ISPGAYA
instituto superior politécnico

Licenciatura em Engenharia Informática
Projeto de Engenharia Informática em Contexto Empresarial
2023

Lara Júlia Medeiros Dantas da Silva Figueiredo

PROCESSO DE TESTES AUTOMATIZADOS COM RESTASSURED

Relatório de estágio orientado pelo Professor Doutor Mário Jorge Dias Lousã e apresentada à Escola Superior de Ciência e Tecnologia

Junho de 2023

Dedicatória

A persistência é o caminho do êxito.

Dedico este trabalho a minha família, esposo e amigos. Seu apoio e amor foram fundamentais para minha jornada. Agradeço a todos por fazerem parte da minha vida.

Agradecimentos

Expresso a minha profunda gratidão a Deus por me guiar e fortalecer a minha determinação, permitindo que eu siga em frente e nunca desista de mim mesmo e dos meus sonhos.

Agradeço imensamente aos meus pais, João Batista e Lindalva, e à minha irmã, Líllian Maria, por todo o amor, cuidado e apoio que me proporcionaram ao longo da vida, especialmente durante a minha jornada académica. Obrigado por estarem sempre ao meu lado.

A toda a minha família e amigos que torcem por mim e me encorajam constantemente. Ao meu esposo, Bruno Figueiredo, por sua companhia constante e pelo carinho demonstrado ao longo dessa caminhada.

Expresso minha gratidão ao professor Mário Lousã por sua orientação e dedicação na elaboração deste trabalho. À empresa ITSector, em especial ao meu gestor de projeto, William Pedrosa, agradeço pela receção da proposta de trabalho e pelo apoio fornecido para sua execução.

Por fim, agradeço aos professores do curso de Engenharia Informática do ISPGAYA pelo ensino de qualidade ao longo dos anos da minha licenciatura.

Sou verdadeiramente grata a cada um de vocês por todo o suporte, orientação e amizade que tornaram possível minha jornada académica e profissional. Vocês foram peças fundamentais nessa conquista.

Resumo

Este relatório final de estágio cumpre o requisito da unidade curricular de Projeto de Engenharia Informática em Contexto Empresarial, do terceiro ano da licenciatura em Engenharia Informática do Instituto Superior Politécnico Gaya (ISPGAYA). O objetivo é relatar a implementação de um processo de testes automatizados de software, visando aumentar a confiabilidade do software e criar uma cobertura de testes de API, que antes era realizada, apenas, manualmente. Para alcançar tais objetivos, utilizou-se a biblioteca RestAssured do Java, uma ferramenta que integra a mesma *stack* de tecnologias utilizadas no projeto, JAVA. A implementação dos testes automatizados resultou num considerável aumento na cobertura dos testes, permitindo uma validação mais eficiente e frequente do software em desenvolvimento. Isso possibilitou a identificação e correção antecipada de problemas, contribuindo para a qualidade do software final. Com o processo de testes automatizados, a equipa foi capaz de detetar falhas com maior agilidade, reduzindo o tempo necessário para as correções e melhorando a entrega do produto. A introdução desta prática no projeto, promoveu uma maior confiabilidade no software, além de otimizar os esforços da equipe de desenvolvimento, tornando o processo mais eficiente e seguro.

Abstract

This final internship report fulfills the requirement of the course Informatics Engineering Project in a Business Context in the third year of the Bachelor's degree program in Informatics Engineering at Instituto Superior Politécnico Gaya (ISPGAYA). The objective is to report on the implementation of an automated software testing process, aiming to increase software reliability and create API test coverage, which was previously done manually only. To achieve these objectives, the RestAssured library of Java, a tool that integrates with the same technology stack used in the project, was utilized. The implementation of automated tests resulted in a considerable increase in test coverage, enabling more efficient and frequent validation of the software under development. This facilitated the early identification and correction of issues, contributing to the quality of the final software. With the automated testing process, the team was able to detect faults more promptly, reducing the time required for corrections and improving product delivery. The introduction of this practice in the project promoted greater software reliability, optimized the efforts of the development team, and made the process more efficient and secure.

Lista de abreviaturas e siglas

API	Application Programming Interface
API REST	Representational State Transfer
BDD	Behavior-Driven Development
CRM	Customer Relationship Management.
CI	Continuous Integration
CD	Continuous Deployment
URL	Uniform Resource Locators
ESI	Exploração de Sistemas de Informação
DSI	Desenvolvimento de Sistemas de Informação
DEV	Desenvolvimento
GSI	Gestão de Sistemas de Informação
HTTP	Hypertext Transfer Protocol
ISPGAYA	Instituto Superior Politécnico Gaya
IDE	Integrated Development Environment
JSON	JavaScript Object Notation

Índice

Índice de figuras

1. Introdução

1.1. Contextualização

Conforme a tecnologia avança e o software se torna cada vez mais presente em todos os aspetos da vida, os requisitos para garantir a confiabilidade, manutenção e segurança tornam-se mais rigorosos. A credibilidade de um software confiável depende de vários fatores de engenharia, como um *design* cuidadoso e uma boa gestão de processos, entre outros. No entanto, realizar testes é a melhor maneira de avaliar o software e garantir a sua confiabilidade (Chaves, 2015).

Durante o processo de desenvolvimento de software, a grande maioria dos defeitos é causada por seres humanos. Mesmo com o uso dos melhores métodos de desenvolvimento e ferramentas, erros muitas vezes persistem nos produtos finais, o que torna a atividade de teste essencial durante todo o processo de desenvolvimento de software (Audy, 2007).

Segundo essas atividades abrangem revisões técnicas formais, várias etapas de teste, gestão da documentação do software e controlo de alterações nos procedimentos, com o propósito de garantir ao cliente que o software desenvolvido estará conforme as especificações e requisitos, atendendo às suas necessidades (Machado, 2018).

Existem diferentes abordagens para o desenvolvimento de software e uma delas tem ganhado bastante destaque, a abordagem ágil. Seja qual for a metodologia de desenvolvimento implementada, as atividades de testes de software são de enorme importância.

No modelo tradicional de desenvolvimento de software, os testes utilizam requisitos para gerar casos de teste, que são usados na implementação de testes manuais e automatizados. Já no desenvolvimento ágil, trabalha-se com pequenas iterações e entregas ao final de cada uma, e, por isso, os testes devem ser executados de forma contínua e integrada desde o início do projeto, tornando-se parte integrante de todo o processo de desenvolvimento (Pressman & Maxim, 2021).

O processo de teste frequentemente envolve uma combinação de testes manuais e automatizados. No teste manual, um testador executa o programa com alguns dados de teste e compara os resultados com as suas expectativas, registando e relatando quaisquer discrepâncias aos desenvolvedores do programa. Já os testes

automatizados são codificados num programa executado sempre que o sistema em desenvolvimento é testado (Barroso, 2022).

Existem diversas ferramentas disponíveis para apoiar a atividade de testes, as quais ajudam a automatizar as atividades do processo. Quando utilizadas corretamente, essas ferramentas trazem melhorias para a equipa de testes e para todo o grupo envolvido num projeto de software, permitindo uma maior cooperação entre todos, mantendo todos os envolvidos atualizados sobre o progresso do projeto, reduzindo o tempo entre a identificação e a correção de erros, e proporcionando maior segurança durante as alterações no código (Finkler, 2018).

Partindo deste pressuposto, na secção 1.2. são descritos os objetivos do presente trabalho.

1.2. Objetivos

Nesta seção, são apresentados o objetivo geral e os objetivos específicos do presente trabalho.

1.2.1. Objetivo geral

O objetivo geral deste relatório de estágio é descrever a experiência da automatização de testes num ambiente empresarial, relatando o processo de transição de testes manuais para testes automatizados num projeto específico, destacando as ferramentas utilizadas, os resultados obtidos e os desafios encontrados durante a implementação do processo. O relatório tem como finalidade documentar as atividades realizadas no estágio, bem como aprimorar o conhecimento e as habilidades da estagiária em relação à automação de testes de software.

1.2.2. Objetivos específicos

Como objetivos específicos, para o presente trabalho, foram definidos os seguintes:

- Selecionar uma ferramenta de automação de testes que atenda às necessidades do projeto e da equipa de testes.
- Elaborar um plano de teste automatizado que descreva as etapas necessárias para implementar a automação do processo de teste.

- Converter os cenários de teste manual existentes em casos de teste automatizados.
- Implementar os casos de teste automatizados na ferramenta selecionada.
- Executar os testes automatizados e comparar os resultados com os resultados obtidos nos testes manuais.
- Identificar e corrigir falhas nos casos de teste automatizados e na ferramenta de automação, conforme necessário.
- Integrar os casos de teste automatizados ao processo de desenvolvimento contínuo, para serem executados automaticamente a cada nova versão ou atualização do software.
- Avaliar o desempenho e a eficácia dos testes automatizados em termos de tempo e esforço.

1.3. Empresa ITSector

Fundada em 2005, a ITSector (https://www.itsector.pt/) – Sistemas de Informação, S.A., é uma empresa portuguesa de desenvolvimento de software especializada em transformação digital para instituições financeiras, criada com o objetivo de oferecer ao mercado soluções de sistemas de informação de elevado valor acrescentado.

A empresa encontra dividida em várias torres de desenvolvimento. Na torre onde a colaborada está atualmente alocada, Torre de AMS Talent Management, é feito trabalho suporte e implementação de novos serviços a aplicações já existentes, o que permite estar em contacto com aplicações feitas de uma forma mais legacy.

Os serviços de *outsourcing* e de desenvolvimento em regime *nearshore*, são a principal valência da ITSector, sendo que a partir de localizações estratégicas em Portugal, desenvolve software para mais de 20 países como: Angola, Moçambique, Reino Unido, Luxemburgo, Polónia, Roménia, Rússia, França, Islândia, Dinamarca, Macau, Timor, África do Sul, Canadá, entre outros.

A cultura da ITSector é o resultado de um conjunto de princípios e valores que inspiram as suas políticas e atuações. Sendo os pilares dessa cultura os seguintes:

- Satisfação do cliente, promovendo a confiança e a credibilidade.
- Ética e responsabilidade social.
- Iniciativa, dinamismo e inovação.

