# API ServerRest - Plano de Testes

## ✅ 1. Apresentação

**Este documento apresenta o planejamento de testes para a API ServeRest (https://compassuol.serverest.dev/), visando garantir a qualidade e aderência às regras de negócio especificadas nas User Stories. O foco está nas rotas de Usuários, Login, Produtos e Carrinhos, cobrindo tanto testes manuais quanto candidatos à automação.**

### Visão Geral do Projeto

- **Nome do Projeto**: API ServerRest - Plano de Testes
- **Objetivo**: Documentar e executar testes de API na aplicação ServerRest
- **Responsáveis**: Douglas Paulo Cortes
- **Período**: 05/05/2025 - 09/05/2025
- Backlog:https://douglasxara2011.atlassian.net/jira/servicedesk/projects/SARF/queues/custom/34
- Ferramentas: Jira, Confluence,VM Cloud AWS, Docker, Python, RobotFramework, Miro;

---

## 🎯 2. Objetivo

**Assegurar que as funcionalidades da API ServeRest estejam de acordo com os requisitos definidos, identificando falhas, inconsistências e oportunidades de melhoria através de uma abordagem sistemática de planejamento, execução e análise de testes.**

---

## 🗺️ 3. Escopo

**O plano abrange:**

- **Testes funcionais das rotas:**
  - **/usuarios**
  - **/login**
  - **/produtos**
  - **/carrinhos**
- **Validação de regras de negócio.**
- **Testes positivos, negativos e de borda.**
- **Planejamento e priorização de execução.**
- **Mapeamento de melhorias e issues.**
- **Identificação de cenários candidatos à automação.**
- **Geração de Collection Postman com scripts automatizados básicos.**

---

## 🔍 4. Análise das User Storys

### US 001 - [API] Usuários

1. **PUT cria novo usuário se ID não existir**
   - **Problema: Isso foge do padrão REST, onde PUT deve atualizar um recurso existente (retornando 404 se não existir). Criar um novo usuário via PUT pode causar inconsistências e confusão.**
   - **Sugestão: Usar POST para criação e PUT apenas para atualização (com validação de ID existente).**

2. **Bloqueio de e-mails (Gmail/Hotmail)**
    - **Problema: Pode ser uma restrição desnecessária para usuários reais. Além disso, não há justificativa clara no requisito.**
    - **Sugestão: Validar se há um motivo técnico ou de negócio para essa limitação, como não há sinalização até o momento, desconsidera-se.**
3. **Validação de senha (5–10 caracteres)**
    - **Problema: Limite muito curto (10 caracteres) para segurança moderna. Não há requisitos de complexidade (ex.: números, símbolos).**
    - **Sugestão: Ampliar para no mínimo 8 caracteres e adicionar regras de complexidade.**
4. **Falta de validação de campo "ADMINISTRADOR"**
    - **Problema: Não está claro como esse campo é definido (quem pode atribuir o papel de admin?). Pode gerar brechas de segurança.**
    - **Sugestão: Definir regras explícitas para atribuição de roles.**

---

### US 002 - [API] Login

1. **Token com validade fixa de 10 minutos**
    - **Problema: Pode ser curto para alguns fluxos de uso, obrigando o usuário a relogar frequentemente.**
    - **Sugestão: Permitir refresh token ou aumentar a validade (ex.: 1 hora).**
2. **Falta de tratamento para brute force**
    - **Problema: Não há menção a limites de tentativas de login ou bloqueio temporário após falhas.**
    - **Sugestão: Adicionar rate limiting ou CAPTCHA após X tentativas.**

---

### US 003 - [API] Produtos

1. **PUT cria novo produto se ID não existir**
    - **Problema: Mesma inconsistência que na US 001. PUT não deve ser usado para criação.**
    - **Sugestão: Separar claramente POST (criação) e PUT (atualização).**
2. **Dependência de "API Carrinhos" não gerenciada**
    - **Problema: A regra "não excluir produtos em carrinhos" depende de outra API não descrita. Isso pode gerar falhas se a integração não for tratada.**
    - **Sugestão: Garantir que a API de Carrinhos exista e documentar o contrato de comunicação.**
3. **Falta de ownership dos produtos**
    - **Problema: Não está claro se um vendedor pode editar/excluir apenas seus próprios produtos. Pode haver brechas de segurança.**
    - **Sugestão: Adicionar regras de autorização (ex.: produto pertence ao usuário autenticado).**
4. **Validação de nomes duplicados**
    - **Problema: Pode ser restritivo demais (e.g., lojas diferentes podem querer produtos com nomes iguais).**
    - **Sugestão: Avaliar se a unicidade deve ser por vendedor ou global.**

