Desenvolvimento de Aplicação Web para Teste Automatizado de Requisições REST

Rui Meirelles Magalhães
ruimaga.meirelles@gmail.com

Claudinei Di Nuno, MSc
professorcleudinei@uol.com.br

Curso de Pós-Graduação *Lato Sensu* em Desenvolvimento Orientado a Objeto com Java
Universidade Estácio de Sá

Resumo
As organizações para sustentarem seus negócios e conseguirem atender a dinâmica dos avanços tecnológicos, precisam de aplicações mais integradas as informações corporativas, que quase sempre se encontram disponibilizadas em sistemas legados com arquitetura monolítica. Com isso, toda tentativa de modernização sistêmica precisa ser conduzida de uma forma que possibilite a incorporação e o compartilhamento das novas implementações aos fluxos de negócios dos antigos sistemas. Com este fim, surge a Arquitetura Orientada ao Serviço (SOA — *Service-Oriented Architecture*), que logo se destaca devido ao advento de serviços Web baseada em padrões, que simplificam a interoperabilidade e o reaproveitamento de código. Tais serviços, conhecidos como APIS (*Application Programming Interface* — Interface de Programação de Aplicações), representam um componente de Software capaz de compartilhar operações na Web entre diferentes sistemas. Nesse contexto, esse artigo teve como iniciativa o desenvolvimento de um *Software* que pudesse auxiliar a grande demanda de testes em aplicações que se utilizam do modelo de arquitetura *REST* (*Representational State Transfer* — Transferência de Estado Representacional). Atualmente, os testes para identificação de defeitos em APIS, quase sempre são realizados de forma manual, que dependendo da quantidade de validações, tornam essas atividades por demais onerosas, cansativas e com grande possibilidade de falha. Portanto, fica evidente a necessidade de novos modelos de testes automatizados para essa finalidade. Nesse sentido, a aplicação desenvolvida se propõe a realizar testes em APIS do estilo REST, que se utilizam dos métodos PUT, GET, DELETE e POST do protocolo HTTP e que somente tenham como resposta o formato JSON *(JavaScript Object Notation* — Notação de Objetos JavaScript). Essa aliciação Web foi desenvolvida na linguagem Java com base no *Framework Spring*.

Palavras-chave: Serviços Web, API REST, *SOAP*, Arquitetura Orientada a Serviços (SOA), Sistemas Legados.

1. Introdução

É consenso entre os grandes especialistas de tecnologia, que o mundo mudou. A revolução digital com adoção completa da Internet vem proporcionando novas possibilidades para os mais diversos setores produtivos da sociedade moderna, como exemplo: indústria, serviços, comércio, saúde e educação. Alguns especialistas chegam até a dizer, que tal mudança assemelha-se a grande revolução industrial, devido ao tamanho das alterações e sua importância para comunicação e interação entre as pessoas. Com este avanço, a Web, que inicialmente só fornecia conteúdo estático, ou seja, para qualquer tipo de alteração do código fonte deveria ser atualizado, atualmente disponibiliza novas tecnologias possibilitando assim,

que os servidores Web também distribuam conteúdos dinâmicos a partir de requisições HTTP (*Hypertext Transfer Protocol* — Protocolo de Transferência de Hipertexto).

Nesse contexto surgiu um novo modelo de Arquitetura orientada a serviços, denominada SOA (*Service Oriented Architecture*), que se utiliza de *Web Services* (Serviços Web) para transportarem dados entre aplicações desenvolvidas nas mais diversas linguagens e plataformas. Esses Serviços podem ser considerados aplicativos que são amplamente difundidos e auto direcionados, sendo encontrados, conectados e integrados via Internet. O valor agregado consiste em reuni-los em grandes aplicações cujas funções de negócio podem ser utilizadas e exibidas através deles. Os consumidores de serviços podem construir aplicativos poderosos aproveitando os benefícios desses serviços através de padrões de comunicação. Existem dois tipos de Web Service, SOAP (*Simple Object Access Protocol* — Protocolo de acesso a objetos simples) e REST (*Representational State Transfer*— Transferência de Estado Representacional). O Protocolo SOAP baseia-se em XML (*Extensible Markup Language* — Linguagem de Marcação Extensível), que permite a transferência e o compartilhamento de dados estruturados com outras aplicações. O modelo REST, que será o foco principal desse artigo, possui uma arquitetura mais flexível podendo ser incorporada em aplicações distribuídas também através do protocolo HTTP, mas se utilizando de outros formatos além do XML, como: JSON e TEXT (Texto Livre sem padrões).

Com o surgimento dessa nova arquitetura, os sistemas legados monolíticos estão sendo modificados, e incorporados a eles Serviços Web. Com a presença desses serviços, surge uma nova preocupação - os Testes. Como realizar testes dos recursos de uma API REST? Um teste de uma requisição tem como finalidade os seguintes controle e validações: estrutura dos dados transportado e o código do *status* de retorno de sucesso ou falha da requisição. Algumas ferramentas com esta finalidade já são disponibilizadas no mercado, mas em sua grande maioria, pagas e outras bastantes limitadas, como: SoapUI e Postman.

Assim sendo, este artigo se objetiva apresentar uma aplicação para realizar testes automatizados em APIS REST desenvolvida na linguagem Java. Na sequência, com o intuito de agregar valor e fornece uma forma simples, mas precisa, de testes automatizados, serão apresentados as funcionalidades, os componentes de *Software* envolvidos, ambiente de desenvolvimento, além dos diferenciais de outras ferramentas.

2. Fundamentação Teórica

2.1 Sistema Legado

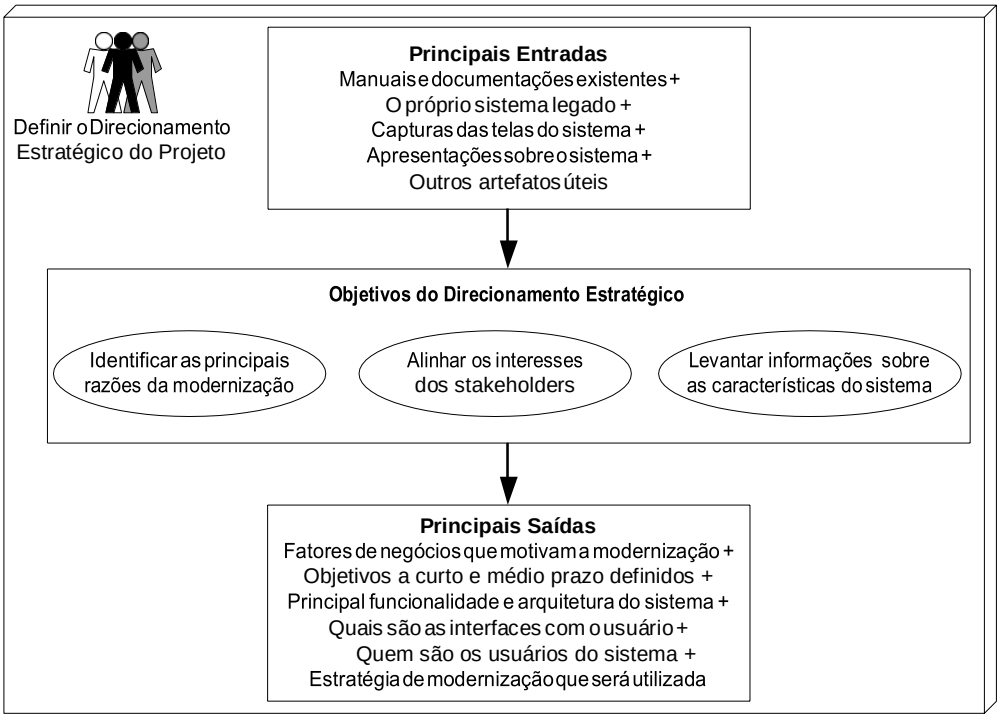
O termo legado geralmente refere-se a conhecimentos repassados entre gerações distintas e anteriores. Normalmente entende-se como algo que foi desenvolvido há muito tempo, com tecnologias e métodos totalmente ultrapassadas, e de difícil manutenção. Alguns autores assinalam que esses sistemas constituem-se um grande problema para organizações, pois suas operações não podem ser descontinuadas, o que dificulta o esforço para evolução do sistema. (FREITAS, 2017, p.25).