- Flexibilidade e autonomia.

A ITSector depende tanto da sua capacidade tecnológica quanto dos seus colaboradores para alcançar o sucesso. Por essa razão, a empresa estimula ativamente o desenvolvimento dos seus colaboradores e privilegia habilidades técnicas.

- Profissionalismo.
- Espírito de equipa.
- Iniciativa.
- Honestidade.

1.4. Estrutura do documento

Este relatório está organizado em quatro secções distintas. Na primeira secção, é apresentada uma contextualização acerca do tema do trabalho, bem como uma apresentação da instituição onde o estágio foi realizado, além de expor os objetivos do trabalho. Na segunda secção, é feito um enquadramento teórico sobre o tema dos testes de software, abordando alguns conceitos essenciais, tais como os diferentes tipos e níveis de testes que podem ser aplicados, destacando a importância dos testes no desenvolvimento de software. Adicionalmente, são abordados os tipos de testes funcionais.

A terceira secção descreve todo o processo do estágio, apresentando exemplos das principais tarefas realizadas durante o estágio, bem como a forma como as atividades de teste ocorreram desde o início dos casos de teste até o registo de erros. Finalmente, a quarta secção apresenta os resultados da implementação e a conclusão do trabalho, abordando algumas das dificuldades encontradas e fornecendo sugestões para a melhoria do processo de testes aplicados na empresa.

2. Base teórica

Nesta secção, serão apresentados os conceitos teóricos necessários para melhor entendimento do trabalho.

2.1. Desenvolvimento de software

O desenvolvimento de software é o processo de criar, modificar, testar e manter programas de computador e sistemas de software. Envolve a identificação de requisitos, a projeção da estrutura do software, a implementação do código, o respetivo teste e implantação num ambiente de produção. A manutenção contínua também é necessária para corrigir bugs e adicionar novas funcionalidades. É uma disciplina complexa que requer competências técnicas, a compreensão dos requisitos dos utilizadores e a colaboração entre equipas (Borba, 2017).

Anteriormente, o desenvolvimento de software era frequentemente realizado seguindo uma abordagem em cascata, também conhecida como modelo tradicional. Nesse modelo, as etapas do processo de desenvolvimento eram realizadas em sequência, com cada fase dependendo da conclusão da anterior. Isso significava que o planeamento, o design, a implementação, os testes e a implementação ocorriam em fases distintas e separadas (Mota, 2018).

No entanto, com o passar do tempo, percebeu-se que esse modelo tinha algumas limitações. As mudanças nos requisitos eram difíceis de serem realizadas, pois qualquer alteração numa etapa inicial poderia exigir retrabalho em todas as etapas subsequentes. Além disso, o feedback dos utilizadores era recebido tardiamente, o que poderia resultar em produtos que não atendiam às suas necessidades reais (Campi, 2012)

Perante essas limitações, surgiu a necessidade de recorrer a abordagens mais flexíveis e adaptáveis ao desenvolvimento de software. Assim, ganhou destaque o desenvolvimento ágil, que se baseia em princípios de flexibilidade, colaboração e resposta rápida às mudanças. Na secção 2.2 serão explicados os principais conceitos desta metodologia.

2.2. Desenvolvimento Agile

Nas últimas duas décadas houve uma necessidade de mudar o modo de desenvolvimento de software. O mercado digital de software exige facilidade na elaboração e alteração de requisitos, levando a entregas cada vez mais rápidas destes mesmos requisitos. Segundo Oliveira e Júnior (2019) para atender a estas exigências, começaram a adotar-se as metodologias ágeis, por apresentarem uma forma moderna e simplista, tornando as equipas de desenvolvimento mais flexíveis e autónomas, permitindo que as tarefas sejam definidas e amplamente discutidas antes de serem executadas.

No início do século XXI, foi escrito "O Manifesto Ágil" em resposta a estas novas necessidades. De acordo com esse documento, a abordagem Ágil pode ser vista como uma filosofia de desenvolvimento baseada em quatro valores fundamentais (e seus consequentes doze princípios):

1. indivíduos e interações sobre processos e ferramentas;

2. software utilizável sobre documentação abrangente;

3. colaboração do cliente sobre negociação de contratos;

4. reagir à mudança sobre seguir um plano (Ozelieri, 2018).

A filosofia Agile tem múltiplas implementações concretas, nomeadamente, Extreme Programming, Crystal Methodologies, SCRUM, entre outras. As empresas, além de utilizar as *frameworks* originais, têm como hábito desenvolver novas *frameworks* que atendam melhor às necessidades dos seus projetos e colaboradores, as quais são frequentemente adotadas por outras empresas com necessidades semelhantes. O conceito Agile é orientado por uma utilização mais consciente dos recursos disponíveis, sejam eles as horas úteis de trabalho ou a produção de documentação. O foco está nas pessoas e na forma como elas realmente trabalham. Onde os imprevistos são comuns, e a comunicação é a chave para o sucesso.

Em contraste com as metodologias tradicionais de desenvolvimento de software, que são mais sequenciais e previsíveis, o desenvolvimento ágil promove uma abordagem iterativa e colaborativa, com foco na entrega de valor ao cliente de forma rápida e adaptativa.

As vantagens do desenvolvimento ágil em relação, por exemplo, à abordagem cascata são:

- Flexibilidade e adaptação a mudanças nos requisitos do projeto.
- Entrega de valor mais rápida e incremental.
- Maior envolvimento do cliente ao longo do processo.
- Visibilidade e transparência para toda a equipa e partes interessadas.
- Maior colaboração e empoderamento da equipa.
- Ênfase na qualidade do software, com práticas de teste contínuo e revisões.

O desenvolvimento ágil desempenha um papel crucial no processo de testes de software, proporcionando uma abordagem dinâmica e iterativa, essencial para garantir a qualidade do produto. Uma das principais vantagens do desenvolvimento ágil é o feedback rápido e contínuo que é obtido ao longo de todo o ciclo de desenvolvimento (Carvalho & Mello, 2012). Isso significa que os testes são realizados regularmente, permitindo a identificação precoce de problemas e bugs. Essa abordagem ajuda a economizar tempo e recursos, pois os defeitos são corrigidos imediatamente, evitando custos e riscos associados à sua descoberta tardia (Mota, 2013).

Além disso, na secção 2.3. são apresentados conceitos importantes para o teste de software. Esses conceitos contribuem para a compreensão e aplicação efetiva dos testes no contexto ágil. A compreensão desses conceitos fundamentais é essencial para garantir a qualidade e eficácia dos testes realizados durante o desenvolvimento ágil.

2.3. Testes de software

O processo de avaliação do sistema ou dos seus componentes, tem como objetivo analisar o software, para detectar as diferenças entre as condições existentes e necessárias, e avaliar as suas características e o respetivo funcionamento, é conhecido como teste de software (Freitas, 2015). Por outras palavras, o teste envolve a execução do sistema para identificar erros nas suas funcionalidades ou requisitos que não foram implementados conforme as necessidades requeridas (Tutorials Point India Private Limited, 2014).

Os testes de software são importantes porque permitem que os desenvolvedores e testadores avaliem o desempenho, a funcionalidade, a segurança e a usabilidade do software. Isso ajuda a identificar possíveis problemas que podem afetar a experiência do utilizador ou comprometer a integridade do sistema. Eles também ajudam a reduzir

os custos de desenvolvimento, pois os erros são detetados mais cedo no processo de desenvolvimento, evitando a necessidade de reescrever todo o código ou fazer correções complexas e caras no futuro (Santos, 2012).

Existem diferentes tipos de testes de software, cada um com um objetivo específico. Nas próximas secções, serão descritos alguns exemplos desses tipos de testes.

2.3.1. Teste funcional ou não funcional

Os testes podem ser agrupados segundo as características do sistema que se deseja testar. Os testes podem avaliar as características funcionais do sistema, como a sua completude, correção e adequação. De igual modo, os requisitos não-funcionais também podem ser analisados, tais como, a fiabilidade do sistema, a eficiência da performance, a segurança, a compatibilidade e a usabilidade do sistema (International Software Testing Qualifications Board, 2018). Então, os testes funcionais verificam o comportamento do sistema do ponto de vista do utilizador, enquanto os testes não-funcionais verificam os atributos de qualidade do sistema. O sistema como um todo também deve ser verificado para garantir a sua completude de acordo com as especificações.

Uma vez que os testes funcionais levam em consideração o comportamento do sistema, eles podem ser elaborados utilizando técnicas de caixa-preta (*black-box*), o que será explicado na secção 2.3.2.

2.3.2. Técnicas de testes black-box, white-box e experience-based

Os testes podem ser divididos em três categorias de acordo com a técnica utilizada: *black-box*, *white-box* e *experience-based*.

A técnica de testes *black-box* tem como foco a análise da relação entre os *inputs* e *outputs* do objeto em teste, sem considerar a estrutura interna. Por isso, é considerada uma técnica que se concentra na vertente comportamental e pode ser aplicada tanto em testes funcionais como não-funcionais (Marsh & Boag, 2013).

A técnica de teste *white-box* envolve a análise da arquitetura interna do objeto de teste, ou seja, estuda a vertente estrutural do mesmo e requer o acesso ao código implementado. Por outro lado, a técnica de teste *experience-based* apoia-se no conhecimento prévio do desenvolvedor ou testador para elaborar e executar os casos

de teste. Essas técnicas podem incluir também técnicas *white-box* e *black-box*, dependendo da abordagem adotada (Dalal, 2004).

Na secção 2.3.3, serão abordados os testes de regressão, que verificam a funcionalidade do software após modificações.

2.3.3. Testes de regressão

Os testes de regressão são normalmente realizados após a correção de algum defeito ou a adição de uma nova funcionalidade ao sistema. O objetivo desses testes é assegurar que nenhum defeito foi introduzido no sistema durante a sua evolução. Não é eficiente testar o sistema uma única vez e verificar que não existem defeitos para aquele conjunto específico de casos de teste, sem executar novamente esses casos após modificações no sistema (Rocha, 2020).