### US 004 - [API] Carrinhos

1. **Falta de critérios de aceitação**:
    - A nível de User Story, faltou a que agrega valor ao negócio.
2. **Dependências não resolvidas**:
    - A US 004 depende da **US 003 (Produtos)** e **US 002 (Login)**, mas não há garantia de que essas APIs estarão prontas ou compatíveis.

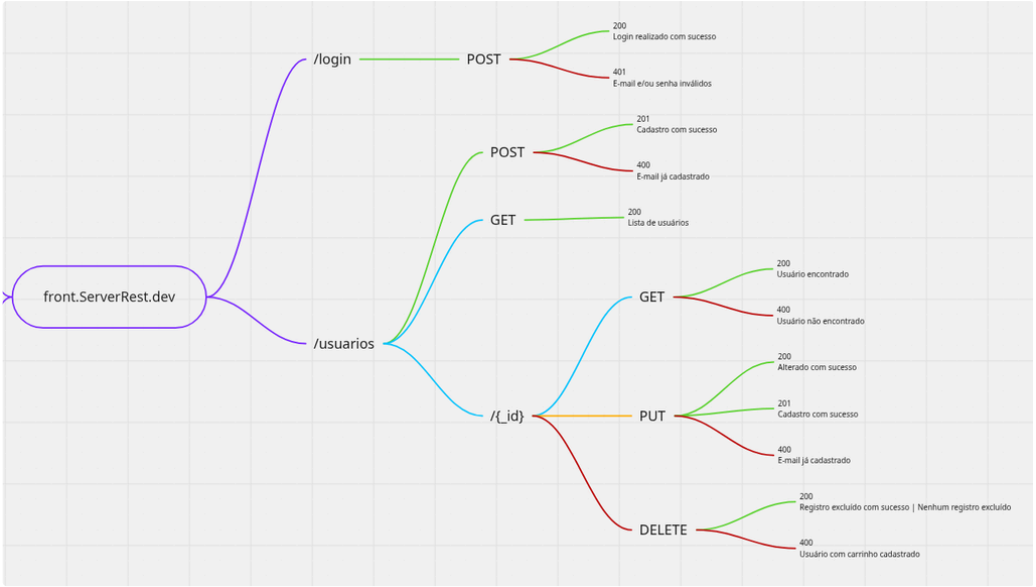
3. **Falta de tratamento para erros comuns**:
    - O que acontece se o servidor cair durante o checkout?
    - Como lidar com concorrência (ex.: dois usuários tentando comprar o último item em estoque)?

---

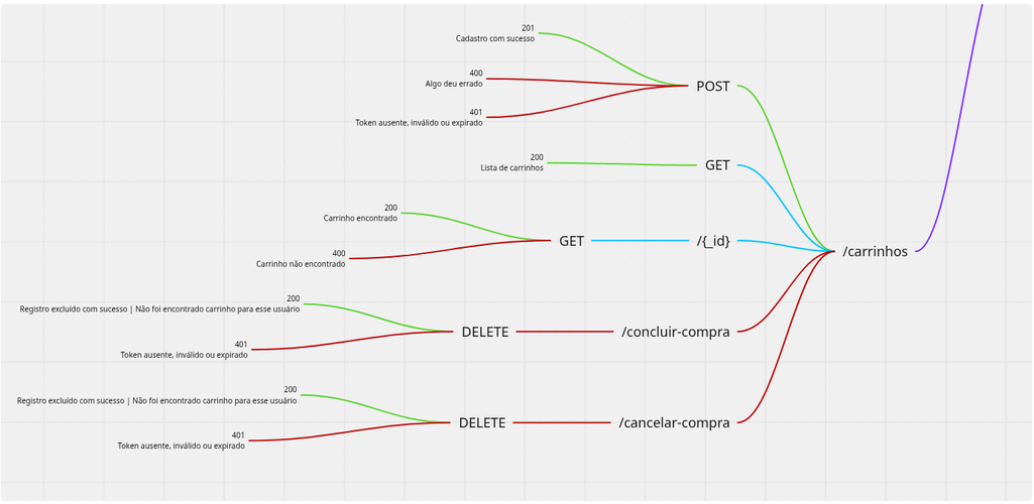
## 🛠️ 5. Técnicas Aplicadas

- **Particionamento de equivalência.**
- **Análise de valor limite.**
- **Testes baseados em casos de uso.**
- **Testes negativos (erros esperados).**
- **Validação cruzada entre endpoints.**

