FACULDADE DE TECNOLOGIA DE SÃO PAULO

LEANDRO RODRIGUES DE SANTANA

AUTOMAÇÃO DE TESTES EM APIS REST NO CONTEXTO ÁGIL

SÃO PAULO

2022

LEANDRO RODRIGUES DE SANTANA

AUTOMAÇÃO DE TESTES EM APIS REST NO CONTEXTO ÁGIL

Trabalho submetido como exigência parcial para a obtenção do grau de tecnólogo em Análise e Desenvolvimento de Sistemas

Orientador: Professor Tiago Filho Francisco da Costa

SÃO PAULO
2022

FACULDADE DE TECNOLOGIA DE SÃO PAULO

LEANDRO RODRIGUES DE SANTANA

AUTOMAÇÃO DE TESTES EM APIS REST NO CONTEXTO ÁGIL

Trabalho submetido como exigência parcial para a obtenção do Grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Parecer do Professor Orientador

_____

_____

_____

_____

_____

Conceito/Nota Final: _____

Atesto o conteúdo contido na postagem do ambiente TEAMS pelo aluno e assinada por mim para avaliação do TCC.

Orientador: Professor Tiago Filho Francisco Da Costa

SÃO PAULO, ____ de _____ de 2022.

Assinatura do Orientador **Assinatura do aluno**

RESUMO

Este trabalho irá analisar como a automação de testes de software influencia e contribui no desenvolvimento de software nas metodologias ágeis, abordando principalmente APIs construídas na arquitetura REST. Para isso, o desenvolvimento do trabalho é construído de modo a apresentar pontos focais sobre o tema abordado, tais como, metodologias ágeis, APIs REST, conceitos e técnicas de teste. Alguns exemplos de testes automatizados serão apresentados utilizando linguagens e frameworks diferentes. Por meio de estudos de casos, conclui-se que os testes automatizados agregam valor na entrega do produto no quesito de agilidade, flexibilidade e qualidade nas entregas.

ABSTRACT

This work will analyze how software testing automation influences and contributes to software development in agile methodologies, mainly addressing APIs built in REST architecture. For this purpose, the development of the work is built to present focal points on the topic addressed, such as agile methodologies, REST APIs, testing concepts and testing techniques. Some examples of automated tests will be presented using different languages and frameworks. Through case studies, it is concluded that automated tests add value in the delivery of the product in terms of agility, flexibility and quality in deliveries.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

A história do desenvolvimento de software possui grandes marcos, um deles aconteceu em 2001, quando foi publicado o Manifesto Ágil. Com a publicação do Manifesto Ágil, foram colocados no papel princípios que começavam a ser praticados com mais frequência dentro do desenvolvimento de software. Com a publicação do Manifesto Ágil, esse movimento ganhou mais força ganhando uma maior estruturação [2].

Esse foi um movimento de grande avanço no mercado, visto que com o desenvolvimento ágil foi possível ter um maior foco no cliente, desenvolvendo produtos de maneira mais rápida, gerando mais competitividade no mercado. [21]

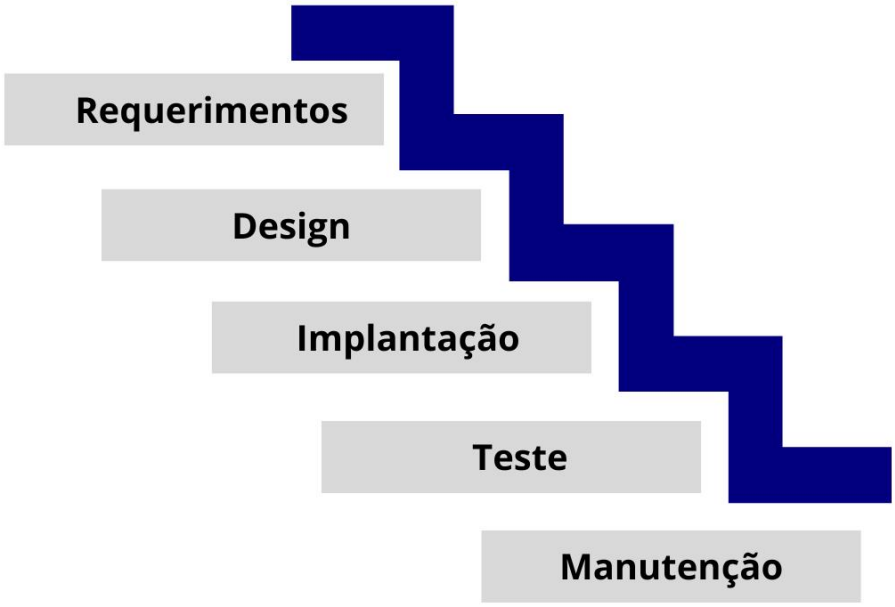
Com o desenvolvimento de software se expandindo, as APIs passaram a ser cada vez mais usadas, possibilitando um desenvolvimento distribuído e mais dinâmico. Hoje, as APIs estão presente no dia a dia tanto de desenvolvedores quanto de usuários, tornando-as um recurso valioso que necessita apresentar um bom funcionamento. [6]

Com isso, a garantia de qualidade tem um papel importante e crescente no desenvolvimento ágil, com os testes automatizados cada vez mais presentes, com o objetivo de realizar entregar de qualidade no menor tempo possível. [1]

2. MODELOS DE DESENVOLVIMENTO

O desenvolvimento de software é uma tarefa complexa, que exige a execução de diversos processos para um resultado de sucesso ser alcançado. Para realizar o desenvolvimento de forma organizada e eficiente, modelos de desenvolvimento, que consistem em um conjunto de processos a serem realizados, visando um mesmo objetivo. O mais famoso desses modelos é o Cascata (Watterfall), popularizado na década de 70, consiste na divisão do desenvolvimento em etapas bem definidas.[20]

Figura 1 - Etapas no modelo Cascata



Fonte: https://blog.betrybe.com/tecnologia/modelo-cascata/

Esse modelo apresentou um avanço muito significativo no desenvolvimento de software, mas possui diversos pontos negativos, dentre eles podemos citar:

- Dificuldade na conclusão da análise de requisitos, podendo ocasionar um grande atraso no desenvolvimento.
- Dificuldade de manutenção, uma vez em produção, seria muito caro e trabalhoso realizar modificações.

- Elevado tempo entre o início do projeto e a primeira versão do software, ocasionado pelo tipo de execução sequencial das atividades.

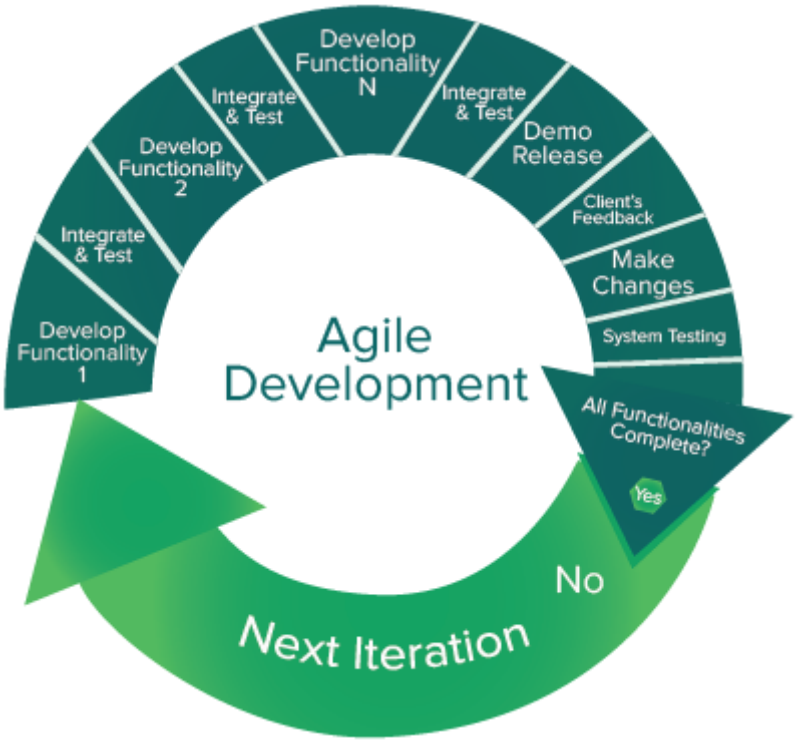
Com todas essas desvantagens, foram criados outros modelos para suprir as necessidades que o modelo cascata não atendia, podemos citar aqui os seguintes modelos:

- Modelo de prototipação - consistia em entender os requisitos do usuário e publicar uma versão simplificada do produto para validar os requisitos.
- Modelo Incremental - combinação do Cascata com o modelo de prototipação, nesse modelo, a cada “incremento” era realizado todo ciclo de desenvolvimento novamente.
- Modelo em Espiral - divide o ciclo de vida em quatro fases que são repetidas várias vezes. [5][19]

3. METODOLOGIA ÁGIL

Visando a melhoria nos processos de desenvolvimento, as metodologias ágeis entram como filosofias para resolver alguns pontos negativos que os diferentes modelos de desenvolvimento possuíam. Elas surgem com um foco no cliente, visando realizar uma entrega de valor em prazos curtos. Dessa forma, o ciclo de desenvolvimento tomou uma forma circular. [2][18]

Figura 2 - Ciclo do modelo ágil



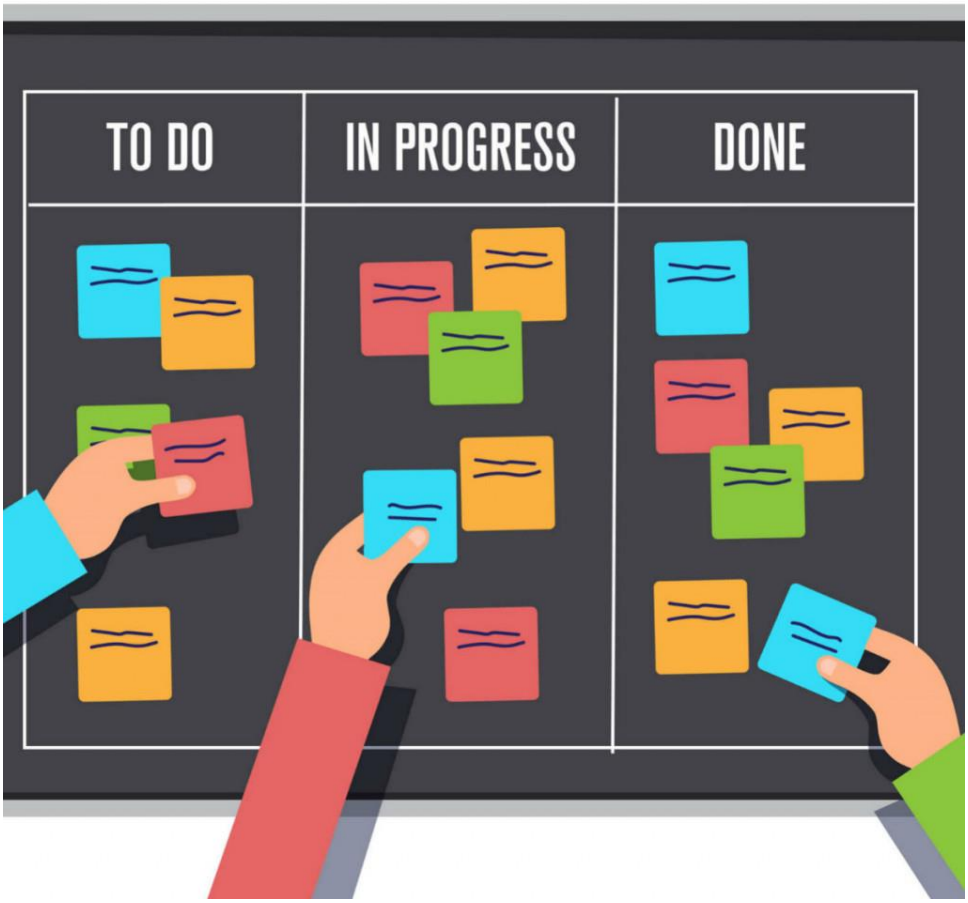
Fonte: https://www.iguansystems.com/agile-development-services.html

Dessa forma, o software vai sendo construído e testado aos poucos, a cada iteração um novo ciclo de desenvolvimento se dá início.

3.1 Kanban

O Kanban é uma das metodologias ágeis mais famosas no mercado atualmente. O Kanban é um sistema de gestão de trabalho que visa conduzir cada tarefa por um fluxo definido de trabalho. Nele, os cartões e colunas de definições são passos muito importantes que marcam essa metodologia. Os cartões representam uma tarefa ou uma ação, de acordo com o processo, o cartão vai sendo movido entre as colunas. As colunas representam os status de cada cartão, sendo as colunas padrões: “A fazer”, “Em execução” e “Pronto”. [17]

Figura 1 - Quadro Kanban



Fonte: https://blog.delogic.com.br/quais-sao-os-tipos-de-kanban-e-como-utilizar/

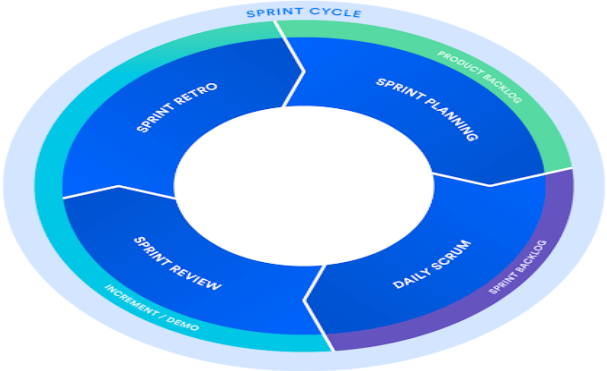
3.2 SCRUM

O Scrum é uma metodologia ágil que pode usar elementos do Kanban em seus processos, agregando a metodologia dentro de si. Porém, o Scrum possui suas próprias características, sendo uma delas a arquitetura de como é realizado o trabalho. A estrutura do Scrum é formada por Sprints, que são ciclos de trabalho bem definidos com começo e fim. Esses ciclos são compostos por cerimônias que marcam o início, fim e consequentemente o início de outro ciclo [18]. Essas cerimônias são:

- Sprint Review - Nesta cerimônia são apresentados os artefatos que foram produzidos ao longo da sprint, quais tarefas foram executadas e tudo que for relevante para os envolvidos.
- Sprint Retro - Nesta etapa são discutidos os impedimentos que aconteceram na sprint, o que pode melhorar, o que merece ser destacado e outros assuntos relacionados aos integrantes do time
- Sprint Planning - Cerimônia que inicia a sprint, nela são apresentadas as tarefas da sprint e os objetivos a serem cumpridos durante a sprint.