Muitos dos Sistemas legados foram desenvolvidos e mantidos em ambientes de grande porte e fortemente baseados no modelo relacional de dados, que necessitam de informações semiestruturadas para comunicação com Sistemas externos. Nesse modelo, a migração de banco de dados para versões mais atuais, quase sempre devido à alta complexidade, se torna inviável. (FLÔRES, 2019, p.12).

Dayani-Fard e seus colegas [Day99] descrevem *Software* Legado da seguinte maneira: “Sistemas de *Software* Legado... foram desenvolvidos décadas atrás e tem sido continuamente modificado para se adequar ás mudanças dos requisitos de negócio e a plataformas computacionais. A proliferação de tais sistemas está causando dores de cabeça

para grandes organizações que os consideram dispendiosos de manter e ariscados de evoluir". (PRESSMAN, 2016, pg.08).

Figura 1 – Exemplo de uma definição de Estratégia de Modernização



Fonte: (AGILAR, 2016, p.34)

Como direcionamento estratégico para modernização de Sistemas legado, as organizações precisam realizar longos debates entre Gestores, Usuários e todos envolvidos (*stakeholders*). Esse debate deverá ter como foco de discussão a real necessidade da modernização, como também, a elaboração da estratégia a ser adotada. (AGILAR, 2016, p.33).

2.2 Arquitetura Orientada a Serviços (SOA)

Os Serviços Web surgiram como uma evolução dos modelos de computação distribuídos utilizados na década de 90. Entre essas tecnologias podemos citar o RMI (*Remote Method Invocation* — Invocação de Método Remoto), (*Remote Method Invocation* — Invocação de Método Remoto), DCOM (*Component Object Model* — Componente Modelo de Objeto) e CORBA (*Common Object Request Broker Architecture* — arquitetura padrão criada pelo Object Management Group). No entanto, só tiveram sucesso na integração de aplicações em ambientes de redes locais e homogêneos. Com o avanço da internet para o mercado corporativo, surgiram a necessidade de integração das aplicações além das redes locais em ambientes heterogêneos. Nesse contexto que surge a tecnologia denominada *Web Services* (Serviços Web) utilizados na arquitetura SOA. (GOMES, 2014, p.13).

A Arquitetura SOA é um conceito de arquitetura corporativa que promove a integração e comunicação entre negócio e TI. Essa comunicação é feita através de um conjunto de Serviços Web acoplados, denominados provedor, que disponibilizados na rede, permitem o compartilhamento de funcionalidades a todos os Sistemas de negócios, denominados consumidor, independentes de plataforma e tecnologia. (GHODDOSI, 2017, p.47).

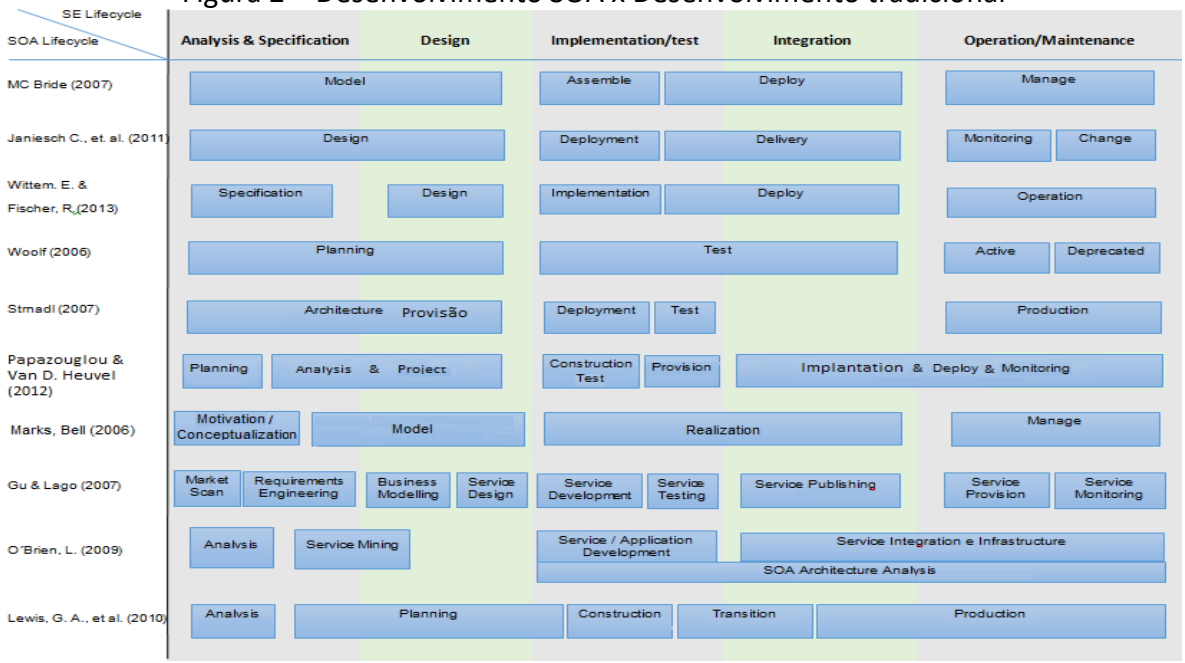
A arquitetura SOA baseia-se nos seguintes princípios:

- Baixo acoplamento: Provedor e Consumidores de serviço são independentes e capazes de tomar decisões sobre a tecnologia adotada. Um consumidor de serviço

não deve ser vinculado a uma determinada tecnologia (por exemplo, linguagem de programação, base de dados, plataforma), a fim de usar um serviço.
- Reutilização: Um serviço reutilizável apresenta funcionalidade auto contida que pode ser utilizada em múltiplos processos de negócio. Serviços agnósticos tem maior possibilidade de reutilização.
- Modularidade: O objetivo final de modularidade é prover a capacidade de mudar os componentes da arquitetura sem impactar os consumidores do serviço.
- Capacidade de Descoberta: Se refere a capacidade de um serviço ser encontrado e que suas interfaces sejam compreendidas pelo consumidor. Todo serviço possui. (RIBEIRO, 2017, p.16).

Os serviços web são componentes de software que se comunicam usando tecnologias Web baseadas em padrões. Uma vez que eles são baseados em padrões abertos, como protocolos baseados em HTTP e XML, incluindo SOAP e WSDL, os serviços web podem ser considerados hardware, linguagem de programação e sistema operacional independente. Provedor e consumidor são conceitos lógicos, pois o mesmo provedor, dependendo das regras de negócio, pode ser também um consumidor. (GHODDOSI, 2017, p.71).