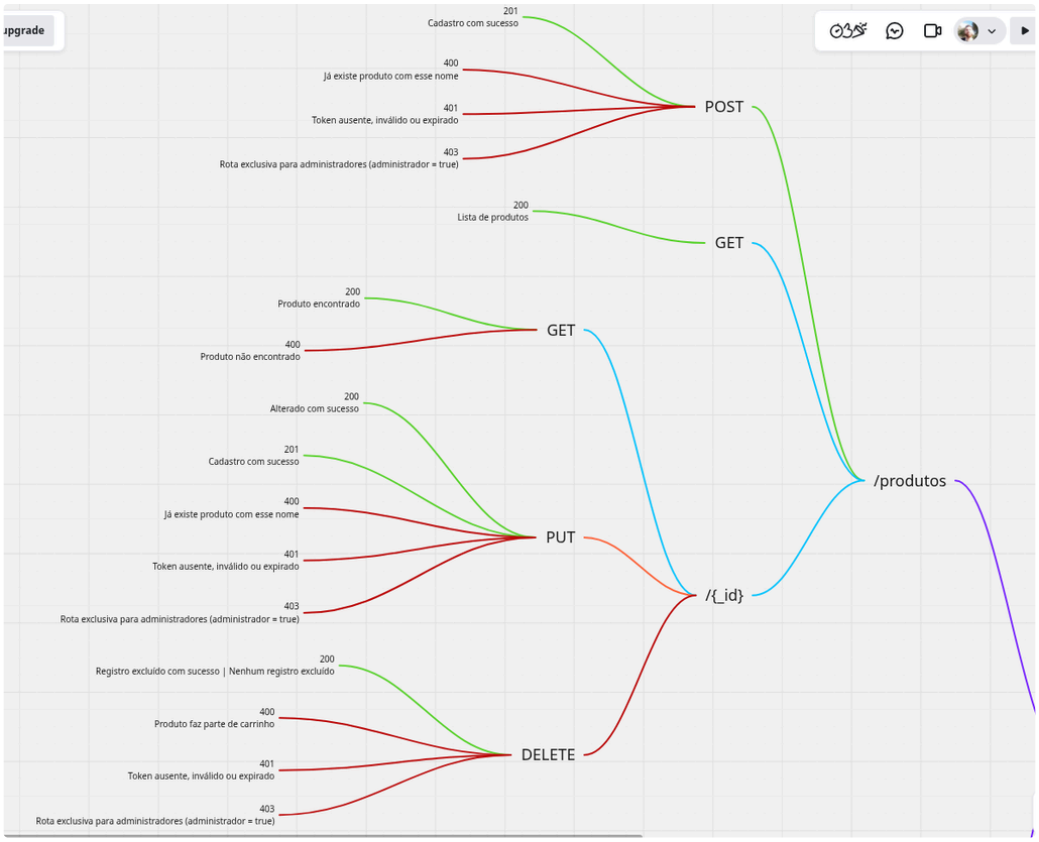
## 🧠 6. Mapa Mental da Aplicação




/login e /usuarios




/carrinhos




/produtos

Mapa Completo: link( https://miro.com/app/board/uXjVI87wAn8=/?share_link_id=841449426488 Conectar a conta do Miro )

