Desafio 1 - Eduardo Luiz da Silva

# Plano de Teste - US 001: API de Usuários

### Apresentação

Este plano de testes tem como objetivo validar a história de usuário US 001, que trata da implementação das operações de CRUD (criar, ler, atualizar e excluir) para o cadastro de vendedores (usuários) na plataforma Marketplace do ServeRest. Os testes visam verificar se a API cumpre todos os requisitos funcionais e não funcionais definidos, especialmente no que diz respeito à criação, atualização, listagem e remoção de usuários. A validação também incluirá regras de negócio específicas e critérios como formato de e-mail, segurança da senha e demais validações necessárias. De acordo com os critérios definidos pelo Definition of Ready (Dor), o ambiente de teste e o banco de dados já se encontram preparados.

### Objetivo

Assegurar que a API de usuários esteja funcionando corretamente de acordo com os critérios de aceitação, contemplando:

- A implementação completa das operações de CRUD: criação, edição, listagem e remoção de usuários.
- A aplicação correta das validações de e-mail e senha.
- A prevenção de comportamentos inválidos, como o registro de usuários com e-mails já existentes ou de domínios proibidos (como Gmail ou Hotmail).
- A realização de testes que abranjam tanto cenários de sucesso quanto de falha, com registros e evidências das execuções.
- A automação dos testes para casos repetitivos, utilizando como base a documentação da API (Swagger) e incluindo variações de cenários.

### Escopo

**Incluso:**

- Testes da API REST no endpoint /usuarios.
- Validação dos campos: **Nome**, **E-mail**, **Senha** e **Administrador**.
- Testes dos verbos HTTP: **POST** (criar), **PUT** (atualizar), **GET** (listar), **DELETE** (deletar).
- Verificação de regras de negócio, como:
  - E-mails únicos e com formato válido.

    - Senha com tamanho mínimo.
- Cobertura de cenários negativos:
    - Ações com usuários inexistentes.
    - Campos obrigatórios ausentes ou inválidos.
- Evidências de teste: logs, capturas de tela e relatórios de execução.

**Excluído:**

- Testes de performance ou carga.
- Validação de interface gráfica (UI).
- Funcionalidades fora do módulo de usuários (ex: Produtos e Carinho)

### Análise

A API de usuários é responsável pelo gerenciamento do cadastro dos vendedores no Marketplace, apresentando as seguintes funcionalidades principais:

- **Endpoints principais**:
    - **POST /usuarios**: cria um novo usuário.
    - **PUT /usuarios/{id}**: atualiza os dados de um usuário existente.
    - **GET /usuarios**: lista todos os usuários.
    - **GET /usuarios/{id}**: recupera um usuário pelo seu ID.
    - **DELETE /usuarios/{id}**: remove um usuário do sistema.
- **Campos obrigatórios**:
    - **NOME**: String.
    - **E-MAIL**: String, deve ter formato válido e não pode ser de domínios Gmail ou Hotmail.
    - **PASSWORD**: String, com tamanho entre 5 e 10 caracteres.
    - **ADMINISTRADOR**: valor booleano indicando se o usuário possui privilégios administrativos.
- **Riscos identificados**:
    - A existência de e-mails duplicados pode comprometer a consistência dos dados no banco.
    - Falhas na validação do formato do e-mail ou da senha podem permitir registros incorretos ou inválidos.
    - O sistema pode apresentar comportamento inesperado ao tentar atualizar um usuário que não existe (PUT com ID inexistente).
    - Possibilidade de erros na automação dos testes devido a respostas inconsistentes ou imprevisíveis da API.

### Técnicas Aplicadas

- **Teste de Contato**: Garante que a API esteja em conformidade com a especificação,ex: Swagger.
- **Teste Baseado em Risco**: Priorização cenários críticos que podem causar impacto significativo, ex: cadastro com e-mails duplicados ou operações com usuários inexistentes.
- **Teste Exploratório**: Execução de testes sem roteiro pré-definido para descobrir comportamentos inesperados, como uso de e-mails com caracteres especiais ou entradas fora do padrão previsto.
- **Teste de API Automatizado**: Utilização de ferramentas como Postman ou Newman para automatizar a execução de testes repetitivos nos endpoints da API
- **Teste de Validação de Dados**: Verificação de regras de formato e restrições aplicadas aos campos obrigatórios

### Mapa Mental da Aplicação

O mapa mental da API de usuários inclui:

**Nó principal:** API de Usuários

**Endpoints:**

- POST /usuarios
- PUT /usuarios/{id}
- GET /usuarios
- GET /usuarios/{id}
- DELETE /usuarios/{id}