Figura 2 – Desenvolvimento SOA x Desenvolvimento tradicional



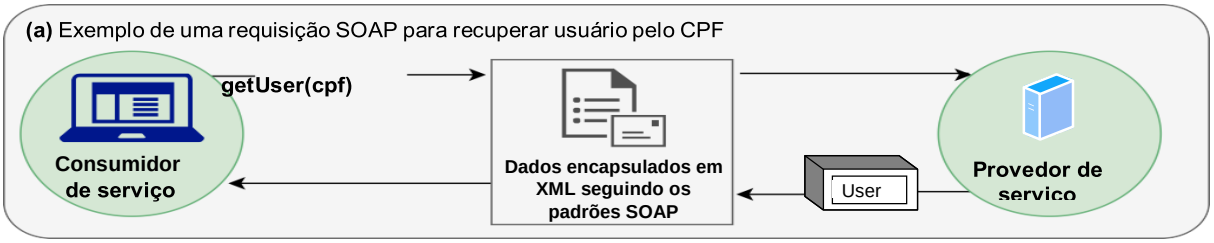
Fonte: (GHODDOSI, 2017, p.58).

Baseados nos princípios da Orientação a Serviço para se disponibilizar serviços Web, é preciso uma arquitetura tecnológica com características distintas. Essa arquitetura deve ser capaz de acomodar comportamentos e requisitos que possibilitem alcançar as metas da Computação SOA, que pode existir em quatro escopos ou níveis diferentes de implementação, sendo que alguns níveis abrangem os outros. (JUSTINO, 2018, p.21).

2.3 Serviço Web Tipo SOAP

As tecnologias SOAP e REST são protocolos usados para implementação da arquitetura SOA. Ambos tem a função de trafegar os dados invocados nos serviços de negócios disponibilizados em ambiente distribuído. Esses dados são basicamente transportados pelo protocolo HTTP ou HTTPS para conexões seguras. (AGILAR, 2016, p.26).

Figura 3 – Exemplo de Requisição SOAP



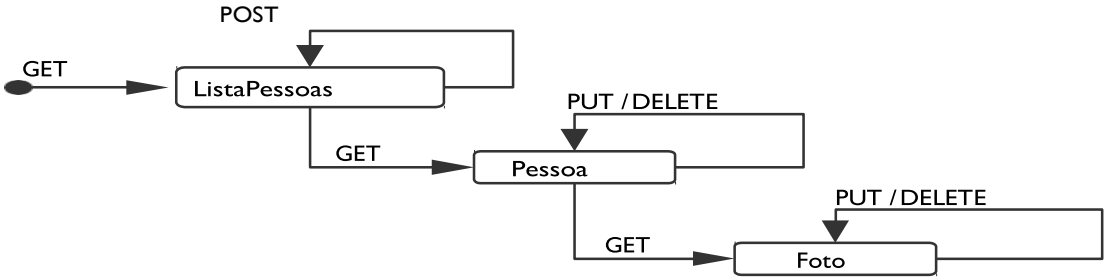
Fonte: (PASSINI, 2020, p.36)

Os Web Service SOAP representam um conjunto de tecnologias e padrões de comunicação independente de plataforma e linguagem, que define um formato baseado em XML para troca de informações sobre o protocolo HTTP usando chamadas de procedimento remoto. (AGILAR, 2016, p26). De acordo com o W3C (*World Wide Web Consortium* – Rede Mundial de Computadores), um serviço web é um sistema de software projetado para suportar a interação Inter operável entre máquinas em uma rede. Possui uma interface descrita em um formato de processo de máquina (especificamente WSDL). Outros sistemas interagem com o serviço web de uma maneira prescrita por sua descrição usando mensagens SOAP, normalmente transmitidas usando o protocolo HTTP com serialização XML em conjunto com outros padrões relacionados à Web (FREITAS, 2017, p.10).

2.4 Serviço Web Tipo REST

Os Serviços REST, representam uma outra forma de serviço web bastante difundida e utilizada em grandes organizações. Também baseado no protocolo HTTP, reúne uma coleção de princípios e restrições arquiteturais para desenvolvimento Web fracamente acoplados. Utilizado como alternativa ao XML, este tipo de serviço, para intercambio dos dados, adotam o uso do formato JSON, que são notações de objetos em Javascripts. (AGILAR, 2016, p26). REST baseia-se em recursos denominados métodos, onde cada um deles pode ser referenciado através de uma URL (Unified Resource Locator – Localizador de Recursos Unificados). As ações dos recursos são implementadas através de verbos existentes no protocolo HTTP, que são: GET para obter dados do recurso, PUT para alterar um recurso, POST para incluir um recurso e DELETE para excluir um recurso existente. (GRAHL et al., 2017, p.2).

Figura 4 – Fluxograma de Operações



Fonte: (SALVATORI, 2015, p.67)

Um recurso REST pode ser identificado por diversas URIs, mas uma URI endereça apenas um recurso. Outra característica importante é que o recurso pode ser representado de diferentes maneiras. A representação de um recurso é uma amostra dos valores de suas propriedades em um determinado momento do tempo. O recurso jamais é acessado diretamente, mas através de uma representação, sendo assim uma URI está sempre associada a pelo menos uma representação, sendo o JSON a representação mais utilizada em Web

Services REST. (SALVATORI, 2015, p.34). A Tabela 1 apresenta um comparativo entre SOAP e REST:

Tabela 1 – Comparativo SOAP x REST

	SOAP	**REST**
Descrição do Serviço	WSDL	Nenhum padronizado
Comunicação	Síncrona / Assíncrona	Síncrona
Web service	É o próprio serviço	Tudo é um recurso
Descoberta	WSDL	WADL
Estado	Com estado	Sem estado
Publicação	ESB ou UDDI	APP
Pesquisa	Baseado em serviços	Baseado em recursos
Formato de Mensagem	SOAP/XML	Não padronizada
Linguagens	C++, C#, Java, .Net, etc	HTML, Json, Javascript, Java, .Net, Ruby
Protocolos	Vários	HTTP
Segurança	HTTPS / WS-Security	HTTPS

Fonte: (Voigt, 2017, p.4)

2.5 *Hypertext Transfer Protocol* - HTTP

O protocolo HTTP atualmente é o principal meio para transmissão e troca de informações através da Web. É a forma mais comum de implementação dos Serviços Web do tipo REST. Cada transação realizada sobe esse protocolo consiste em uma requisição realizada por uma aplicação chamada cliente e uma resposta enviada pelo servidor. (FREITAS, 2017, p47). Os serviços Web são compostos de técnicas destinadas a integrar sistemas através das redes de computadores. Essa comunicação é realizada pelo protocolo HTTP geralmente realizada através de envio de mensagens no formato XML ou JSON (GRAHL et al., 2017, pg.02).

O estilo arquitetural de serviço Web do tipo REST foi proposto por Roy Fielding em 2000 em sua tese de doutorado. Pode ser descrita como um conjunto de princípios arquiteturais que podem ser utilizados para o desenvolvimento de serviços web. Utilizam o protocolo HTPP para realizar as trocas de mensagens. O HTTP efetua essa troca através das seguintes: POST - Cria um novo recurso; GET - Utilizado para recuperar um informação; PUT - Atualiza os dados de um recurso e DELETE - Apaga os dados de determinado recurso. (RIBEIRO, 2017, p.17).