As novas mudanças podem trazer defeitos para o sistema, tornando necessária a realização de testes de regressão para garantir que o software continua a funcionar corretamente após as mudanças realizadas. Estes tipos de testes são, normalmente, realizados através de ferramentas de automação de testes, porque um problema encontrado durante estes testes é a falta de tempo para executar novamente casos de testes já executados (Pessoa & others, 2020).

Os testes regressivos têm relação direta com os testes contínuos (*continuous testing*) ao longo do processo de desenvolvimento de software. Na secção 2.3.4. é explicado o conceito de *continuous testing*.

2.3.4. Continuous testing

A abordagem de *continuous testing* no desenvolvimento de software implica que o produto seja testado de uma forma automatizada com antecedência, frequência e abrangência em todas as componentes, visando obter *feedback* pertinente o mais cedo possível (ISTQB, 2020). Com a aplicação do *continuous testing*, os desenvolvedores recebem *feedback* imediato após modificarem uma parte do código, já que a alteração dispara a execução dos testes automatizados.

Ao antecipar a análise da qualidade do software, é possível ajustar a direção do trabalho. Caso os testes indiquem uma boa qualidade, o desenvolvimento pode prosseguir com confiança. Caso contrário, os eventuais defeitos são corrigidos prontamente, aumentando a produtividade. De acordo com (Saff & Ernst, 2004) a

diminuição de produtividade é diretamente proporcional ao tempo de vida de um erro. Quando um erro não é descoberto imediatamente, são necessárias mais mudanças para corrigi-lo.

Ao aplicar *continuous testing* no desenvolvimento, o uso de recursos importantes, como o tempo e a energia do desenvolvedor, é reduzido, como comprovado num estudo realizado por Staff e Ernst (2004) que referem: *"os participantes que utilizaram uma ferramenta de continuous testing tinham três vezes mais probabilidade de completar a tarefa corretamente comparativamente àqueles que não tinham suporte de continuous testing*". Além disso, o mesmo estudo constatou que os participantes que utilizaram *continuous testing* eram três vezes mais propensos a concluir a tarefa antes do prazo. Na secção 2.3.4. será descrito o conceito de continuos integration que está intimamente relacionado com o *continuos testing*, permitindo a integração frequente de código e testes.

2.3.5. Continuous integration

A *Continuous Integrationin* (CI) consiste na fusão regular de código produzido ou alterado pelos desenvolvedores. Todas as ações necessárias ao software, como a compilação, o *deployment* e os testes, devem integrar um processo único, automático e cíclico (Coelho, 2015). O objetivo da CI é permitir que os desenvolvedores integrem o código frequentemente, de modo que possíveis problemas possam ser detetados e corrigidos rapidamente (Rocha, 2014).

Este processo automático, que inclui vários tipos de testes ao software, deve ser executado o mais frequentemente possível, consoante as prioridades da empresa ou produto, sendo possível a deteção e correção de defeitos o mais rápido possível (ISTQB, 2020).

A integração contínua é uma prática que pode ser encarada como um controlo de qualidade contínuo. Além de contribuir para a melhoria da qualidade do produto, ela também é capaz de reduzir o tempo necessário para a entrega do software, já que o controlo de qualidade não é feito apenas ao final de cada ciclo, mas sim de forma contínua e constante durante todo o processo de desenvolvimento (Humble & Farley, 2010).

Na secção 2.3.5., serão descritos os testes de aceitação, que desempenham um papel fundamental na correlação entre *continuous integration* e garantia da conformidade dos requisitos do software.

2.3.6. Testes de aceitação

Na área de testes de software, há diversos níveis, tais como os testes de componentes, os testes de integração, os testes de sistema e os testes de aceitação. Esses níveis de teste estão diretamente relacionados às fases do desenvolvimento do sistema e, portanto, cada um deles possui objetivos específicos (Rocha, 2014).

- Testes de componentes – foco em componentes que podem ser testadas independentemente, por exemplo, estruturas de dados, classes, módulos de base de dados;
- Testes de integração – foco nas interações entre componentes, sejam elas bases de dados, interfaces, micro-serviços;
- Testes de sistema – foco no comportamento do sistema como um todo, avaliando tanto a vertente funcional como a não-funcional;
- Testes de aceitação – foco no comportamento do sistema como um todo, relativamente às expetativas do cliente.

Os testes elaborados para o presente relatório enquadram-se no segundo nível – testes de integração – portanto, esta seção dedica-se a aprofundar uma explicação sobre os mesmos.

O principal objetivo de testes de integração é verificar a construção da estrutura do software em desenvolvimento e a comunicação adequada entre os seus módulos. Este teste é fundamental para evitar a perda de dados na interface entre os módulos ou a ocorrência de resultados inadequados (Kriger, 2021).

Assim, a estratégia de teste é uma técnica sistemática para construir arquitetura de software, além de realizar teste para descobrir erros relacionados com as interfaces (Pressman & Maxim, 2021).

Na secção 2.3.6., serão discutidas as diferenças e características do teste manual e do teste automático.

2.3.7. Teste manual x Teste automático

Testes manuais e testes automáticos são duas abordagens diferentes para garantir a qualidade do software.

O teste manual envolve a reprodução de todas as funcionalidades pelo testador, seguindo o documento previamente definido. Geralmente, quando a interface do utilizador do software está envolvida, o teste manual é amplamente utilizado, pois uma pessoa real pode detetar facilmente problemas potenciais e reais durante o uso do programa (Pessoa, 2020).

No contexto ágil, os testes manuais, geralmente, são considerados como testes exploratórios (Bortoluci & Duduchi, 2015). Eles envolvem a execução de testes através de um ciclo rápido de etapas que incluem o planeamento, o *design* e a execução.

O teste manual consiste em testar o software de forma manual, ou seja, sem o uso de ferramentas automatizadas ou *scripts*. Nesse tipo de teste, o testador desempenha o papel de um utilizador final e testa o software para identificar qualquer comportamento ou *bug* inesperado (Silva, 2012)

Uma das principais desvantagens de testar software manualmente é a possibilidade de erros humanos. Ao depender exclusivamente de testadores humanos para executar os testes, existe uma maior probabilidade de ocorrerem equívocos, omissões ou inconsistências nos procedimentos de teste (Moraes, 2013). Os testadores podem cometer erros ao inserir manualmente os dados de teste, executar passos incorretos ou negligenciar certos cenários de teste. Além disso, os testes manuais são suscetíveis à fadiga, diminuindo a concentração e a precisão ao longo do tempo.

Esses erros podem resultar em falhas não detetadas, levando a software com problemas, baixa qualidade e insatisfação dos utilizadores. Além disso, os testes manuais, geralmente, são demorados e consomem muitos recursos, já que requerem a presença constante de testadores para executá-los repetidamente, o que pode ser custoso e ineficiente em termos de tempo e esforço.

No âmbito do projeto em questão, serão automatizados os testes manuais de API. Portanto na secção 2.4. serão definidos os conceitos de API e testes de API.

2.4. Testes de API

API é uma ponte de comunicação que permite que diferentes softwares se comuniquem entre si de maneira organizada. Ela define as regras e os protocolos que os programas devem seguir para trocar informações e interagir uns com os outros. É como um conjunto de instruções que os softwares usam para se entenderem e

compartilharem dados de forma padronizada. Através da API, os aplicativos, serviços e sistemas podem se conectar e trabalhar juntos de forma mais eficiente (Costa, 2022).

De acordo com (Smith, 2020) ao testar uma API, existem várias áreas e aspetos que podem ser avaliados. Pode-se citar alguns pontos-chave a serem considerados nos testes de API. Dentre eles:

- Funcionalidade: verificar se a API está a executar corretamente as suas funções e atendendo aos requisitos funcionais definidos. Devem ser testadas todas as operações e *endpoints* disponíveis para garantir que eles produzam os resultados esperados.
- Validação de entrada: verificar o modo como a API lida com diferentes tipos de dados de entrada. Teste casos de uso com dados válidos, dados inválidos, limites de tamanho, formatos específicos, entre outros. Certificar-se se a API valida e trata corretamente os dados fornecidos.
- Autenticação e autorização: verificar o modo como a API lida com a autenticação e autorização de utilizadores ou clientes. Testar se os mecanismos de segurança estão implementados corretamente e se apenas os utilizadores autorizados têm acesso aos recursos adequados.
- Tratamento de erros: testar a forma como a API responde a erros e exceções. Confirmar se a API retorna códigos de erro apropriados, mensagens de erro claras e informações úteis para facilitar a depuração e resolução de problemas.
- Desempenho e carga: avaliar o desempenho da API em diferentes cenários de carga. Realizar testes de carga para verificar como a API lida com um grande número de solicitações simultâneas e avaliar o seu tempo de resposta, taxa de transferência e escalabilidade.
- Segurança: realizar testes de segurança para identificar vulnerabilidades na API, como, por exemplo, ataques de injeção, exposição de informações confidenciais. Garantir que a API está protegida contra ameaças e implementar práticas adequadas de segurança.
- Integração: verificar como a API se integra com outros sistemas, serviços ou APIs. Testar a interoperabilidade e a compatibilidade da API com outros componentes do ecossistema em que ela está inserida.

Estas são apenas algumas áreas a serem consideradas durante os testes de uma API. O âmbito e a abrangência dos testes podem variar dependendo dos requisitos e do contexto específico do projeto. É importante elaborar um plano de testes abrangente e explorar diferentes cenários para garantir a qualidade e o bom

funcionamento da API. Algumas validações mais específicas que se faz necessário ter em conta:

- Verificar o *status* retornado pela API.
- Validar o *header* de retorno.
- Verificar o corpo da resposta da API.
- Testar o comportamento da API quando o serviço está indisponível.
- Testar o comportamento da API com dados inválidos.
- Verificar se a API retorna os resultados esperados para diferentes tipos de dados.

Uma abordagem popular para testes de API é BDD (*Behavior-Driven Development*). O BDD utiliza a estrutura "*Given-When-Then*" para descrever o comportamento esperado da API. O BDD promove a colaboração e uma melhor compreensão dos testes de API. Na secção 2.5. o BDD é descrito de um modo mais aprofundado, explorando os princípios fundamentais desta técnica.

