


# Plano de Testes [ Challenge Compass ]

## 1. Introdução

Este plano de testes visa garantir a qualidade da API ServeRest () , um marketplace REST que possibilita:

- Cadastro e gerenciamento de usuários.
- Login simples e seguro.
- Gerenciamento de produtos com regras de negócio.

## 2. Objetivo Geral

Estabelecer uma base para garantir a qualidade da API ServeRest, assegurando que as funcionalidades implementadas estejam corretas, seguras e alinhadas com as expectativas do usuário final. O plano de testes tem como propósito identificar falhas, validar regras de negócio e apoiar a entrega de um produto confiável por meio de:

- Análise de rotas com base no Swagger.
- Testes de fluxos principais e alternativos.
- Priorização de cenários e gestão de riscos.
- Automação via Postman com scripts de validação.
- Documentação de issues e melhorias.

Os testes serão entregues em fases, com apresentação dos resultados e discrepâncias entre o planejado e o executado.

### 2.1 Escopo dos Testes

Este plano contempla testes manuais e automatizados dos seguintes recursos da API ServeRest:

- **Usuários** (US 001): Cadastro, edição, consulta e exclusão com validação de regras específicas.
- **Login** (US 002): Autenticação e emissão de token.
- **Produtos** (US 003): CRUD autenticado, com foco em regras como nomes únicos e vínculo com carrinhos.
- **Integração entre funcionalidades**: Testes que envolvem dependências entre usuários, login e produtos.

## 3. Análise dos Testes

Mapear funcionalidades, regras de negócio, riscos e critérios da API ServeRest, com base no Swagger e User Stories (US 001, US 002, US 003), para orientar a criação e priorização de testes.

### 3.1. Funcionalidades e Regras de Negócio

- **US 001 - Usuários:**
  - CRUD (**POST**, **GET**, **PUT**, **DELETE**) com Nome, E-mail, Password, Administrador.
  - Regras: E-mails únicos, válidos, sem Gmail/Hotmail. Senhas de 5-10 caracteres. PUT cria usuário se ID inexistente. Ações em usuários inválidos falham.
  - Risco: Validação fraca de e-mails ou senhas.
- **US 002 - Login:**
  - Autenticação (**POST** /login) com token de 10 minutos.
  - Regras: Usuários inválidos ou senhas incorretas retornam 401. Token exigido para rotas protegidas.

- Risco: Tratamento falho de tokens expirados.

- **US 003 - Produtos:**

- CRUD (**POST**, **GET**, **PUT**, **DELETE**) para usuários autenticados.
- Regras: Nomes únicos. Produtos em carrinhos não podem ser excluídos. PUT cria produto se ID inexistente. Não autenticados retornam 401.
- Risco: Falhas na validação de nomes ou dependência com carrinhos.

### 3.2. Base para Testes [↗](#)

Testes funcionais e de integração baseados no Swagger (rotas, parâmetros, respostas) e User Stories (critérios de aceitação), cobrindo fluxos principais e alternativos.

