Relatório de Melhorias

Projeto: Seleção QA PGE

Candidato: Lucas Araújo de Almeida

Versão 1.0
03/ 08 / 2025

Sumário

1. Introdução

Durante a execução do desafio técnico para a vaga na PGE, a API oficial disponibilizada para testes esteve fora do ar durante todo o período da análise, impedindo a execução direta dos testes propostos.

Para garantir a continuidade da entrega e manter o comprometimento com os requisitos do desafio, tomei a iniciativa de desenvolver uma API simulada, replicando a estrutura e os comportamentos esperados da API original da PGE. Essa solução própria foi adaptada para:

- Manter endpoints com a mesma semântica (/login, /contribuintes, /inscrições/:cpf);
- Suportar autenticação via token JWT;
- Reproduzir mensagens, erros e fluxos de negócio conforme o caderno de testes.

Todos os artefatos foram mantidos como se estivessem sendo utilizados com a API da PGE: plano de testes, caderno de testes, massa de dados via fixtures, comandos customizados no Cypress e organização por fluxos positivos e negativos. A adaptação não comprometeu a estrutura do projeto, apenas os dados e endpoints foram ajustados para refletir a nova fonte.

2. Melhorias Observadas

2.1. Integração com Pipeline de CI/CD

Uma melhoria relevante para o projeto seria a integração dos testes automatizados com uma pipeline de integração contínua. A proposta é utilizar o GitHub Actions para executar os testes automaticamente a cada push ou pull request no repositório. Isso garante que os testes sejam validados continuamente, evitando regressões e facilitando a detecção imediata de falhas. Além disso, traz mais confiabilidade para a equipe de desenvolvimento e QA, que passa a contar com um processo de validação automatizado em tempo real. Para este projeto, já entrego a suíte Cypress integrada com uma pipeline funcional configurada no GitHub Actions, pronta para ser usada ou expandida conforme necessidade.
Além disso, utilizei integração com Discord, enviando uma notificação com o resultado dos testes para um canal específico, garantindo visibilidade e transparência para a equipe.

2.2. Tratamento de Mensagens Genéricas

Durante os testes, identifiquei que a API retorna a mesma mensagem de erro genérica (“Usuário ou senha inválidos”) tanto para credenciais erradas quanto para campos vazios no login. Para melhorar a experiência do usuário final e até o debug em produção, a sugestão é diferenciar a mensagem quando os campos estão vazios para “Usuário e senha são obrigatórios”.

2.3. Implementação de Validação de JSON Schema

Atualmente, os testes automatizados com Cypress já validam corretamente os códigos de status e as mensagens de resposta da API. No entanto, para deixar a suíte de testes mais robusta e deixa-la mais preparada para escalabilidade, é interessante implementar a validação do schema dos objetos retornados (se os campos obrigatórios estão presentes, se os tipos estão corretos e se a resposta segue o contrato definido).

2.4. Adoção de Quality Gate (SonarQube)

Mesmo com a suíte completa e automatizada, não coloquei nenhum mecanismo que restrinja ou bloqueie código de baixa qualidade ou testes com cobertura insuficiente. A adoção de um quality gate como o SonarQube (que eu sei que é utilizado na PGE) permitiria que a qualidade do projeto fosse verificada automaticamente a cada novo commit ou pull request, nos dando maior confiança nas entregas.

2.5. Modularização e Escalabilidade da Suíte de Testes

Durante o desenvolvimento dos testes com Cypress foi realizada uma refatoração para organizar os casos de testes em arquivos separados por funcionalidade, o que contribui diretamente para a escalabilidade e manutenção do projeto. Os testes foram distribuídos em 3 arquivos principais, cobrindo a autenticação, cadastro e consultas. Além disso, foram criados comandos customizados para encapsular chamadas repetitivas, como login e requisições autenticadas, reduzindo a duplicidade de código e facilitando a leitura.

Também foi adotada uma estratégia de uso de dados dinâmicos, como a geração de CPFs aleatórios, para garantir que os testes sejam idempotentes e não causem conflitos na base simulada. O uso de fixtures para armazenar dados reutilizáveis também foi implementado a fim de garantir boa prática na automação.

Essa estrutura modular favorece o crescimento da suíte, facilita a manutenção e garante clareza no fluxo de testes, sendo adequada para projetos reais com várias rotas e lógicas de negócio distintas.