Automação de Testes de APIs REST no SisNIR

João Victor de Morais
Fundação de Desenvolvimento Científico e Cultural
Lavras, Minas Gerais, Brasil
joaomorais.ti@fundecc.org.br

Neumar Costa Malheiros
Universidade Federal de Lavras
Lavras, Minas Gerais, Brasil
neumar@ufla.br

Ricardo Terra
Universidade Federal de Lavras
Lavras, Minas Gerais, Brasil
terra@ufla.br

RESUMO

Garantir a confiabilidade e a qualidade de APIs REST é um desafio, sobretudo em sistemas de setores regulados que demandam transparência, rastreabilidade e auditabilidade. Este artigo apresenta o Zettalenium, um *framework* de testes automatizados para o desenvolvimento, execução e documentação de casos de teste em APIs RESTful. O estudo de caso relata sua aplicação no Sistema Nacional de Informações sobre Irrigação (SisNIR), onde viabilizou a automação de testes que verificaram funcionalidades essenciais do sistema, assegurando confiabilidade e consistência nas operações. O Zettalenium combina a execução de requisições HTTP com asserções automatizadas, oferece utilitários reutilizáveis e gera relatórios abrangentes – recursos que apoiaram os processos de aceite do Ministério da Integração e do Desenvolvimento Regional (MIDR). Os resultados evidenciam melhorias na estrutura e na manutenibilidade dos testes, além da redução de redundâncias e de intervenções manuais.

PALAVRAS-CHAVE

Testes automatizados de software, API REST

1 Introdução

As APIs REST (*Representational State Transfer*) são um estilo arquitetural amplamente adotado no desenvolvimento de sistemas distribuídos devido à sua simplicidade, escalabilidade e eficiência [5]. Fundamentadas no modelo cliente-servidor, utilizam o protocolo HTTP para padronizar interações por meio de métodos bem definidos, como GET, POST, PUT e DELETE [2, 3]. Além disso, operam sobre recursos identificados por URIs e comumente representados no formato JSON por sua legibilidade e facilidade de integração entre sistemas heterogêneos [6, 8].

A crescente dependência de sistemas baseados em APIs REST para integrar serviços e aplicações trouxe um aumento significativo na complexidade e na necessidade de qualidade desses sistemas [4]. Esse cenário, associado à natureza dinâmica das APIs, gera desafios específicos, como a validação de múltiplos cenários de integração e o tratamento de respostas variadas [7].

No desenvolvimento do Sistema Nacional de Informações sobre Irrigação (SisNIR), o Ministério da Integração e do Desenvolvimento Regional (MIDR) estipulou como requisito essencial a criação e documentação de testes funcionais automatizados para todas as rotas da API REST. No entanto, ferramentas tradicionais como *Postman* e *Insomnia* apresentaram limitações importantes: exigem configurações manuais, não geram relatórios detalhados e dificultam a padronização e manutenção de testes extensos com diversas assertivas [1, 10].

Diante desse cenário, este trabalho descreve a aplicação do Zettalenium, um *framework* de testes automatizados desenvolvido para atender às demandas do projeto SisNIR, conduzido em parceria com o MIDR. O *framework* foi empregado para simplificar a automação de testes de APIs REST, integrando de forma unificada o envio de requisições, validações e geração de relatórios. Com essa abordagem, buscou-se reduzir a duplicação de código, facilitar a manutenção dos testes e garantir maior transparência e confiabilidade durante o desenvolvimento do sistema.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta o *framework* Zettalenium, explicando sua proposta, funcionamento e principais componentes. A Seção 3 descreve a sua aplicação no desenvolvimento do SisNIR, ressaltando os benefícios obtidos. Por fim, a Seção 4 encerra o artigo com considerações finais e indicações de trabalhos futuros.

2 O *framework* Zettalenium

O Zettalenium, desenvolvido neste artigo, tem o objetivo de simplificar e estruturar o processo de automação de testes funcionais em APIs REST. Trata-se de um *framework* que unifica o envio de requisições, a validação de respostas e a geração automatizada de relatórios em um único fluxo, promovendo eficácia, padronização e facilitando a manutenção. Apesar de estender o JUnit, o Zettalenium se diferencia por sua abordagem unificada, eliminando a necessidade de integrar e configurar múltiplas bibliotecas manualmente.

