UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE
CENTRO DE CIÊNCIA E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ RENAN ALVES PEREIRA

USO DE TESTES EXPLORATÓRIO E DE INTEGRAÇÃO EM UMA PLATAFORMA MICROSERVIÇOS

CAMPINA GRANDE
2025

UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE
CENTRO DE CIÊNCIA E TECNOLOGIA

JOSÉ RENAN ALVES PEREIRA

USO DE TESTES EXPLORATÓRIO E DE INTEGRAÇÃO EM UMA PLATAFORMA MICROSERVIÇOS

Relatório de Estágio apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Computação.

Orientador: Prof. Ma. Ana Isabella Muniz Leite.

CAMPINA GRANDE
2025

P436u Pereira, Jose Renan Alves.

Uso de testes exploratório e de integração em uma plataforma microserviços [manuscrito] / Jose Renan Alves Pereira. - 2025.

41 p.

Digitado. Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia, 2025. "Orientação : Prof. Me. Ana Isabella Muniz Leite, Coordenação do Curso de Computação - CCT. "

1. Suíte de testes. 2. Teste de integração. 3. Testes exploratórios. 4. Microserviços. I. Título

21. ed. CDD 005.3

Elaborada por Bruno R. F. de Lima - CRB - 15/1021 BC/UEPB

JOSE RENAN ALVES PEREIRA

USO DE TESTES EXPLORATÓRIO E DE INTEGRAÇÃO EM UMA PLATAFORMA MICROSERVIÇOS

Relatório de Estágio apresentado à Coordenação do Curso de Ciência da Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de Bacharel em Computação

Aprovada em: 13/02/2024.

BANCA EXAMINADORA

Documento assinado eletronicamente por:

- **Ana Isabella Muniz Leite** (***.834.864-**), em **19/02/2025 09:38:07** com chave **5edae4caeebe11ef8dfb1a1c3150b54b**.
- **Fábio Luiz Leite Júnior** (***.848.564-**), em **19/02/2025 09:45:00** com chave **54f03572eebf11ef8b5d1a7cc27eb1f9**.
- **Wellington Candeia de Araujo** (***.655.074-**), em **19/02/2025 14:41:59** com chave **d1c75a66eee811ef82462618257239a1**.

Documento emitido pelo SUAP. Para comprovar sua autenticidade, faça a leitura do QrCode ao lado ou acesse https://suap.uepb.edu.br/comum/autenticar_documento/ e informe os dados a seguir.
Tipo de Documento: Folha de Aprovação do Projeto Final
Data da Emissão: 20/02/2025
Código de Autenticação: 005bf5

Aos meus pais Domingos e Aparecida, por toda dedicação e esforço para que eu chegasse até aqui, DEDICO.

AGRADECIMENTOS

À Deus por sempre estar comigo nessa trajetória me guiando e sendo meu alicerce em todos os momentos, me ajudando a passar por várias tribulações durante essa jornada. E também a Nossa Senhora de Fátima, padroeira do sítio ao qual morei por muito tempo e que é meu local de conforto e encontro comigo mesmo, obrigado por sempre interceder a Jesus por mim.

Ao meu pai e mãe, por sempre estarem comigo e serem minha base e inspiração, vocês são a razão de tudo na minha vida e eu os amo incondicionalmente. Vocês me sustentam e sempre me deram ânimo nos momentos mais difíceis. Com vocês aprendi coisas imensuráveis, sou eternamente grato por tudo isso. Vocês são minha vida.

Aos meus irmãos, por sempre me ajudarem e estarem presentes em momentos de dificuldade, amo vocês incondicionalmente.

“Eu acredito que às vezes são as pessoas que ninguém espera nada que fazem as coisas que ninguém consegue imaginar.” Alan Turing

RESUMO

No cenário atual do desenvolvimento de software, há uma tendência clara para a ampla adoção de sistemas baseados em microsserviços, especialmente aqueles ligados à computação em nuvem. Embora essa abordagem ofereça benefícios em termos de escalabilidade e flexibilidade, também apresenta desafios significativos relacionados à segurança, conformidade, confiabilidade e qualidade de serviço. Uma maneira de manter esses princípios é por meio da implementação de testes automatizados. No entanto, devido à complexidade inerente da arquitetura de microsserviços e à necessidade de que todos os serviços coexistam e persistam simultaneamente, extrair o comportamento do sistema torna-se uma tarefa desafiadora. Este relatório explorará os desafios e benefícios associados ao desenvolvimento de uma suíte de testes voltada sistemas baseados em microsserviços. Especificamente, no contexto do projeto de Saúde Animal, cujo objetivo é centralizar todas as atividades relacionadas ao bem-estar animal, tais como o cadastro de ONGs, cadastro de animais, bem como o cadastro de castração e adoção.

Palavras-Chave: Metodologias ágeis, Teste de integração, Teste exploratório.

ABSTRACT

In the current software development landscape, there is a clear trend toward the widespread adoption of microservices-based systems, especially those related to cloud computing. While this approach offers benefits in terms of scalability and flexibility, it also presents significant challenges related to security, compliance, reliability, and quality of service. One way to uphold these principles is through the implementation of automated tests. However, extracting the system's behavior becomes a challenging task due to the inherent complexity of microservices architecture and the need for all services to coexist and persist simultaneously. This report will explore the challenges and benefits associated with developing a test suite for microservices-based systems. Specifically, it will focus on the context of the Animal Health project, which aims to centralize all activities related to animal welfare, such as the registration of NGOs, and animals, as well as castration and adoption records.

Keywords: Agile methodologies, Integration testing, Exploratory integration.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicativos
HTTP	Hypertext Transfer Protocol
NUTES	Núcleo de Tecnologias Estratégicas em Saúde
QA	Quality Assurance
REGPET	Sistema universal do Estado da Paraíba que centraliza as ações relacionadas à causa animal
UEPB	Universidade Estadual da Paraíba

SUMÁRIO

1 INTRODUÇÃO

Uma das partes fundamentais no desenvolvimento de software é garantir sua qualidade e a exatidão dos resultados, segundo o IEEE (2014). Uma das formas de adquirir as informações necessárias sobre a qualidade do produto é la engenharia de testes (AZEVÊDO; SILVA, 2015). Há diversos tipos de testes como: unidade, integração, funcional, aceitação, dentre outros. Alinhado a isso, ferramentas que ajudam a executar alguns tipos de teste, como Selenium, [1]TestLink[2], TestComplete [3].

Como exibidos e detalhados por Sneha e Malle (2017), os testes automatizados, em contexto de mudanças e da necessidade de entrega rápida de incrementos do software, surge como uma alternativa para garantir a confiabilidade e qualidade do software com detecção rápida de defeitos que podem surgir durante os ciclos de desenvolvimento. A automação permite a execução de baterias de teste de forma automatizada e rápida por meio de ferramentas e scripts (ANICHE, 2015). Segundo o Digital Business Assurance (2024), em complemento com esse cenário, os testes exploratórios ajudam a detectar problemas que geralmente não são identificados com testes automatizados, como questões de usabilidade e experiência do usuário, permitindo uma exploração mais profunda com grande flexibilidade para se adaptar a mudanças.

Apesar desses benefícios, a implementação deste em projetos reais é algo mais complexo e que deve ser analisado com cuidado. São muitos os desafios que surgem durante todas as fases de automação de testes. Xu e Hoje (2023) refletem esses desafios de testes em metodologias ágeis por não haver uma etapa separada apenas para essa fase, tornando difícil criar e executar testes abrangentes no intervalo de tempo definido, manter casos de teste, dados documentos e outros artefatos atualizados.

Diante disso, este relatório tem como objetivo descrever minha participação no projeto REGPET, destacando minha atuação no planejamento e na execução de testes exploratórios e de integração. O projeto resulta de uma parceria entre o Governo do Estado da Paraíba e o Núcleo de Tecnologias Estratégicas em Saúde da UEPB, onde atuei como bolsista na equipe de qualidade de software.

[1] https://www.selenium.dev/pt-br/documentation/ide/
[2] https://testlink.org/
[3] https://smartbear.com/product/testcomplete/

Este trabalho está organizado em cinco capítulos. O primeiro capítulo, dedicado à fundamentação teórica, apresenta os conceitos essenciais para o entendimento do estudo. O segundo capítulo explora o contexto do projeto, seus requisitos e as regulamentações legais dentro da política de saúde do Estado da Paraíba. No terceiro capítulo, são descritas a metodologia adotada para a realização das tarefas e minhas principais atividades. O quarto capítulo detalha a execução dessas atividades. Por fim, o quinto capítulo apresenta a conclusão, resumindo como o objetivo foi alcançado e as principais lições aprendidas.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Engenharia De Software