2.6 *Extensible Mark-up Language* – XML

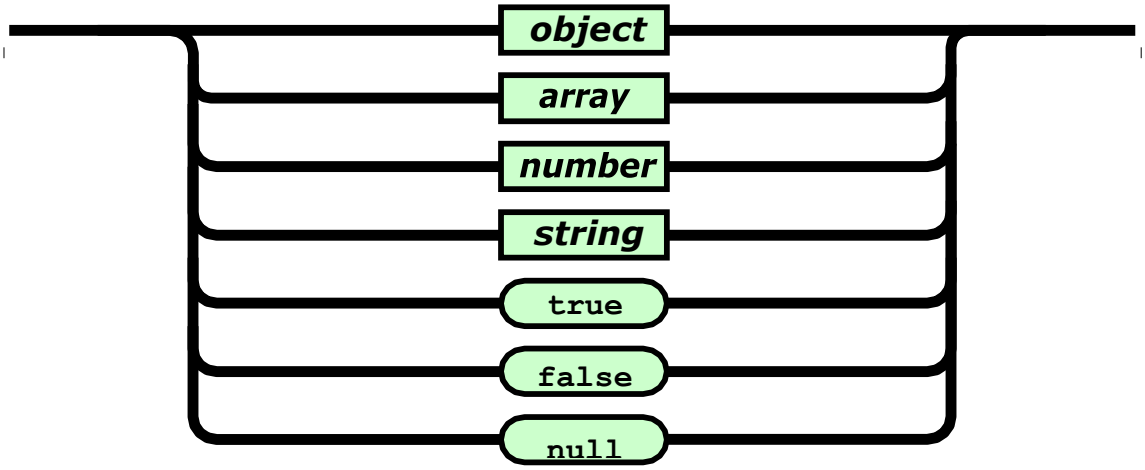
O *Extensible Markup Language* (XML) é arquivo disponibilizado em um formato simples baseado em texto utilizado para representar informações estruturadas, como: Documentos, Dados, Configuração, Livros, Transações, Faturas e outros. A sua derivação foi estabelecida de um formato padrão antigo denominado SGML (ISO 8879), para ser mais adequado ao uso na web. Atualmente ele é um dos formatos mais utilizados para compartilhar informações estruturadas entre: Programas, Pessoas, Computadores através da rede de computadores. W3C. (2016, p.01).

Apesar de ser muito semelhante ao HTML, as regras do XML são mais rígidas, pois as ferramentas que os processam não aceitam erros, fornecendo mensagens de erro para que sejam corrigidas. Isso significa que quase todos os documentos XML podem ser processados de forma confiável por diversos tipos de aplicações. O XML se diferencia sobre muitos outros formatos, pois permite que os dados sejam facilmente categorizados, permitindo uma consulta mais consistente pela aplicação. W3C. (2016, p.01).

2.7 *JavaScript Object Notation* – JSON

JSON é um tipo de armazenamento e transmissão de dados em formato texto que agrupa pares de chave/valor. Surgido em 2001 ele possibilita a transferência e a troca de informações entre aplicações de linguagens distintas. JUNGES, Renata, 2020. O JSON pode ser constituído por nome e valor ou um grupo de valores como uma lista, array ou vetor. Os nomes devem conter caracteres do tipo Strings e seus respectivos valores podem ser dos seguintes tipos: booleanos, strings, numéricos, nulo, ou array. Sua estrutura é formada por uma chave de abertura e fechamento (“{”) (“}”) cada elemento é seguido por dois pontos (“:”) e cada valor dos respectivos elementos. Como separação dos elementos utilizamos a vírgula. Já no uso do array utiliza-se o colchete (“[”) (“]”) de abertura e fechamento e os valores separados por (“,”) (ECMA-262, 2020, p.01).

Figura 5 – Valores JSON



Fonte: (ECMA-404, 2017, p.2)

JSON é um formato de serialização de dados legível por humanos baseado em texto com especificação padronizada e parcialmente descritivo. O JSON foi criado por Douglas Crockford que tinha como objetivo prático representar informações de forma simplificada, leve e flexível, e que pudesse obter desempenho superior ao formato XML. Com o passar do tempo, JSON se tornou popular, tendo se adaptado muito bem em ambientes de aplicações distribuídas. O JSON foi construído com base em quatro tipos primitivos de dados e outros dois para composição. Cada tipo possui seu respectivo correspondente na maior parte das linguagens de programação, embora possam ser identificados por nomes distintos (DROETTBOOM, 2020, p.01).

2.8 Linguagem de Programação JAVA

Com base em técnicas de funcionamento as linguagens de programação são qualificadas e definidas por meio de paradigmas, que são características determinantes no funcionamento da Linguagem. Cada linguagem de programação acopla a sua essência algumas regras que deverão ser seguidas quando implementadas. Como principais paradigmas temos o imperativo (PI), que se preocupa com os detalhes do funcionamento do algoritmo, e o Declarativo (PD) baseado na programação funcional. O Imperativo pode ser dividido em dois novos paradigmas, o Procedimental (PP) e o Orientado a Objetos (POO) sendo o PP orientador das linguagens como C e Pascal e o POO linguagens como Java e C#, havendo algumas linguagens hibridas como C++. (RONSZCKA, 2019, p.25).

A Orientação a Objetos tem se destacado em diversos tipos de *Software* ao longo de muitos anos, mas os desenvolvedores Web que utilizam Javascript e linguagens do lado

servidor não implementaram com muita rapidez. Inicialmente as páginas Web apresentavam necessidades relativamente simples, fazendo com que a orientação a objetos não fosse tão importante e necessária. Atualmente, com a complexidade das Aplicações Web, esse conceito tornou-se fundamental se quisermos tratar o desenvolvimento dessas aplicações com o mesmo rigor de outros tipos de Software. (KYLE, 2018, p.29).

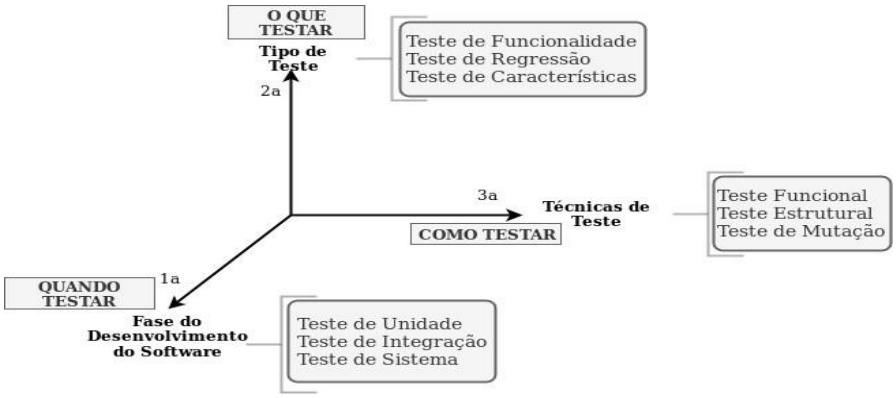
O Java é mais do que uma linguagem de programação orientada a objetos, podemos destaca-la como um poderoso *framework* que possibilita desenvolver *Aplicações*, por possuir uma grande variedade de bibliotecas que tornam a programação mais eficiente e um compilador altamente confiável. Java é uma linguagem de programação portável compatível com os principais sistemas operacionais do mercado, pois graças a sua JVM (*Java Virtual Machine* - Máquina Virtual JAVA), um componente de Software que simula um computador, oculta o sistema operacional e o hardware permitindo a execução de um programa JAVA. Assim sendo, basta que essa máquina virtual seja implementada nos diversos sistemas operacionais para que um programa JAVA seja executado em multiplataformas. (MACHADO,2016, p.48).

Através das expressões Lambda na versão 8 do Java foi incluído o suporte ao estilo funcional de programação. Com esse suporte atualmente é possível passar funções anônimas (expressões lambda) como argumentos para outras funções e diminuir a quantidade de declaração de classes anônimas nos programas que podem também ser substituídas por expressões lambda. (MENDONÇA, 2019, p.4).