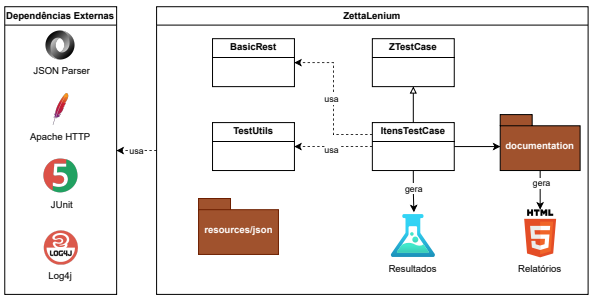
Seu uso permite validar desde fluxos simples até cenários complexos de integração, nos quais múltiplas rotas possuem dependências entre si — como, no caso do SisNIR, o cadastro de um Projeto Público de Irrigação, que é pré-requisito para o registro de um Plano Operativo Anual. O Zettalenium foi projetado para atender a essas demandas, eliminando tarefas repetitivas e centralizando utilitários essenciais para a execução dos testes.

A Figura 1 ilustra a arquitetura do Zettalenium, que integra diversas dependências externas para viabilizar suas operações. O *framework* utiliza a biblioteca *JUnit*[1] para realizar as asserções de respostas durante os testes; a *Apache HttpClient*[2] para o envio de requisições HTTP; e a biblioteca JSON[3] para a manipulação de arquivos e estruturas de dados em formato JSON. Também emprega o *Log4j*[4]

[1] https://junit.org/junit5/
[2] https://hc.apache.org/httpcomponents-client-4.5.x/index.html
[3] https://www.json.org/json-en.html
[4] https://logging.apache.org/log4j/

para registrar logs detalhados de cada etapa do teste, facilitando o rastreamento e a auditabilidade da execução. Internamente, o desenvolvedor cria casos de teste estendendo a classe base `ZTestCase` que centraliza métodos reutilizáveis e assegura padronização. A execução envolve componentes como `BasicRest`, para as operações HTTP, e `TestUtils`, para manipulações e comparações de dados. A geração automática do relatório ocorre ao final dos testes, consolidando informações em um arquivo HTML. Em suma, Zettalenium se apresenta como uma solução unificada para testes de APIs REST, reduzindo a complexidade e o esforço de codificação – um diferencial significativo em relação a ferramentas tradicionais, como o uso puro do JUnit.

Figura 1: Arquitetura do Zettalenium



O Código 1 exemplifica um teste funcional utilizando o Zettalenium. O fluxo é composto por duas etapas principais: (i) o cadastro de um item via requisição POST (linhas 13 a 16) e (ii) a validação desse cadastro com uma requisição GET (linhas 22 a 24), garantindo que os dados persistidos sejam consistentes. A comparação com os dados esperados ocorre entre as linhas 18 e 20. Operações como o envio de requisições e a validação de respostas são encapsuladas por métodos utilitários como `post`, `get` e `assertJsonContains`, reduzindo a repetição de código, melhorando a legibilidade e facilitando a manutenção dos testes.

O *framework* também disponibiliza um conjunto abrangente de funções utilitárias, como leitura e comparação de arquivos JSON, manipulação de datas e tratamento de textos, reduzindo significativamente o esforço e o risco de erros durante a implementação dos testes. Por exemplo, funções como `readJson` e `assertJsonContains` foram amplamente utilizadas no projeto para validar as respostas das APIs de forma eficiente e confiável.

Por fim, o Zettalenium inclui mecanismos de suporte à auditoria: ao executar os testes, ele gera *logs* detalhados que permitem acompanhar a execução de forma *headless* no servidor, uma vez que todas as funções do *framework* já possuem *logs* embutidos. Essa automação não apenas evita que o testador precise adicionar *logs* manualmente, mas também garante a padronização e a consistência dos registros de auditoria em todos os testes, o que é fundamental para a auditabilidade. Além disso, gera um relatório que sintetiza informações sobre os testes em um formato claro e acessível, facilitando a análise dos resultados.

```
1  public class ItensTestCase extends ZTestCase {
2    private static Logger LOGGER = LogManager.getLogger(
         ItensTestCase.class);
3
4    private final static String path = "/itens";
5
6    @Test
7    @ZTest(tester = "Joao Victor",
8        description = "Metodo que cadastra um item, depois o
             busca e confere se foi salvo como esperado.",
9        createdAt = "06/08/2024",
10       revisedAt = "07/08/2024"
11   )
12   public void testFluxoSimples() {
13     LOGGER.trace("Cadastra um novo item.");
14     JSONObject resultCreate = post(this.path, 201, "item001.
           json", JSONObject.class);
15     Integer generatedId = resultCreate.getInt("id");
16     assertNotNull(generatedId);
17
18     LOGGER.trace("Checa se a criacao foi bem-sucedida");
19     JSONObject expectedResultCreate = TestUtils.readJsonFile("
           item-create-result.json");
20     this.assertJsonContains(resultCreate,expectedResultCreate);
21
22     LOGGER.trace("Busca o item e verifica os dados.");
23     JSONObject resultRetrieve =  get(this.path + "/" +
           generatedId, 200, null, JSONObject.class);
24     this.assertJsonContains(resultCreate, resultRetrieve);
25   }
26 }
```