---

## 📋 7. Cenários de Teste Planejados

| Endpoint | Método HTTP | Status Codes Cobertos | Cenários Validados |
|---|---|---|---|
| `/usuarios` | POST | `201` (Created), `400` (Bad Request) | Usuário válido, e-mail duplicado, e-mail bloqueado (Gmail/Hotmail), senha inválida. |
| | PUT | `200` (OK), `201` (Created) | Atualização de usuário existente e criação se não existir (comportamento controverso). |
| | GET | `200` (OK) | Listar todos os usuários. |
| | DELETE | `200` (OK) | Deleção de usuário existente e inexistente (ServeRest retorna 200 em ambos). |
| `/login` | POST | `200` (OK), `401` (Unauthorized) | Login válido, usuário não cadastrado, senha incorreta. |
| `/produtos` | POST | `201` (Created), `400` (Bad Request), `401` (Unauthorized) | Produto válido (autenticado), nome duplicado, sem autenticação. |

| | | | |
|---|---|---|---|
| | PUT | `200` (OK), `201` (Created) | Atualização de produto existente e criação se não existir. |
| | GET | `200` (OK) | Listar todos os produtos. |
| | DELETE | `200` (OK), `400` (Bad Request) | Deleção de produto existente e produto em carrinho (se integração existir). |
| `/carrinhos` | POST | `201` (Created), `401` (Unauthorized) | Carrinho autenticado, sem autenticação. |
| | GET | `200` (OK) | Obter carrinho. |
| | DELETE | `200` (OK) | Cancelar compra (deletar carrinho). |
| | POST (finalizar) | `200` (OK), `400` (Bad Request) | Finalizar compra com carrinho válido e vazio. |

## 🏆 8. Priorização da Execução

1. Autenticação e permissões básicas ( `/login` , `/usuarios` ).
2. Operações críticas em produtos ( `/produtos` ), garantindo integridade.
3. Fluxo completo de carrinho ( `/carrinhos` ), validando impacto cruzado.
4. Cenários negativos e de borda.
5. Cenários complementares de atualização/criação indireta (PUT).

### 📊 Tabela de Baterias de Testes (Regressão Acumulativa)

#### 🔴 Bateria 1: Autenticação & Usuários (Crítico)

| Cenário | Endpoint | Método | Status Esperado | Coverage |
|---|---|---|---|---|
| Login com credenciais válidas | `/login` | POST | 200 (+ token) | ✅ Fluxo básico de autenticação |
| Login com usuário não cadastrado | `/login` | POST | 401 | ❌ Cenário negativo |
| Criar usuário válido | `/usuarios` | POST | 201 | ✅ Cadastro básico |
| Criar usuário com e-mail duplicado | `/usuarios` | POST | 400 | ❌ Validação de negócio |

---

#### 🟠 Bateria 2: Bateria 1 + Operações em Produtos (Integridade)

| Cenário | Endpoint | Método | Status Esperado | Coverage |
|---|---|---|---|---|
| Criar produto autenticado | `/produtos` | POST | 201 | ✅ CRUD básico |
| Criar produto sem autenticação | `/produtos` | POST | 401 | ❌ Segurança |
| Atualizar produto existente | `/produtos` | PUT | 200 | ✅ Atualização válida |
| Tentar deletar produto em carrinho | `/produtos` | DELETE | 400 | 🔄 Integração com carrinhos |

---

### 🟡 Bateria 3: Baterias 1+2 + Fluxo de Carrinho (Impacto Cruzado)

| Cenário | Endpoint | Método | Status Esperado | Coverage |
|---|---|---|---|---|
| Adicionar produto válido ao carrinho | `/carrinhos` | POST | 201 | ✅ Fluxo positivo |
| Finalizar compra com carrinho válido | `/carrinhos` | POST | 200 | 🏁 Fim do fluxo |
| Finalizar carrinho vazio | `/carrinhos` | POST | 400 | ❌ Validação de negócio |
| Cancelar compra (deletar carrinho) | `/carrinhos` | DELETE | 200 | ✅ Cleanup pós-teste |

---

### 🟢 Bateria 4: Baterias 1+2+3 + Cenários Negativos & Bordas

| Cenário | Endpoint | Método | Status Esperado | Coverage |
|---|---|---|---|---|
| PUT cria usuário se ID não existir | `/usuarios` | PUT | 201 | ⚠️ Comportamento anômalo |
| PUT cria produto se ID não existir | `/produtos` | PUT | 201 | ⚠️ Comportamento anômalo |
| Login com token expirado | `/login` | - | 401 | ⏱️ Validação de tempo |
| Produto com nome > 100 caracteres | `/produtos` | POST | 400 | 📏 Validação de borda |