2.9 Teste de Software

Teste de Software é definido como o processo de executar um programa com o objetivo de encontrar falhas. De fato, quando se testa um programa, deseja-se adicionar algum valor a ele. Adicionar valor através de teste significa incrementar a qualidade ou confiabilidade do programa. Um aumento na confiabilidade do programa significa achar e remover erros. Todavia, não se testa um programa para provar que ele funciona, ao invés disso, é assumido que o programa contém erros e testamos o programa para encontrá-los. Podemos dividi-los em 3 dimensões (PEREIRA, 2019, p.28).

Figura 6 – Relação entre níveis, tipos e técnicas de teste



Fonte: (PEREIRA, 2019, p.28)

Por muito tempo, a qualidade do software dependia da construção cuidadosa de um projeto de software e das habilidades de implementação do. Nesse sentido, poucas estratégias eram adotadas para conter os defeitos presentes no software. No entanto, em decorrência do advento de áreas como a engenharia de software, que dentre seus objetivos visa à produção de software de alta qualidade; técnicas e atividades especializadas e eficientes foram emergindo, fornecendo subsídios para o desenvolvimento de software com maior qualidade. Dentre as atividades que fazem parte do processo de desenvolvimento de

software, o teste de software busca revelar defeitos antes do software ser disponibilizado para uso. (PASCHOAL, 2019, p.37).

3. Materiais e Métodos

3.1 Etapas do Desenvolvimento

O desenvolvimento do projeto foi realizado em três etapas, como segue: 1 - Pesquisa bibliográfica para coleta de todo referencial teórico que envolve abordagens e ferramentas de apoio ao desenvolvimento de APIs REST. 2 – Teste e Avaliação algumas APIs REST disponibilizadas no mercado, de forma a aprofundar o conhecimento técnico sobre essa tecnologia; 3 – Desenvolvimento e Avaliação do Software de teste de API REST proposta nesse artigo.

As atividades foram delineadas através do processo de desenvolvimento ágil, interativo e incremental, onde cada interação estabelece as fases de análise, projeto, desenvolvimento, teste, integração e entrega. A avaliação do projeto foi realizada através de testes do Software com URIs de APIS REST disponibilizadas na internet. O objetivo desse *Software* é realizar verificações em APIs REST de forma a poder agilizar as validações de respostas no formato JSON.

3.2 Camadas da Arquitetura de Software

A arquitetura desta ferramenta de testes foi desenvolvida em duas camadas denominadas *Back-end (Processo Interno – Regras de Negócio)* e Front-end (Interface do Usuário). O *Front-end, processo interno,* é composto pelos componentes responsáveis pela interface do sistema com o usuário. O *Back-end* caracteriza-se pelos componentes executados no lado do servidor, onde são implementadas toda regra de negócio.

No desenvolvimento do Front-end, foi utilizado a linguagem JavaScript para complementar as rotinas das páginas em HTML com estilos definidos no CSS (Cascading Style Sheets – Estilo de Folhas em Cascatas). Além disso, também integrado ao framework Spring MVC, foi usado o mecanismo de Template Java Thymeleaf, que fornece uma gama de funcionalidades na camada de visão, capaz de processar arquivos em HTML, XML, JavaScript, CSS e Texto.

O desenvolvimento do *Back-end* foi feito através da linguagem de programação Java versão 8. Por se tratar de uma aplicação Web foi adotado o *Framework Spring,* que implementa o padrão em camadas MVC – Model (Modelo), View (Visão) e Controller (Controlador).

Por meio desse Framework foi possível implementar ORM (*Object-Relational Mapping - M*apeamento objeto relacional), que faz uma relação dos objetos da aplicação, segundo a Orientação a Objetos, com as tabelas do banco de dados relacional. Além disso, via seu utilitário RestTemplate, proporcionou a interação da aplicação Java com os serviços REST permitindo o consumo dos serviços e o mapeamento de uma lista de objetos no formato JSON.

3.4 Banco de Dados

Para persistência de dados foi utilizado o banco *H2 Database Engine (*H2 Motor de Banco de Dados*)*, que vem incorporado ao *Framework Spring Boot*. Ele é um banco de dados *Open Source* (de Código Aberto)) que funciona em memória com um console dentro do contexto da aplicação que pode ser acessado via *Browser* (Navegador Web). Por ter seu funcionamento em memória todo seu armazenamento é desfeito e reconstruído a cada reinicio da aplicação. Sua principal intenção é oferecer maior produtividade no

desenvolvimento de *Software*, pois é um banco de configuração rápida, precisando apenas da inclusão de parâmetros no arquivo de propriedades “application.properties”.

Para integração do Banco de dados H2 com o framework Spring Boot é preciso acessar o arquivo pom.xml da aplicação e adicionar sua dependência. Além disso, precisamos configurar o acesso ao banco de dados, para isso acesse a pasta src/main/resources e incluir no arquivo chamado application.properties as propriedades abaixo:

Figura 7 – Arquivo application.properties da Aplicação testrest

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.datasource.url=jdbc:h2:file:/C:/DADOS/Pos-Estacio/TCC/DB/TCC.db
spring.datasource.username=rui
spring.datasource.password=admin
spring.datasource.driver-class-name=org.h2.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
spring.jpa.open-in-view=true
```

Fonte: Autoria própria

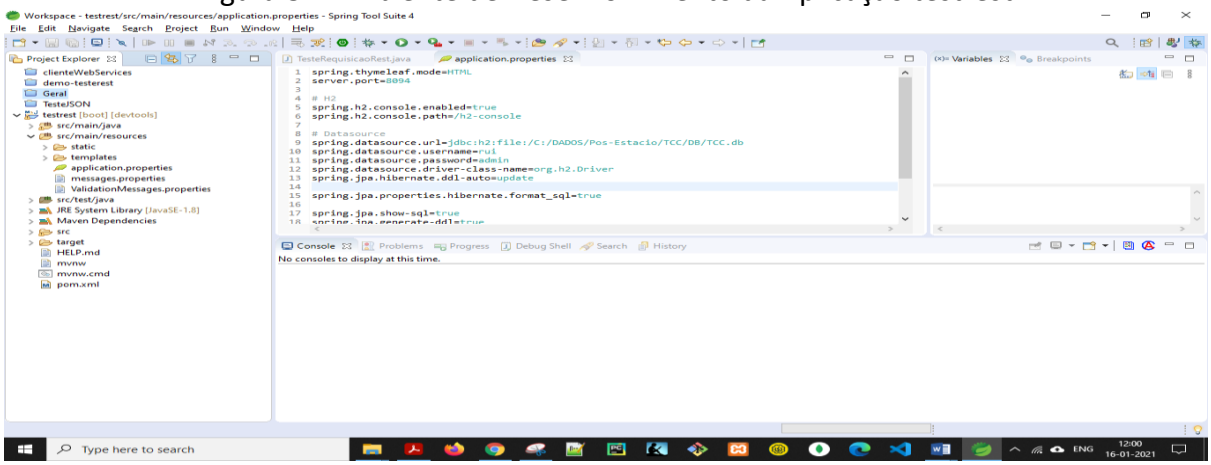
3.5 IDE Spring Tool Suite 4

O Spring Tool Suite é um ambiente de desenvolvimento projetado e baseado na IDE Eclipse customizado e personalizado para o desenvolvimento de aplicação que utilizam o Framework Spring. Ele fornece um ambiente completo e pronto para usar e implementar, configurar, depurar e disponibilizar aplicação Java.

De forma fácil e simples, você ganha mais produtividade não só em projetos com Spring MVC, mas também em projetos Spring Core, Spring Boot, Spring Data JPA, e Spring Web. Para disponibilizar um projeto Spring MVC utilizando o Spring Tool Suite com Spring Boot é bem mais rápido do que apenas Spring MVC na IDE Eclipse.

Além disso o Spring Tool Suite incorpora em seu ambiente ferramentas personalizadas para simplificar e acelerar o desenvolvimento de aplicações com Groovy e Grails. Groovy é uma linguagem orientada a objetos com tipagem dinâmica que roda na JVM e Grails é um framework web baseado em MVC que utiliza a linguagem Groovy.

Figura 8 – Ambiente de Desenvolvimento da Aplicação testrest

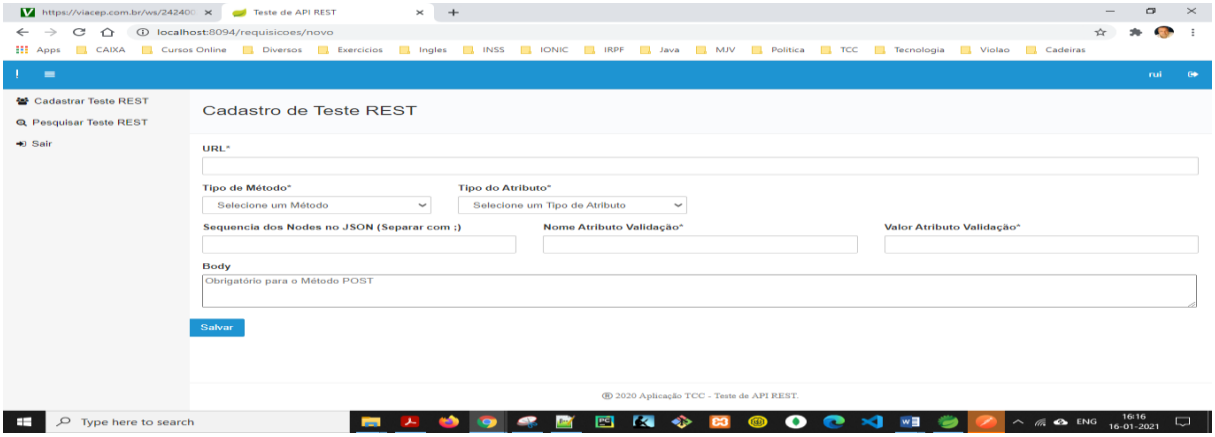


Fonte: Autoria própria

3.6 Aplicação Desenvolvida testrest

Nessa parte será demonstrado o funcionamento básico da Aplicação desenvolvida testrest, começando da tela principal conforme a Figura 9. Nela é possível observar um menu à esquerda contendo os dois principais módulos da Aplicação: Cadastro Teste REST e Pesquisa Teste REST.

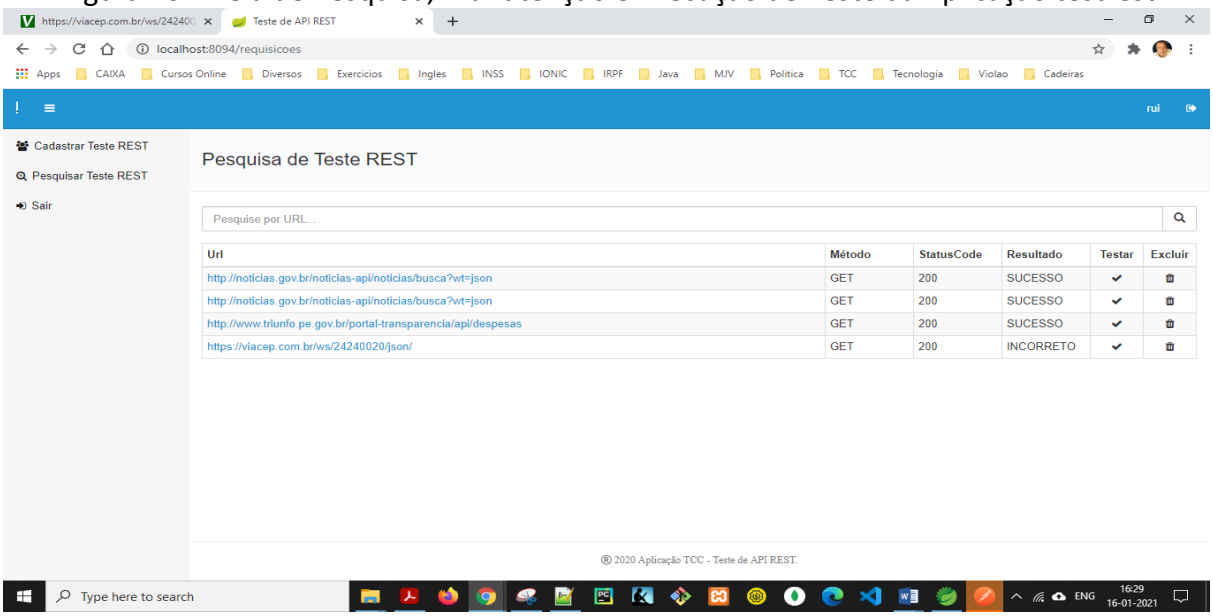
Figura 9 – Tela Inicial de Cadastro da Aplicação testrest



Fonte: Autoria própria

Na página de Cadastro Teste REST o usuário realiza um novo cadastro de uma nova requisição de API para teste. Nela é preciso inserir a URL a ser testada e seus respectivos parâmetros obrigatórios para validação: Tipo do Método, Tipo do Atributo, Nome Atributo Validação e Valor Atributo Validação. Os demais campos opcionais são: Sequencia dos Nodes no JSON (Separar com ;) e Body.

Figura 10 – Tela de Pesquisa, Manutenção e Execução de Teste da Aplicação testrest



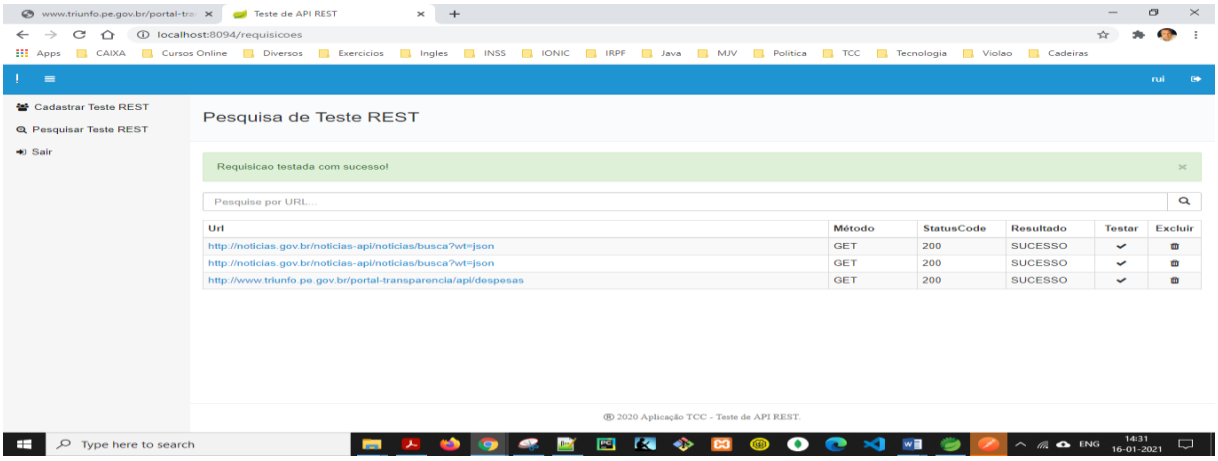
Fonte: Autoria própria

4. Resultados

Como resultado deste projeto foi desenvolvido a aplicação Web testrest, que é capaz de realizar testes automatizados em requisições do tipo de requisições REST com maior velocidade e a possibilidade de validar o conteúdo de entrada e saída de atributos dos serviços. Um dos grandes destaque dessa aplicação consiste na possibilidade de validação dos

atributos contido na estrutura de entrada e saída de dados do formato JSON. Basta o usuário informar o nome do atributo que deverá ser testado e seu valor de retorno. Esta funcionalidade não foi observada nas ferramentas similares disponíveis no mercado.

Figura 11 – Resultado do teste pe.gov. Validar Atributo "status"

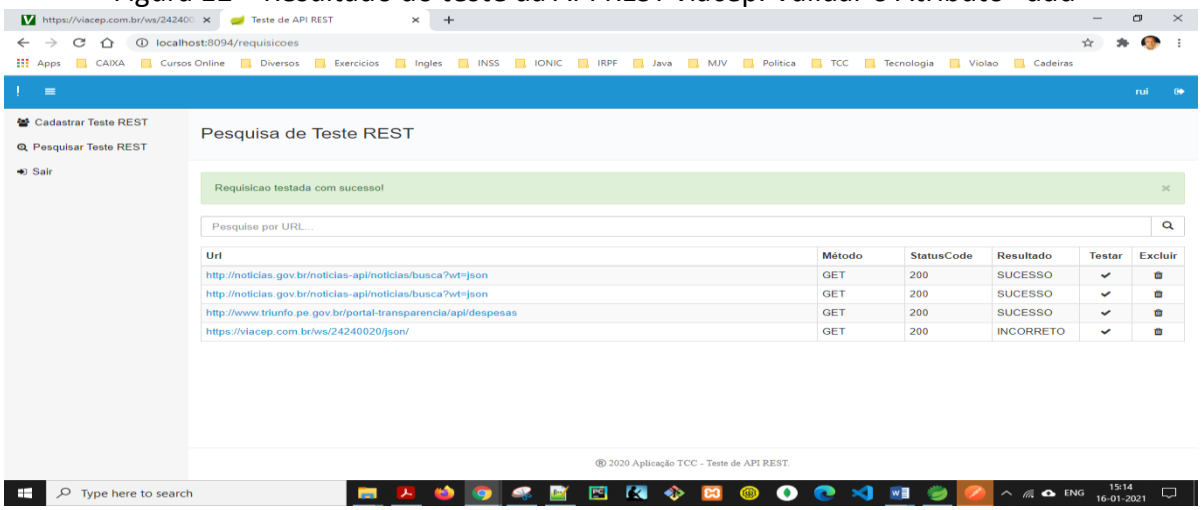


Fonte: Autoria própria

Apesar dos serviços REST poderem retornar diversos formatos, esta aplicação somente realiza testes com resultados no formato JSON. Seguem algumas APIS utilizadas para os testes REST: http://www.triunfo.pe.gov.br/portal-transparencia/api/despesas, https://gturnquist-quoters.cfapps.io/api/random, https://viacep.com.br/ws/24240020/json/, https://viacep.com.br/ws/RS/Porto%20Alegre/Domingos/json/.

A URI http://www.triunfo.pe.gov.br/portaltransparencia/api/despesas foi testada e a condição de “Sucesso” foi apresentada. Isto ocorre porque o atributo validado “status” possui em seu conteúdo o valor compatível com o parâmetro imputado pelo usuário na Tela de Cadastro da Aplicação testrest, conforme demonstra a figura 12 (Tela de Cadastro da Aplicação testrest).

Figura 12 – Resultado do teste da API REST viacep. Validar o Atributo "ddd"



Fonte: Autoria própria

A URI http://www.triunfo.pe.gov.br/portaltransparencia/api/despesas foi testada e a condição de “incorreto” foi apresentada. Isto ocorre porque o atributo validado “ddd” não possui em seu conteúdo o valor compatível com o parâmetro imputados pelo usuário na Tela de Cadastro da Aplicação testrest.

4.1 Métricas

Para fins comparativos, foram realizados testes utilizando várias requisições de serviços REST disponibilizadas na internet, como a dos portais Cep, Amazon e aplicações de CRM. Em resposta, constatou-se que o tempo de espera em milissegundos dos resultados em comparação com as ferramentas de mercado Postman e SoapUI foram bastante satisfatórios, conforme indica a tabela abaixo:

Tabela 2 - Métricas do Teste com a Ferramenta Postman