2.5. Behavior - driven development

BDD que significa "desenvolvimento orientado ao comportamento", e é uma metodologia na qual os desenvolvedores, *testers* e clientes trabalham em equipa para analisar os requisitos do sistema de software e formulá-lo numa linguagem compreendida por todos. Em seguida, esses requisitos são verificados automaticamente para garantir que o sistema esteja funcional conforme o esperado (Roman et al., 2018).

Antes do surgimento da metodologia ágil, havia um modelo conhecido como Cascata que se baseava em cronogramas rígidos, extensa documentação e pouca interação com o cliente, o que significava que o cliente só teria acesso ao produto final no final do projeto (Belém & Rezende, 2021). O BBD então surgiu para ultrapassar estas lacunas, definindo as especificações em termos do comportamento desejado da aplicação, e não do estado desejado, e sempre com um nível de abstração que facilite a transposição dos mesmos para testes automatizados.

De acordo com (North, 2006) que descobriu a necessidade do BDD: "*Se pudéssemos desenvolver um vocabulário consistente para analistas, testers, desenvolvedores e para o negócio, estaríamos num bom caminho para eliminar parte da ambiguidade e das falhas de comunicação que ocorrem quando as pessoas da área técnica falam com empresários"*.

Na próxima secção, será explicado o *Gherkin*, a sua sintaxe e o modo como é usado para escrever as especificações no contexto do BDD.

2.5.1. Linguagem Gherkin

O *Gherkin* é uma linguagem utilizada para descrições de comportamentos da aplicação. O *design* desta linguagem de programação foi concebido para ser facilmente compreensível por qualquer ser humano que conheça linguagem natural, independentemente da língua que fale, uma vez que ela suporta cerca de 70 idiomas diferentes, se tornando assim um componente importante no âmbito BDD (Wynne et al., 2017). Existem *keywords* (ou "palavras-chave") a serem utilizadas para estruturar frases em *Gherkin*, tais como: *Given*, *When*, *Then*, *And*, *But*.

De acordo com (Wynne et al., 2017) a ordem geralmente seguida nas frases construídas em Gherkin é a seguinte:

- "*Given*/Dado o contexto inicial" – pré-condição;
- "*When*/Quando um evento acontece" – trigger/ação catalisadora;
- "*Then*/Então acontece o resultado esperado" – pós-condição.

Considerando um software de uma máquina de café, uma das funcionalidades é a de servir café. Com base nessa funcionalidade, é possível escrever vários casos de teste para testar cenários de sucesso e erro. A Figura 1 ilustra um exemplo de como é testada a funcionalidade de servir café quando existe apenas mais uma dose de café.

Figura 1 - Estrutura de testes usando Gherkin

```
Scenario: Buy last coffee
  Given there are 1 coffees left in the machine
  And I have deposited 1 dollar
  When I press the coffee button
  Then I should be served a coffee
```

Fonte: https://www.valuebound.com/resources/blog/introduction-behavior-driven-development

No contexto deste trabalho, a nomenclatura do *Gherkin* será aplicada à automatização. Pois a *framework* escolhida utiliza o *Gherkin* para escrever cenários de testes. Na seção 2.5.2 serão explicados os conceitos da *framework* RestAssured.

2.5.2. RestAssured

O RestAssured é uma ferramenta desenvolvida para facilitar a criação de testes automatizados para APIs REST. Esta biblioteca oferece suporte para validar protocolo HTTP e requisições em JSON (Cabral, 2019).

A estrutura do RestAssured trabalha no formato de BDD, realizando chamadas para o serviço, utilizando a sintaxe *given*, *when* e *then*. No *given* passa-se as pré-condições como, por exemplo, token de autenticação e body; no *when* efetua-se a chamada para o método a ser testado, passando parâmetros quando forem necessários; e o *then* tem como função trabalhar com os retornos da requisição, extraindo respostas e efetuando assertivas (Oliveira, 2020).

Quando se trata de testes de serviços, há alguns elementos fundamentais que devem ser validados, incluindo o *status code*, o corpo da resposta (onde é possível validar mensagens e valores dos atributos), autenticação, permissões de acesso e tipo de conteúdo, entre outros. Embora cada contexto apresente cenários de teste distintos, esses elementos são a base para a validação. Para realizar as validações, pode-se utilizar recursos nativos do JUnit, como o AssertEquals(), ou a biblioteca Hamcrest, que oferece recursos mais específicos para a escrita de testes (Borba, 2017).

Na secção 3 são apresentadas as atividades desenvolvidas durante o estágio.

3. Protótipo de Automação de Testes

Nesta secção descrevem-se as atividades realizadas na empresa, ao longo dos meses, em que decorreu o estágio.

3.1. Descrição do Projeto API IT Sector

O presente trabalho descreve um projeto cujo foco primordial reside no desenvolvimento de APIs personalizadas para um cliente do setor da banca. O objetivo central desse empreendimento consiste em criar soluções escaláveis e seguras que possibilitem a integração de sistemas e a troca de informações sensíveis no ambiente financeiro. O cliente em questão já possuía soluções pré-existentes, porém estas eram representadas por sistemas legados, incapazes de proporcionar escalabilidade e segurança.

Diante dessa situação, o principal objetivo do cliente é migrar esses sistemas para uma arquitetura de microserviços, visando obter maior flexibilidade, modularidade e capacidade de escalabilidade horizontal. Essa transição para microserviços permitirá

ao cliente uma arquitetura mais moderna e ágil, capaz de atender às demandas do mercado de forma mais eficiente e adaptável.

O projeto é conduzido por meio de uma colaboração multidisciplinar, com a participação de equipas de desenvolvimento de software, testes e produto. No presente estudo, destaca-se o papel desempenhado pela estagiária na equipa de qualidade e testes. Os requisitos, na sua essência, compreendiam as características básicas de uma API REST, porém, com um foco especial na segurança e desempenho. Durante o processo de desenvolvimento, foram definidos *endpoints* e métodos de solicitação adequados, a fim de garantir que as APIs atendessem às necessidades específicas de integração do cliente.

A segurança e a proteção das informações confidenciais foram prioridades ao longo de todo o projeto. Adotamos medidas de autenticação e autorização, assegurando que apenas utilizadores autorizados tivessem acesso às funcionalidades fornecidas pelas APIs. Além disso, seguimos os padrões de segurança estabelecidos pelo setor bancário, visando à salvaguarda dos dados sensíveis durante as etapas de transmissão e armazenamento.

Na secção 3.2. será descrito o funcionamento do atual workflow de desenvolvimento do projeto.

3.2. Processo de desenvolvimento e testes

A Figura 2 ilustra o fluxo do processo de desenvolvimento e testes do projeto, envolvendo três entidades principais: Produto, Desenvolvimento e Qualidade. Essas entidades representam diferentes responsabilidades e abstrações das pessoas envolvidas.

No contexto desse processo, a entidade "Produto" refere-se às pessoas envolvidas no negócio ou à equipa responsável pela execução do projeto. Essas pessoas desempenham um papel fundamental no desenvolvimento e no sucesso do produto final. O grupo pode incluir gerentes de projeto, analistas de negócio, desenvolvedores, testadores, especialistas em qualidade e outros membros da equipa.

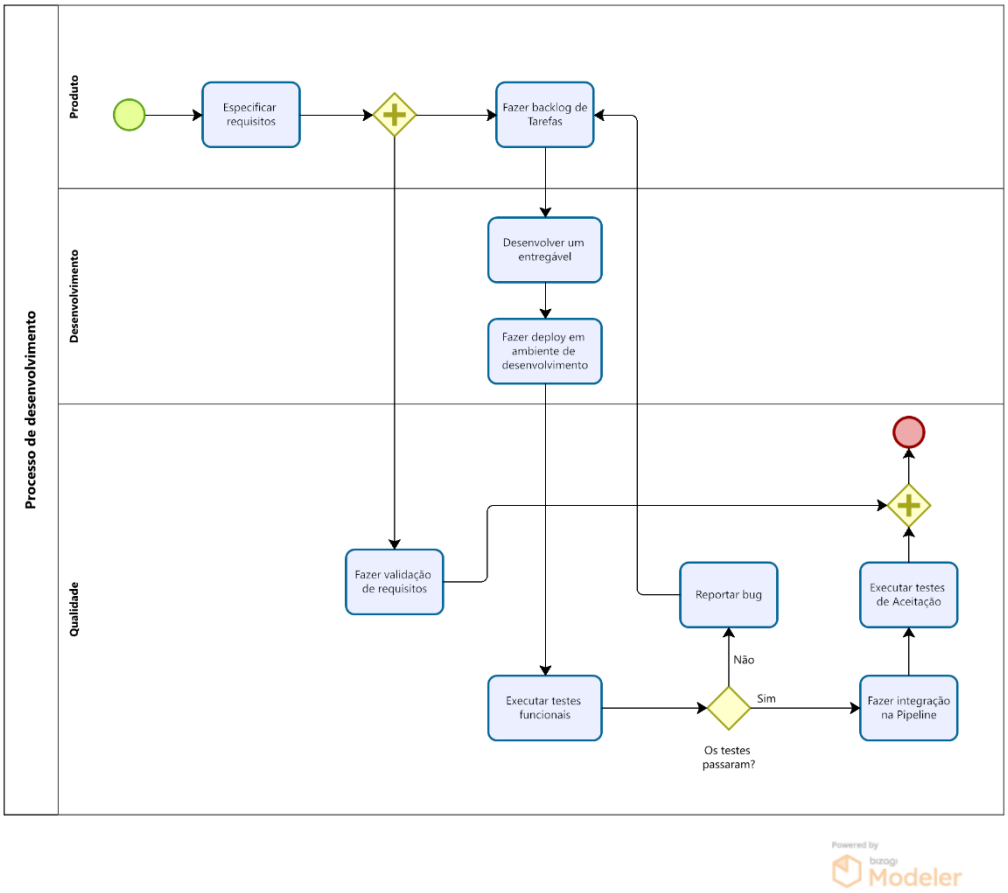
A entidade "Desenvolvimento" é a equipa responsável por todas as atividades relacionadas com a criação e construção do produto. Nesse contexto, inclui-se a análise de requisitos, codificação, configuração, integração de sistemas e outras etapas envolvidas na construção do produto.