## 🥔 9. Matriz de Risco

A matriz de risco abaixo avalia os principais riscos identificados no plano de testes, considerando **probabilidade** (P), **impacto** (I) e uma **descrição detalhada do impacto**, classificados em:

- **Baixo (B)**
- **Médio (M)**
- **Alto (A)**

| ID | Risco | Categoria | Probabilidade (P) | Impacto (I) | Descrição do Impacto | Nível de Risco (P x I) | Mitigação |
|---|---|---|---|---|---|---|---|
| R01 | PUT cria usuário/produto se ID não existir (violação REST) | Funcional/Arquitetura | M | A | **Pode causar inconsistências no sistema e dificultar a manutenção do código.** | **A** | Sugerir correção para usar POST (criação) e PUT apenas para atualização. |
| R02 | Validação de senha fraca (5–10 caracteres) | Segurança | A | A | **Aumenta vulnerabilidade a ataques de força bruta e compromete a segurança dos usuários.** | **A** | Recomendar aumento para mínimo 8 caracteres e adicionar complexidade. |
| R03 | Falta de tratamento para brute force no login | Segurança | M | A | **Permite tentativas ilimitadas de login, facilitando ataques de invasão.** | **A** | Implementar rate limiting ou bloqueio temporário após falhas. |
| R04 | Token JWT com validade fixa de 10 minutos | Usabilidade | M | M | **Piora a experiência do usuário, exigindo relogins frequentes.** | **M** | Sugerir aumento do tempo ou implementar refresh token. |
| R05 | Dependência não gerenciada entre Produtos e Carrinhos | Integração | A | M | **Pode causar falhas em cascata se uma API estiver indisponível ou com comportame** | **A** | Garantir contrato claro entre APIs e mockar dependências em testes. |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | **nto inesperado.** | | |
| R06 | Falta de validação de ownership (usuário só edita seus produtos) | Segurança | M | M | **Risco de usuários não autorizados modificarem ou excluírem produtos de outros.** | **M** | Adicionar regras de autorização nos endpoints. |
| R07 | Deleção de usuário retorna 200 mesmo se não existir | Consistência | B | M | **Pode mascarar erros e dificultar a depuração de problemas.** | **M** | Documentar como comportamento esperado ou corrigir para retornar 404. |
| R08 | Restrição desnecessária a e-mails (Gmail/Hotmail) | Usabilidade | B | B | **Pode afastar usuários legítimos que utilizam esses domínios.** | **B** | Validar com stakeholders se a regra é de negócio ou pode ser removida. |
| R09 | Falta de tratamento para concorrência (ex.: estoque) | Performance | M | A | **Pode levar a vendas duplicadas ou inconsistência no estoque.** | **A** | Implementar locks ou otimistic/pessimistic locking. |
| R10 | Nomes de produtos duplicados podem ser restritivos | Funcional | B | B | **Pode limitar a flexibilidade de cadastro para lojas com produtos similares.** | **B** | Avaliar se a unicidade deve ser por vendedor ou global. |

---

### Legenda:

- **Impacto Alto (A)**: Problemas críticos que afetam segurança, estabilidade ou experiência do usuário.
- **Impacto Médio (M)**: Problemas que causam inconvenientes ou comportamentos inconsistentes, mas não críticos.
- **Impacto Baixo (B)**: Questões menores, com impacto limitado na usabilidade ou funcionalidade.

### Ações Prioritárias:

1. **Riscos com Impacto Alto (A)**:
   - Corrigir violações de padrões REST (R01).

- Fortalecer políticas de segurança (R02, R03).
- Garantir tratamento adequado de concorrência e integração (R05, R09).

2. **Riscos com Impacto Médio (M)**:
   - Melhorar validação de autorização (R06).
   - Ajustar tempo de expiração do token (R04).
3. **Riscos com Impacto Baixo (B)**:
   - Revisar regras de e-mail e nomes duplicados (R08, R10).