Operação	Software	Postman
Teste da Requisição	245 ms	324 ms
Criação do Teste	1902 ms	2409 ms
Alteração do Teste	7 S	10 S
Consulta do Teste	10 S	15 S

Fonte: Autoria própria

Observando o resultado da avaliação, é visto que a ferramenta desenvolvida obtém dados de tempo próximos a ferramentas já disponíveis no mercado. Os valores obtidos na comparação entre a ferramenta desenvolvida e a ferramenta Postman não podem ser interpretados literalmente, pois os dados são médias de contagens realizadas em teste, e podem variar dependendo do momento e situação dos envolvidos, como a própria ferramenta, o servidor e principalmente a rede entre cliente e servidor.

Tabela 3 - Métricas do Teste com a Ferramenta SoapUI

Operação	Software	Postman
Teste da Requisição	321 ms	528 ms
Criação do Teste	1876 ms	2054 ms
Alteração do Teste	8S	13 S
Consulta do Teste	12 S	18 S

Fonte: Autoria própria

5. Conclusão

Como demonstrado nesse artigo o teste de API é bastante importante como altamente necessário, pois ajuda a garantir o bom funcionamento, desempenho e a confiabilidade de diferentes aplicativos e sistemas baseados em dados. Isso ocorre através da certificação das trocas e comunicações entre aplicativos, sistemas, bancos de dados e redes. A maneira como temos nos movido rapidamente em direção a novas propostas tecnológicas fará com que logo haja uma demanda por testes mais criteriosos de API. Assim, seguir algumas etapas importantes para iniciar o teste de API, além de ter boas ferramentas de teste automatizados, ajudarão a cobrir cada vez mais as necessidades dos testes de APIs e implementará aplicativos com segurança e qualidade no tempo certo.

Nesse artigo foi desenvolvido uma Aplicação bastante eficiente capaz de realizar testes em requisições do tipo REST com respostas em formato JSON. Aplicação capaz de proporcionar maior velocidade no teste, uma vez que é possível cadastrar diversos testes e executá-los quantas vezes forem necessários com apenas um clique.

A Aplicação desenvolvida parte do princípio que exista uma definição anterior do teste, que deve ser cadastrado, explicitando o tipo da resposta e os valores que devem ser retornados no retorno da requisição. Nos testes são validados o tipo da resposta da requisição e o valor específico do atributo a ser validado.

Com a finalização do desenvolvimento ficou evidente a melhoria no tempo de execução de testes dado que agora todos os testes são cadastrados apenas uma vez e executados em diversos momentos. É importante destacar também que a Aplicação desenvolvida será de grande ajuda no caso de *Softwares* desenvolvidos utilizando o processo de desenvolvimento orientado por testes, ajudando na verificação e validação durante os ciclos de repetição de testes em aplicações desenvolvidas utilizando a técnica de TDD (*Test Driven Development* — Desenvolvimento Orientado por Testes).

Referências Bibliográficas

AGILAR, E. de V. **Uma Abordagem Orientada a Serviços para a Modernização de Sistemas Legados**. Tese (Doutorado) — Universidade de Brasília, 2016. 2016. Disponível em https://repositorio.unb.br/handle/10482/22250. Acesso em 26 nov. 2020.

DROETTBOOM, M. **Understanding JSON Schema**. 2015. Disponível em: https://json-schema.org/understanding-json-schema/. Acesso em 10 dez. 2020.

ECMA-262. **The JSON Data Interchange Syntax**. 2nd Edition/December 2017. 2020. Disponível em: https://262.ecma-international.org/11.0/. Acesso em 10 dez. 2020.

ECMA-404. **The JSON Data Interchange Syntax.** 2nd Edition/December 2017. 2017. Disponível em: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. Acesso em 10 dez. 2020.

FLÔRES, Fabiano Niederauer. **Um processo para a geração automatizada de instâncias de esquemas JSON a partir de consultas SQL.** Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCC), da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de Mestre em Ciência da Computação. 2019. Disponível em: http://repositorio.ufsm.br/handle/1/18773. Acesso em 20 dez. 2020.

FREITAS, B. **Modernização De Sistemas Legados Para Disponibilização Em Dispositivos Móveis Com Arquitetura Baseada Em Microservices**. 2017. Universidade Federal de Pernambuco. Disponível em: https://repositorio.ufpe.br/handle/123456789/25235. Acesso em 05 out 2020.

GHODDOSI, Nader. **Um Método de Apoio à Decisão na Estimação de Software baseado em Serviços em uma Perspectiva SOA**. Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas. 2017. Disponível em: https://repositorio.ufsc.br/handle/123456789/189291. Acesso em 11 dez. 2020.

GOMES, Daniel Adorno. **Web Services SOAP em Java** - Guia prático para o desenvolvimento de Web Services JAVA. 2. Ed. Editora NOVATEC. 2014. Disponível em: https://books.google.com.br/books?id=0iysBAAAQBAJ&printsec=frontcover&dq=web+services&hl=pt-BR&sa=X&ved=2ahUKEwjuwerrncnuAhV7HLkGHSMrDigQ6AEwA3oECAEQAg#v=onepage&q=web%20services&f=false. Acesso em 02 jan. 2021.