Código 1: Teste Funcional Automatizado com Zettalenium

3 Estudo de Caso

O Zettalenium foi inicialmente concebido e aplicado no âmbito do SisNIR, uma plataforma estratégica prevista na Política Nacional de Irrigação, cujo objetivo é subsidiar o planejamento da expansão da agricultura irrigada no Brasil. O sistema foi desenvolvido para coletar, armazenar e recuperar informações sobre irrigação, consolidando dados essenciais para a gestão de programas voltados ao desenvolvimento sustentável do setor.

O SisNIR foi desenvolvido por meio de uma parceria entre o *Ministério da Integração e do Desenvolvimento Regional* (MIDR) e a *Universidade Federal de Lavras* (UFLA). Essa colaboração resultou em uma solução robusta, confiável e alinhada às demandas institucionais e setoriais.

Durante o desenvolvimento, o MIDR definiu como requisito essencial a implementação e documentação de testes funcionais automatizados para todas as rotas da API REST do sistema. Inicialmente, a verificação era conduzida manualmente com ferramentas como o Postman, porém demandava esforço considerável para a execução repetitiva e a verificação manual dos resultados. Para superar essas limitações, o Zettalenium foi adotado, oferecendo uma abordagem padronizada, auditável e eficiente, capaz de assegurar a qualidade e a rastreabilidade ao longo do ciclo de vida do software.

Ao todo, foram desenvolvidos 41 casos de teste, totalizando 78 métodos, organizados de forma modular: para cada funcionalidade da API, foi criado um *testcase* específico. Essa estrutura favoreceu a escalabilidade e a manutenção dos testes, sobretudo em funcionalidades mais complexas, onde se validaram fluxos completos de operações *create*, *read*, *update* e *delete* (CRUD). Cenários de erro também foram contemplados, incluindo tentativas de cadastro com campos inválidos ou dados mal formatados, elevando a robustez e a resiliência da API frente a entradas inesperadas.

A Figura 2 apresenta o fluxo de execução dos testes automatizados com o Zettalenium, adotado antes de cada *deploy* como parte do processo de testes de regressão. O processo se inicia com a atualização do código ou o desenvolvimento de uma nova funcionalidade. Em seguida, os testes são executados de forma automatizada e *headless*, dispensando a necessidade de interação manual ou interface gráfica. Após a execução, os resultados são validados por meio dos relatórios e logs de auditoria gerados pelo *framework*, permitindo identificar eventuais falhas ou inconsistências. Quando os testes são aprovados, o *deploy* é autorizado, garantindo que alterações recentes não comprometam funcionalidades existentes. Essa abordagem contribuiu diretamente para a estabilidade e confiabilidade do sistema ao longo de seu desenvolvimento [9].

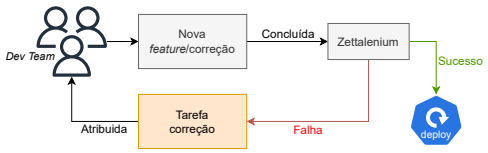


Figura 2: Fluxo de execução dos testes no SisNIR

As funções utilitárias do Zettalenium, como `readJson` para carregar dados e `assertJsonContains` para validar respostas, contribuíram para testes mais claros, padronizados e fáceis de manter. A automação proporcionada pelo *framework* se traduziu em um esforço significativamente menor por parte da equipe. Embora não tenha sido realizado um estudo formal, tal redução se justifica pela eliminação da necessidade de executar e validar testes manualmente, o que era um processo repetitivo e propenso a falhas.

Outro diferencial relevante foi o suporte à auditoria automática. A cada execução, o *framework* gera *logs* detalhados e relatórios em HTML, permitindo o acompanhamento da execução de forma *headless* e sem necessidade de intervenção manual. Esses relatórios, além de servirem como documentação técnica, foram utilizados como critério formal de aceite pelo MIDR, reforçando a transparência do processo.