Por sua vez, a entidade "Qualidade" é responsável por garantir que o produto atenda aos requisitos e padrões de qualidade estabelecidos. Essa equipa engloba todas

as atividades relacionadas com a verificação e validação do produto, incluindo testes de qualidade, revisões técnicas, análise de conformidade, auditorias e implementação de processos de controlo de qualidade.

O processo se inicia com a etapa de especificação dos requisitos por parte do cliente. Em seguida, é criado um *backlog* de tarefas, normalmente estruturadas em *requirements* e *tasks* associadas. Posteriormente, os programadores iniciam o desenvolvimento, resultando num entregável e implantação no ambiente de desenvolvimento (DEV).

Figura 2 - Workflow de desenvolvimento do projeto



Fonte: Própria

Após o *deploy*, a equipa de testes assume a responsabilidade pelos testes funcionais. Nesta fase, os testes são executados e validados, e quaisquer erros ou problemas identificados são registados para posterior correção. É importante ressaltar que os testes começaram a ser estruturados já na fase de conceção dos requisitos, com o intuito de validar a sua clareza, testabilidade e atendimento às solicitações do cliente.

Uma vez concluídos os testes e na existência de *bugs* a equipa de desenvolvimento faz as correções necessárias com base no relatório de *bugs* e

problemas encontrados. Após as correções, é realizada uma nova execução de testes para verificar se os problemas foram resolvidos de forma satisfatória. Esse ciclo de desenvolvimento, teste, correção e reteste continua até que as APIs estejam totalmente funcionais, atendendo aos requisitos do cliente.

Cabe ressaltar que a equipa de qualidade desempenha um papel fundamental durante todo o processo, garantindo a conformidade com os padrões de qualidade estabelecidos, realizando revisões técnicas e garantindo que os testes sejam abrangentes e eficazes. Dessa forma, o projeto segue um fluxo bem definido, procurando assegurar a entrega de APIs personalizadas de alta qualidade e desempenho.

3.3. Processo manual de testes

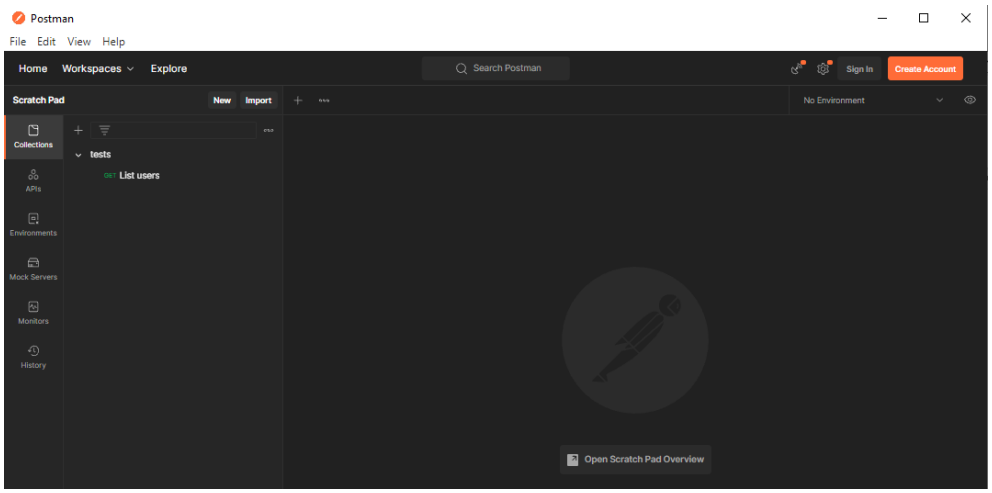
No estágio atual do projeto, a realização dos testes funcionais de API é feita de forma manual, sendo o Postman a ferramenta principal utilizada para esse propósito.

O Postman é uma solução abrangente que oferece suporte à documentação detalhada das requisições feitas pela API. Além disso, ele disponibiliza um ambiente completo para a execução de testes de APIs e realização de requisições em geral.

A interface gráfica do Postman é altamente intuitiva, permitindo que os utilizadores especifiquem facilmente os parâmetros e detalhes necessários para cada chamada de API. Através dessa interface, é possível definir a URL de destino da chamada, selecionar o método HTTP adequado (como GET, POST, PUT, DELETE), adicionar parâmetros de consulta, definir cabeçalhos personalizados e incluir o corpo da requisição, caso seja necessário.

Uma das vantagens do uso do Postman é a sua capacidade de facilitar a validação e teste das respostas da API. A ferramenta oferece recursos avançados para a criação de asserções, que são regras específicas para verificar se os dados retornados estão corretos. Essas asserções podem englobar a verificação de códigos de *status*, estrutura do JSON/XML retornado, valores esperados e muito mais, permitindo uma avaliação completa do comportamento da API. A Figura 3 demonstra a interface gráfica da ferramenta.

Figura 3 - Interface do Postman

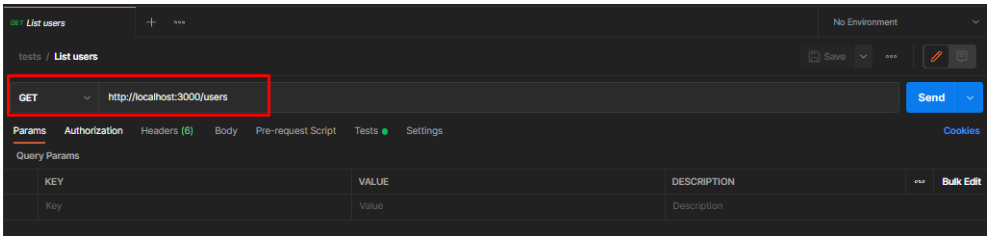


Fonte: Própria

Para ilustrar as validações realizadas, dado que há restrições de confidencialidade da empresa, exemplos fictícios gerados por um mecanismo de mock são utilizados. Os *mocks* são uma forma de simular as respostas da API, sem depender de dados reais do projeto. Ao utilizar um mecanismo de *mock*, é possível definir respostas personalizadas para cada requisição, possibilitando a simulação de diferentes cenários, como respostas bem-sucedidas, erros específicos ou situações de falha. Essa abordagem permite testar o comportamento da aplicação cliente frente a diversas respostas simuladas.

No primeiro passo do processo de teste, é necessário especificar a URL para onde a chamada será enviada, seguido pela definição do método da chamada, como no caso específico em que é utilizado o método GET. A Figura 4 apresenta uma representação visual da estrutura dessa chamada, auxiliando na compreensão do fluxo de testes realizado.

Figura 4 - Invocação do método GET



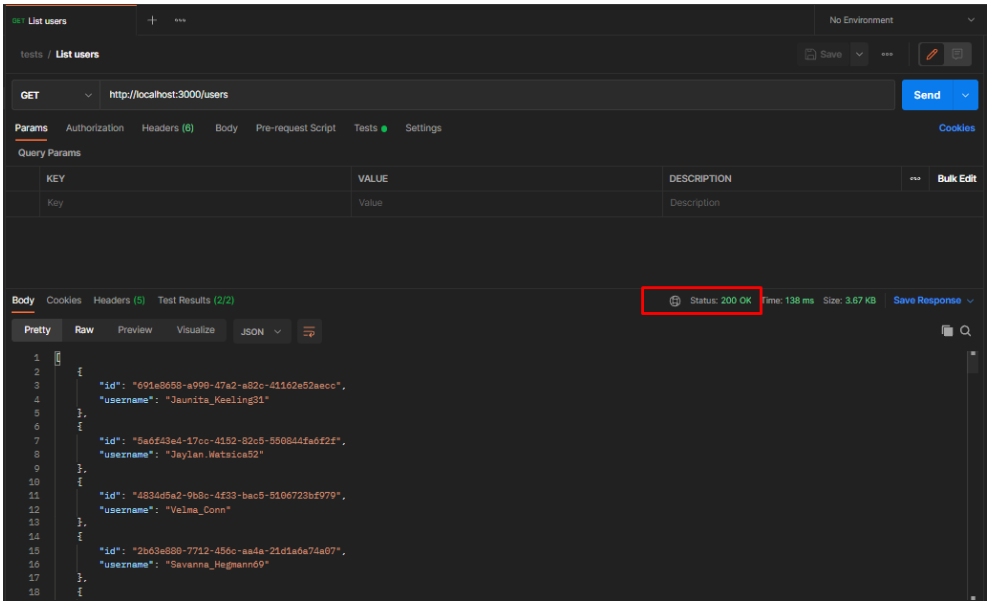
Fonte: Própria

Após especificar a URL e informar o método, é possível criar um caso de teste para validar o retorno. No exemplo a seguir, iremos verificar se o recurso retorna o código de *status 200*, que indica que a requisição foi bem-sucedida:

Caso de Teste: Verificar o código de status 200

A requisição retorna um corpo de resposta contendo uma lista de dados obtidos. Como mostra na Figura 5.

Figura 5 - Validação de status code



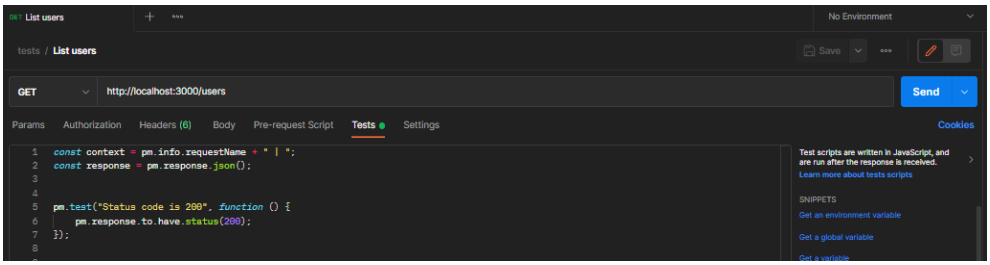
Fonte: Própria

Para validar o código de *status* da requisição corretamente, podemos utilizar a aba "*Tests*" e adicionar o seguinte código em JavaScript, que é a sintaxe utilizada pelo Postman para realizar as asserções:

```
1. pm.test("Verificar código de status 200", function () {
2.     pm.response.to.have.status(200);
3. });
```

A Figura 6 ilustra o resultado esperado.