JUSTINO, Yan de Lima. **Do monólito, do monólito aos micros serviços: um relato de migração de sistemas legados da Secretaria de Estado da Tributação do Rio Grande do Norte.** Dissertação (Mestrado Profissional em Engenharia de Software) - Instituto Metrópole Digital, Universidade Federal do Rio Grande do Norte, Natal. 2018. Disponível em: https://repositorio.ufrn.br/handle/123456789/26370. Acesso em 04 jan. 2021.

KYLE, Loudon, **Desenvolvimento de Grandes Aplicações Web**. Editora NOVATEC. 2018. Disponível em:

https://books.google.com.br/books?id=g_6ZDwAAQBAJ&pg=PT290&dq=xml&hl=pt-BR&sa=X&ved=2ahUKEwj6jur8s8nuAhXAILkGHXTPAf0Q6AEwAHoECAYQAg#v=onepage&q=xml&f=false. Acesso em 02 jan. 2021.

MACHADO, Felipe Nery R. **Análise e Gestão de Requisitos de Software – onde nascem os sistemas**. 3 Ed. 2016. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=lang_pt&id=MYdiDwAAQBAJ&oi=fnd&pg=PT5&dq=an%C3%A1lise+e+projeto+de+sistema&ots=DkVeozgWI0&sig=y1QO55-sJaMjiGDJ3L7x8zRdt0M#v=onepage&q=an%C3%A1lise%20e%20projeto%20de%20sistema&f=false. Acesso em 10 dez. 2020.

MENDONÇA, Walter Lucas Monteiro. **Análise do impacto na compreensão de programas Java com a introdução de expressões lambda.** Dissertação apresentada como requisito parcial para conclusão do Mestrado em Informática. Universidade de Brasília Instituto de Ciências Exatas Departamento de Ciência da Computação. 2019. Disponível em: https://repositorio.unb.br/handle/10482/36024. Acesso em 28 dez. 2020.

PASCHOAL, Leo Natan. **Contribuições ao ensino de teste de software com o modelo flipped classroom e um agente conversacional.** Dissertação de Mestrado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC). 2019. Disponível em: https://www.teses.usp.br/teses/disponiveis/55/55134/tde-13062019-140507/publico/LeoNatanPaschoal_revisada.pdf. Acesso em 04 jan. 2021.

PASSINI, William Filisbino. **Desenvolvimento de serviços compostos auto adaptativos: um framework baseado em implantação dinâmica, métricas de QoS e informação semântica**. Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Geociências e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de Rio Claro. 2020. Disponível em: https://repositorio.unesp.br/bitstream/handle/11449/192811/passini_wf_me_sjrp.pdf?sequence=5&isAllowed=y. Acesso em 20 dez. 2020.

PEREIRA, Iuri Guerra de Freitas. **Avaliação da efetividade de uma suíte de teste de sistema aplicada ao contexto do Middleware Ginga.** Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Software da Universidade Federal do Rio Grande do Norte como requisito para a obtenção do grau de Mestre em Engenharia de Software. 2019. Disponível em: https://repositorio.ufrn.br/jspui/bitstream/123456789/28577/1/Avaliacaoefetividadesuite_Pereira_2019.pdf. Acesso em 28 dez. 2020.

PRESSMAN, Roger S.; Maxim, Bruce R., 2016. **Engenharia de Software** - 8ª Edição. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=wexzCwAAQBAJ&oi=fnd&pg=PR1&dq=Pressman&ots=0OWGlKPx50&sig=fSTHfdFLDy6jftVv6jtwWHxNJTg&redir_esc=y#v=onepage&q=Pressman&f=false. Acesso em 02 jan. 2021.

RIBEIRO, Alysson de Sousa. **Uma Implementação do Protocolo OAuth 2 em Erlang para uma Arquitetura Orientada a Serviço.** Dissertação apresentada como requisito parcial para conclusão do Mestrado Profissional em Computação Aplicada. Universidade de Brasília Instituto de Ciências Exatas Departamento de Ciência da Computação. 2017. Disponível em: https://repositorio.unb.br/bitstream/10482/24694/1/2017_AlyssondeSousaRibeiro.pdf. Acesso em 21 dez. 2020.

RONSZCKA, A. F. **Método para a Criação de Linguagens de Programação e Compiladores para o Paradigma Orientado a Notificações em Plataformas Distintas**. Tese de doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática

Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Doutor em Ciências” – Área de Concentração: Engenharia da Computação. 2019. Disponível em: https://repositorio.utfpr.edu.br/jspui/bitstream/1/4234/1/CT_CPGEI_D_Ronszcka%2C%20Adriano%20Fransico_2019.pdf. Acesso em 21 dez. 2020.

SALVATORI, Ivan Luiz. **Desenvolvimento de Web APIS RESTFUL Semânticas baseadas em JSON**. Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação. 2015. Disponível em: https://repositorio.ufsc.br/handle/123456789/132469. Acesso em 11 dez. 2020.

VOIGT, Ricardo; Junior, Osmar de Oliveira Braz. **Análise de desempenho de arquitetura SOAP e REST para comunicação entre sistemas.** Centro de Educação Superior do Alto Vale do Itajaí (CEAVI) – Universidade do Estado de Santa Catarina (UDESC) – Ibirama, SC – Brasil. 2017. Disponível em: http://periodicos.unesc.net/sulcomp/article/view/3120. Acesso em 20 dez. 2020.

W3C - World Wide Web Consortium. **Principal organização de padronização da World Wide Web.** Disponível em: https://www.w3.org/standards/xml/core. 2020. Acesso em 02 fev. 2021.