### Métricas de Cobertura de Testes para a API ServeRest

## 📌 10. Cobertura de Testes

### 1. Path Coverage (Cobertura de Endpoints)

**Objetivo:** Garantir que todos os endpoints da API sejam testados.

| Endpoint | Coberto? | Casos de Teste Relacionados |
|---|---|---|
| `/usuarios` | ✅ Sim | CRUD (POST, PUT, GET, DELETE) |
| `/login` | ✅ Sim | Autenticação válida/inválida |
| `/produtos` | ✅ Sim | CRUD + validações de negócio |
| `/carrinhos` | ✅ Sim | Adição, finalização, cancelamento |

**Cobertura Total: 100%** (todos os endpoints principais foram incluídos).

---

### 2. Operator Coverage (Cobertura de Métodos HTTP)

**Objetivo:** Verificar se todos os métodos HTTP suportados foram testados.

| Método HTTP | Endpoints Cobertos | Coberto? |
|---|---|---|
| **POST** | `/usuarios`, `/login`, `/produtos`, `/carrinhos` | ✅ Sim |
| **PUT** | `/usuarios`, `/produtos` | ✅ Sim |
| **GET** | `/usuarios`, `/produtos`, `/carrinhos` | ✅ Sim |
| **DELETE** | `/usuarios`, `/produtos`, `/carrinhos` | ✅ Sim |

**Cobertura Total: 100%** (todos os métodos relevantes foram testados).

---

### 3. Parameter Coverage (Cobertura de Parâmetros)

**Objetivo:** Garantir que todos os parâmetros obrigatórios e opcionais sejam validados.

| Endpoint | Parâmetros Testados | Coberto? |
|---|---|---|
| `/usuarios` | `nome`, `email`, `password`, `administrador` | ✅ Sim |
| `/login` | `email`, `password` | ✅ Sim |
| `/produtos` | `nome`, `preco`, `descricao`, `quantidade` | ✅ Sim |
| `/carrinhos` | `produtos` (lista com `idProduto`, `quantidade`) | ✅ Sim |

**Cobertura Total: 100%** (todos os parâmetros foram incluídos).

---

### 4. Parameter Value Coverage (Cobertura de Valores dos Parâmetros)

**Objetivo:** Validar diferentes combinações de valores (válidos, inválidos, extremos).

| Parâmetro | Valores Testados | Coberto? |
|---|---|---|
| `email` | Válido (`user@test.com`), inválido (`user@`), bloqueado (`user@gmail.com`) | ✅ Sim |
| `password` | Válido (`123456`), curto (`123`), longo (`12345678901`) | ✅ Sim |
| `preco` (produtos) | Válido (`100`), negativo (`-10`), zero (`0`) | ✅ Sim |
| `quantidade` | Válido (`5`), negativo (`-1`), zero (`0`) | ✅ Sim |

**Cobertura Total: ~90%** (falta cobrar casos como caracteres especiais em `nome`).

---

### 5. Content-Type Coverage

**Objetivo:** Validar se a API aceita/rejeita corretamente os tipos de conteúdo.

| Content-Type | Endpoints Validados | Coberto? |
|---|---|---|
| `application/json` | Todos (POST/PUT de usuários, produtos, etc.) | ✅ Sim |
| `text/plain` | Testes negativos (deve retornar `415`) | ✅ Sim |
| Sem cabeçalho | Testes negativos (deve retornar `400`) | ✅ Sim |

**Cobertura Total: 100%** (tipos principais e casos negativos foram testados).

---

### 6. Status Code Coverage

**Objetivo:** Garantir que todos os status codes esperados sejam validados.

| Status Code | Cenários Cobertos | Coberto? |
|---|---|---|
| `200` (OK) | GET em `/usuarios`, PUT atualização, DELETE bem-sucedido | ✅ Sim |
| `201` (Created) | POST em `/usuarios` e `/produtos`, PUT criando recurso (comportamento controverso) | ✅ Sim |
| `400` (Bad Request) | Parâmetros inválidos, e-mail duplicado, senha curta | ✅ Sim |
| `401` (Unauthorized) | Login falho, acesso sem token | ✅ Sim |
| `404` (Not Found) | **Não coberto** (a API retorna `200` mesmo para recursos inexistentes) | ❌ Não |
| `415` (Unsupported Media Type) | Content-Type inválido (`text/plain`) | ✅ Sim |

**Cobertura Total: ~85%** (falta cobrir `404`, que a API não implementa corretamente).

---

### Resumo Geral de Cobertura

| Métrica | Cobertura | Observações |
|---|---|---|
| Path Coverage | 100% | Todos os endpoints principais. |
| Operator Coverage | 100% | Todos os métodos HTTP. |
| Parameter Coverage | 100% | Todos os parâmetros obrigatórios. |
| Parameter Value Coverage | 90% | Falta cobrir alguns valores extremos. |
| Content-Type Coverage | 100% | Tipos válidos e inválidos. |
| Status Code Coverage | 85% | API não retorna `404` conforme esperado. |