Figura 6 – Função para validar status code



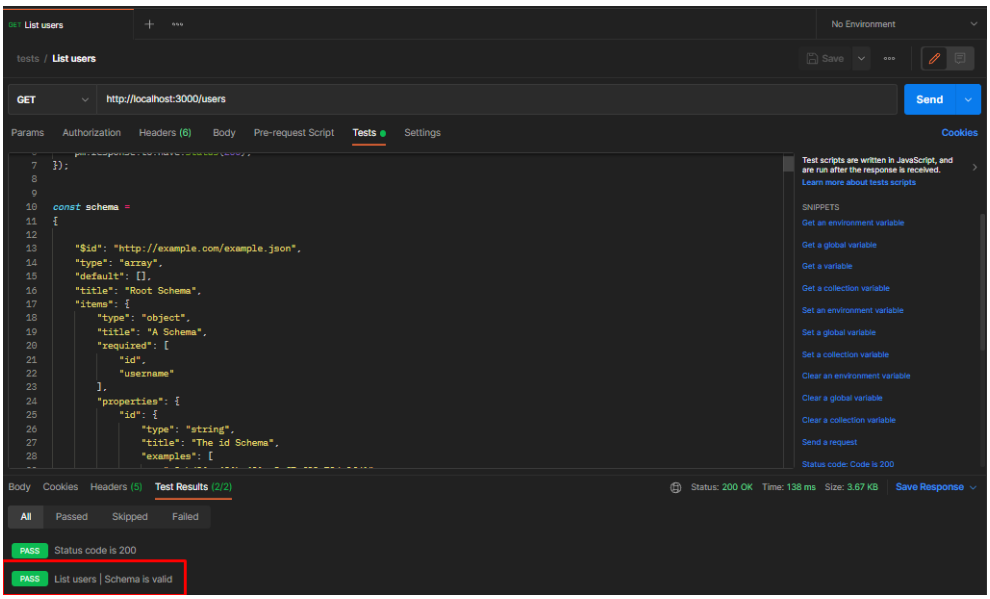
Fonte: Própria

Test Case: Validação do JSON

A validação do JSON é importante para garantir a integridade e a conformidade dos dados retornados por uma API. Ela verifica se a estrutura, os tipos de dados e as regras estão corretos, de acordo com um esquema definido. Isso ajuda a evitar erros, assegura a conformidade com as especificações e melhora a qualidade das APIs. Para criar o esquema e facilitar a validação, utiliza-se a ferramenta *online* https://jsonschema.net. Essa ferramenta permite gerar o esquema JSON com base em exemplos de dados e auxilia na validação dos dados em conformidade com o esquema definido. Com o uso dessa ferramenta, permite uma abordagem eficiente para garantir a qualidade e a consistência dos dados retornados pela API.

Para ilustrar como é a validação do JSON no postman ilustra-se na Figura 7:

Figura 7 - Estrutura do schema



Fonte: Própria

Nesta figura, é demonstrado o fluxo de validação do JSON utilizando o esquema gerado pelo JSON Schema Generator. O esquema é aplicado à resposta JSON recebida pela API, e a validação é realizada para garantir que a estrutura e os dados estejam em conformidade com o esquema definido. Caso ocorram erros de validação, eles são identificados e podem ser visualizados no *console* do Postman.

As validações mencionadas anteriormente são apenas exemplos de alguns tipos de validações que podem ser realizadas. No entanto, existem várias outras validações que podem ser aplicadas, dependendo dos requisitos específicos do projeto.

Com o objetivo de otimizar o tempo e o esforço envolvidos nos testes de regressão manuais, a automação dos testes de API tornou-se uma necessidade. Essa abordagem traz benefícios significativos, como a redução do tempo e do esforço necessários, além de proporcionar maior eficiência e precisão nos resultados. Através da automação, é possível criar *scripts* de teste que podem ser executados repetidamente, identificando de forma rápida regressões e problemas de integração.

Na próxima secção 3.4. - Desenho e automatização do processo de testes, será fornecido um detalhe mais completo sobre o processo de testes automatizados, abordando aspectos como ferramentas utilizadas, estratégias de automação e a sua importância no contexto do projeto em questão.

3.4. Desenho e automatização do processo de testes

3.4.1. Porquê automatizar este processo?

A automação do processo de testes é essencial no projeto devido à frequente adição de novas funcionalidades ou modificações em métodos existentes nas API´s desenvolvidas. Automatizar os testes traz uma série de vantagens. Em primeiro lugar, permite que os testes sejam executados de forma rápida e consistente, economizando tempo e esforço. Além disso, a automação facilita a deteção precoce de regressões e problemas de integração, permitindo que sejam corrigidos rapidamente antes que afetem negativamente o sistema.

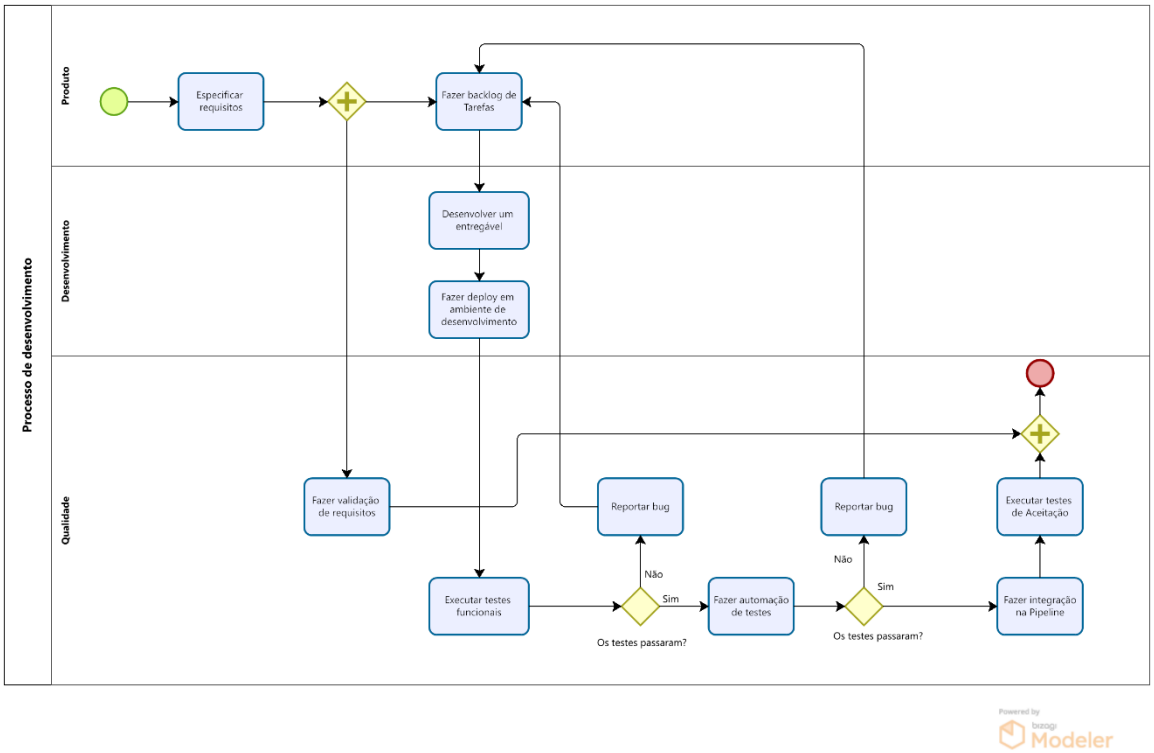
A automação dos testes também proporciona uma cobertura abrangente dos casos de teste. É possível criar *scripts* de teste que abrangem diferentes funcionalidades e cenários, garantindo que todas as partes críticas do software sejam testadas de maneira consistente.

Outro benefício importante é a agilidade no ciclo de desenvolvimento. Com a automação dos testes, as equipas podem verificar e validar as mudanças implementadas de forma mais rápida, acelerando a entrega do software. Além disso, os testadores podem se concentrar em tarefas mais complexas e de maior valor, enquanto os testes automatizados cuidam dos casos repetitivos.

Em resumo, automatizar o processo de testes é fundamental para lidar eficientemente com as mudanças no projeto, garantindo a qualidade do software. Isso proporciona uma execução consistente, deteção precoce de problemas e maior agilidade no ciclo de desenvolvimento.

Neste caso, foi necessário adaptar o processo de desenvolvimento para incluir a tarefa de automação. Essa modificação foi realizada para garantir a eficiência e a qualidade do projeto, permitindo a incorporação da automação dos testes de forma adequada. A Figura 8 mostra o novo processo:

Figura 8 - Workflow com adição dos testes automatizados



Fonte: Própria

3.4.2. Como automatizar?

No contexto das APIs, existem várias ferramentas disponíveis para automatizar os testes. No entanto, no projeto em questão, foi selecionada a ferramenta Rest Assured para a automação dos testes de API. A escolha pelo Rest Assured deve-se ao fato de ser uma ferramenta que faz parte da mesma *stack* de tecnologias utilizadas no projeto.

Ao escolher uma ferramenta que está alinhada com a stack do projeto, há uma maior integração e sinergia entre as diferentes partes do sistema. O Rest Assured, sendo uma biblioteca Java, permite uma fácil integração com o código existente, facilitando o desenvolvimento dos testes automatizados.

Além disso, ao utilizar uma ferramenta da mesma *stack*, a equipa de desenvolvimento já está familiarizada com a sintaxe e a abordagem utilizada. Isso reduz a curva de aprendizado e agiliza o processo de automação, permitindo que a equipe aproveite ao máximo os recursos oferecidos pela ferramenta.

Ao utilizar o Rest Assured, é possível aproveitar as suas funcionalidades específicas para testes de API, como a facilidade de envio de requisições HTTP, a validação de respostas e a extração de dados. Essas características tornam o Rest Assured uma escolha adequada para automatizar os testes de API no contexto do projeto em questão.

3.4.3. Fundamentos técnicos sobre RestAssured

No processo de desenvolvimento dos testes automatizados de APIs, foi crucial adquirir conhecimentos técnicos na utilização do RestAssured. Para isso, foram explorados diversos recursos, tais como artigos do Medium, como os de Thilakumara (2020) e Oliveira (2023), além de um tutorial em vídeo por Raghav (2020). Adicionalmente, a documentação oficial do RestAssured foi estudada minuciosamente.