Além das três cerimônias que acontecem uma vez por sprint, há uma cerimônia diária chamada Daily Scrum, nesta cerimônia todos os membros do time descrevem as atividades que realizaram desde a última Daily e se há algum impedimento para a realização das tarefas. Sendo essa uma reunião rápida e diária, não deve durar mais do que 15 minutos. [8] [18]

Figura 4 - Ciclo de uma sprint Scrum



Fonte: https://www.atlassian.com/br/agile/scrum

4. APPLICATION PROGRAMMING INTERFACE

APIs são aplicações que permitem que dois ou mais serviços de software se comuniquem por meio de protocolos pré-estabelecidos. As APIs facilitam o desenvolvimento de software, permitindo que aplicações intercalem dados e funcionalidades de maneira rápida e segura, além disso, aceleram a inovação, já que os desenvolvedores podem construir aplicações baseadas em dados e funcionalidades já existentes. [6]

4.1 APIs REST

Este é um tipo de arquitetura de APIs muito utilizado, principalmente nas aplicações Web. A sigla significa Representational State Transfer (Transferência Representacional de Estado), nesse modelo um conjunto de métodos é definido para representar cada tipo de ação executada e um código de status informa qual o resultado da ação requisitada.

Geralmente, as APIs REST são orientadas a recursos, que podem ser recursos simples ou em conjunto (collection). Um recurso, representa um objeto que pode ser acessado pelo cliente. [6]

A troca de dados acontece utilizando o protocolo HTTP, por meio de URLs, essa comunicação funciona por meio de requisições, no modelo de cliente – servidor, a aplicação que envia a requisição é o cliente e a API faz o papel do servidor. Normalmente, o cliente acessa um endpoint, que age como uma interface entre a API e o consumidor, a requisição é processada, iniciando funções internas do servidor e uma resposta é retornada ao cliente. [12]

A requisição é composta por um conjunto de dados, que são dispostos separadamente dentro da requisição, onde são ser divididos por:

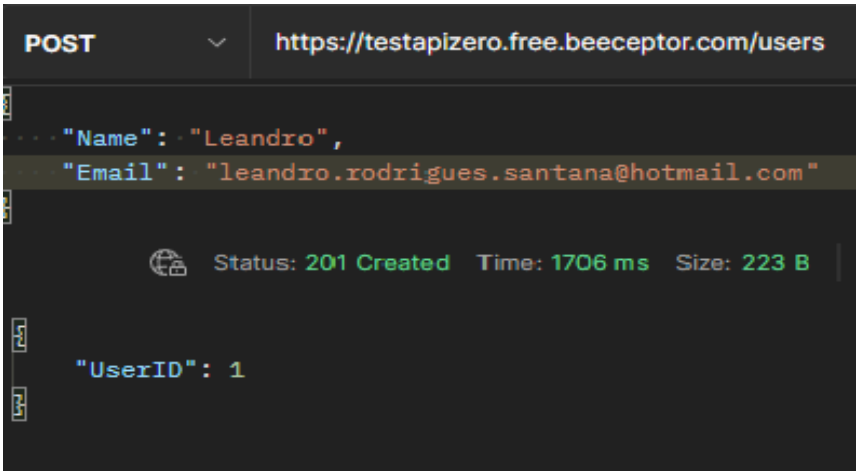
- Cabeçalho (Header) - Dados que podem conter a chave para utilizar a API, tipo do conteúdo, tamanho do conteúdo, origem, além de qualquer outra informação usada para enriquecer a requisição.
- Parâmetros (Query Params) - Parâmetros que podem ser utilizados quando a requisição é uma consulta, indicando dados específicos para a busca

- Corpo (Body) - Campo onde a mensagem da requisição será escrita, quando tratamos de APIs REST essa mensagem geralmente será escrita no formato JSON (JavaScript Object Notation)

Para enviarmos as requisições, utilizamos dos métodos, sendo os mais famosos POST, GET, PUT e DELETE. [7]

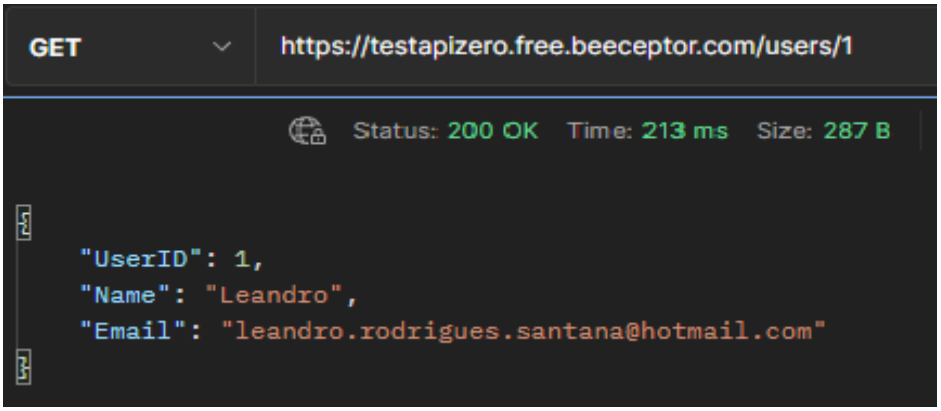
- POST - Método utilizado para manipular objetos, de modo a adicionar um recurso ou iniciar uma ação que resultará em um recurso criado.
- PUT - Método que atualiza um recurso já existente.
- GET - Método que retorna o conteúdo de um recurso.
- DELETE - Método utilizado para deletar recursos.

Figura 2 - Requisição POST



Fonte: Autor

Figura 6 - Requisição GET



Fonte: Autor

4.2 SEGURANÇA DAS APIS

Visando a segurança e proteção de dados, a comunicação com APIs geralmente necessita de algum tipo de autenticação. Essa autenticação pode ocorrer de diversas maneiras, mas os mais comuns são: [13]

- Tokens de Autenticação - São tokens utilizados para autorizar os usuários a realizar chamadas na API. Para obter o token, uma chamada é realizada em um endpoint específico, passando as informações necessárias para autenticação, como usuário e senha.
- Chaves de API – Ao contrátrio dos Tokens, não identificam um usuário específico, são utilizadas para verificar se uma aplicação possui os direitos para o uso da API.

5. TESTES DE SOFTWARE

Desde seu início, a tecnologia vem auxiliando no desenvolvimento de inúmeras áreas, com a crescente informatização, o software está cada vez mais presente na sociedade, desempenhando um papel importante no mundo atual. Com essa disseminação, os riscos e prejuízos que os softwares defeituosos podem causar aumentaram muito e, com isso, as atividades que visam diminuir os riscos ganharam mais importância. [1]

O processo de Garantia de Qualidade de Software, sempre esteve presente no desenvolvimento e, assim como o processo de desenvolvimento em si, o processo de qualidade também passou por modificações e atualizações ao longo dos anos. Dentro da Garantia de Qualidade temos todos os processos que fazem parte de verificação e validação, desde a revisão das especificações ao teste de execução do produto.

Atualmente, a atividade de teste não mais se limita a executar o software e realizar as validações. Dentro do ambiente ágil, o teste está presente em todas as fazes do desenvolvimento, não só no final. Dentro do processo de teste, podemos incluir diversas atividades, como o planejamento de testes, criação dos cenários de teste, criação das massas de teste, implementação dos testes, relatórios e avaliação de qualidade. [4] [11]

Tendo as metodologias ágeis como base, um manifesto de teste ágil foi criado por Samantha Laing e Karen Greaves.

Figura 7 - O Manifesto de Teste

Fonte: https://www.growingagile.co.za/2015/04/the-testing-manifesto/

Com o manifesto é possível ter uma compreensão maior de como o processo de teste é realizado no ambiente ágil. Um dos pontos do manifesto, propõe que a responsabilidade pela qualidade não é de uma pessoa só, mas sim, do time como um todo. Visando este propósito, o teste pode ser utilizado como uma ferramenta que auxilia no desenvolvimento. [8]

5.1 TIPOS DE TESTES

Os testes podem ser divididos em alguns tipos, cada tipo de teste representa um grupo de atividades definidas para validar algumas características específicas de um sistema.