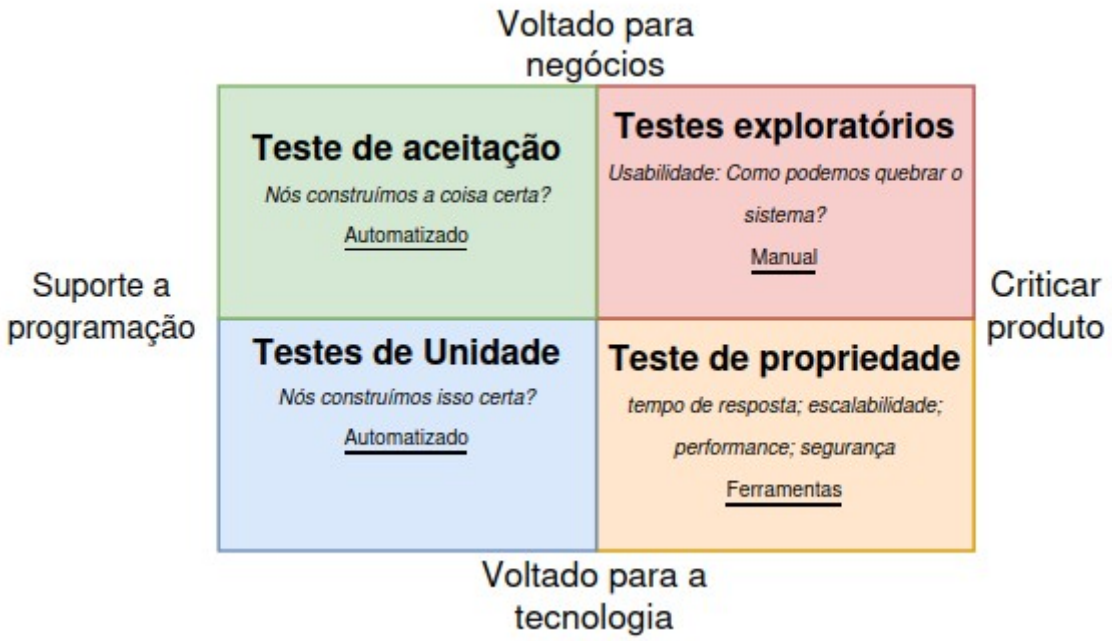
Pressman e Maxim (2021, p. 312) definem Qualidade de Software como um processo capaz de criar um produto útil que oferece um valor mensurável para quem o produz e consome. Em uma definição mais abrangente, Bartié (2002) conceitua o termo como “um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos” (BARTIÉ, 2002, p.16). A Garantia de Qualidade (QA) é definida por Pressman e Maxim (2021, p. 323) como a infraestrutura que suporta um conjunto de funções e relatórios que avaliam a integridade de diversas ações e serviços, de acordo com métodos da engenharia de software. Dessa forma, fornece dados sobre a qualidade do produto, tornando-a mensurável. Complementando essa visão, Lewis (2004) descreve a QA como “o conjunto de atividades de apoio para fornecer confiança de que os processos estão estabelecidos e continuamente aprimorados para produzir produtos que atendam às especificações e sejam adequados ao uso pretendido”. Diante dessas definições, pode-se concluir que ambos os conceitos estão interligados. A qualidade de software é responsável por definir atributos e métricas de qualidade, como as propostas por McCall e Cavano (1978) ou pela ISO 9126-1, enquanto a garantia de qualidade atua no monitoramento e na melhoria contínua dos processos, assegurando o cumprimento dos requisitos estabelecidos. Segundo Kokol (2022), há uma ampla quantidade de pesquisas na área, mas essas pesquisas precisam ser moldadas e adaptadas às subdisciplinas da área e aos domínios de aplicação.

2.2 Teste De Software

O IEEE Standard 729 de 2017 define teste de software como um “processo de avaliar um software ou um componente de software para verificar se ele satisfaz os requisitos especificados ou identificar diferenças entre resultados esperados e obtidos”(IEEE Standard 729, 2017). Os testes em metodologias ágeis podem variar em relação a metodologias tradicionais, como Newman (2021)

discorre sobre as diferentes vertentes de teste, conforme a Figura 1, cada uma com seu escopo, fase e tipos de testes executados. A parte de voltada para tecnologia inclui testes que auxiliam os desenvolvedores a criarem o software em um estágio inicial, enquanto a parte de voltada para negócios ajuda os *stakeholders* a entenderem como o sistema funciona. Esses quadrantes auxiliam na compreensão de como os testes ágeis são realizados e do que eles buscam responder. Os testes exploratórios possuem um quadrante específico, realizado de forma manual. Esse quadrante se localiza entre o escopo voltado para negócios, voltado para a validação com ênfase no usuário final, e o de criticar o produto, visando explorar limites do sistema.

Figura 1 - Quadrante de teste para testes ágeis



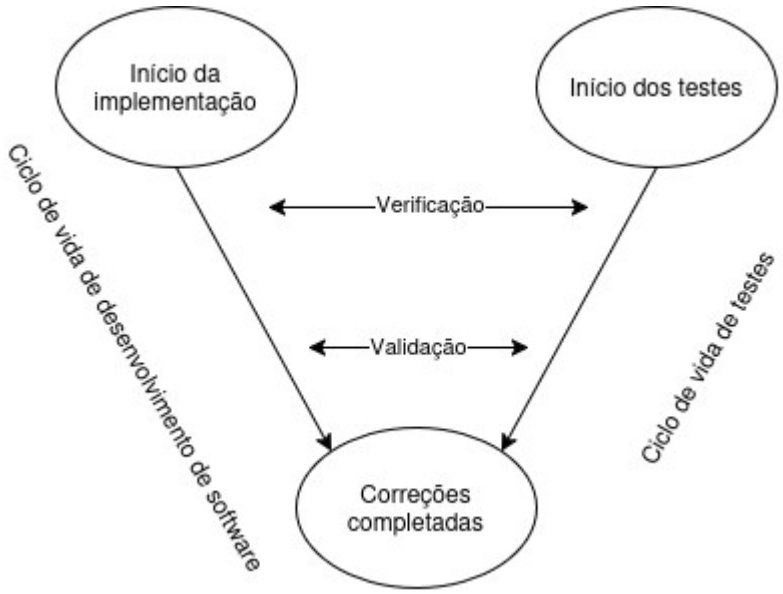
Fonte: Marick apud CRISPIN; GREGORY, 2009.

Dessa forma, os testes de integração há duas possibilidades de quadrantes e eles podem variar de acordo com o escopo dos testes. Se o objetivo é validar a interação entre diferentes módulos do sistema em um nível técnico, podemos encaixá-los entre o lado tecnológico e de suporte a programação, no quadrante de testes unitários. Se o foco é garantir que os módulos trabalham juntos para que atendam aos requisitos de negócios, o quadrante muda para testes de aceitação, entre o suporte a programação e o lado de negócios.

O quadrante de testes unitários está relacionado com a etapa de verificação de software. Segundo o site LucidChart, a verificação é o processo responsável por analisar se o sistema corresponde às expectativas de um ponto de vista técnico. O quadrante de testes de aceitação está

mais relacionado a etapa de validação de software, que segundo o site LucidChart é etapa de avaliação se a funcionalidade construída atende os requisitos do usuário. Como podemos ver na Figura 2, o modelo V[4] possui dois lados que exibe graficamente a relação de ambos esses conceitos com as etapas de implementação e teste. No final, ambas as fases devem convergir para a funcionalidade do produto corrigida e implementada.

Figura 2 - Exemplo de modelo V

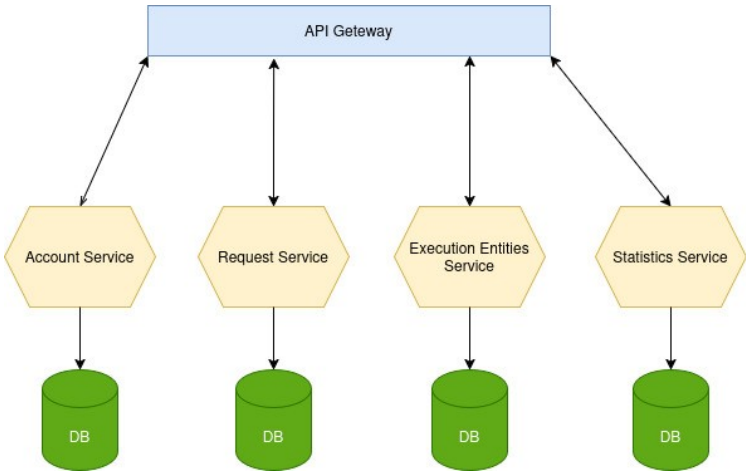


Fonte: Base de conhecimento em testes de software, 2007, pg. 41.

A Figura 3 apresenta os quatro serviços que compõem a arquitetura de microserviços que o REGPET foi construído. Cada serviço possui seu próprio banco de dados e suas funções específicas se comunicando com a API Geteway. Detalhes mais específicos não são necessários para o entendimento de como o processo de testes eram realizados.

[4] https://www.lucidchart.com/blog/pt/v-model-verificacao-e-validacao

Figura 3 - fluxograma dos serviços contidos na arquitetura.



Fonte: elaborado pelo autor, 2025.

2.3 Teste De Integração

Os testes de integração são um dos tipos de testes que podem ser utilizados para identificar defeitos no sistema. Existe uma controvérsia em relação ao nome, visto que eles também podem ser chamados de testes end-to-end, como afirma Newman (2021, p. 282), mas essa distinção é uma mera formalidade. Testes de integração são definidos por Hoplling et al. (2016) como o tipo de teste responsável por validar o comportamento entre diferentes módulos e componentes do sistema, incluindo as etapas de definição dos casos de teste, do número de componentes que serão testados e do ambiente de teste.

