

Automação de testes de API no CSOnline

**Da Necessidade ao Sistema: Como Construímos uma Suite de Testes Automatizados para APIs REST**

**[[](https://www.linkedin.com/in/chmulato/)](https://www.linkedin.com/in/chmulato/)**

**[Christian Mulato](https://www.linkedin.com/in/chmulato/)**

Desenvolvedor Java Sênior | Especialista em Back-end | Jakarta, Spring Boot, REST APIs, Docker | Engenheiro Químico

20 de agosto de 2025

**1.1     O Desafio Inicial**

Durante o desenvolvimento do **CSOnline**, um sistema de gestão de entregas enterprise, nos deparamos com um problema comum mas crítico: como garantir que nossas APIs REST funcionassem consistentemente durante o desenvolvimento iterativo?

Com 7 módulos (Usuários, Entregadores, Clientes, Entregas, Equipes, SMS e Preços) e múltiplos endpoints CRUD, testar manualmente cada funcionalidade a cada deploy tornou-se insustentável.

**1.2     A Primeira Abordagem**

Tudo começou com um script simples: **test-couriers.ps1**. Era interativo, testava apenas um endpoint e requeria input manual. Funcionava, mas não escalava.

# Primeira versão - manual e limitada

Write-Host "Testando endpoint de couriers..."

$response = Invoke-RestMethod -Uri "http://localhost:8080/csonline/api/couriers"

**1.3     A Virada: Pensamento Sistemático**

Em agosto de 2025, decidimos abordar o problema de forma estruturada. A ideiafoi simples: se temos padrões na API, podemos ter padrões nos testes.

**Decisões arquiteturais tomadas:**

* **Um script por endpoint** - Cada módulo teria seu próprio script de teste
* **Operações CRUD completas** - Não apenas GET, mas POST, PUT e DELETE
* **Tratamento de erros padronizado** - Feedback claro e actionable
* **Automação total** - Zero interação manual

**1.4     A Implementação**

Em uma tarde intensa, nasceram 7 scripts especializados:

* test-users.ps1
* test-customers.ps1
* test-deliveries.ps1
* test-teams.ps1
* test-sms.ps1
* test-login.ps1
* health-check-endpoints.ps1

Cada script seguia o mesmo padrão:

function Test-EntityCRUD {

param([string]$BaseUrl, [string]$Entity)

try {

# GET - Lista

$list = Invoke-RestMethod -Uri "$BaseUrl/$Entity"

Write-Host "GET /$Entity : SUCESSO ($($list.Count) registros)" -ForegroundColor Green

# GET - Individual (se lista não estiver vazia)

if ($list.Count -gt 0) {

$individual = Invoke-RestMethod -Uri "$BaseUrl/$Entity/$($list[0].id)"

Write-Host "GET /$Entity/{id} : SUCESSO" -ForegroundColor Green

}

# POST, PUT, DELETE...

} catch {

Write-Host "ERRO em $Entity : $($\_.Exception.Message)" -ForegroundColor Red

}

}

**1.5     As Ferramentas de Orquestração**

Não bastava ter scripts individuais. Precisávamos de ferramentas para executá-los de forma inteligente:

* **test-all-endpoints.ps1** - O maestro que executa todos os testes em sequência, com opções de filtro.
* **health-check-endpoints.ps1** - Diagnóstico rápido que gera relatórios de status.
* **run-tests.ps1** - Interface simplificada na raiz do projeto.

**1.6     As Descobertas**

Os testes revelaram insights valiosos:

* **Taxa de sucesso: 80%** (8 de 10 endpoints funcionando)
* **Problemas identificados:** Serialização circular, endpoints individuais com 404
* **Padrões de falha:** ***LocalDateTime*** causando problemas de ***deserialização***

**1.7     Os Benefícios Realizados**

* **Detecção proativa de problemas** - Bugs descobertos antes do deploy
* **Feedback imediato** - Saber em segundos se uma mudança quebrou algo
* **Documentação viva** - Os testes servem como especificação da API
* ***Onboarding* facilitado** - Novos desenvolvedores entendem rapidamente o estado do sistema

**1.8     Lições Aprendidas**

**Técnicas:**

* ***PowerShell*** é uma ferramenta poderosa para automação de testes de API
* Padronização é fundamental para manutenibilidade
* Feedback colorido e estruturado acelera o debugging

**Organizacionais:**

* Investir tempo em automação de testes economiza horas de trabalho manual
* Testes devem ser executáveis por qualquer membro da equipe
* Documentação de testes é tão importante quanto a documentação da API

**1.9     O Resultado**

Hoje, qualquer mudança no **CSOnline** pode ser validada em minutos:

# Um comando para testar tudo

.\run-tests.ps1

# Ou verificação rápida de saúde

.\run-tests.ps1 -HealthCheck

**1.10  Próximos Passos**

A jornada não para aqui. Estamos expandindo para:

* Testes de carga automatizados
* Integração com CI/CD
* Testes de contratos com Pact
* Monitoramento de performance

**1.11  Reflexão Final**

Construir ferramentas de teste não é apenas sobre encontrar bugs. É sobre criar confiança. Confiança para ***refatorar***, para experimentar, para inovar.

No desenvolvimento de software, a pergunta não é “se” algo vai quebrar, mas “quando”. Ter uma rede de segurança robusta transforma o medo de mudança em coragem para evolução.

**1.12  Recursos e Contato**

**Call to Action:** E você, como garante a qualidade das suas APIs durante o desenvolvimento? Compartilhe suas estratégias nos comentários!

*Artigo baseado na experiência real de desenvolvimento do sistema CSOnline - um projeto enterprise de gestão de entregas com Jakarta EE, Vue 3 e WildFly.*