### 3.3 Cobertura de Testes [↗](#)

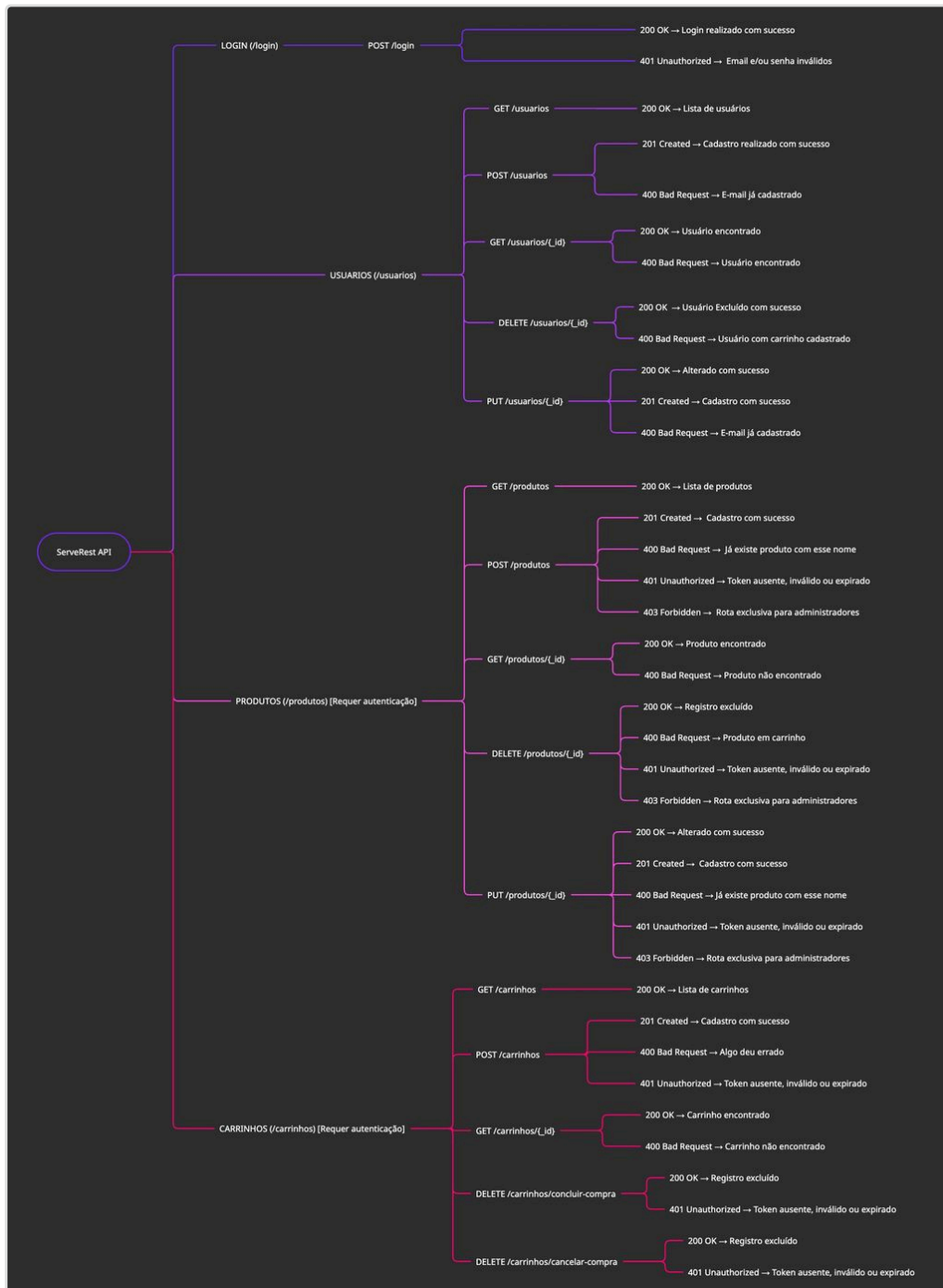
A cobertura de testes da API ServeRest contempla diferentes abordagens para garantir que as funcionalidades estejam corretas, seguras e estáveis. Os testes funcionais validam o comportamento esperado das rotas, incluindo status code, mensagens, headers e estrutura da resposta. Os testes negativos avaliam o comportamento do sistema diante de entradas inválidas, como campos obrigatórios ausentes, formatos incorretos ou autenticação ausente.

Também foram considerados testes de borda, que exploram os limites das regras de negócio, como o tamanho mínimo e máximo de senhas ou a validade de formatos de e-mail. Os testes de segurança asseguram que apenas usuários autorizados possam acessar rotas protegidas e que tokens inválidos ou expirados sejam corretamente rejeitados. A cobertura inclui ainda testes de integração entre funcionalidades dependentes, como a tentativa de exclusão de produtos que estão vinculados a um carrinho. Por fim, os testes de regressão garantem que os principais fluxos funcionais continuem operando corretamente após alterações ou novas implementações.

A cobertura estimada contempla aproximadamente 90% dos fluxos principais e alternativos definidos nas User Stories (US 001, US 002, US 003), com foco especial em regras de negócio, autenticação e permissões.

