicação a ser testada, se faz necessário que o teste de regressão tenha cobertura total da aplicação.

4 RESULTADO E DISCUSSÃO

A utilização dessas ferramentas trouxe bastante agilidade quando falamos de testes de API, fizemos o processo de teste completo e obtivemos os seguintes resultados, onde foi dividido em tópicos para melhor entendimento.

Sempre importante frisar que o teste de regressão é um teste caro e de muito esforço do time de desenvolvimento de teste, já que é necessário validar todos os cenários, as rotas e demais necessidades da aplicação, e é recomendado esse teste para aplicações críticas e que sofrem muitas alterações no dia a dia do desenvolvimento.

4.1 Teste com o Postman

Vimos no desenvolvimento que o Postman pode executar o teste automatizado de forma fácil e tranquila, utilizando a ferramenta de execução de coleções, mas ele não possui integração contínua, isso que faz toda a diferença quando o assunto é teste automatizado.

O Postman não possui um relatório, ele traz um retorno simples do que foi executado.

4.2 Teste com o Newman

No desenvolvimento, utilizamos o Newman, lembrando que ele precisa ser instalado, e utiliza o Node.JS, que é uma biblioteca da linguagem de programação JavaScript. Depois que criamos todo o fluxo de teste utilizando o Postman, fizemos a execução por linha de comando, como é necessário para executar o teste na integração contínua.

O Newman possui um pequeno relatório de execução, e ele permite a utilização de outras bibliotecas para gerar relatórios mais elaborados, integrado com o teste e por linha de comando.

5 CONCLUSÃO FINAL

A utilização do teste de regressão é muito eficaz em grandes aplicações com criticidade alta e com modificações cotidianas, onde seu objetivo é validar a cobertura total da aplicação e evitar que erros sejam replicados em produção sem o conhecimento do time de desenvolvimento.

Utilizando o teste de regressão com Newman, a integração com a pipeline é fácil e sem alto esforço, tornando mais fácil o gerenciamento da integração contínua e implantação contínua.

Por fim, o teste de regressão é bem cansativo para ser feito, mas quando estamos falando de qualidade, o trabalho nunca para, temos que buscar o nível máximo de qualidade para os usuários e assim aumentando a credibilidade dos nossos serviços, e evitando que um erro esteja em produção, já que sua resolução quando está no ambiente de produção é muito custoso para a empresa.

5.1 Trabalhos futuros

Para próximos trabalhos, vou efetuar o teste de regressão em uma aplicação maior, com integração com o Azure Devops, que é uma aplicação de desenvolvimento da Microsoft.

REFERÊNCIAS

BARTIÉ, Alexandre. *Garantia da qualidade de software*. Gulf Professional Publishing, 2002.

Boehm, B., and V. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer*, vol. 34, no. 1, January 2001, pp. 135–137.

Documentação Postman. Disponível em: <https://learning.postman.com/docs/>

Documentação Newman. Disponível em: <https://www.npmjs.com/package/newman>

GOUVEIA, Cidinha Castro. **Teste de Integração para Sistemas Baseados em Componentes**. 2004. Orientador: Patrícia Duarte de Lima Machado. Dissertação de Mestrado, Universidade Federal de Campina Grande, Campina Grande, Paraíba, fevereiro de 2004. Disponível em: <http://dspace.sti.ufcg.edu.br:8080/xmlui/handle/riufcg/10582>. Acessado em 01 de março de 2022.

ISO/IEC 25010:2011(en): Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Disponível em: <https://www.iso.org/standard/35733.html>. Acessado em 01 de março de 2022.

NIELSEN, Jakob. **The Myth of the Genius Designer**. 2007. Disponível em <https://www.nngroup.com/articles/the-myth-of-the-genius-designer/>. Acessado em 01 março 2022.

P. Bourque and R.E. Fairley, eds., **Guide to the Software Engineering Body of Knowledge**, Version 3.0, IEEE Computer Society, 2014; www.swebok.org.

PRESSMAN, R. S. **Software Engineering: A Practitioner's Approach**. 7. ed. New York: McGraw-Hill Education, 2010, ISBN 9780073375977.

RIOS, Emerson; MOREIRA, Trayahú. **Teste de software**. Alta Books Editora, 2006.

RUBIN, Jeffrey. **Handbook of Usability Testing: How to Plan, Design and Conduct Effective Tests**. 2. Ed. Indiana: Wiley Publishing, Inc., 2008. ISBN: 978-0-470-18548-3

SOMMERVILLE, Ian. **Engenharia de Software**. 9. Ed. São Paulo: Pearson Prentice Hall, 2011. ISBN 978-85-7936-108-1.

Tricentis. **SOFTWARE FAILURES: \$1.1 Trillion Impacted by Software Defects: A Testing Fail?** Fevereiro, 2017. Disponível em <https://www.tricentis.com/blog/1-1-trillion-in-assets-impacted-by-software-defects-a-software-testing-fail/>. Acessado em 01 de junho de 2022.