2.4 Teste Exploratório

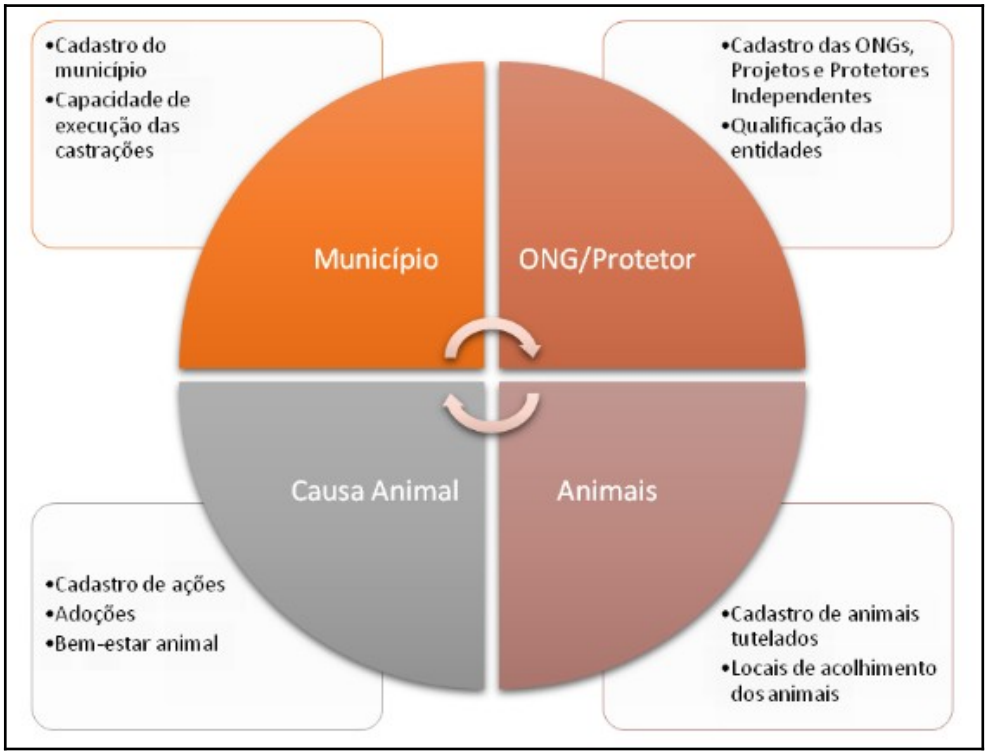
Segundo Parmar, trata-se de uma abordagem de teste de software com foco na descoberta, cujo escopo varia conforme o testador. O teste exploratório de software é, por natureza, aleatório ou não estruturado, sem definição de etapas. Analisado dessa forma, esse tipo de teste atua como um precursor da automação, auxiliando na formalização de descobertas.

Complementando, Crispin e Gregory (2009) definem como uma ferramenta investigativa, sem script e que permite que você vá além das variações que já foram testadas. Dessa forma, o aprendizado que é gerado nesses tipos de teste pode ajudar a entender o sistema de uma forma mais ampla, e para além disso, contribuir para gerar mais testes automatizados. Ou seja, os testes exploratórios combinam aprendizado, design de teste e execução dentro de uma abordagem de teste definida.

3 PROJETO REGPET

De acordo com a Resolução CIN-PB Nº 814, de 2023, do Governo do Estado da Paraíba que aprovou o Programa Estadual de Incentivo à Castração e Bem-Estar Animal, com o objetivo de incentivar e desenvolver políticas públicas da causa animal nos municípios paraibanos. Ainda de acordo com essa resolução, o REGPET é um sistema Web que visa centralizar todas as atividades relacionadas ao bem-estar animal organizadas em 4 dimensões, como apresentada na Figura 4.

Figura 4 - As 4 dimensões do REGPET

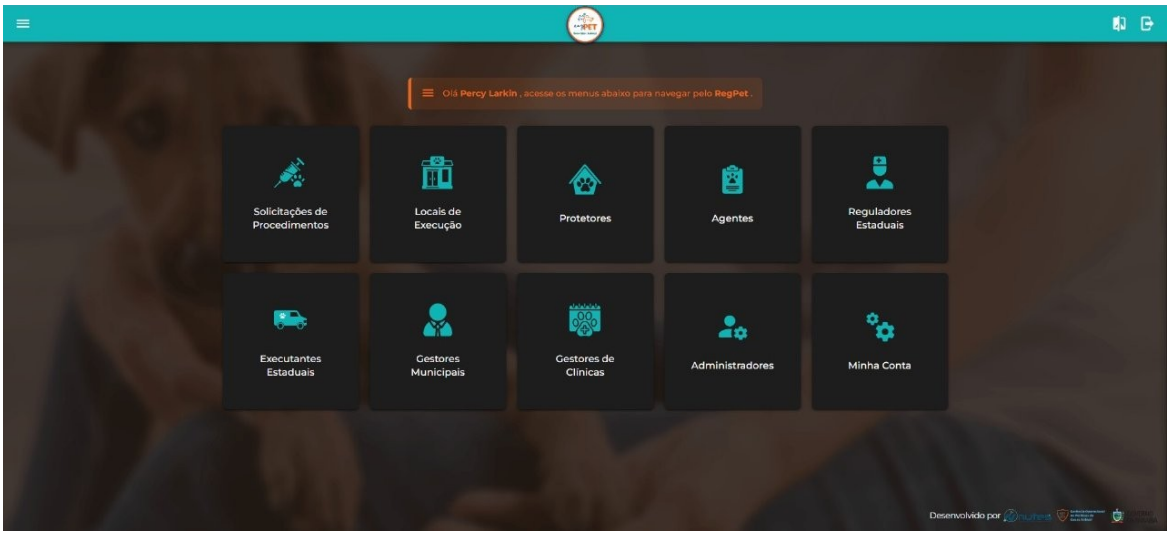


Fonte: Secretaria de Estado da Saúde, 2023

- A dimensão referente aos animais é responsável pelo registro detalhado de todos os animais que estarão sob a supervisão e tutela do município.
- A dimensão de ONGs e Protetores é responsável pelo cadastro dos dados dos responsáveis pelos animais tutelados pelo município.
- A dimensão responsável pelos municípios executa toda a gestão dos municípios de forma geral, como a capacidade de castrações.
- A dimensão referente e Causa Animal compreenderá todas as ações de monitoramento do bem-estar animal.

Além dessas, também existe a dimensão Administrativa, que não está presente na figura e é responsável pela obtenção dos relatórios sobre todo o estado da população animal do Estado da Paraíba. O REGPET está sendo desenvolvido pelo Núcleo de Tecnologias Estratégicas em Saúde (NUTES) da Universidade Estadual da Paraíba (UEPB) com o objetivo de regulamentar procedimentos da causa animal, com ação inicial voltada para procedimentos de castração em cães e gatos. As primeiras etapas de desenvolvimento do software se iniciaram em novembro de 2023, com as primeiras reuniões com os representantes do projeto perante o estado. A Figura 5 é a visão do usuário administrador ao inserir suas credenciais corretamente e acessar o sistema:

Figura 5 – Tela de login do REGPET WEB.



Fonte: Elaborado pelo autor, 2025

4 METODOLOGIA

4.1 Scrum Como Metodologia Para Gerenciamento Do Time

Em 1990, Jeff Sutherland e a sua equipe foram responsáveis por desenvolver um método de desenvolvimento ágil de software chamado Scrum[5] (PRESSMAN; MAXIM, 2021). Os princípios que regem o Scrum estão alinhados com o manifesto ágil e orientam as atividades em um processo com as seguintes atividades metodológicas: requisitos, análise, projeto, evolução e entrega (PRESSMAN; MAXIM, 2021). O Scrum foi escolhido para o projeto pela ótima adaptação da metodologia para gerenciar equipes pequenas, formando um time Scrum.

Ao longo de cada *sprint* de desenvolvimento o foco dos testes mudam, variando conforme as funcionalidades que estão sendo desenvolvidas. Dessa forma, o escopo de implementação e execução dos testes de integração e exploratório estão diretamente relacionados com isso, visto que os testes são para validar alterações e funcionalidades que estão sendo desenvolvidas ao longo das *sprints*. O escopo diz respeito às funcionalidades que serão testadas durante a *sprint*. Essas alterações que serão testadas são produzidas pelo time de desenvolvimento na *sprint* anterior e disponibilizada quando pronta na próxima *sprint*, para que possa ser testada e validada antes de ir para produção.

A definição do escopo é feita no início de cada sprint e serve como base para cadastrar as tarefas referente aos testes de software. Para isso, é realizada uma reunião de planejamento da sprint com toda a equipe para que isso possa ser esclarecido. Com isso em mãos, as tarefas ligadas a testes de integração e exploratório podem ser criadas no Azure DevOps[6] com base no que foi definido. O escopo, portanto, refere-se às funcionalidades que serão testadas durante a *sprint* em questão. As alterações a serem testadas eram desenvolvidas pela equipe de programação na *sprint* anterior e, uma vez finalizadas, eram disponibilizadas para testes na *sprint* seguinte. Isso garantia que as novas funcionalidades desenvolvidas passariam por uma bateria mínima de testes.

O Azure DevOps foi a ferramenta de apoio utilizada para gerenciar as atividades dentro do Scrum. Este software facilitou o planejamento e a execução das *sprints*, permitindo que o escopo de cada *sprint*, com duração de duas semanas, fosse claramente definido. A partir desse escopo, as tarefas foram distribuídas entre os membros da equipe, especificando as