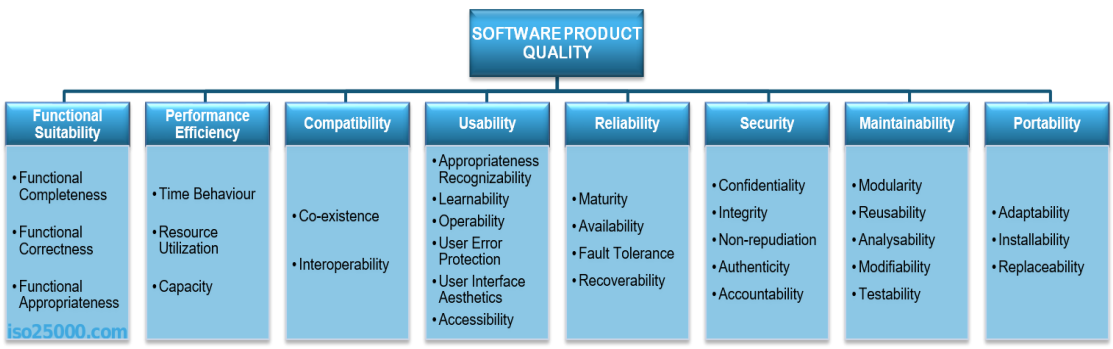
5.1.1 TESTE FUNCIONAL

Os testes funcionais são aqueles que envolvem a verificação das funcionalidades do software, com base em um conjunto de pré-requisitos estabelecidos pela regra de negócio. Para esses requisitos, podem ser usados casos de uso, especificações de requisitos e outros documentos que indiquem quais funções o software deve realizar. [1]

5.1.2 TESTES NÃO FUNCIONAL

Os testes não funcionais estão relacionados as características comuns de um software, como por exemplo usabilidade, performance e segurança. A ISO 25010 indica quais são as características de qualidade de um software. [1] [10]

Figura 8 - Características de qualidade de um software



Fonte: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

6. AUTOMAÇÃO DE TESTES

Dentro do ambiente ágil, todo o desenvolvimento acontece de forma mais rápida. A cada pequeno ciclo um novo artefato deve estar pronto para produção, com alguma mudança, mesmo que pequena, da sua última versão. Porém, como abordado anteriormente, sempre há riscos em uma nova versão de software. Todo o processo manual de teste poderia ser aplicado para diminuir esses riscos, mas pensando no ambiente ágil, os testes devem acompanhar a velocidade de desenvolvimento. [3]

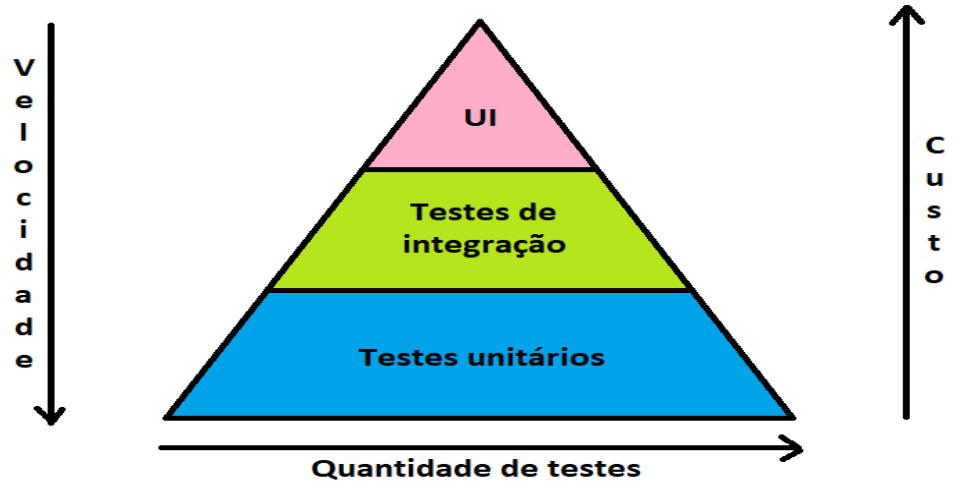
Pensando em uma abordagem mais rápida, sem comprometer a qualidade de entrega do produto, os testes automatizados auxiliam no processo de teste no quesito de custos e tempo, justamente quando testes de regressão são necessários. Esses testes visam garantir que as funcionalidades já implementadas não foram comprometidas com a nova atualização do software. [1]

A automação pode ser utilizada tanto para realizar processos repetitivos de forma rápida e assertiva, garantindo que sempre sejam executados da mesma maneira, quanto realizar processos complexos que exigiriam muitas etapas manuais.

Os testes automatizados também fornecem um retorno rápido para os envolvidos, gerando relatórios assim que os testes são executados, essa documentação pode ser utilizada para auxiliar tanto em uma possível reparação de um problema, quanto pode ser utilizada como um critério de aceite para o software ir para produção. [11]

No estudo dos testes automatizados, frequentemente é utilizada a abordagem proposta por Mike Cohn, que criou uma pirâmide de testes estabelecendo uma relação entre tempo, custo e volume de testes. A pirâmide é composta por três níveis, tendo em sua base os testes unitários, seguidos por testes integrados e, por fim, testes de interface gráfica. [22]

Figura 9 - Pirâmide de testes



Fonte: https://www.alura.com.br/artigos/por-que-e-o-que-e-possivel-testar

De acordo com a pirâmide, os testes unitários são os mais baratos e mais rápidos em sua execução, sendo mais isolados e se concentrando apenas no código a ser executado. À medida que subimos na pirâmide, o custo eleva e a velocidade de execução também, até chegamos nos testes de interface gráfica. [22]

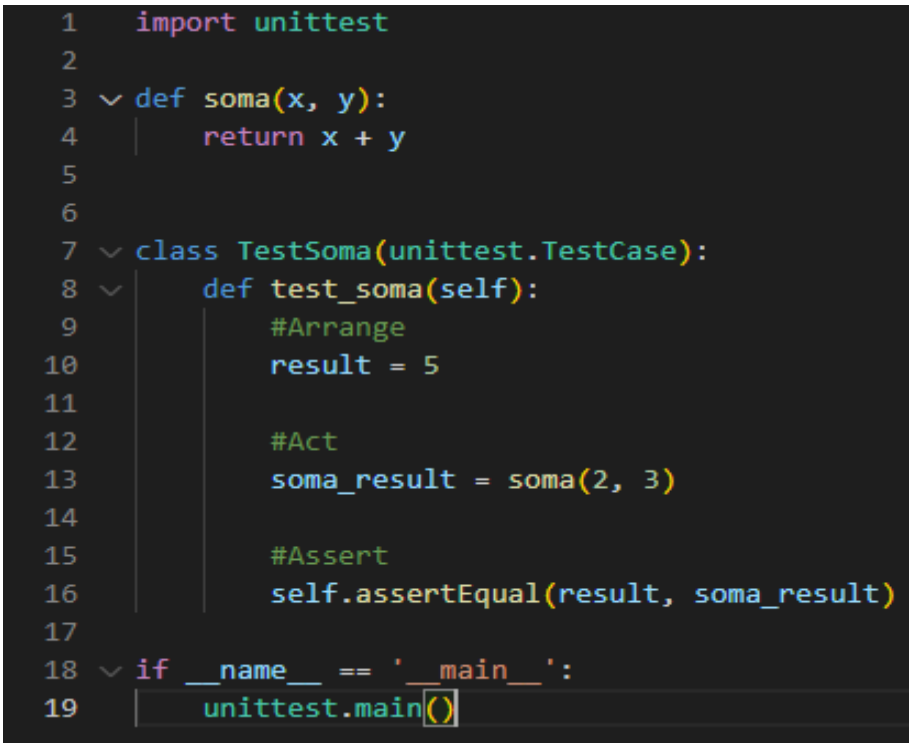
6.1 TESTES UNITÁRIOS

Os testes unitários são utilizados para testar pequenos trechos de código chamados de métodos, a menor unidade de código no desenvolvimento orientado a objeto. [3]

O objetivo desse tipo de teste é validar que um método está realizar exatamente a função que deveria realizar. Esses testes fazem parte do repositório principal do software e geralmente são organizados na estrutura: Arrange, Act, Assert. [9]

O teste unitário pode ser realizado com o auxílio de vários frameworks, para diversas linguagens diferentes. A figura abaixo representa um exemplo utilizando a linguagem Python com o framework Unittest.

