Plano de Testes Melhorado - API Serverest

1. Apresentação

Este documento apresenta o planejamento de testes para a API **ServeRest**, com o objetivo de validar o cumprimento das regras de negócio, assegurar a qualidade das APIs REST e viabilizar a automação de testes.

O planejamento inclui cenários de teste detalhados, priorização baseada em criticidade, matriz de risco, cobertura total de endpoints, e práticas voltadas para testes automatizados utilizando **Robot Framework**.

2. Objetivo

Garantir que todos os endpoints da API ServeRest funcionem conforme os requisitos de negócio, assegurando:

- Validação de funcionalidades principais (CRUD de usuários, produtos e carrinhos).
- Conformidade com regras de negócio críticas (ex.: estoque, carrinho único por usuário).
- Identificação de falhas, vulnerabilidades e riscos de integridade.
- Viabilidade e implementação de **testes automatizados** de APIs REST.
- Cobertura de cenários de sucesso, falha, limites e concorrência.

3. Escopo

O escopo abrange testes **funcionais e de regras de negócio** das APIs REST:

- **Autenticação** → `/login`
- **Usuários** → `/usuarios`
- **Produtos** → `/produtos`
- **Carrinhos** → `/carrinhos`

Abrange:

- Regras de negócio específicas (ex.: carrinho vinculado a um usuário, controle de estoque).
- Cenários de sucesso, erro, limite e dados inválidos.
- Cenários de concorrência (múltiplos usuários manipulando o mesmo produto).
- Preparação e limpeza automatizada de dados para testes repetíveis.

4. Análise

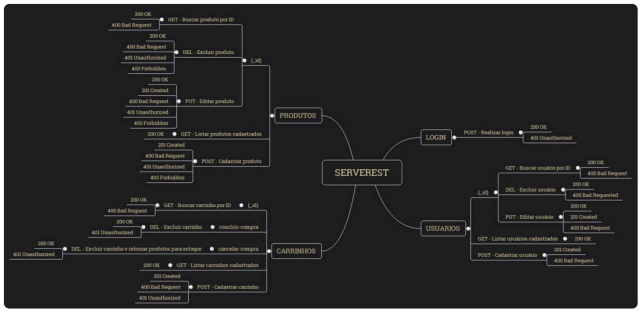
A API ServeRest é um simulador de e-commerce com operações CRUD. Pontos críticos a serem testados:

- Estoque reduzido automaticamente ao inserir produto no carrinho.
- Apenas um carrinho por usuário.
- Exclusão de usuários com carrinho ativo não permitida.
- Exclusão de produtos vinculados a carrinhos não permitida.
- Autenticação obrigatória para operações restritas.
- Manutenção da integridade de dados em operações simultâneas.

5. Técnicas aplicadas

- **Path Coverage** → Garantir cobertura completa dos endpoints da API.
- **Testes parametrizados** → Execução de cenários com múltiplos dados (usuários, produtos, quantidades).
- **Validação de contrato JSON** → Checagem de schemas para consistência de payload.
- **Testes de concorrência** → Verificação da integridade de estoque e carrinhos em múltiplas requisições simultâneas.

6. Mapa mental da aplicação



7. Cenários de teste

Estrutura utilizada:

- **Pré-condição**: estado inicial.
- **Passos**: ações executadas.
- **Resultado esperado**: resposta esperado.

7.1 Usuários

Cenário 1: Criar usuário com dados válidos

- Pré-condição: Nenhum usuário cadastrado com email `novo@qa.com`.
- Passos: Enviar `POST /usuarios` com dados válidos.
- Resultado esperado: **201 Created** e mensagem `"Cadastro realizado com sucesso"`.

Cenário 2: Criar usuário já existente

- Pré-condição: Usuário com email `teste@qa.com` já cadastrado.
- Passos: Enviar `POST /usuarios` com mesmo email.
- Resultado esperado: **400 Bad Request**, mensagem `"Este email já está sendo usado"` ou algo relacionado.

Cenário 3: Excluir usuário com carrinho ativo

- Pré-condição: Usuário com carrinho em aberto.
- Passos: Enviar `DELETE /usuarios/:id`.
- Resultado esperado: **400 Bad Request**, mensagem `"Não é permitido excluir usuário com carrinho"` ou algo relacionado.

7.2 Produtos

Cenário 1: Criar produto com token válido

- Pré-condição: Usuário administrador autenticado.
- Passos: Enviar `POST /produtos` com dados válidos.
- Resultado esperado: **201 Created**, mensagem `"Cadastro realizado com sucesso"` ou algo relacionado.

Cenário 2: Criar produto sem token

- Pré-condição: Nenhuma autenticação enviada.
- Passos: Enviar `POST /produtos`.
- Resultado esperado: **401 Unauthorized**, mensagem `"Token de acesso ausente ou inválido"` ou algo relacionado.

