Plano de Testes — Desafio Técnico QA

Autor: João Vìctor Santos Campos

Data: 06/09/2025

# 1. Objetivo

O objetivo deste plano de testes é garantir a qualidade da **aplicação web Automation Testing** **Demo**, da **API DummyJSON** e do **endpoint de performance JSONPlaceholder**, assegurando que seus requisitos funcionais e não funcionais sejam atendidos. Para isso, serão realizados testes manuais, automatizados (Web e API) e de performance, de forma a aumentar a confiança nos resultados, identificar riscos e fornecer evidências claras de conformidade com os critérios de aceitação definidos.

👉 Esta seção existe para que todos entendam claramente a missão dos testes — evita que o trabalho fique sem propósito definido.

# 2. Escopo

Os testes deste plano abrangem:

- Aplicação Web: [Automation Testing Demo] — testes de cadastro, login, formulários, upload, alertas, iframes.

- API REST: [DummyJSON] — validação de endpoints de produtos e usuários (GET, POST, PUT, DELETE, validação de contrato).

- Teste de Performance: [JSONPlaceholder] — endpoint /posts, validando capacidade de resposta e comportamento sob carga.

Estão fora do escopo: testes mobile, integrações com banco de dados, cenários não listados no desafio.

|  |  |
| --- | --- |
| Objetivo | Escopo relacionado |
| Garantir a qualidade da aplicação Automation Testing Demo | Testes funcionais na aplicação Web: cadastro, login, formulários, upload, alertas e iframes. |
| Validar a conformidade da API DummyJSON com requisitos funcionais e de contrato | Testes de API: CRUD de produtos e usuários (GET, POST, PUT, DELETE) e validação de contrato via JSON Schema. |
| Avaliar o desempenho do endpoint JSONPlaceholder (/posts) sob diferentes cargas | Testes de performance no endpoint /posts, analisando capacidade de resposta, throughput e taxa de erro. |
| Fornecer evidências que suportem decisões sobre confiabilidade e estabilidade dos sistemas | Resultados documentados em relatórios, métricas de execução e registros de falhas. |

👉 O escopo limita o que será testado. Isso evita gastar tempo em funcionalidades que não fazem parte do desafio e mostra foco.

# 3. Critérios de Sucesso / Aceitação

- **Funcionalidade (Web e API):** pelo menos 95% dos casos de teste automatizados e manuais executados devem ser aprovados. Casos críticos de cadastro, login e CRUD não podem falhar.

- **Conformidade da API:** 100% das respostas devem estar em conformidade com os contratos JSON Schema definidos.

- **Performance:** no teste de carga do endpoint /posts, o tempo de resposta no p95 deve ser inferior a 5 segundos e a taxa de erro não pode ultrapassar 10%.

- **Cobertura mínima:** todos os fluxos funcionais obrigatórios descritos no escopo (cadastro, login, upload, formulários, alertas/frames e CRUD da API) devem estar cobertos por casos de teste documentados e automatizados.

- **Evidências e rastreabilidade:** todas as execuções de teste devem gerar relatórios com logs, métricas e, em caso de falha, evidências (prints, request/response).

- **Entrega:** o projeto deve ser versionado em repositório Git, contendo README com instruções de execução, código de automação (Web + API), scripts de performance e relatório final.

👉 Sem critérios claros, não há como saber se o trabalho terminou ou se a qualidade é aceitável.

# 4. Estratégia de Testes

A estratégia de testes será conduzida em camadas (manuais, API, Web, performance), de forma a garantir cobertura funcional, integração de APIs, validação da experiência do usuário e resiliência da aplicação sob carga:

**- Testes Manuais:** utilizados inicialmente para validação exploratória e cobertura de cenários críticos da aplicação web (cadastro, login, upload de arquivos, formulários, alertas/frames). Também servirão para descobrir comportamentos inesperados não previstos na automação.

**- Testes Automatizados de API:** implementados com **RestAssured + JUnit**, cobrindo as operações de CRUD de produtos e usuários. Cada endpoint será validado quanto a: códigos de status, consistência dos dados retornados, tempo de resposta e conformidade com contratos JSON Schema. Casos negativos (ex.: payload inválido, ID inexistente) também serão incluídos.