Através dos artigos do Medium, foi possível aprofundar as funcionalidades e recursos oferecidos pelo RestAssured. A leitura desses artigos proporcionou uma boa compreensão da biblioteca, que permitiu assim a escrita de testes automatizados de forma eficiente. Além disso, a comunidade no Medium costuma compartilhar *insights* valiosos e abordar casos de uso práticos, o que facilitou a aprendizagem. Os tutoriais gratuitos do YouTube também desempenharam um papel fundamental. Ao assistir a esses cursos, foi possível acompanhar instrutores especializados e seguir exemplos práticos de uso do RestAssured.

No entanto, adquirir o conhecimento teórico foi apenas o primeiro passo. Após absorver as informações dos artigos e cursos, investiu-se tempo e esforço para implementar os testes automatizados utilizando o RestAssured. Essa etapa exigiu a criação de cenários de teste relevantes, a escrita de código em conformidade com as melhores práticas e a validação das respostas da API.

No total, foram dedicadas cerca de 10 horas para adquirir o conhecimento teórico sobre o RestAssured. De seguida, na fase de implementação dos testes automatizados, foram despendidos, aproximadamente, 30 horas de execução, distribuídas ao longo de cerca de um mês, sendo o projeto executado em paralelo com as tarefas já existente internamente. Essas 30 horas foram preenchidas com atividades como a configuração do ambiente de teste, a elaboração dos casos de teste, a construção dos *scripts* automatizados, a depuração de problemas e a otimização dos testes. Foi uma etapa desafiadora, mas extremamente valiosa para consolidar o conhecimento adquirido e obter experiência prática na aplicação do RestAssured.

3.4.4. Automação de Testes com RestAssured

Inicialmente, o planeamento do trabalho visava ampliar a cobertura de testes para várias APIs. No entanto, é comum em projetos de software, depararmo-nos com restrições de tempo e recursos que nos obrigaram a repensar a nossa abordagem. Dado que a automação não é a prioridade principal deste projeto, decidiu-se iniciar com uma prova de conceito. Para esta prova de conceito, utilizou-se uma API de simulação, como mostra na Figura 9.

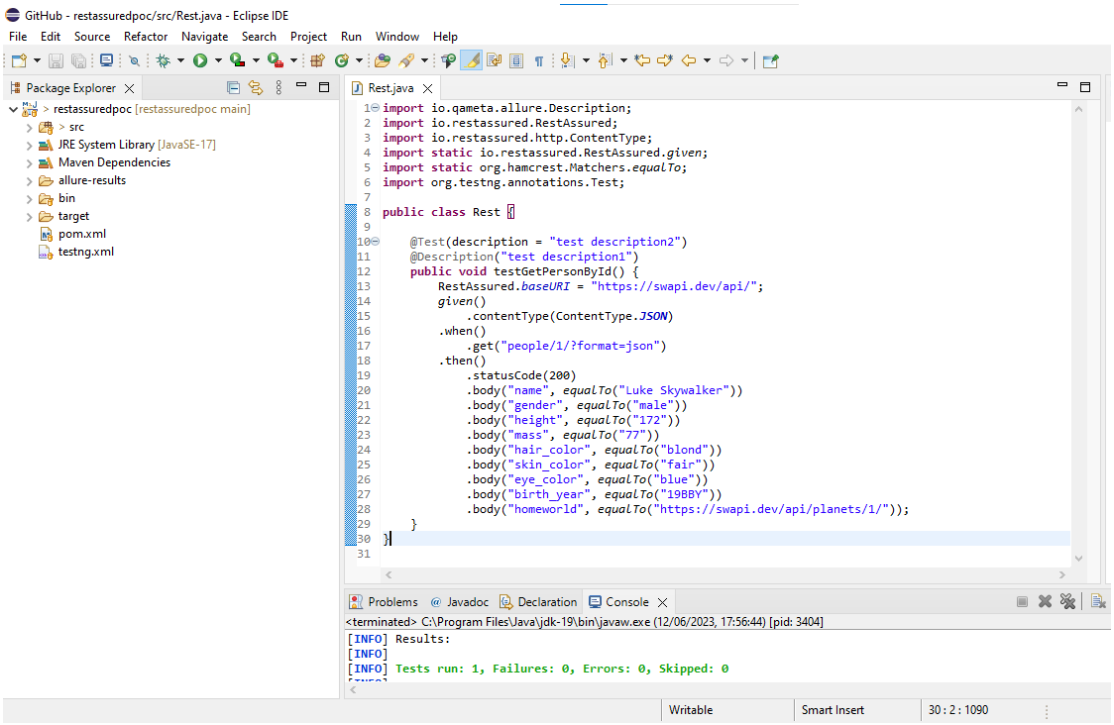
Figura 9 - API de simução

Fonte: https://swapi.dev/api/planets/1/

Para a prova de conceito, escolheu-se o RestAssured para validar um *endpoint* do tipo GET. Selecionamos o *endpoint* público https://swapi.dev/api/, uma API de testes que oferece diversas informações sobre personagens de Star Wars, de entre outras coisas. É importante salientar que, devido a questões de confidencialidade, não podemos exibir todos os testes desenvolvidos. Portanto, neste trabalho, apresentaremos apenas uma parte dessa prova de conceito.

Na imagem 10 a seguir mostra uma classe de teste em que utiliza o test assured para validar dados.

Figura 10 - Exemplo de estrutura de teste com RestAssured



Fonte: Própria

Neste teste, estamos a utilizar a biblioteca RestAssured para realizar uma requisição HTTP do tipo GET num *endpoint* específico.

Inicialmente, definimos a base da URI como "https://swapi.dev/api/" utilizando o método RestAssured.baseURI. Em seguida, utilizando o método given(), configuramos o tipo de conteúdo da requisição como JSON, especificado por ContentType.JSON.

Na sequência, utilizamos o método when() para indicar que estamos prontos para realizar a ação da requisição, que no caso é um GET no *endpoint* "people/1/?format=json".

Logo após, utilizamos o método then() para realizar asserções nos dados da resposta da requisição. Primeiramente, verificamos se o código de *status* retornado é 200, utilizando o método statusCode(200). Em seguida, utilizamos o método body() para verificar os valores de diferentes campos no corpo da resposta.

Neste caso específico, estamos a verificar se os valores dos seguintes campos são iguais aos esperados:

"name" é igual a "Luke Skywalker"

"gender" é igual a "male"

"height" é igual a "172"

"mass" é igual a "77"
"hair_color" é igual a "blond"
"skin_color" é igual a "fair"
"eye_color" é igual a "blue"
"birth_year" é igual a "19BBY"
"homeworld" é igual a "https://swapi.dev/api/planets/1/"

Dessa forma, o teste verifica se o *endpoint* está a retornar os dados esperados para o personagem com o ID 1, no caso, Luke Skywalker.

Durante a execução dos testes, encontramos desafios relacionados com a utilização do RestAssured num ambiente seguro. Foi necessário adicionar certificados para possibilitar os testes na API, o que acabou por ser uma dificuldade inesperada. No entanto, com o apoio dos desenvolvedores, que possuíam um maior conhecimento sobre o ambiente de teste e o desenvolvimento da API, conseguimos superar esse obstáculo.

Uma vez solucionada essa questão, foi possível validar as respostas esperadas para a API em questão. Mais uma vez, a colaboração dos desenvolvedores foi fundamental para contornar essa dificuldade e garantir o sucesso dos testes.

Para compartilhar o código desenvolvido, optou-se por utilizar o Git. Foi criada uma *branch* remota para trabalhar nesse código, porém, não foi realizada a fusão (*merge*) dessa *branch*. Isso ocorreu porque o planeamento inicial previa a integração desse código numa *pipeline*, na qual seria possível executar os testes não apenas localmente, apontando para o ambiente de teste, mas também por meio dessa *pipeline*. O objetivo era automatizar a execução desses testes após cada *commit* ou imediatamente após o lançamento de uma nova versão (*release*). Dessa forma, a integração contínua e a entrega contínua seriam suportadas pela *pipeline*, garantindo a execução dos testes de forma eficiente e consistente.

3.4.5. Resultados falando sobre cobertura de automação de teste

Como mencionado anteriormente, o objetivo inicial do trabalho era ampliar a cobertura de testes para várias APIs. No entanto, devido às limitações comuns em projetos de software, como restrições de tempo e recursos, precisamos de rever a nossa abordagem. É importante destacar que, inicialmente, a API em questão não possuía nenhuma cobertura de testes, o que tornou ainda mais crucial o desenvolvimento dos testes.

A API em questão consistia num projeto de pequeno porte, composto por apenas dois *endpoints*. Focamos os nossos esforços na realização de testes abrangentes para esses dois *endpoints*, o que resultou numa cobertura de 100%. Embora esse resultado seja significativo em termos absolutos, ele não atende às solicitações atuais do projeto.

Ao analisar mais detalhadamente, identificou-se que a API possui, em média, cerca de 30 *endpoints*. Contudo, até o momento, apenas os mesmos dois *endpoints* foram cobertos pelos testes. Isso indica que a nossa cobertura ainda é muito baixa, deixando a maior parte da funcionalidade da API sem validação.

Dessa forma, fica evidente a necessidade de expandir a nossa cobertura de testes para os demais *endpoints*, a fim de garantir uma abordagem abrangente e sólida em relação à qualidade do software.

Quanto ao tempo de execução dos testes, os resultados obtidos são absolutamente significativos e trazem ganhos consideráveis. Antes da automação, eram necessários cerca de 30 minutos para executar os dois testes manualmente. Com a automação implementada, esses mesmos testes são concluídos em menos de 10 segundos. Esse resultado é impressionante, pois permite a execução dos mesmos testes num tempo significativamente menor.

Essa redução no tempo de execução dos testes traz benefícios práticos na validação do comportamento das APIs. Com uma execução mais rápida, podem ser testados mais cenários, aumentando a abrangência dos testes e proporcionando uma validação mais ampla da API.