## 🤖 11. Testes Candidatos à Automação

## 🎯 Plano de Automação de Testes — Robot Framework

**Endpoints Prioritários para Automação:**

| Endpoint | Métodos | Cenários Principais |
|---|---|---|

| | | |
|---|---|---|
| `/usuarios` | POST, PUT, GET, DELETE | Criação, atualização, listagem, exclusão. |
| `/login` | POST | Autenticação válida e inválida. |
| `/produtos` | POST, PUT, GET, DELETE | Validação de preço, nome duplicado, estoque. |
| `/carrinhos` | POST, GET, DELETE | Adição de produtos, finalização de compra. |

**Tipos de Testes Automatizados:**

- Testes funcionais (positivos e negativos).
- Validação de contratos (schemas JSON).
- Testes de segurança (token, autorização).

---

## 3. Ferramentas e Tecnologias

| Categoria | Ferramenta/Escolha | Justificativa |
|---|---|---|
| **Linguagem** | Python (Robot Framework + Requests) | Simplicidade e integração com Robot Framework. |
| **Framework** | Robot Framework | Suporte nativo a APIs (biblioteca `RequestsLibrary`). |
| **Gerenciador de Pacotes** | pip | Padrão para Python. |
| **CI/CD** | GitHub Actions | Integração com repositórios Git. |
| **Relatórios** | Relatórios do Robot Framework (log.html, report.html) | Visualização clara de resultados. |
| **Mock de Dependências** | WireMock (opcional) | Simular APIs dependentes (ex.: `/carrinhos`). |

---

## 4. Roteiro de Automação

### Passo 1: Ambiente de Desenvolvimento

- **Configurar Python** (versão 3.8+) e pip.
- **Instalar dependências**:

```
pip install robotframework requests robotframework-requests
```

- **Criar estrutura de pastas**:

```
/api-tests
├── /resources
│   ├── keywords.robot       # Keywords reutilizáveis
│   └── variables.robot      # Variáveis globais (URLs, credenciais)
├── /tests
│   ├── usuarios.robot       # Testes de /usuarios
│   ├── login.robot          # Testes de /login
│   └── ...                  # Demais endpoints
└── run_tests.robot          # Suite principal
```

**Passo 2: Implementação dos Testes**

**Exemplo: Teste de Criação de Usuário (`/usuarios`)**

```
*** Settings ***
Library    RequestsLibrary
Resource    ../resources/keywords.robot
Resource    ../resources/variables.robot

*** Test Cases ***
Criar Usuário Válido
    [Tags]    REGRESSION
    ${headers}=    Create Dictionary    Content-Type=application/json
    ${payload}=    Create Dictionary    nome=Fulano    email=fulano@test.com    password=123456    administrador
    ${response}=    POST    ${BASE_URL}/usuarios    json=${payload}    headers=${headers}
    Status Should Be    201    ${response}
    Should Be Equal As Strings    ${response.json()["message"]}    Cadastro realizado com sucesso
```

**Keywords Reutilizáveis (`keywords.robot`)**

```
*** Keywords ***
Gerar Email Único
    [Arguments]    ${prefixo}
    ${timestamp}=    Get Current Date    result_format=%Y%m%d%H%M%S
    [Return]    ${prefixo}_${timestamp}@test.com
```

**Passo 3: Validações Avançadas**

- **Schemas JSON**: Usar a biblioteca `jsonschema` para validar respostas.

```
Validate Schema    ${response.json()}    ${USER_SCHEMA}
```

- **Testes de Segurança**:
  - Validar token JWT retornado no login.
  - Testar acesso não autorizado a endpoints protegidos.