Cenário 3: Excluir produto vinculado a carrinho

- Pré-condição: Produto já em carrinho ativo.
- Passos: Enviar `DELETE /produtos/:id`.

- Resultado esperado: **400 Bad Request**, mensagem `"Não é permitido excluir produto que faz parte de carrinho"` ou algo relacionado.

7.3 Carrinhos

Cenário 1: Criar carrinho com produto em estoque

- Pré-condição: Produto com quantidade ≥ 1.
- Passos: Enviar `POST /carrinhos` vinculando produto.
- Resultado esperado: **201 Created**, estoque reduzido.

Cenário 2: Tentar criar dois carrinhos para o mesmo usuário

- Pré-condição: Usuário já possui carrinho ativo.
- Passos: Enviar `POST /carrinhos` novamente.
- Resultado esperado: **400 Bad Request**, mensagem `"Não é permitido ter mais de um carrinho"` ou algo relacionado.

Cenário 3: Concluir compra e validar redução de estoque

- Pré-condição: Carrinho ativo com produtos em estoque.
- Passos: Enviar `DELETE /carrinhos/concluir-compra`.
- Resultado esperado: **200 OK**, estoque atualizado corretamente.

7.4 Autenticação e segurança

Cenário 1: Login com credenciais válidas

- Passos: Enviar `POST /login` com email e senha corretos.
- Resultado esperado: **200 OK**, token retornado válido com expiração de 600s.

Cenário 2: Acesso a endpoint administrativo com token de usuário comum

- Pré-condição: Usuário autenticado sem privilégios.
- Passos: Enviar `POST /produtos` com token comum.
- Resultado esperado: **403 Forbidden**, mensagem `"Rota exclusiva para administradores"` ou algo relacionado.

8. Priorização da execução dos cenários

Prioridade	Cenários	Priorização

Alta	CRUD de usuários e produtos, criação/conclusão/cancelamento de carrinhos, autenticação, regras críticas de estoque, concorrência de carrinhos/estoque.	CRUD de usuários/produtos, carrinho, login, regras críticas de estoque e concorrência.
Média	Cenários de campos inválidos, duplicidade de cadastro, manipulação de carrinhos com produtos fora de estoque, atualização de dados inválidos.	Campos inválidos, duplicidade de cadastro, manipulação de carrinhos sem estoque.
Baixa	Cenários extremos e incomuns, performance de endpoints não críticos, filtros de pesquisa complexos.	Casos extremos (inputs muito grandes, filtros complexos, performance não crítica).

9. Matriz de risco

Risco	Impacto	Probabilidade	Nível
Estoque incorreto após compra	Alto	Alta	Crítico
Usuário com mais de um carrinho	Alto	Média	Alto
Autenticação falha permite	Alto	Baixa	Alto

acesso restrito			
Exclusão de produto vinculado a carrinho permitida	Médio	Alta	Alto
Inconsistência de estoque em operações simultâneas	Alto	Média	Crítico
Falha na revogação de token	Médio	Média	Médio

10. Cobertura de testes

- 100% dos endpoints principais.
- 100% das regras de negócio documentadas.
- Cenários de sucesso e falha para cada operação CRUD.
- Cenários com dados no limite e inválidos.
- Cenários de concorrência e simultaneidade.
- Validação de schema JSON e mensagens de erro.

11. Testes candidatos à automação (Robot Framework)

- **Login** → credenciais válidas e inválidas
- **CRUD completo de usuários e produtos**
- **Criação, atualização e manipulação de carrinhos**
- **Cenários de estoque** → redução, reposição e concorrência
- **Cenários de erro comuns** → status code e mensagens detalhadas
- **Validação de schema JSON** → integridade do payload
- **Preparação e limpeza de dados automatizada** → usuários, produtos e carrinhos

Ferramentas e bibliotecas sugeridas:

- **Robot Framework**
- **RequestsLibrary** → para chamadas HTTP e validação de respostas

- **JSONLibrary** → validação de payload
- **Data-driven tests** → execução de cenários com múltiplos dados parametrizados

12. Boas práticas para automação

1. **Separação de cenários críticos e não críticos** para integração contínua (CI/CD).
2. **Execução parametrizada** para múltiplos usuários, produtos e quantidades.
3. **Validação de payload** e **status codes** em todas as respostas.
4. **Preparação e limpeza automática de dados** para garantir consistência entre execuções.
5. **Testes de concorrência** para verificar integridade do estoque e regras de carrinho.
6. **Relatórios automatizados** do Robot Framework para análise de resultados.

13. Integração com ferramentas

- **QALity (Jira)** → registro de casos, execução manual, evidências.
- **GitHub** → versionamento de código e plano de testes.
- **Robot Framework** → automação.