Figura 10 - Teste unitário em Python



```python
import unittest

def soma(x, y):
    return x + y


class TestSoma(unittest.TestCase):
    def test_soma(self):
        #Arrange
        result = 5

        #Act
        soma_result = soma(2, 3)

        #Assert
        self.assertEqual(result, soma_result)

if __name__ == '__main__':
    unittest.main()
```

Fonte: Autor

No exemplo acima, o método soma deve retornar a soma de dois números. Para o teste, uma classe de teste é criada, dentro desta classe um método de teste é criado, dentro do método há a preparação dos dados (Arrange), a ação que deve ser testada (Act) e a validação dos dados obtidos (Assert).

A execução deste tipo de teste acontece de maneira muito rápida, visto que não é necessária nenhuma integração ou processo de entrada e saída. Quando alguma integração é necessária, como a leitura de um banco de dados, por exemplo, pode-se usar os mocks, que simula o comportamento de outros objetos independentes do método em teste. [9] [11]

6.2 TESTES DE INTERFACE GRÁFICA

Os testes de Interface Gráfica (UI) representam os testes que realizam as verificações e validações na interface gráfica do software a ser testado, tendo como objetivo realizar o teste como se um usuário estivesse utilizando o sistema. [1]

Como observado na pirâmide de testes anteriormente, esse tipo de teste leva mais tempo tanto em sua construção quanto em sua execução, consequentemente

tendo um custo mais elevado para ser realizado. Em contrapartida, esse tipo de teste pode oferecer ganhos relacionados ao funcionamento da aplicação como um todo, visto que o teste é realizado pela ótica do usuário, ou seja, haverá integração com todos os sistemas que compõe o software.

Os testes UI podem ser realizados utilizando de frameworks no code, que realizam a gravação das ações e reproduzem os passos executados, mas esta não é a maneira mais usada de criar os testes, visto que os testes criados desse modo apresentam problemas de performance, manutenção e flexibilidade.

Dessa forma, os frameworks que necessitam de código são mais utilizados na construção de testes UI. Os frameworks acabam mudando de acordo com o tipo de sistema a ser testado. Podem ser citados como exemplo os frameworks: Selenium, Cypress e Playwright para sistemas Web, AutoIt para aplicações desktop e Appium para aplicações mobile. [11]

Tendo em vista que os testes UI tendem a serem construídos e executados da ótica do usuário, a metodologia BDD (Behavior Driver Development) é frequentemente usada para construir e documentar estes testes.

O BDD utiliza da linguagem Gherkin para interpretar os cenários de teste. Com essa linguagem, são escritos cenários seguindo um modelo específico, onde são usadas palavras chaves para descrever os passos a serem executados. Segue abaixo um exemplo de um cenário descrito utilizando a linguagem Gherkin. [3]

Figura 11 - Cenário de teste utilizando BDD

```
#language: pt-BR
Funcionalidade: Login Fatec

Cenario: Login Sucesso
    Dado que acesso o site da Fatec São Paulo
    E insiro o numero de matricula 16200325
    E insiro a senha 1
    Quando clico no botão Acessar Sistema
    Então a pagina do portal do aluno deve ser exibida
```

Fonte: Autor

Tendo este cenário descrito, pode-se usar um framework BDD para a interpretação ser realizada, assim como a escrita dos passos e execução do teste. A classe abaixo representa a definição de passos a serem executados.

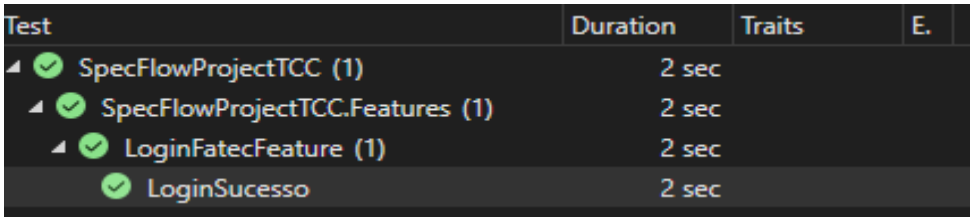
Figura 12 - Classe de passos para cada ação do cenário de teste

```
{
    [Binding]
    0 references
    public class LoginFatecStepDefinitions
    {
        IWebDriver _driver = null;
        LoginPage _loginPage = null;

        [Given(@"que acesso o site da Fatec São Paulo")]
        0 references
        public void GivenQueAcessoOSiteDaFatecSaoPaulo()
        {
            _loginPage = new LoginPage(_driver);
            _loginPage.AccessLoginPage();
        }

        [Given(@"insiro o numero de matricula (.*)")]
        0 references
        public void GivenInsiroONumeroDeMatricula(int login)
        {
            _loginPage.InputLogin(login);
        }

        [Given(@"insiro a senha (.*)")]
        0 references
        public void GivenInsiroASenha(int password)
        {
            _loginPage.InputPassword(password);
        }

        [When(@"clico no botão Acessar Sistema")]
        0 references
        public void WhenClicoNoBotaoAcessarSistema()
        {
            _loginPage.ClickOnAccessSystem();
        }

        [Then(@"a pagina do portal do aluno deve ser exibida")]
        0 references
        public void ThenAPaginaDoPortalDoAlunoDeveSerExibida()
        {
            string welcomeMessage = _loginPage.GetWelcomeMessage();
            string expectedMessage = "Bem vindo LEANDRO RODRIGUES DE SANTANA";
            Assert.AreEqual(expectedMessage, welcomeMessage);
            _loginPage.CloseDriver();
        }
    }
}
```

Fonte: Autor

Para complementar o teste, foi utilizado o framework Selenium, utilizando o padrão Page Object para organização do projeto. Com os passos definidos, pode-se utilizar um framework como o NUnit para executá-lo.