---

## 4. Mapa mental da Aplicação



## 5. Técnicas Aplicadas

As seguintes técnicas serão utilizadas para testar a API ServeRest, com foco nas User Stories (US 001, US 002, US 003) e no Swagger:

- **Teste Funcional:** Testar endpoints e parâmetros com Postman, validando regras e erros sem acessar o código fonte da API.
- **Teste de Fluxo:** Cobrir os principais fluxos de dados e os cenários de erro, conforme os critérios das User Stories.
- **Teste de Validação de Limite:** Validar os limites das regras de negócio, como o tamanho das senhas e outros parâmetros críticos.

- **Teste de Integração de Sistema:** Verificar a interação entre diferentes componentes da API, como login, produtos e carrinhos.
- **Teste de Segurança:** Validar o processo de autenticação, incluindo tokens e controle de acesso às rotas protegidas.
- **Automação de Testes de API:** Utilizar scripts no Postman para validar respostas e automatizar testes repetitivos.

## 6. Cenários de Teste Planejados

Os cenários de teste são organizados de acordo com a ação a ser testada, a prioridade e a justificativa para sua execução, considerando impacto, criticidade e dependências. A execução desses testes busca garantir que todos os fluxos, tanto principais quanto alternativos, sejam validados adequadamente de acordo com as regras de negócio.

### 6.1. Expansão dos Cenários de Teste

Além dos cenários principais, serão adicionadas variações importantes que cobrem situações comuns e críticas. No módulo de usuários, serão testadas combinações inválidas de senha, e-mail e nome, assim como operações com IDs inexistentes. No login, serão verificadas falhas de autenticação com dados incorretos ou malformados, além de acessos indevidos com tokens inválidos ou expirados.

Para os produtos, serão incluídos testes para nomes duplicados, criação por usuários sem permissão administrativa, e manipulações com produtos inexistentes ou vinculados a carrinhos. Esses cenários complementam os fluxos principais e contribuem para uma validação mais completa da API.

### 6.1. US 001 - Usuários

User Story	Cenário	Ação	Resultado Esperado
US 001 - Usuários	Criar usuário com dados válidos	Testar POST <code>/usuarios</code> com e-mail único, senha de 5-10 caracteres, administrador válido	Status 201, usuário criado
	Criar usuário com e-mail inválido	Testar POST <code>/usuarios</code> com e-mail de Gmail/Hotmail ou malformado	Status 400, erro de validação
	Criar usuário com e-mail duplicado	Testar POST <code>/usuarios</code> com e-mail já cadastrado	Status 400, erro de duplicidade
	Atualizar usuário com ID inexistente	Testar PUT <code>/usuarios/{id}</code> com ID não existente	Status 201, novo usuário criado
	Consultar usuário inexistente	Testar GET <code>/usuarios/{id}</code> com ID inválido	Status 400, erro de usuário não encontrado
	Excluir usuário válido	Testar DELETE <code>/usuarios/{id}</code> com ID existente	Status 200, usuário excluído
	Testar senha no limite	Testar POST <code>/usuarios</code> com senha de exatamente 5 ou 10 caracteres	Status 201, usuário criado
	Atualizar usuário com e-mail duplicado	Testar PUT <code>/usuarios/{id}</code> com e-mail já cadastrado	Status 400, erro de duplicidade
	Excluir usuário inexistente	Testar DELETE <code>/usuarios/{id}</code> com ID inválido	Status 400, erro de usuário não encontrado

## 6.2. US 002 - Login

US 002 - Login	Login com credenciais válidas	Testar POST <code>/login</code> com e-mail e senha corretos	Status 200, token gerado
	Login com senha inválida	Testar POST <code>/login</code> com senha incorreta	Status 401, erro de autenticação
	Login com usuário não cadastrado	Testar POST <code>/login</code> com e-mail inexistente	Status 401, erro de autenticação
	Acessar rota protegida com token expirado	Testar acesso a <code>/produtos</code> com token após 10 minutos	Status 401, erro de token inválido