**- Testes Automatizados de Web:** desenvolvidos em **Selenium** seguindo o padrão **Page Object Model (POM)**, para garantir manutenção e reuso. O foco estará em fluxos críticos (cadastro, login, upload e interações com formulários e frames), cobrindo cenários positivos e negativos. Serão implementados mecanismos de captura de evidências (screenshots e logs) em caso de falha.

**- Testes de Performance:** realizados com **k6**, simulando aumento progressivo de carga no endpoint /posts até identificar o ponto de falha. Serão coletadas métricas como tempo de resposta (p95), taxa de erros e throughput, com critério de aceitação de ≥95% de sucesso. O relatório final apresentará análise detalhada do comportamento sob carga.

Essa abordagem combina testes manuais exploratórios, automação funcional (API e Web) e testes não-funcionais (performance), permitindo validar tanto a corretude quanto a robustez e a escalabilidade da aplicação.

👉 Aqui se detalha o como. Essa seção mostra seu raciocínio sobre tipos de testes, ferramentas e abordagens.

# 5. Ambientes de Teste

**- Web:** execução em **Google Chrome (última versão estável)**, com testes locais.

**- API:** consumo direto dos endpoints públicos **DummyJSON** (CRUD de produtos e usuários) e **JSONPlaceholder** (endpoint /posts), respeitando limitações de uso e conectividade.

**-** **Performance:** execução local via CLI com **k6.**

👉 Definir ambiente evita discrepâncias — o avaliador saberá em que condições você rodou os testes.

# 6. Ferramentas

**- Controle de versão: Git + GitHub**

- **Automação de API**: Java 17, Maven, **RestAssured**, **JSON Schema Validator** para contrato, e **JUnit 5** como framework de testes.

- **Automação Web**: Java 17, **Selenium WebDriver**, **WebDriverManager** (gestão de drivers), **JUnit 5** e padrão **Page Object Model** para organização do código.

- **Performance**: **k6**, com scripts em JavaScript para cenários de carga, stress e análise de throughput/latência.

👉 Ferramentas documentadas dão transparência e mostram maturidade técnica nas escolhas.

# 7. Riscos

- Limitações de APIs públicas (rate limiting ou respostas inconsistentes).

- Alterações de layout na aplicação web de demo (pode quebrar testes UI).

- Diferenças de ambiente local (drivers, versões de navegador).

👉 Listar riscos mostra visão crítica: você sabe que nem tudo depende só do código do teste.

# 8. Plano de Dados de Teste

- **API**: utilização da biblioteca **Faker** para geração dinâmica de dados (nomes, e-mails, produtos), evitando duplicações e garantindo reprodutibilidade. IDs criados serão validados e descartados ao final dos testes (quando o endpoint suportar DELETE).

- **Web**: criação de usuários temporários e execução de cenários com dados controlados, garantindo independência entre testes. Sempre que possível, dados serão gerados dinamicamente para evitar conflitos com execuções anteriores.

**- Performance**: uso de payloads simples e repetitivos, simulando acessos de múltiplos usuários ao endpoint /posts, de forma a isolar gargalos de infraestrutura e não de massa de dados.

👉 Dados controlados garantem reprodutibilidade; evita usar dados “sujos” que quebram a automação.

# 9. Cronograma

1. Criação do Plano de Testes

2. Definição dos Casos de Teste.

3. Setup do repositório Git e estruturação inicial do projeto (pastas, dependências).

4. Implementação da automação de API

5. Implementação da automação Web

6. Execução de testes de performance

7. Documentação final e relatórios.

👉 Ter um cronograma mostra que você sabe priorizar e organizar sua entrega.

# 10. Entregáveis

- **Documento de Plano de Testes:** incluindo estratégia, riscos, dados e cronograma.

- **Casos de Teste**: em formato estruturado, com IDs e resultados esperados.

- **Repositório Git**: contendo código de automação de API e Web, scripts de performance e documentação de execução.

- Relatório final com evidências e métricas.

👉 Esta seção fecha o ciclo: define claramente o que será entregue, para que ninguém fique em dúvida sobre o resultado do trabalho.