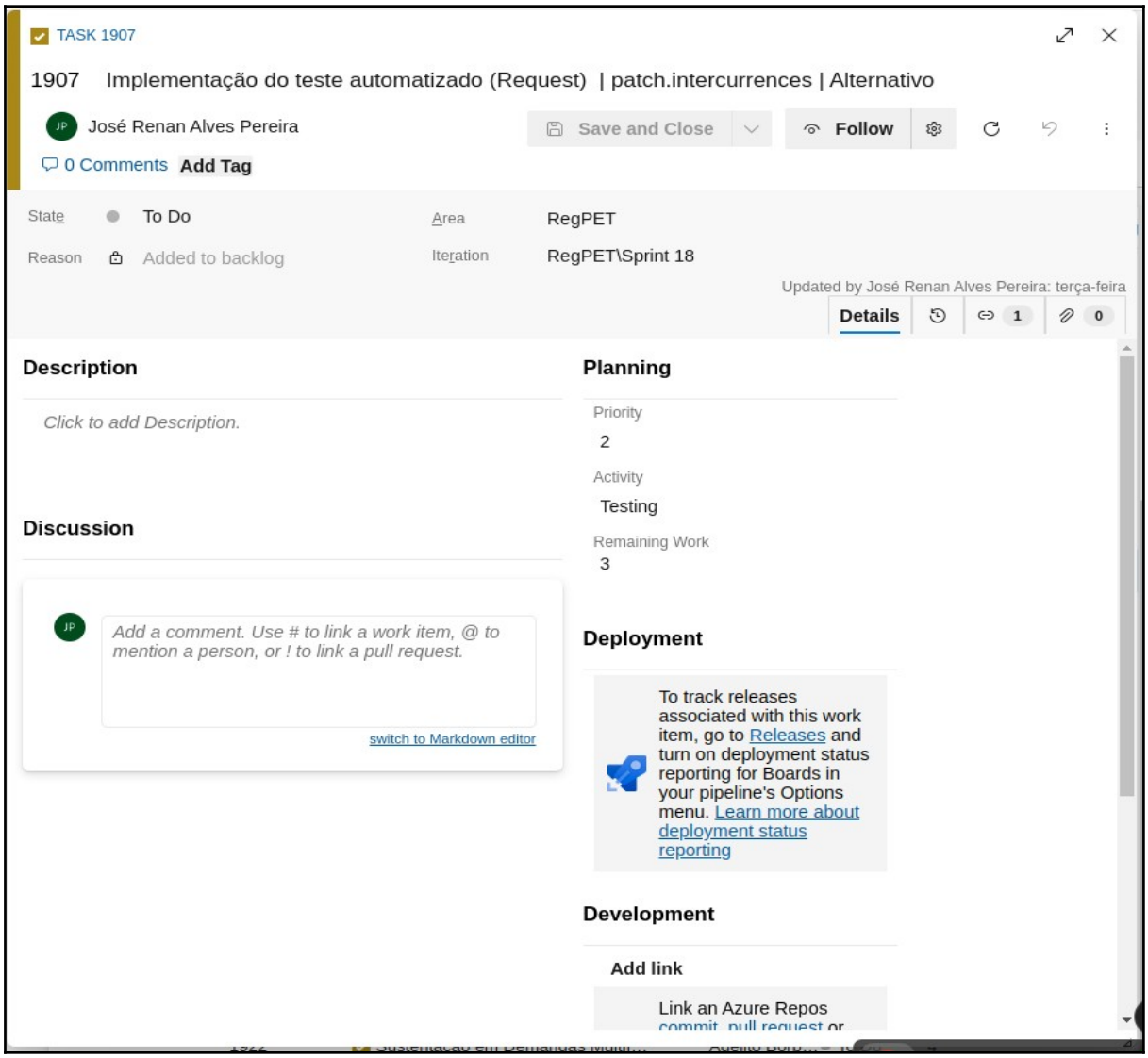
[5] https://scrumguides.org/

[6] https://azure.microsoft.com/pt-br/products/devops

responsabilidades de cada um, o tempo necessário para a conclusão e os recursos alocados.Na sequência, como mostrado na Figura 6, é possível observar um exemplo de uma tarefa atribuída a um membro da equipe, detalhando a sprint à qual ela pertencia, o responsável pela execução, as horas estimadas para a conclusão, entre outras informações relevantes. Cada tarefa no sistema era classificada em três estados distintos: *to do*, *doing* e *done*, indicando seu progresso. A tarefa começava no estado *to do*, passava para *doing* quando estava sendo executada, e, finalmente, para *don*e quando era concluída. Esse processo de acompanhamento garantiu que o andamento das tarefas fosse monitorado de forma eficiente e transparente. Como ilustrado na Figura 6, o exemplo de tarefa exibia o título da tarefa, que descrevia o módulo, a rota e o tipo de teste (se padrão ou alternativo), além do estado da tarefa e das horas necessárias para sua execução.

Quando uma tarefa era movida para o estado *to do*, as informações registradas no Azure DevOps eram usadas para criar a *branch* correspondente, onde as alterações seriam feitas. Após a conclusão da tarefa, o estado era alterado para *don*e, sinalizando que a tarefa estava finalizada.

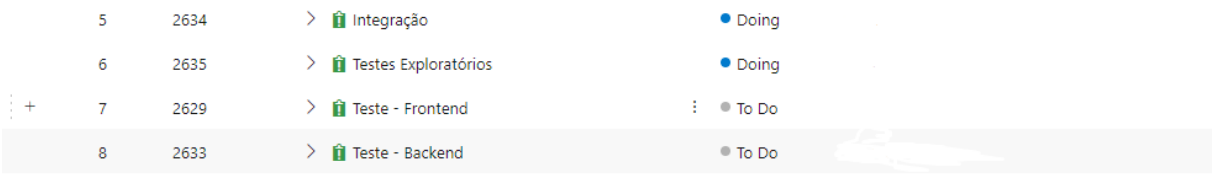
Figura 6 - Exemplo de tarefa cadastrada no Azure DevOps



Fonte: Elaborado pelo autor, 2024

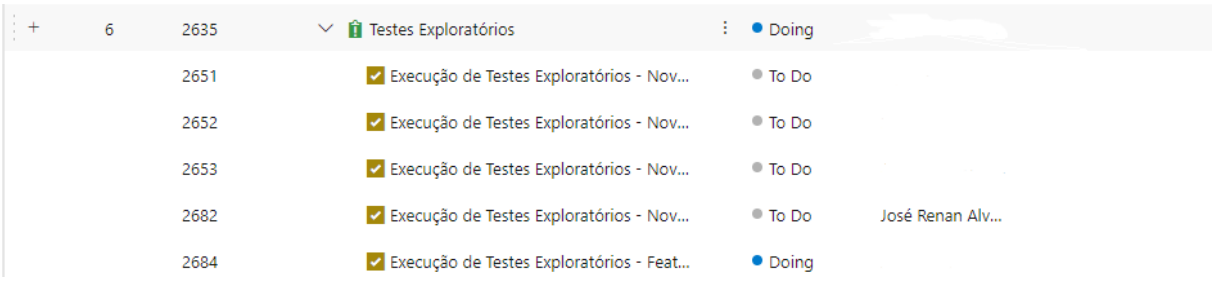
Como mostrado na Figura 7, as tarefas são armazenadas em PBIs (*Product Backlog Item*) que fragmentam as tarefas em escopos específicos de teste. O Guia do Scrum, gerenciado e mantido por Ken Schwaber e Jeff Sutherland, define as PBIs como uma lista ordenada com todas as atividades necessárias para melhorar o produto. Durante a etapa de planejamento da *sprint* e seu escopo, as atividades são detalhadas da forma que está sendo descrita e exibida na Figura 7, e com isso posicionada em seu PBI mais adequado, como exemplificado na Figura 8.

Figura 7 - PBIs dos testes de software



Fonte: Elaborado pelo autor, 2025

Figura 8 - Tarefas posicionadas no PBI referente aos testes exploratórios



Fonte: Elaborado pelo autor, 2025

Uma *sprint* tinha duração média de 2 semanas e a cada semana eram realizadas duas reuniões de acompanhamento. Essas reuniões tinham como objetivo a troca de informações entre o time acerca do estado e andamento de suas atividades, além de possíveis impedimentos e dificuldades encontradas durante a realização das tarefas. Ao final da *sprint* eram realizadas reuniões de revisão, com os desenvolvedores demonstrando o que foi desenvolvido durante a *sprint*, coleta de informações sobre o que foi implementado por parte do time e ajustes na visão do produto.

4.2 Minha Atuação

Diante deste cenário, minhas atividades estiveram relacionadas a minha participação na equipe de qualidade de software. Ao longo das sprints, participei das reuniões de planejamento, reuniões de acompanhamento e de revisão com o objetivo de discutir melhor sobre as tarefas associadas a mim com todo o time. Diante deste cenário, o Quadro 1 abaixo lista as atividades realizadas por mim que serão discutidas na seção de resultados.

Quadro 1: Lista de atividades desenvolvidas pelo autor no projeto

1	Planejamento e execução dos testes exploratórios
2	Planejamento dos cenários dos testes de integração
3	Criação e execução dos casos de teste dos cenários de integração
4	Geração de relatórios referentes aos cenários de teste
5	Reporte de comportamentos inesperados

Fonte: Elaborado pelo autor, 2025

5 RESULTADOS

5.1 Planejamento E Execução Dos Testes Exploratórios

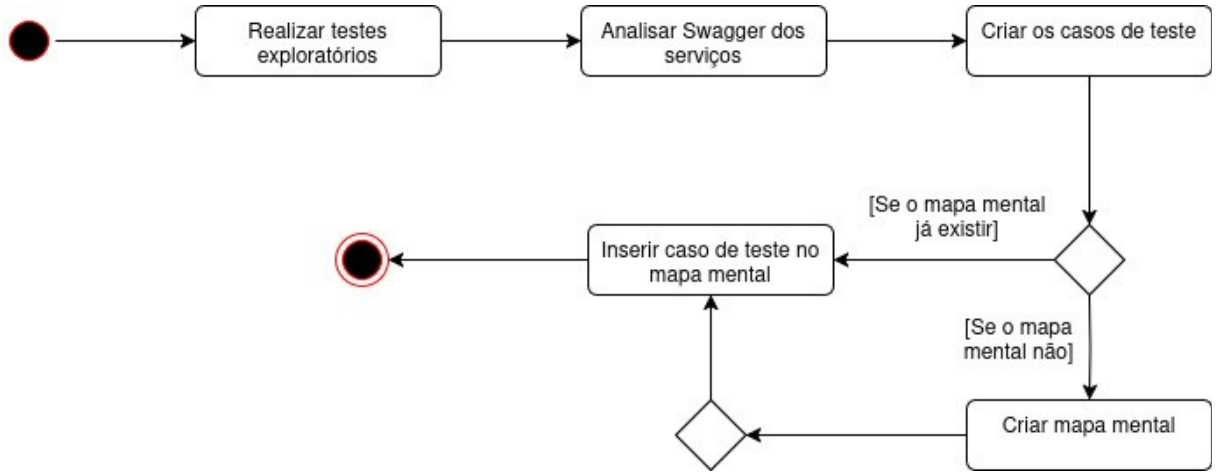
Com base no que foi definido na sprint de planejamento, as funcionalidades críticas eram primeiramente testadas utilizando os testes exploratórios. Dessa forma, a equipe de testes conseguia entender melhor as peculiaridades da implementação, que contribuia para o entendimento do comportamento do sistema. Além de identificar comportamentos suspeitos que deviam ser investigados por toda a equipe de desenvolvimento, esses tipos de teste também eram utilizados como uma das formas de planejar cenários para os testes de integração que seriam executados posteriormente.