## 6.3. US 003 - Produtos [↗](#)

US 003 - Produtos	Criar produto com dados válidos	Testar POST <code>/produtos</code> com token válido e nome único	Status 201, produto criado
	Criar produto com nome duplicado	Testar POST <code>/produtos</code> com nome já existente	Status 400, erro de duplicidade
	Excluir produto vinculado a carrinho	Testar DELETE <code>/produtos/{id}</code> com produto em carrinho	Status 400, erro de dependência
	Atualizar produto com ID inexistente	Testar PUT <code>/produtos/{id}</code> com ID não existente	Status 201, novo produto criado
	Acessar produtos sem autenticação	Testar GET <code>/produtos</code> sem token	Status 401, erro de autenticação
	Fluxo integrado usuário-produto-carrinho	Testar sequência de POST <code>/usuarios</code> , POST <code>/login</code> , POST <code>/produtos</code> , adicionar ao carrinho	Status 200/201 em cada etapa, integração funcional
	Atualizar produto com nome duplicado	Testar PUT <code>/produtos/{id}</code> com nome já existente	Status 400, erro de duplicidade
	Criar produto com preço/quantidade inválidos	Testar POST <code>/produtos</code> com preço ou quantidade $\leq 0$	Status 400, erro de validação
	Criar produto como não-admin	Testar POST <code>/produtos</code> com token de usuário não-admin	Status 403, erro de permissão

## 7. Matriz de Riscos [↗](#)

Neste tópico, são apresentados os riscos identificados para a API ServeRest, com foco nas User Stories (US 001, US 002, US 003). A matriz classifica esses riscos em termos de probabilidade e impacto, além de propor ações específicas para mitigar possíveis falhas, orientando a priorização dos testes.

Risco	User Story	Probabilidade	Impacto	Mitigação
Validação fraca de e-mails	US 001	Média	Alto	Testar e-mails inválidos e duplicados em POST e PUT (cenários 2, 3, 18).
Validação fraga de senhas	US 001	Média	Médio	Testar senhas fora do padrão e no limite (cenários 2, 7).
Ações em usuários inválidos	US 001	Baixa	Baixo	Testar GET e DELETE com ID inválido (cenários 5, 21).
Tratamento falho de tokens expirados	US 002	Alta	Alto	Testar acesso com token expirado (cenário 11).
Falhas na validação de credenciais	US 002	Média	Alto	Testar login com senha inválida e usuário inexistente (cenários 9, 10).
Falhas na validação de nomes de produtos	US 003	Média	Alto	Testar nomes duplicados em POST e PUT (cenários 13, 19).
Erros na dependência com carrinhos	US 003	Alta	Alto	Testar exclusão de produto em carrinho e fluxo integrado (cenários 14, 17).
Acesso não autorizado a rotas protegidas	US 002, US 003	Média	Alto	Testar acesso sem token ou com token inválido (cenários 11, 16).
Falhas em fluxos integrados	US 001, US 002, US 003	Baixa	Médio	Testar sequência completa de usuário, login, produto e carrinho (cenário 17).
Validação fraca de preço/quantidade	US 003	Média	Médio	Testar preço/quantidade inválidos (cenário 20).
Acesso não autorizado por usuários não-admin	US 003	Média	Alto	Testar criação de produto como não-admin (cenário 22).

## 8. Testes Candidatos a Automação

Os cenários abaixo foram selecionados para automação via Postman, utilizando scripts na aba Tests para validar respostas e status codes, devido à sua criticidade, repetitividade e impacto nas regras de negócio da API ServeRest. A automação prioriza fluxos principais, validações críticas e testes repetitivos, cobrindo as User Stories (US 001, US 002, US 003).

**8.1. Robot Framework** será utilizado para automação de testes de API, substituindo gradualmente os testes automatizados no Postman.

A biblioteca utilizada é a `RequestsLibrary`, permitindo chamadas HTTP diretas e validação de status codes, corpo da resposta, tokens e headers.

**Estrutura dos testes automatizados:**

```

1 /tests
2 |— login.robot
3 |— usuarios.robot
4 |— produtos.robot
5 |— resources/
6   |— variables.robot

```

Cada arquivo `.robot` contém testes organizados por funcionalidade (US 001 a US 003), permitindo reuso e execução modular via terminal ou CI.