Figura 13 - Resultado da execução dos testes UI



Fonte: Autor

6.3 TESTES DE API

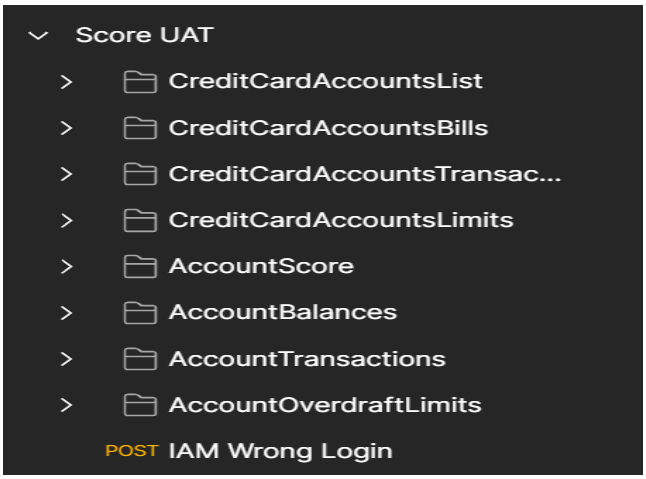
Na camada de testes de serviço temos os testes de API, com a utilização crescente das APIs, principalmente em aplicações Web, esses testes acabaram ganhando mais espaço, tendo um crescimento de frameworks que auxiliam nesse tipo de teste. [11]

Para os testes de API pode-se usar uma heurística com o acrônimo VADER. Criada por Stuart Ashman, está heurística tem como objetivo apresentar o que deve ser testado dentro de uma API. Abaixo, uma explicação do que cada sigla significa: [14]

- V - Verbos - São os métodos HTTP, dentro desse escopo de testes, são realizados testes utilizando os métodos para realizar as chamadas na API, validando o resultado esperado para cada endpoint de acordo com o verbo utilizado.
- A - Autenticação/Autorização - Nesse escopo devem ser realizados testes em toda parte de autenticação da API, independentemente do tipo de autorização a ser utilizada.
- D - Dados - Aqui são realizados testes com foco nos dados da API, observando se tanto os dados que são recebidos quanto os que são retornados, estão de acordo com as regras de negócio.
- E - Erros - Nessa parte, os testes são focados em validar os tipos de erro que a API retorna, tendo um foco maior em validar se os códigos de erro estão sendo retornados corretamente.
- R - Responsividade - Nesta etapa, são tratados todos os comportamentos que a API apresenta quando em situações adversas e/ou de mudanças rápidas. Os testes de desempenho da API entram nesta etapa.

Para criar e executar os testes de API, podem ser usados diversos frameworks, um dos mais famosos é o Postman. Neste framework os testes são escritos utilizando JavaScript e podem ser executados dentro da aplicação do Postman. A seguir, será apresentado um modelo de automação de testes com o Postman. [16]

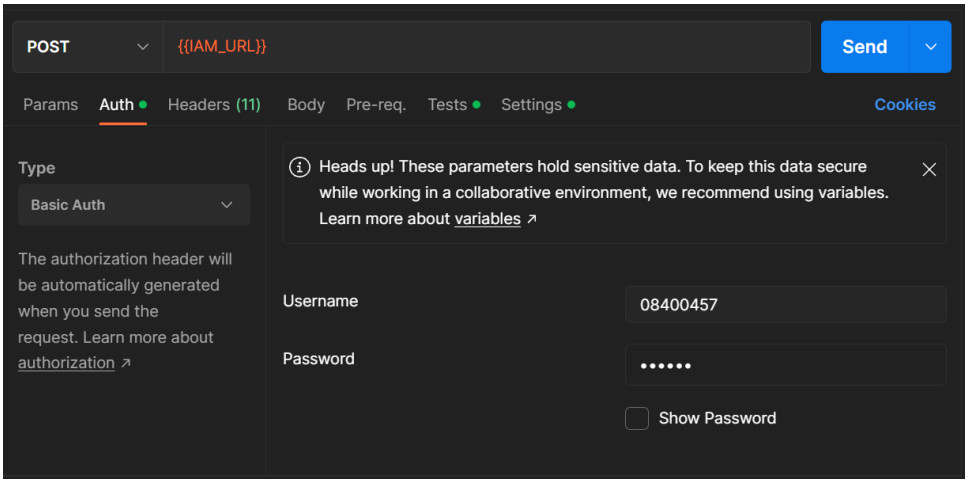
Figura 14 - Coleção de requisições no Postman



Fonte: Autor

Primeiramente, as requisições são organizadas em pastas, separadas por conjuntos de dados que são enviados para a API. Cada pasta contém um ou mais requisições que representam um cenário de teste cada.

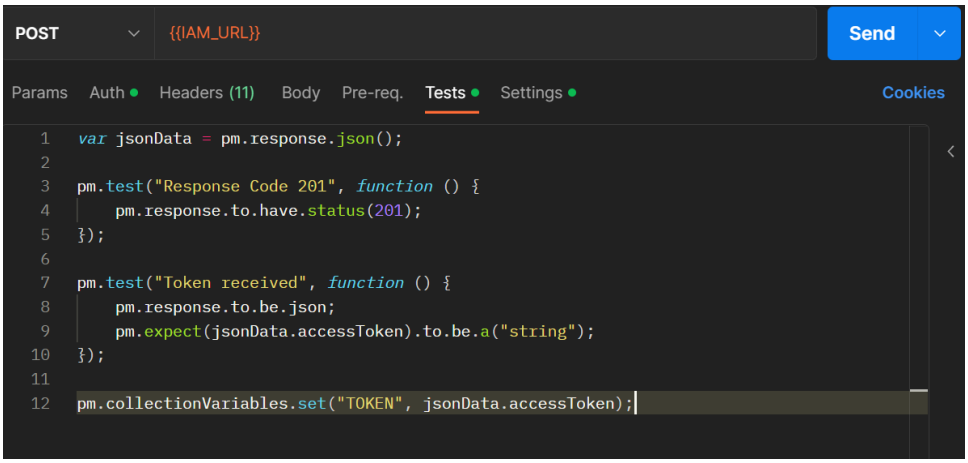
Figura 15 - Requisição de autorização



Fonte: Autor

Uma requisição de autorização é realizada para obtenção do token de acesso, que deve ser utilizado nas demais requisições. Mesmo essa sendo uma requisição de autorização, já são realizados testes, como pode ser observado na aba “Tests”.

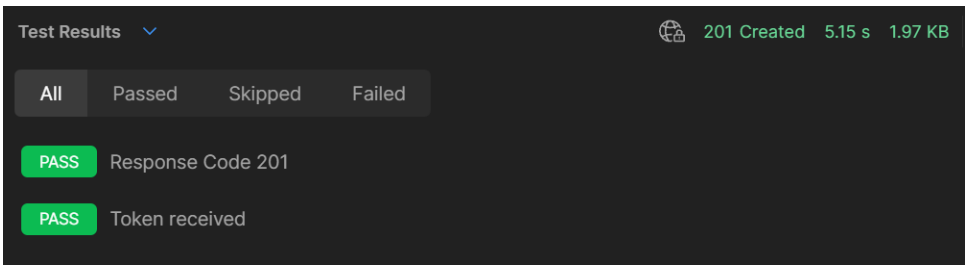
Figura 16 - Testes da requisição de autorização



Fonte: Autor

Nessa aba, os testes são escritos em JavaScript, utilizando a biblioteca do postman para realizar as validações. Cada trecho de código que inicia com “pm.test” representa um teste dentro da requisição. Outras funções da linguagem também são utilizadas, como o armazenamento da resposta em uma variável e o armazenamento do valor do campo “TOKEN” dentro de uma variável de coleção (global).

Figura 17 - Resultado dos testes da requisição de autorização



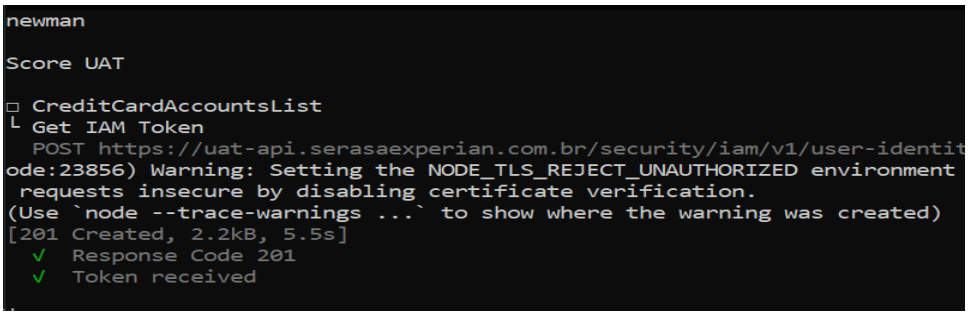
Fonte: Autor

Ao ser realizada a requisição, os resultados dos testes podem ser observados na aba “Test Results”.