A execução desses testes consistia em utilizar o REGPET simulando o comportamento dos usuários finais do sistema. As funcionalidades eram testadas utilizando a interface do software que os utilizadores tinham acesso no seu dia a dia, e com isso, eram realizadas as tarefas que incluíram as novas funcionalidades implementadas e que estavam planejadas para serem exercitadas no escopo da sprint. Com isso, podemos notar a contribuição que Crispin e Gregory (2009) afirmam ser possível ao identificar que os testes exploratórios contribuem para gerar conhecimento e contribuir para a criação de mais casos de testes automatizados, que nesse caso, são de integração.

5.2 Planejamento Dos Cenários Dos Testes De Integração

A sequência de planejamento dos testes de integração está detalhada no diagrama de sequência da Figura 9. Ela revela o fluxo padrão em que esses casos de testes eram mapeados, e ao longo deste tópico, a forma e as ferramentas utilizadas serão detalhadas para melhor compreensão do diagrama e de cada atividade presente nele.

Figura 9 - Diagrama da sequência de planejamento dos testes de integração



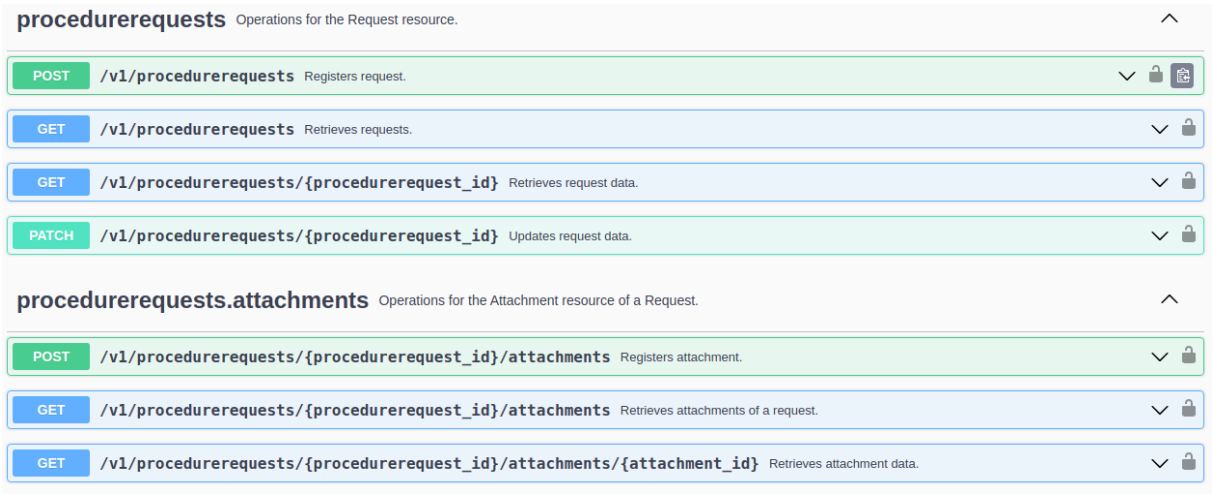
Fonte: Elaborado pelo autor, 2025

Aliado ao conhecimento gerado com os testes exploratórios, o planejamento final dos cenários de teste foi realizado com o auxílio do Swagger[7], uma ferramenta de documentação que especifica detalhadamente as rotas da API, conforme mostrado na Figura 10. Essa documentação incluiu informações sobre códigos de resposta, mensagens de erro, e os corpos das requisições que podem ser feitas para uma determinada rota de cada serviço. Com base nessas especificações das rotas da API, foram mapeados tanto os casos de teste padrão quanto os alternativos. Os casos de teste representam cenários nos quais os testes foram conduzidos, visando validar se os resultados esperados correspondiam aos recebidos.

Nos casos de teste padrão, o objetivo era garantir que, ao seguir todos os passos de acordo com as regras estabelecidas, a solicitação fosse processada corretamente, resultando na resposta esperada da API. Já nos casos de teste alternativos, buscava-se validar a capacidade da API de retornar uma mensagem de erro adequada, caso algum passo fosse executado de maneira incorreta na solicitação de um recurso.

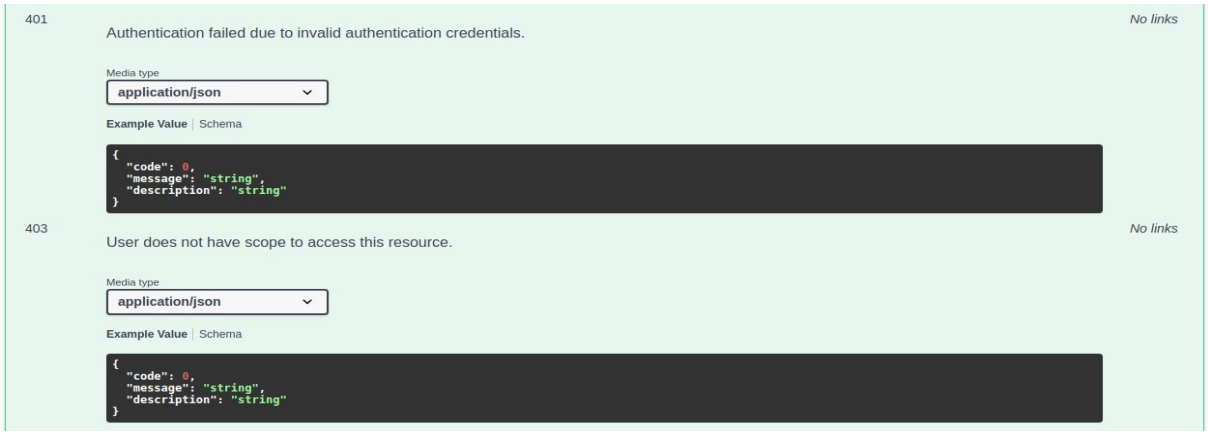
[7] https://swagger.io/

Figura 10 - Exemplo de documentação presente no swagger



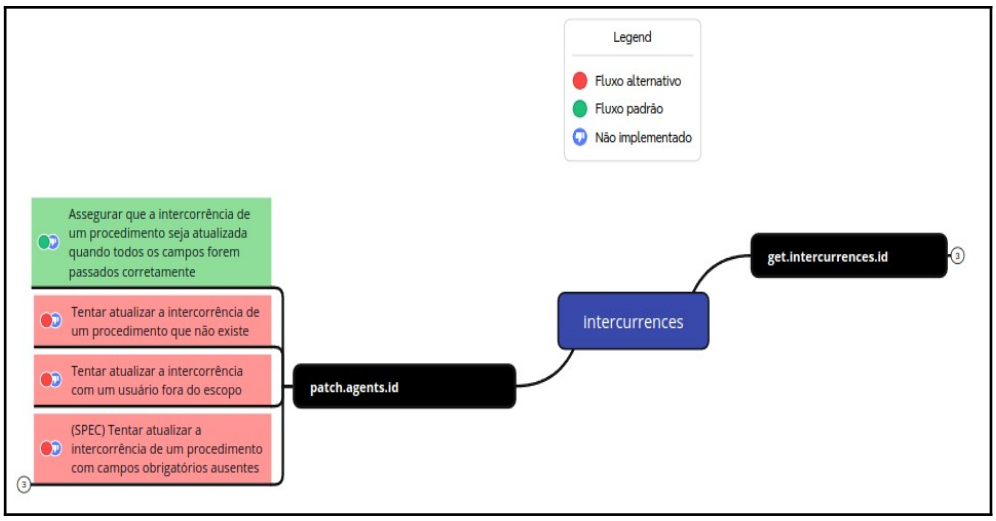
Fonte: Elaborado pelo autor, 2025

Figura 11 - Detalhamento de respostas de uma rota de API detalhada usando o Swagger



Fonte: Elaborado pelo autor, 2025

Figura 12 - Mapa mental: casos de testes padrão e alternativo para uma rota de API

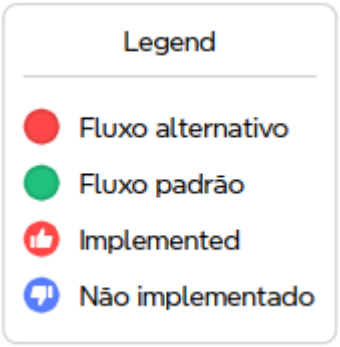


Fonte: Elaborado pelo autor, 2025

Com base nos códigos e mensagens de resposta especificados na documentação do Swagger para cada rota presente em cada serviço e as informações obtidas nos testes de exploratórios, foram definidos os casos de testes, que posteriormente foram organizados em mapas mentais criados utilizando o Xmind[8]. Esses mapas mentais continham todos os verbos HTTP[9] presentes em cada rota, junto aos respectivos casos de teste padrão e alternativo mapeados para cada um deles. Nos mapas, os casos de teste de fluxo padrão foram destacados em verde, enquanto os de fluxo alternativo foram indicados em vermelho. Além disso, símbolos específicos foram empregados para fornecer informações adicionais, como o estado de implementação de cada caso de teste.