### US 001 - Usuários:

- **Criar usuário com dados válidos:** Testar POST /usuarios com e-mail único, senha de 5-10 caracteres e administrador válido. Justifica-se por ser um fluxo principal, de alta prioridade e repetitivo em cadastros.
- **Criar usuário com e-mail inválido:** Testar POST /usuarios com e-mail de Gmail/Hotmail ou malformatado. Necessário para validar uma regra crítica de e-mail, com alta prioridade.
- **Criar usuário com e-mail duplicado:** Testar POST /usuarios com e-mail já cadastrado. Testa um erro comum, repetitivo em validações de unicidade.
- **Atualizar usuário com e-mail duplicado:** Testar PUT /usuarios/{id} com e-mail já cadastrado. Valida a unicidade crítica no PUT, sendo de alta prioridade e repetitivo.

### US 002 - Login:


- **Login com credenciais válidas:** Testar POST /login com e-mail e senha corretos. Essencial como fluxo principal, pré-requisito para rotas protegidas e de alta prioridade.
- **Login com senha inválida:** Testar POST /login com senha incorreta. Valida a segurança da autenticação, com alta prioridade e repetitivo.
- **Acessar rota protegida com token expirado:** Testar acesso a /produtos com token após 10 minutos. Valida a regra de expiração do token, de alta prioridade.

### US 003 - Produtos:

- **Criar produto com dados válidos:** Testar POST /produtos com token válido e nome único. Fluxo principal, de alta prioridade e repetitivo para cadastros de produtos.
- **Criar produto com nome duplicado:** Testar POST /produtos com nome já existente. Testa um erro comum, repetitivo em validações de unicidade.
- **Excluir produto vinculado a carrinho:** Testar DELETE /produtos/{id} com produto em carrinho. Valida uma dependência crítica com carrinhos, de alta prioridade.
- **Acessar produtos sem autenticação:** Testar GET /produtos sem token. Valida a segurança de rotas protegidas, com alta prioridade e repetitivo.
- **Atualizar produto com nome duplicado:** Testar PUT /produtos/{id} com nome já existente. Valida a unicidade crítica no PUT, de alta prioridade e repetitivo.
- **Criar produto como não-admin:** Testar POST /produtos com token de usuário não-admin. Valida uma restrição crítica de permissão, de alta prioridade e repetitivo.

## 9. Ferramentas de Teste

As seguintes ferramentas serão utilizadas para a execução dos testes, garantindo a validação eficiente das funcionalidades da API ServeRest:

- **Postman:** Ferramenta principal para testes manuais e automatizados, utilizada para testar endpoints, rotas, parâmetros e métodos (conforme técnica de Teste Funcional). Scripts na aba Tests serão criados para validar respostas e status codes nos 13 cenários automatizados.
- **Swagger:** Documentação de referência para mapear rotas, parâmetros e respostas esperadas, usada na criação dos cenários de teste.
- **Ambiente de Teste:** API ServeRest (  ), acessada diretamente para execução dos testes.

## 10. Conclusão [↗](#)

Este plano de testes garante a validação completa das regras de negócio das User Stories (US 001 - Usuários, US 002 - Login, US 003 - Produtos) da API ServeRest, cobrindo fluxos principais, alternativos e casos de erro. Com 22 cenários de teste planejados, priorizados e mapeados contra riscos, o plano assegura a qualidade das funcionalidades de cadastro, autenticação e gerenciamento de produtos. A automação de 13 cenários via Postman otimizará a execução repetitiva, enquanto testes manuais complementarão a cobertura.

Para garantir escalabilidade e organização, foram adotadas as seguintes práticas:

- **Separação entre testes manuais e automatizados**, com documentação paralela.
  - Uso de arquivos de variáveis ( `variables.robot` ) para centralizar dados reutilizáveis, como tokens, URLs e payloads padrão.
  - Documentação de passos de execução no README e planejamento de **integração com GitHub Actions** para execução automática dos testes automatizados a cada push/PR.
  - Todos os testes automatizados são versionados junto ao código-fonte, permitindo rastreabilidade de falhas e regressões.
-