Para a execução de todos os cenários de teste sem a necessidade de uma interface gráfica, é utilizado o framework Newman. Para isso, a coleção contendo os testes deve ser exportada em formato Json. Após isso, a execução pode ser realizada através de uma linha de comando: [16]

newman run “collection_name.postman_collection.json” -r cli,htmlextra

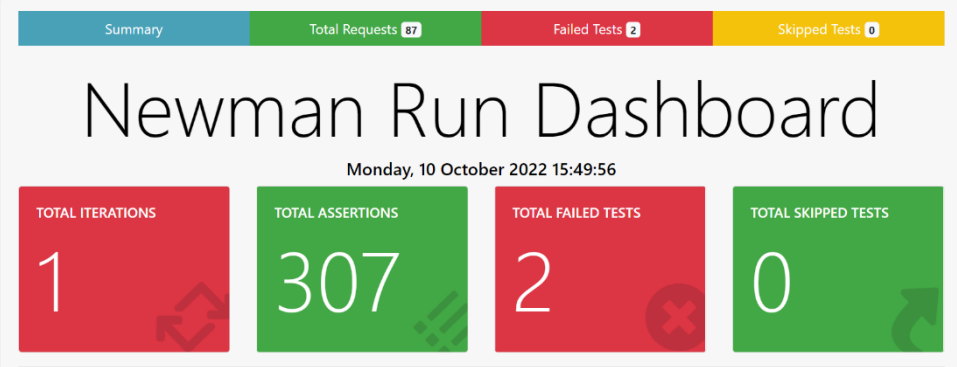
Figura 18 - Testes sendo executados via linha de comando utilizando Newman



Fonte: Autor

A execução inicia, exibindo os resultados de teste à medida que as requisições são executadas. Com os parâmetros de execução “htmlextra” é possível observar os resultados dos testes em um relatório html.

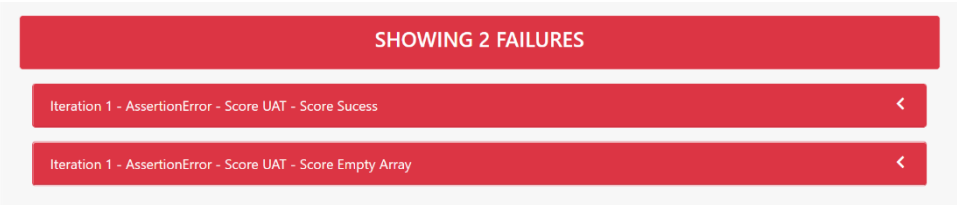
Figura 19 - Relatório dos testes: visão geral da execução



Fonte: Autor

Na aba inicial é apresentado um resumo da execução, com o total de iterações, total de asserções realizas, total de testes que falharam e total de testes que não foram executados.

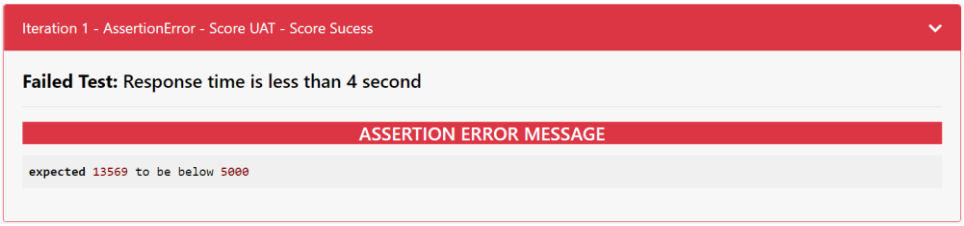
Figura 20 - Relatório dos testes: visão dos testes que falharam



Fonte: Autor

Entrando na aba de testes que falharam, são apresentados em qual iteração falharam, que tipo de falha ocorreu e o nome da requisição.

Figura 21 - Relatório dos testes: visão em detalhe de um teste que falhou



Fonte: Autor

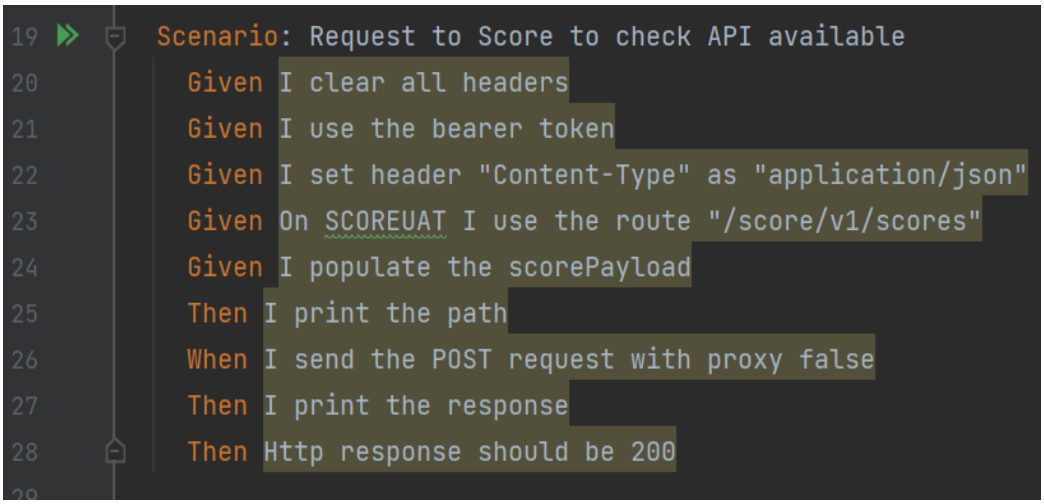
Entrando no detalhe de cada requisição, é possível observar o porquê de o teste ter falhado.

Figura 22 - Relatório dos testes - visão geral de testes com sucesso e sem sucesso

Fonte: Autor

Como abordado anteriormente, os testes de API podem utilizar diferentes frameworks, a seguir pode-se observar um exemplo da mesma API sendo testada com um projeto construído na linguagem Java, utilizando os frameworks RestAssured para realizar as requisições e Cucumber para os cenários em BDD.

Figura 23 - Cenário de teste utilizando BDD no IntelliJ



Fonte: Autor

O cenário é escrito em arquivo “feature”, para ser lido pela classe principal que irá realizar os testes.

Figura 24 - Exemplo de uma requisição POST utilizando Java e RestAssured

Fonte: Autor

Para cada sentença, uma função que representa o passo a ser executado é criada. Cada passo pode ser reutilizado dentro dos cenários.