Na Figura 12, pode-se observar um exemplo de mapa mental que inclui os casos de teste padrão e alternativo para a rota de "intercurrences", com suas respectivas legendas. Sempre que um caso de teste fosse implementado ou invalidado, a legenda deveria ser atualizada, garantindo o controle sobre os casos de teste já implementados e eventuais impedimentos que surgissem durante a execução. A Figura 13 ilustra as diferentes legendas que poderiam ser utilizadas nos mapas mentais, proporcionando uma maneira clara de identificar o status de cada caso de teste.

Figura 13 - Legendas que podem ser utilizadas em casos de teste



Fonte: Elaborado pelo autor, 2025

De acordo com Crispin e Gregory (2009) os mapas mentais são uma forma simples e eficaz de representar ideais sobre casos de teste para uma funcionalidade do sistema. Podemos vincular conceitos e ideais que são ligados a um conceito-chave central, que para o caso dos mapas mentais elaborados para o REGPET, representam as rotas da API e seus respectivos casos de teste, como mostrado na Figura 12. Para o projeto REGPET, os mapas mentais organizaram os casos de

[8] https://xmind.app/
[9] https://www.cloudflare.com/pt-br/learning/ddos/glossary/hypertext-transfer-protocol-http/

teste de acordo com as rotas, que mais tarde seriam transformadas em tarefas gerenciadas e executadas no Azure DevOps, conforme mostrado na Figura 11. Contudo, mesmo se tratando do planejamento de testes de integração, não existia nenhuma metodologia específica para garantir que todos os serviços que estão presentes na arquitetura eram exercitados. Ficou a cargo do testador, após executar os testes exploratórios, garantir de forma manual que os casos de testes que eram criados com a ajuda da documentação no Swagger atuariam exercitando a comunicação entre os serviços.

5.3 Criação E Execução Dos Casos De Teste Dos Cenários De Integração

Com os cenários de teste devidamente planejados e mapeados nos mapas mentais, como ilustrado na Figura 12, a implementação foi realizada conforme o foco de cada sprint e as tarefas registradas no Azure DevOps. A codificação dos testes de integração foi feita utilizando o Cucumber[10], Cypress.io[11] e a linguagem de programação JavaScript[12]. Essas tecnologias foram empregadas de forma complementar, com cada uma desempenhando um papel específico no processo de testes de integração.

De acordo com Hellesoy e Wynne (2012), o Cucumber é uma ferramenta de linha de comando que lê especificações de arquivos chamados *features*. Um exemplo desse tipo de arquivo pode ser observado na Figura 14, que permite a definição de diversos cenários. Para que o Cucumber possa compreender esses arquivos, foi adotado o padrão de escrita *Gherkin*[13], que estabelece as regras sintáticas a serem seguidas. Cada cenário é composto por uma lista de passos, denominados *step definitions*, que traduzem, em linguagem compreensível por seres humanos, cada ação executada pelo código.

Segundo Mwaura (2021), o Cypress.io é uma framework destinada à automação de testes escritos em JavaScript. A integração entre o Cucumber e o Cypress.io foi realizada por meio da biblioteca chamada *cypress-cucumber-preprocessor*[14], que, em conjunto, são responsáveis pela implementação dos passos definidos, conforme demonstrado na Figura 14. Esse conjunto de ferramentas proporcionou o suporte necessário para a implementação dos testes de integração.

[10] https://cucumber.io/
[11] https://www.cypress.io/
[12] https://aws.amazon.com/pt/what-is/javascript/
[13] https://cucumber.io/docs/gherkin/
[14] https://github.com/badeball/cypress-cucumber-preprocessor

Figura 14 - Exemplo de caso de teste escrito pelo Cucumber

```
Feature: Cenários alternativos para o buscar agentes pelo ID

    @regression @back @alternative @get_agents{agent_id} @account
    Scenario: Tentar buscar um Agente com ID inexistente
        Given Um administrador cadastrado com token capturado
        When Criar um ID inexistente
        Then Deve ser retornado uma mensagem de erro ao tentar buscar um agente com ID inexistente
```

Fonte: Elaborado pelo autor, 2025

Na sequência, a Figura 15 apresenta um exemplo de como os *step definitions* foram escritos utilizando o idioma português. O *Given* descreve os passos necessários para a realização de uma ação no *When*. Após a execução dos passos e ações, espera-se uma resposta no *Then*.

Figura 15 - Exemplo de passos implementados com JavaScript utilizando o Cucumber

```
22    Given("Um administrador cadastrado com token capturado", () => {
23        cy.getToken(Config.USER_ADMIN, Config.PASS_ADMIN).then(getDataResponse => {
24            this.tokenAuth = getDataResponse;
25        })
26    })
27
28    When("Criar um ID inexistente", () => {
29        this.id = "sj41541ASD2515QA745Q";
30    })
31
32    Then("Deve ser retornado uma mensagem de erro ao tentar buscar um agente com ID inexistente", () => {
33        requestGetAgentIdAlternative.getAgentId(this.tokenAuth, this.id).then(getDataResponse => {
34            assertionsGetAgentIdAlternative.notNull(getDataResponse);
35            assertionsGetAgentIdAlternative.shouldContainDuration(getDataResponse, 1500);
36            assertionsGetAgentIdAlternative.shouldContainStatus(getDataResponse, 404);
37            assertionsGetAgentIdAlternative.shouldContainStatusText(getDataResponse, "Not Found")
38        })
```

Fonte: Elaborado pelo autor, 2025

Para cada tarefa de implementação no Azure DevOps, uma nova branch foi criada, e os casos de teste presentes no mapa mental da rota correspondente foram analisados. Com isso, cada caso de teste específico foi escrito utilizando o Cucumber, conforme mostrado na Figura 14, e implementado com o Cypress.io e JavaScript, como ilustrado na Figura 15. A integração entre essas ferramentas foi crucial para garantir que os casos de teste fossem implementados de maneira clara e compreensível.

Na Figura 14, pode-se observar *tags* como @back e @account, que foram utilizadas para facilitar a busca e execução dos casos de teste por meio de comandos. O Tópico 5.4 abordará com mais detalhes esses comandos e demonstrará a importância das *tags*. Essas tags serviram para

identificar os cenários de teste e, no caso do exemplo da Figura 14, permitiram identificar que se tratava de um teste de backend, alternativo, relacionado à rota correspondente e pertencente ao serviço *account*. As tags também eram utilizadas para executar os testes específicos por meio do Cypress.io.

A integração das três ferramentas possibilitou um melhor entendimento dos cenários, visto que, ao utilizar o Cucumber, temos uma descrição de alto nível que pode ser interpretada por qualquer pessoa que saiba o idioma ao qual os arquivos foram escritos. Isso contribui diretamente nos relatórios dos testes, como mostrado no tópico 5.4. O Cypress.io e o JavaScript desempenharam bem o seu papel de codificação e execução ao lado do Cucumber, com uma ótima integração utilizando a biblioteca mencionada.

5.4 Geração De Relatórios Referentes Aos Cenários De Teste

Como ilustrado na Figura 14, os casos de teste escritos com o Cucumber foram mapeados utilizando *tags*, como a tag "@back", que se refere a todos os testes do *backend* do sistema. Essas *tags* facilitaram a geração de relatórios, considerando diferentes critérios, como o fluxo padrão, alternativo e as rotas da API, entre outros. Além disso, foi possível combinar essas *tags* para otimizar o processo. Como exemplo, consideraram-se os seguintes comandos:

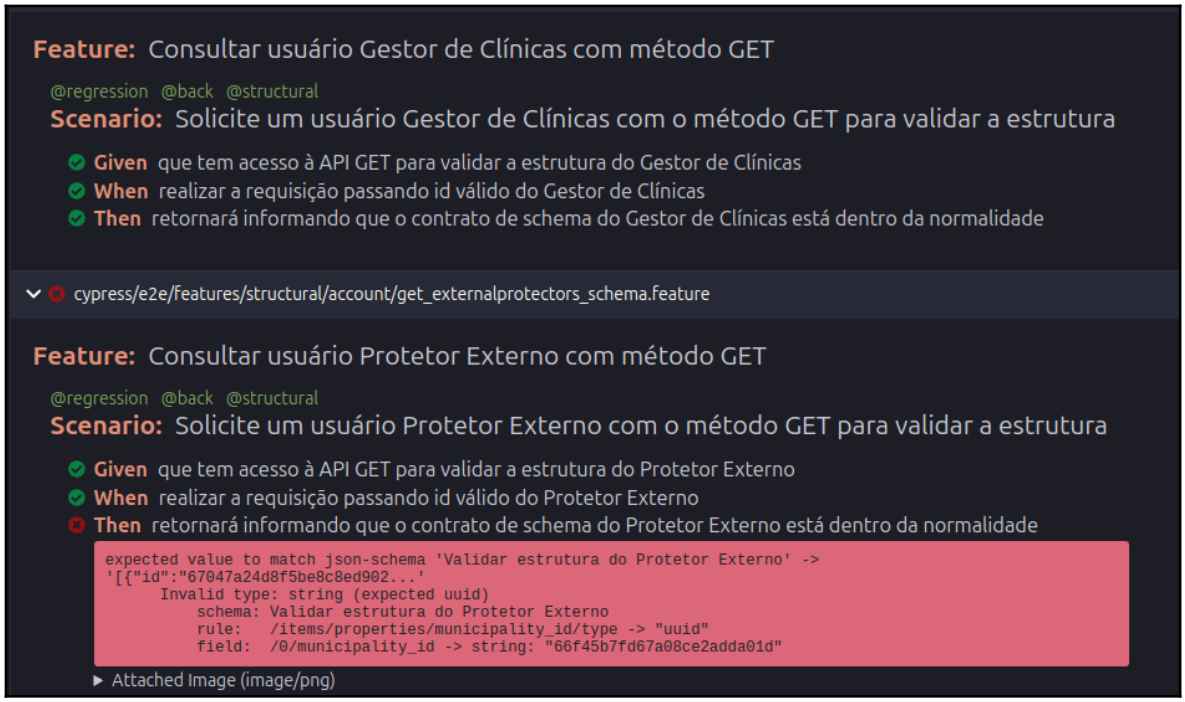
- cypress run –browser chrome –config video=false: executa todos os testes e gera o relatório.
- cypress run –env tags=back –browser chrome –config video=false: executa todos os testes do backend e gera o relatório correspondente.
- cypress run –env tags=account –browser chrome –config video=false: executa todos os testes de um único microserviço, nesse caso, o account.