## CASOS DE TESTE

A tabela abaixo organiza os casos de teste propostos em um formato estruturado para automação com Robot Framework:

| Test Case ID | API Endpoint | Cenário | **Passo a Passo** | **Res. Esperado** | Dados de Teste |
|---|---|---|---|---|---|
| TC_USR_001 | `/usuarios` | Criar usuário válido | Criar usuário com nome "Ana Silva", email "ana@exemplo.com", senha "senhaSegura123", administrador "false" | Status code 201; Mensagem: "Cadastro realizado com sucesso" | nome=Ana Silva; email=ana@exemplo.com; password=senhaSegura123; administrador=false |
| TC_USR_002 | `/usuarios` | Atualizar usuário criado | Atualizar usuário ID [ID_ANA] com nome "Ana Souza", email "ana.souza@exemplo.com" | Status code 200 | nome=Ana Souza; email=ana.souza@exemplo.com |
| TC_USR_003 | `/usuarios` | Listar usuários e validar atualização | Listar todos os usuários | Usuário ID [ID_ANA] tem nome "Ana Souza" | N/A |
| TC_USR_004 | `/usuarios` | Deletar usuário | Deletar usuário ID [ID_ANA] | Status code 200 | N/A |
| TC_USR_005 | `/usuarios` | Tentar criar usuário com e-mail Gmail | Criar usuário com nome "João", email "joao@gmail.com", senha "123", administrador "false" | Status code 400; Mensagem: "Cadastro com e-mail de provedor não permitido" | email=joao@gmail.com; password=123 |
| TC_USR_006 | `/usuarios` | Tentar criar usuário com senha curta | Criar usuário com nome "Maria", email "maria@exemplo.com", senha "123", administrador "false" | Status code 400; Mensagem: "Senha deve ter entre 5 e 10 caracteres" | password=123 |

| Test Case ID | API Endpoint | Cenário | **Passo a Passo** | **Res. Esperado** | Dados de Teste |
|---|---|---|---|---|---|
| TC_LOGIN_001 | `/login` | Login com credenciais corretas | Realizar login com email "ana@exemplo.com" e senha "senhaSegura123" | Status code 200; Token JWT gerado | email=ana@exemplo.com; password=senhaSegura123 |
| TC_LOGIN_002 | `/login` | Validar expiração do token | Extrair expiração do token | Expiração = 600 segundos | N/A |
| TC_LOGIN_003 | `/login` | Tentar login com usuário não cadastrado | Realizar login com email "inexistente@exemplo.com" e senha "senha123" | Status code 401 | email=inexistente@exemplo.com |
| TC_LOGIN_004 | `/login` | Tentar login com senha incorreta | Realizar login com email "ana@exemplo.com" e senha "senhaErrada" | Status code 401 | password=senhaErrada |

| Test Case ID | API Endpoint | Cenário | **Passo a Passo** | **Res. Esperado** | Dados de Teste |
|---|---|---|---|---|---|
| TC_PRD_001 | `/produtos` | Criar produto autenticado | Adicionar token Bearer [TOKEN], Criar produto com nome "Notebook", preço "3500", descrição "Dell i7" | Status code 201 | nome=Notebook; preço=3500; descrição=Dell i7 |
| TC_PRD_002 | `/produtos` | Atualizar produto | Atualizar produto ID [ID_NOTEBOOK] com preço "4000" | Status code 200 | preço=4000 |
| TC_PRD_003 | `/produtos` | Deletar produto | Deletar produto ID [ID_NOTEBOOK] | Status code 200 | N/A |
| TC_PRD_004 | `/produtos` | Tentar criar produto sem token | Remover token Bearer, Criar produto com nome "Mouse", preço "150", descrição "Sem fio" | Status code 401 | nome=Mouse; preço=150 |

| Test Case ID | API Endpoint | Cenário | **Passo a Passo** | **Res. Esperado** | Dados de Teste |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| TC_CART_001 | `/carrinhos` | Criar carrinho autenticado | Adicionar token Bearer [TOKEN], Criar carrinho para usuário ID [ID_ANA] | Status code 201 | N/A |
| TC_CART_002 | `/carrinhos` | Adicionar produto ao carrinho | Adicionar produto ID [ID_PRODUTO] ao carrinho ID [ID_CARRINHO] | Status code 201 | N/A |
| TC_CART_003 | `/carrinhos` | Finalizar compra | Finalizar compra do carrinho ID [ID_CARRINHO] | Status code 200 | N/A |
| TC_CART_004 | `/carrinhos` | Tentar finalizar carrinho vazio | Criar carrinho vazio para usuário ID [ID_ANA], Finalizar compra do carrinho ID [ID_CARRINHO_VAZIO] | Status code 400 | N/A |