Além disso, é importante destacar que os benefícios da automação de testes vão além da economia de tempo. Com a automação, é possível executar os testes de forma repetitiva, permitindo a deteção precoce de regressões e problemas de compatibilidade. Além disso, a automação garante maior consistência e precisão nos testes, reduzindo a margem de erro humana.

Para ilustrar melhor os benefícios da automação de testes, podemos considerar um cenário hipotético. Suponha que, com a automação, a cobertura dos testes seja expandida para incluir todos os 30 *endpoints* da API. Nesse caso, poder-se-ia executar os testes num curto período de tempo, obtendo uma visão abrangente do comportamento da API em questão. Essa abordagem permitiria identificar problemas de forma mais rápida, facilitando a manutenção e evolução do software.

4. Conclusões

Ao finalizar o estágio curricular, é imprescindível destacar a relevância de trabalhar com tecnologias inovadoras, mesmo já atuando na área profissional. O processo de implementação da automação de testes foi extremamente desafiador, exigindo esforço para adquirir o conhecimento técnico necessário e realizar o desenvolvimento do protótipo.

A abordagem de automatizar os testes revelou-se fundamental para aprimorar a eficiência e a qualidade dos processos de teste. Embora tenha exigido um esforço adicional para dominar as tecnologias e conceitos envolvidos, os benefícios foram significativos.

Primeiramente, a automação permitiu a execução rápida e repetitiva dos testes, reduzindo o tempo gasto nessa etapa e aumentando a produtividade. Além disso, a consistência dos resultados obtidos por meio da automação garantiu uma avaliação mais precisa e confiável da qualidade do software.

Outro ponto crucial foi o ganho em termos de cobertura de testes. A automação possibilitou a criação de testes abrangentes, capazes de explorar diferentes cenários e fluxos de dados. Com isso, foi possível identificar falhas e comportamentos indesejados numa fase inicial do processo de desenvolvimento, contribuindo para a entrega de um produto mais confiável e de alta qualidade.

Embora a aprendizagem e a implementação da automação de testes tenham sido desafiadoras, o esforço investido resultou num valioso crescimento profissional. A aquisição de novas competências técnicas e a compreensão dos benefícios da automação abriram portas para oportunidades futuras e fortaleceram a minha base de conhecimento na área de testes de software.

Perante isto, conclui-se que é essencial procurar constantemente a aprendizagem e a adoção de tecnologias inovadoras. A implementação da automação de testes, embora desafiadora, revelou-se uma escolha acertada, proporcionando melhorias significativas nos processos de teste e enriquecendo a minha trajetória profissional.

Existem alguns pontos planeados para implementação que não puderam ser concluídos, o que é comum em projetos de desenvolvimento de software. A automação de testes de software não foi considerada uma prioridade no projeto. Foi explicado que a automação exige tempo e recursos, e sem essa prioridade ou mesmo algum equilíbrio

entre as outras exigências, a atuação da estagiária ficou limitada. Entre os principais pontos que não foram concluídos destacam-se:

- Cobrir um maior número de *endpoints* de diversas API´s e os respetivos cenários de testes com o RestAssured.
- Integrar os testes automatizados no *continuous testing* e *continuous deployment*.

Essa integração é importante para garantir que os testes sejam executados automaticamente em cada etapa do *pipeline* de entrega, fornecendo *feedback* rápido sobre a qualidade do código e permitindo a entrega contínua do produto.

Embora esses pontos não tenham sido alcançados neste momento, é importante reconhecer as limitações e priorizar as entregas de acordo com os recursos disponíveis. No futuro, pode ser considerada a continuidade dessas tarefas para aprimorar a cobertura de testes e integrar os testes automatizados ao fluxo de CI/CD, visando melhorar a qualidade e a eficiência do processo de desenvolvimento.

Referências bibliográficas

Audy, J. L. N. (2007). *Desenvolvimento distribuido de software*. Elsevier.

Barroso, R. de O. (2022). *Investigando o processo de testes de software nas startups de software brasileiras*.

Belém, H. de L., & Rezende, Í. N. (2021). *BDD-CLI: um arcabouço para geração automática de artefatos de testes unitários de software*.

Borba, P. E. S. (2017). *Sistemas legados: uma proposta para centralização de autenticação*.

Bortoluci, R., & Duduchi, M. (2015). Um estudo de caso do processo de testes automáticos e manuais de software no desenvolvimento ágil. *X WORKSHOP DE PÓS-GRADUAÇÃO E PESQUISA DO CENTRO PAULA SOUZA*.

Cabral, J. (2019, dezembro 1). *Testing Spring Boot RESTful APIs using MockMvc/Mockito, Test RestTemplate and RestAssured*. https://medium.com/swlh/https-medium-com-jet-cabral-testing-spring-boot-restful-apis-b84ea031973d.

Campi, M. E. (2012). *Análise de qualidade e desempenho em relação a processos internos: o caso do departamento de materiais indiretos do setor de suprimentos de uma empresa de tecnologia avançada*. Faculdade de gestão e negócios, Universidade Metodista de Piracicaba Piracicaba.

Carvalho, B. V. de, & Mello, C. H. P. (2012). Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica. *Gestão & Produção*, *19*, 557–573.

Chaves, L. E. (2015). *Gerenciamento da comunicação em projetos*. Editora FGV.

Coelho, A. S. M. (2015). *Desenvolvimento de software com integração contínua*. Instituto Politecnico do Porto (Portugal).

Costa, P. J. B. da. (2022). *Garantia de qualidade de software (SQA)*.

Dalal, J. G. (2004). Software Testing Fundamentals-Methods and Metrics. *Software Quality Professional*, *6*(3), 42.

de Freitas, T. M. (2015). *Aplicaçao de Uma Estratégia de Automaçao Contnua de Testes no Desenvolvimento de Software*.

Finkler, F. E. (2018). *Análise comparativa de testes estruturados e testes exploratórios de especialistas*.

Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

International Software Testing Qualifications Board. (2018). *ISTQB Foundation Level Syllabus.*

ISTQB. (2020). *ISTQB Foundation Level Syllabus*.

Kriger, D. (2021, outubro 8). *O QUE É TESTE DE INTEGRAÇÃO E QUAIS SÃO OS TIPOS DE TESTE?*

Machado, F. N. R. (2018). *Análise e Gestão de Requisitos de Software Onde nascem os sistemas*. Saraiva Educação SA.

Marsh, T., & Boag, S. (2013). Evolutionary and differential psychology: conceptual conflicts and the path to integration. *Frontiers in psychology*, *4*, 655.

Moraes, M. A. F. de. (2013). *AUTOMAÇÃO DE TESTES DE REGRESSÃO E SMOKE TEST EM APLICAÇ\~OES WEB COM O SELENIUM*.

Mota, E. B. (2013). *ABORDAGEM DIAMANTE NO GERENCIAMENTO DE RISCOS EM PROJETOS*. Fundação Getulio Vargas.

Mota, E. B. (2018). *GANHOS DE PRODUTIVIDADE EM PROJETOS DE SOFTWARE UTILIZANDO METODOLOGIA ÁGIL FRAMEWORK SCRUM VS MODELO TRADICIONAL PMBOK*. Fundação Getúlio Vargas.

North, D. (2006). *Introducing BDD. Better Software*.

Oliveira, C. (2020, julho 12). *Testando API com Cucumber, JAVA, Rest Assured*. https://medium.com/@carlos.st.oliveira/testando-api-com-cucumber-java-rest-assured-c2e72a7c40a6.

Oliveira, C. (2023, junho 26). *Testando API com Cucumber, JAVA, Rest Assured*. https://medium.com/@carlos.st.oliveira/testando-api-com-cucumber-java-rest-assured-c2e72a7c40a6

Oliveira, D. A., & Júnior, I. G. (2019). Uma solução algoritmica para projetos de TI utilizando metodologia ágil SCRUM: um estudo de caso. *Revista Inovação, Projetos e Tecnologias*, *7*(1), 80–91.

Ozelieri, L. G. (2018). *Como a metodologia Scrum possibilita o compartilhamento do conhecimento: um estudo em uma empresa multinacional de tecnologia da informação*.

Pessoa, C. H. M. (2020). *Avaliação de ferramentas de automação de teste: uma análise documental e comparativa*.

Pessoa, C. H. M., & others. (2020). *Avaliação de ferramentas de automação de teste: uma análise documental e comparativa*.

Pressman, R. S., & Maxim, B. R. (2021). *Engenharia de software-9*. McGraw Hill Brasil.

Raghav, P. (2020, dezembro 22). *REST API Testing with RestAssured* . https://www.youtube.com/watch?v=oVNbaBlrhbo&list=PLhW3qG5bs-L8xPrBwDv66cTMlFNeUPdJx

Rocha, A. (2020, julho 29). *A importância do Teste de Regressão*.

Rocha, L. F. da. (2014). Aplicação da integração contínua no desenvolvimento de software. *Sistemas de Informação-Pedra Branca*.

Roman, A., ROMAN., & Gerstner. (2018). *Study Guide to the ISTQB Foundation Level 2018 Syllabus*. Springer.

Saff, D., & Ernst, M. D. (2004). An experimental evaluation of continuous testing during development. *ACM SIGSOFT Software Engineering Notes*, *29*(4), 76–85.

Santos, A. P. O. dos. (2012). *Aplicação de práticas de usabilidade ágil em software livre*. Universidade de São Paulo.

Silva, C. B. P. (2012). *Apoiando a Construção de Testes de Aceitação Automatizados a partir da Especificação de Requisitos*. Universidade Federal do Rio Grande do Norte.

Smith, J. (2020). Testes de API: Pontos-chave a serem considerados. *Proceedings of the International Conference on Software Testing and Quality Assurance*, 123–136.

Thilakumara, C. (2020, maio 11). *Rest Assured API Testing - Part 1*. https://medium.com/chaya-thilakumara/rest-assured-api-testing-part-1-e96a2f284a6e

Tutorials Point India Private Limited. (2014).

Wynne, M., Hellesoy, A., & Tooke, S. (2017a). *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf.

Wynne, M., Hellesoy, A., & Tooke, S. (2017b). *The cucumber book: behaviour-driven development for testers and developers*. Pragmatic Bookshelf.