O relatório gerado exibia todos os testes executados, juntamente com os resultados obtidos. Na Figura 16, são apresentados dois exemplos de arquivos de teste exibidos no relatório, ambos escritos utilizando o Cucumber: o primeiro mostra um cenário em que o teste foi executado com sucesso e o resultado esperado foi alcançado; o segundo exemplifica um caso de falha, no qual foi identificado um comportamento inesperado.

Ao utilizar o Cucumber, temos uma documentação em forma de relatório sobre o resultado da execução de uma determinada bateria de testes que exercita partes diferentes do sistema, de

acordo com as *tags*. Esta ferramenta possibilita, por exemplo, que este artefato gerado esteja em português, facilitando o entendimento de pessoas que estão fora do escopo de entendimento dos detalhes técnicos. Para além disso, a descrição do passo a passo realizado para a execução dos testes também contribuem para o entendimento das funcionalidades do sistema para alguém leigo que tenha acesso a este artefato.

Figura 16 - Exemplo de como os casos de teste são exibidos no relatório



Fonte: Elaborado pelo autor, 2025

5.5 Reporte De Comportamentos Inesperados

Nas etapas 5.2 e 5.3, foram descritas duas atividades essenciais dos testes de integração. Contudo, durante essas fases, surgiram comportamentos inesperados, como defeitos, que precisaram ser investigados. Para essa finalidade, utilizou-se a ferramenta Bugzilla[15], um sistema de rastreamento de erros, que foi utilizado pela equipe de qualidade para registrar os comportamentos anômalos, os quais deveriam ser analisados pelo restante da equipe. Na Figura 17, é possível observar um exemplo de comportamento inesperado, contendo informações cruciais, como o responsável pelo reporte, o estado do defeito, os passos necessários para sua reprodução,

[15] https://www.bugzilla.org/

além de outros dados relevantes que permitiram à equipe localizar e corrigir o defeito o mais rapidamente possível.

O uso desta ferramenta durante o processo foi essencial para compartilhar as informações de erros, falhas e defeitos encontrados durante os vários testes exploratórios e de integração realizados.

Figura 17 - Defeito cadastrado no Bugzilla

Fonte: Elaborado pelo autor, 2025.

6 CONCLUSÃO

A participação no projeto REGPET permitiu a aplicação prática de conceitos de qualidade de software, especialmente no planejamento e na execução de testes exploratórios e de integração. Durante o desenvolvimento das atividades, foi possível aprimorar habilidades técnicas e metodológicas, além de contribuir para a garantia da qualidade do sistema. A experiência também proporcionou um maior entendimento sobre processos de testes em ambientes reais, reforçando a importância da colaboração em equipe e do uso de boas práticas para assegurar a confiabilidade do software. Dessa forma, a atuação no projeto REGPET representou um avanço significativo na minha formação acadêmica e profissional, consolidando conhecimentos essenciais adquiridos ao longo do curso.

Entre os principais desafios enfrentados, destaca-se a dificuldade de manutenção da base de testes em um ambiente de ciclos de desenvolvimento acelerados. Além disso, a ausência de uma rotina de integração contínua no ambiente de testes dificultou a execução completa da bateria de testes, comprometendo a obtenção de insights essenciais para a evolução das etapas de validação. Por fim, a falta de um mapeamento claro, ao longo das versões, sobre os cenários de testes implementados, não implementados e aqueles que apresentavam falhas também impactou diretamente a qualidade do processo de teste como um todo.

REFERÊNCIAS

ALURA. *Git Flow: O que é, como e quando utilizar*. Disponível em: https://www.alura.com.br/artigos/git-flow-o-que-e-como-quando-utilizar?utm_term=&utm_campaign=%5BSearch%5D+%5BPerformance%5D+-+Dynamic+Search+Ads+-+Artigos+e+Conte%C3%BAdos&utm_source=adwords&utm_medium=ppc&hsa_acc=7964138385&hsa_cam=11384329873&hsa_grp=164240702375&hsa_ad=712801351854&hsa_src=g&hsa_tgt=dsa-2276348409543&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad_source=1&gclid=CjwKCAjw6JS3BhBAEiwAO9waFzTzfn-dwnWLKABKgSRhgHY59LnWiJE6vdAGNR_L3dSCiTShbsEDJhoCc18QAvD_BwE. Acesso em: 12 nov. 2024.

ANICHE, M. T*estes automatizados de software: um guia prático*. 1. ed. Brasil: Casa do Código, 2015.

ATLASSIAN. *Exploratory testing*. Disponível em: https://www.atlassian.com/br/continuous-delivery/software-testing/exploratory-testing. Acesso em: 12 nov. 2024.

ATLASSIAN. T*ipos de teste de software*. Disponível em: https://www.atlassian.com/br/continuous-delivery/software-testing/types-of-software-testing. Acesso em: 15 fev. 2025.

AZEVÊDO, Livyson Saymon Leão; SILVA, Rafael Bezerra Correia da. *A importância da engenharia de testes para a garantia da qualidade de software*. Revista Eletrônica Científica Inovação e Tecnologia, Universidade Tecnológica Federal do Paraná, Câmpus Medianeira, Medianeira, PR, v. 2, n. 12, p. 96-110, jul./dez. 2015. ISSN 2175-1846.

BARTIÉ, Alexandre. *Garantia da Qualidade de Software: Adquirindo Maturidade Organizacional*. 1. ed. Brasil: Casa do Código, 2002.

COODESH. *O que é Cucumber?*. Disponível em: https://coodesh.com/blog/dicionario/o-que-e-cucumber/. Acesso em: 12 nov. 2024.

CRISPIN, Lisa; GREGORY, Janet. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Boston: Addison-Wesley, 2009.

HELLESOY, Aslak; TOOKE, Steve; WYNNE, Matt. *The cucumber book: behaviour-driven development for testers and developers*. 2017.

IEEE. *Adoption of agile methodology in software development*. IEEE Conference Publication, 2013. Disponível em: https://ieeexplore.ieee.org/abstract/document/6596295. Acesso em: 29 out. 2024.

IEEE. *Profiting from Unit Tests for Integration Testing*. IEEE Xplore, 2016. Disponível em: https://ieeexplore.ieee.org/document/7515486. Acesso em: 5 nov. 2024.

IEEE. *Research on software testing techniques and software automation testing tools*. IEEE Xplore, 2024. Disponível em: https://ieeexplore.ieee.org/document/8389562. Acesso em: 28 out. 2024.

IEEE STANDARD 729. *IEEE Standard for Software Reliability*. 1. ed. New York: Institute of Electrical and Electronics Engineers, 2017. 25 p.

IEEE STANDARD 730. *IEEE Standard for Software Quality Assurance Plans*. IEEE Std 730-2014, New York: IEEE, 2014.

ISO 9126-1. *Software engineering — Product quality — Part 1: Quality model*. 1. ed. Genève: International Organization for Standardization, 2001. 24 p.

JYOLSNA, Jyolsna; ANUAR, Syahid. *Modern web automation with cypress. IO*. Open International Journal of Informatics, v. 10, n. 2, p. 182-196, 2022.

KOKOL, Peter. *Software Quality: How Much Does It Matter?* Electronics, v. 11, n. 16, p. 2485, 2022. Disponível em: https://doi.org/10.3390/electronics11162485. Acesso em: 12 nov. 2024.

LUCIDCHART. *V-Model: Verificação e validação*. Disponível em: https://www.lucidchart.com/blog/pt/v-model-verificacao-e-validacao. Acesso em: 17 fev. 2025.

MEDIUM. *ISO 9126: O que é e por que é importante para a qualidade de software*. Disponível em: https://medium.com/@nicolasnmg/iso-9126-o-que-%C3%A9-e-por-que-%C3%A9-importante-para-a-qualidade-de-software-33faa81d2cf9. Acesso em: 12 nov. 2024.

MICROSOFT. *Azure DevOps*. Disponível em: https://azure.microsoft.com/pt-br/solutions/devops#:~:text=O%20Azure%20DevOps%20traz%20integra%C3%A7%C3%B5es,integre-os%20perfeitamente%20no%20Azure. Acesso em: 12 nov. 2024.

MTP. *A importância dos testes exploratórios no desenvolvimento de software*. 2025. Disponível em: https://mtp.com.br/a-importancia-dos-testes-exploratorios-no-desenvolvimento-de-software?utm_source=chatgpt.com. Acesso em: 17 fev. 2025.

PARAÍBA. *Resolução CIB-PB nº 814/2023 - Programa de incentivo à causa animal*. Disponível em: https://paraiba.pb.gov.br/diretas/saude/arquivos-1/cib-2023/resolucao-cib-pb-no-814-2023-programa-de-incentivo-a-causa-animal.pdf. Acesso em: 12 nov. 2024.