Adicionalmente, observou-se um desempenho satisfatório: o tempo médio de execução por método foi de aproximadamente 300 ms, mesmo em ambiente remoto e com múltiplas dependências. Essa agilidade na execução possibilitou a validação contínua das funcionalidades do sistema, o que foi fundamental para o ciclo de desenvolvimento do SisNIR.

Em síntese, a adoção do Zettalenium no SisNIR demonstrou ser uma solução eficaz e confiável para estruturar, automatizar e auditar testes de APIs REST em projetos de sistemas para o setor público, indicando seu potencial como ferramenta de referência também para outros setores regulados, como o financeiro e o de saúde, com severos requisitos de conformidade, que exigem alto grau de controle, rastreabilidade e qualidade.

4 Considerações Finais

Este trabalho relatou a adoção do Zettalenium como solução de automação de testes funcionais no desenvolvimento do SisNIR. Embora o estudo se restrinja a um único projeto, o que representa uma limitação à generalização dos resultados, a utilização do *framework* possibilitou a estruturação de um conjunto consistente de testes automatizados, promovendo padronização, reduzindo duplicidades e facilitando a manutenção do código de testes ao longo do projeto. Além disso, atendeu aos requisitos de documentação e auditabilidade estabelecidos, contribuindo para a garantia da qualidade, confiabilidade e transparência do sistema durante todo o seu ciclo de desenvolvimento.

A aplicação no SisNIR também evidenciou a viabilidade do uso de *frameworks* personalizados em projetos governamentais, demonstrando que soluções especializadas podem atender de forma eficiente a requisitos específicos de conformidade e controle exigidos em ambientes públicos. Esses resultados indicam o potencial de adoção do Zettalenium em outros contextos organizacionais com demandas semelhantes.

Atualmente, o Zettalenium está sendo ampliado com funcionalidades de automação de testes ponta a ponta, integrando o *Selenium*[5] para possibilitar a validação de interfaces de usuário. Essa evolução busca transformar a ferramenta em uma solução mais abrangente e integrada, capaz de abranger tanto a validação de APIs quanto a verificação funcional de aplicações completas, atendendo a múltiplas etapas do ciclo de testes de software.

AGRADECIMENTOS

Os autores agradecem à *Zetta* a colaboração no desenvolvimento do Zettalenium, e ao *Ministério da Integração e do Desenvolvimento Regional (MIDR)* o apoio institucional e a oportunidade de aplicar e validar a ferramenta no contexto do *Sistema Nacional de Informações sobre Irrigação (SisNIR)*.

REFERÊNCIAS

[1] API Expert. 2023. The Limitations of Postman. Disponível em: <https://apiexpert.medium.com/the-limitations-of-postman-bff619577a7b>. Acesso em: 04 maio 2024.
[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. 1997. *Hypertext Transfer Protocol – HTTP/1.1.* RFC 2068. Internet Engineering Task Force (IETF).
[3] David Gourley and Brian Totty. 2002. *HTTP: The Definitive Guide.* O'Reilly Media, Sebastopol, Estados Unidos.
[4] Yi Liu. 2024. *Semantics-aware Human-computer Interaction Software Testing.* Ph. D. Dissertation. College of Computing and Data Science, Nanyang Technological University, Singapura.
[5] Sam Newman. 2015. *Building Microservices: Designing Fine-Grained Systems.* O'Reilly Media, Sebastopol, Estados Unidos.
[6] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON Schema. In *25th International World Wide Web Conference (WWW).* 263–273.
[7] Valeria Pontillo and Maxime Vandercammen. 2024. FERRARI: FailurE RepRoduction through automatic test cAse generation and stack tRace analysIs. In *23rd Belgium-Netherlands Software Evolution Workshop (BENEVOL).* 45–56.
[8] Amazon Web Services. 2024. O que é a API RESTful? Disponível em: <https://aws.amazon.com/what-is/restful-api/>. Acesso em: 29 maio 2024.
[9] Ian Sommerville. 2015. *Software Engineering.* Addison-Wesley, Boston, Estados Unidos.
[10] Testsigma Engineering Team. 2024. Insomnia vs Postman: Which is better for your project? Disponível em: <https://testsigma.com/blog/insomnia-vs-postman/>. Acesso em: 04 jul. 2024.

[5] https://www.selenium.dev/