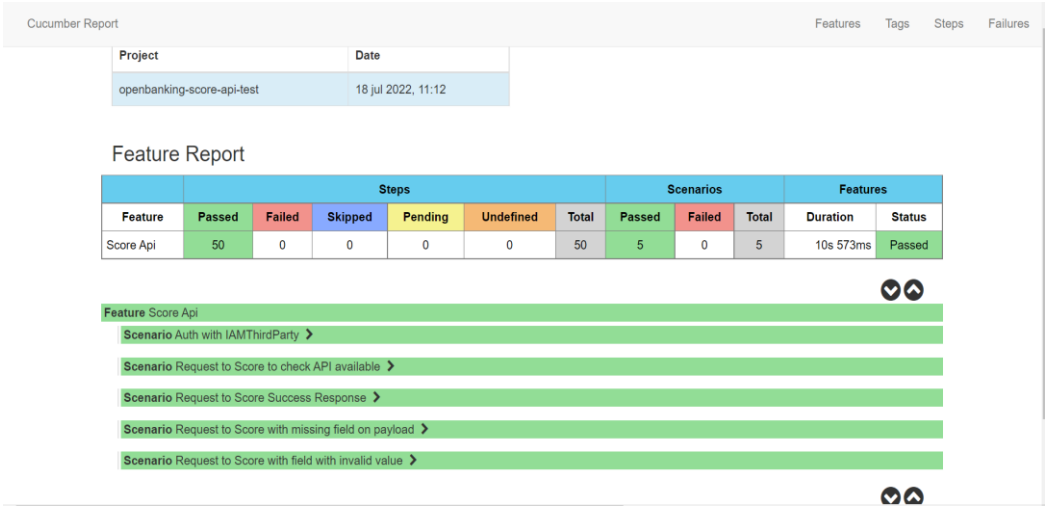
Figura 25 - Comando para execução de testes com Cucumber

```
try {
        cucumber.api.cli.Main.run(arguments, Thread.currentThread().getContextClassLoader());
} catch (Throwable e) {
        e.printStackTrace();
}
ReportJson.generateFinalReport(reportDir, reportDir,  projectName: "openbanking-score-api-test");
String reportAbsPath = new File(reportDir).getAbsolutePath();
String report = reportAbsPath + "/cucumber-html-reports/overview-features.html";
```

Fonte: Autor

Uma classe principal é criada, contendo os comandos necessários para executar os testes e criar um relatório com o resultado das execuções.

Figura 26 - Relatório Cucumber: visão geral dos testes realizados



Fonte: Autor

Após a execução um relatório é gerado com as informações de cada cenário executado.

7. CONCLUSÃO

Com a crescente utilização das metodologias ágeis no desenvolvimento de software, foi necessária uma adaptação na parte de testes e qualidade de software. Dessa forma, os testes automatizados ganharam espaço e continuam evoluindo, tendo como objetivo amenizar os riscos de eventuais defeitos no software, assim como fornecer um feedback rápido de qualidade do produto.

Com o que foi apresentado, é possível observar que os testes automatizados em APIs são de grande importância no quesito de qualidade. Pode-se observar que, com uma suíte de testes robustas, é possível identificar erros que podem aparecer com uma eventual mudança na API. Conforme a API vai evoluindo, é necessário realizar a manutenção nos casos de testes, assim como incluir mais casos de testes que contemplem as novas funcionalidades da API.

O projeto deve ser construído e organizado de maneira a facilitar essa manutenção, afinal, a manutenção da massa de dados pode apresentar um desafio caso não haja uma estratégia que facilite a mudança de massa de teste.

Com isso em mente, os testes automatizados em APIs são de grande valor para economizar tempo e recurso, sendo complementares aos outros testes que compõem a pirâmide de testes. Se construídos, organizados e com manutenção periódica, são testes indispensáveis em sistemas que são, compõe ou utilizam de APIs.

8. REFERÊNCIAS

1. BSTQB**. Certified Tester Syllabus – Agile Tester**, 2014. BSTQB, 2014. Disponível em: https://bstqb.org.br/b9/doc/syllabus_ctfl_at_2014br.pdf. Acesso em: 09 out. 2022.

2. BECK, Kent. et al. **Princípios por trás do Manifesto Ágil**. [S. I.], 2001. Disponível em: https://agilemanifesto.org/iso/ptbr/principles.html. Acesso em: 10 nov. 2022.

3. ASTELS, David. **Test-Driven Development**: A Practical Guide. New Jersey: Prentice Hall, 2003.

4. BSTQB. **Certified Tester Syllabus – Foundation Level**, 2018. BSTQB, 2018. Disponível em: https://bstqb.org.br/b9/doc/syllabus_ctfl_3.1br.pdf. Acesso em: 09 nov. 2022.

5. CENÊZ, Adonai. **Modelos de Ciclo de Vida de Software**. Adonai, 2014. Disponível em: https://www.adonai.eti.br/2014/01/modelos-de-ciclo-de-vida-de-software/. Acesso em: 10 nov. 2022.

6. POETKET, Bridget. **What Is an API? How APIs Improve Application Development.** G2, 2022. Disponível em: https://www.g2.com/articles/what-is-an-api. Acesso em: 10 nov. 2022.

7. NOLLE, Tom. **The 5 essential HTTP methods in RESTful API development. Techtarget**, 2021. Disponível em: https://www.techtarget.com/searchapparchitecture/tip/The-5-essential-HTTP-methods-in-RESTful-API-development. Acesso em: 11 nov. 2022.

8. GREAVES, Karen; LAING, Samantha**. Growing Agile**: A Coach's Guide to Agile Testing. [S. I.]: Growing Agile, 2015.

9. CRISPIN, Lisa; GREGORY, Janet. **Agile Testing Condensed**: A Brief Introduction. Canada: Government of Canada, 2019.

10. International Organization for Standardization. ISO/IEC 25010. ISO 25000. Disponível em: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010. Acesso em 02 dez. 2022.

11. ISTQB. **Certified Tester Advanced Level**: Test Analyst. ISTQB, 2012. Disponível em: https://istqb.com/media/documents/ISTQB_CTAL-TA_Syllabus_v3.1.2.pdf. Acesso em: 12 nov. 2022.

12. Microsoft. **Design da API Web RESTful**. Microsoft, 2022. Disponível em: https://learn.microsoft.com/pt-br/azure/architecture/best-practices/api-design. Acesso em: 22 nov. 2022.

13. Google. **Por que e quando usar chaves de API**. Google, 2021. Disponível em: https://cloud.google.com/endpoints/docs/openapi/when-why-api-key. Acesso em: 10 nov. 2022.

14. ALVES, Maximiliano. **VADER: Heurística para teste de API na prática**. Medium, 2020. Disponível em: https://maximilianoalves.medium.com/vader-heuristica-para-teste-de-api-na-pratica-fcf78c6acec. Acesso em: 08 dez. 2022.

16. Postman. **Writing tests**. Learning Postman, 2022. Disponível em: https://learning.postman.com/docs/writing-scripts/test-scripts/. Acesso em 27 nov. 2022.

17. CAVALLARI, Rodrigo. **Quais são os tipos de Kanban e como utilizar?**. Delogic, 2019. Disponível em: https://blog.delogic.com.br/quais-sao-os-tipos-de-kanban-e-como-utilizar/. Acesso em 07 dez. 2022.

18. DRUMOND, Claire. **Scrum**. Atlassian, 2019. Disponível em: https://www.atlassian.com/br/agile/scrum. Acesso em 06 dez. 2022.

19. PRESSMAN, Roger S. **Engenharia de software**: uma abordagem profissional. Tradução por Ariovaldo Griesi. 7. Ed. Porto Alegre: AMGH, 2011.

20. NOLETO, Cairo. **Modelo cascata:** o que é e por que está ultrapassado?. Betrybe, 2020. Disponível em: https://blog.betrybe.com/tecnologia/modelo-cascata/. Acesso em: 06 out. 2022.

21. FIA. **Desenvolvimento Ágil: o que é, princípios e como aplicar**. FIA, 2019. Disponível em: https://fia.com.br/blog/desenvolvimento-agil/#:~:text=O%20desenvolvimento%20%C3%A1gil%20%C3%A9%20uma%20abordagem%20em%20que%20softwares%20s%C3%A3o,%2C%20de%20certa%20forma%2C%20limitada. Acesso em: 10 nov. 2022.

22. LARISSA, Gabriela. **Por que e o que é possível testar?.** Alura, 2021. Disponível em: https://www.alura.com.br/artigos/por-que-e-o-que-e-possivel-testar. Acesso em: 25 nov. 2022.