PRESSMAN, Roger S. *Engenharia de Software: Uma Abordagem Profissional*. 6. ed. São Paulo: McGraw-Hill, 2005.

RADIX Web. *42+ Most Important Agile Statistics for 2024*. Disponível em: https://radixweb.com/blog/agile-statistics. Acesso em: 28 out. 2024.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. Scrum.org. Disponível em: <ScrumGuides.org>.

SOFTDESIGN. *Cypress: Passo a passo para começar a usar*. Disponível em: https://softdesign.com.br/blog/cypress_passo_a_passo_para_comecar_a_usar/. Acesso em: 12 nov. 2024.

SOUZA, Aderson Bastos de; RIOS, Emerson; CRISTALLI, Ricardo; MOREIRA, Trayahú. *Base de conhecimento em teste de software*. 3. ed. São Paulo: Martins Fontes, 2007. 264 p. ISBN 978-85-8063-053-4. Disponível em: https://www.amazon.com.br/Base-Conhecimento-em-Teste-Software/dp/8580630533?utm_source=chatgpt.com. Acesso em: 17 fev. 2025.

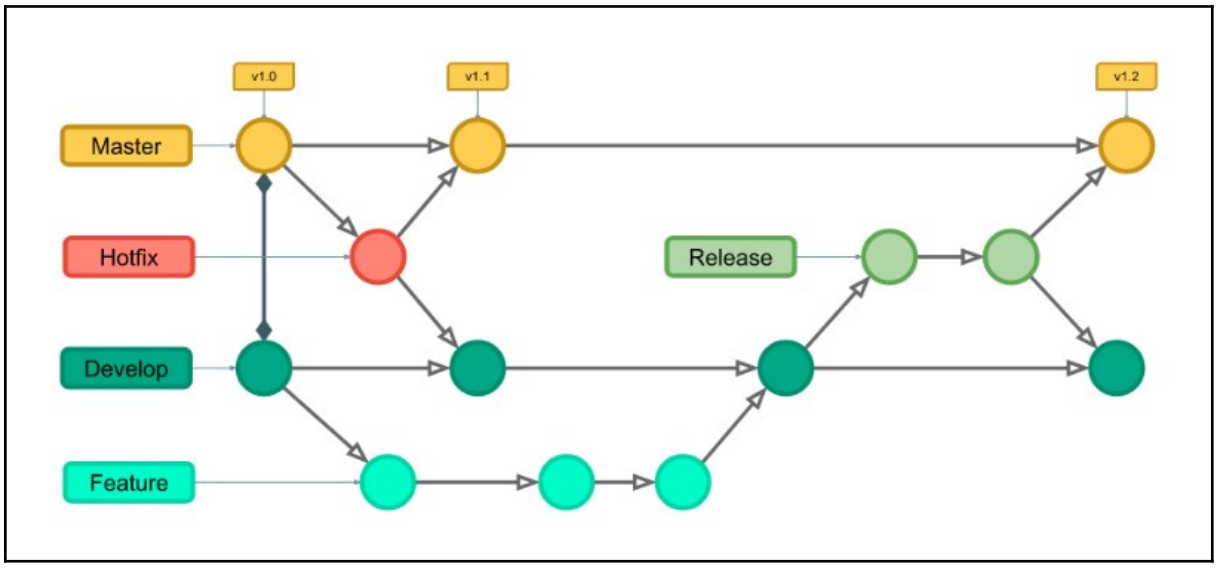
XU, M. *Agile Software Requirements Engineering Challenges-Solutions—A Conceptual Framework from Systematic Literature Review*. Information, v. 14, n. 6, p. 322, 2023. Disponível em: https://doi.org/10.3390/info14060322. Acesso em: 12 nov. 2024.

APÊNDICE A - MANUTENÇÃO E GERENCIAMENTO DAS ALTERAÇÕES UTILIZANDO REPOSITÓRIOS

Durante o desenvolvimento do REGPET, todas as alterações e manutenções foram gerenciadas por meio de repositórios no GitHub[16], adotando o fluxo de trabalho Git Flow[17]. Esse método permitiu um controle eficiente do versionamento do código, garantindo que as entregas ocorressem de maneira organizada e com rastreabilidade adequada. O Git Flow define um conjunto de *branches* específicas para diferentes fases do desenvolvimento, o que possibilitou à equipe gerenciar as versões do software de maneira estruturada.

O fluxo seguiu o modelo apresentado na Figura 18. A *branch master* armazenou a versão estável do código em produção, enquanto a *branch hotfix* foi utilizada para a correção de falhas emergenciais. As novas funcionalidades foram desenvolvidas a partir da *branch develop*, onde cada tarefa foi organizada em *branches feature*, que posteriormente foram integradas à *develop*. Ao final de cada sprint, uma *branch release* foi criada para consolidar todas as novas implementações antes da liberação para produção.

Figura 18 - Fluxo do Git Flow



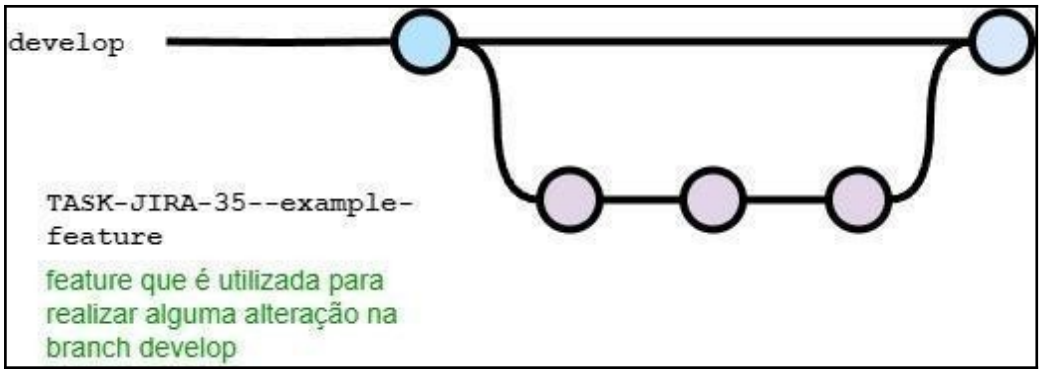
Fonte: GODOI, Murilo; CASTRO, Carolina, 2023

No decorrer do projeto, a equipe de desenvolvimento implementou alterações diretamente no ambiente de teste, assegurando que cada funcionalidade fosse devidamente

[16] https://github.com/
[17] https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

validada antes de ser disponibilizada para uso. FoFigura 19ram criadas *branches feature* para cada nova funcionalidade planejada na *sprint*, seguindo um padrão de nomenclatura que incluía o identificador da tarefa no Azure DevOps, garantindo rastreabilidade. Cada *feature* foi associada a uma tarefa específica registrada no Azure DevOps, permitindo um monitoramento detalhado das atividades em andamento e uma melhor distribuição do trabalho entre os membros da equipe. A Figura 19 ilustra esse processo, destacando a criação de uma *feature* a partir da *branch develop*.

Figura 19 - Criação de uma feature a partir da branch develop

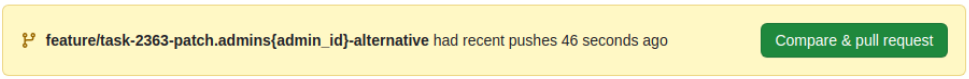


Fonte: Elaborado pelo autor, 2025

Como mostrado na Figura 19, para a equipe de testes, são criadas *branches de feature* que são usadas para desenvolver novas funcionalidades a partir da *branch develop*. Após a conclusão, essas funcionalidades são integradas por meio de uma mesclagem com a *branch develop*. Ou seja, para implementar testes de uma rota específica da API, é necessário criar uma *branch* de *feature* que descreva a rota a ser testada. Após a conclusão, a *branch* é finalizada e mesclada com a *develop*.

A *branch* de *feature* criada segue um padrão de nomeação, como podemos ver na Figura 19, que consiste no nome "task" seguido pelo ID da tarefa no Azure DevOps, mais a rota específica que está sendo testada e a indicação se é um teste padrão ou alternativo. Usando a Figura 20 como exemplo, a *branch* de *feature* criada é usada para realizar as alterações necessárias para a criação dos testes para a rota de atualização dos usuários administradores por meio do verbo HTTP PATCH. Por fim, os cenários de teste que serão implementados para essa rota serão os alternativos.

Figura 20 - Exemplo de nome para branch feature



Fonte: Elaborado pelo autor, 2025

Para realizar o que foi comentado acima, são utilizados comandos que são inseridos por meio do terminal. Esses comandos são definidos pelo Git Flow, e criam toda a dinâmica de *branchs* mencionada acima. São eles:

- **git flow feature start + nome da tarefa:** esse comando serve para iniciar a *branch feature* que servirá para realizar as alterações necessárias para criação dos testes, com o nome seguindo o padrão mencionado acima.
- **git flow feature finish + feature branch name:** após as alterações concluídas, a *branch* deve ser mesclada com a *develop* com esse comando, substituindo tudo após o símbolo de soma pelo nome da *branch feature*.
- **git push:** esse comando envia todas as alterações após a mesclagem para a *branch develop* no repositório remoto.

A adoção dessa metodologia permitiu um fluxo de trabalho estruturado, garantindo que todas as alterações fossem devidamente controladas e documentadas ao longo do desenvolvimento do REGPET. Com a separação das etapas de implementação, revisão e teste, foi possível entregar versões mais estáveis e seguras do sistema. Além disso, a rastreabilidade proporcionada pelo uso do Git Flow facilitou a identificação de eventuais problemas, permitindo correções rápidas e eficientes, reduzindo o impacto de falhas no ambiente de produção.