**Campos de Entrada:**

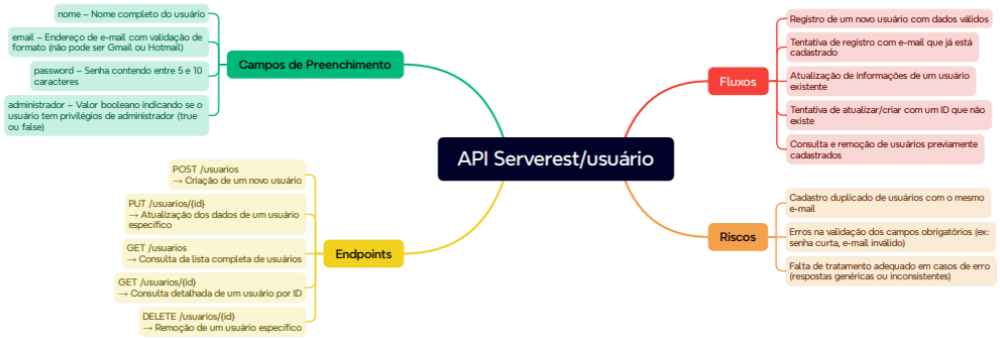
- **NOME**
- **EMAIL** (formato válido, exceto Gmail/Hotmail)
- **PASSWORD** (mín. 5 – máx. 10 caracteres)
- **ADMINISTRADOR** (true/false)

**Fluxos:**

- Criação de usuário com dados válidos.
- Tentativa de criação com e-mail já existente.
- Atualização de usuário existente.
- Busca de usuário por ID válido.
- Remoção de usuário previamente cadastrado.

**Riscos:**

- Cadastro duplicado de e-mail.
- Falhas na validação dos campos obrigatórios.
- Falta de tratamento adequado em erros de negócio.




## Cenários de Teste Planejados

| ID | Descrição do Cenário | Pré-condições | Passos | Resultado Esperado | Dados de Teste |
|---|---|---|---|---|---|
| TC01 | Cadastro de usuário com dados válidos | Banco sem registros prévios | Enviar requisição POST /usuarios com dados corretos | Retorno 201 com ID do novo usuário | {nome: "Carlos Freitas", email: "carlos@exemplo.com", password: "ABC123", administrador: true} |

| | | | | | |
|---|---|---|---|---|---|
| TC02 | Cadastro com e-mail já existente | Usuário com e-mail já cadastrado | Enviar POST /usuarios com o mesmo e-mail | Retorno 400 com mensagem: "Este email já está sendo usado" | {nome: "Maria Alice", email: "roberto@exemplo.com", password: "ABC123", administrador: true} |
| TC03 | Cadastro com e-mail de provedor bloqueado (Gmail) | Banco de dados limpo | Enviar POST /usuarios com e-mail do Gmail | Retorno 400 com mensagem de email inválido | {nome: "Maria Betania", email: "maria.betania@gmail.com", password: "123ABC", administrador: true} |
| TC04 | Cadastro com senha inválida (menos de 5 caracteres) | Banco de dados limpo | Enviar POST /usuarios com senha curta | Retorno 400 com mensagem de senha inválida | {nome: "Gustavo Martins", email: "gustavo.m@exemplo.com", password: "EGO", administrador: true} |
| TC05 | Atualização de | ID de usuário | Enviar PUT /usuarios/{ | Retorno 200 com dados | {nome: "Carlos2", email: |

| | | | | | |
|---|---|---|---|---|---|
| | usuário existente | válido disponível | id} com novos dados | atualizados | "carlos.freitas@exemplo.com", password: "123456", administrador: false} |
| TC06 | Cadastro com e-mail e nome composto pro sinais | Banco de dados limpo | Enviar POST /usuarios com e-mail e nome composto pro sinais | Retorno 400 com mensagem de email inválido | {nome: "$%¨@#%", email: "¨&@##%@exemplo.com", password: "###$$$", administrador: false} |
| TC07 | Listagem de usuários | Ao menos um usuário cadastrado | Enviar GET /usuarios | Retorno 200 com lista de usuários | N/A |
| TC08 | Cadastro com e-mail mal formatado | Banco de dados limpo | Enviar POST /usuarios com e-mail inválido | Retorno 400 com erro de validação | {nome: "Teste", email: "inexistente", password: "ABC123", administrador: true} |
| TC09 | Cadastro com senha em branco | Banco de dados limpo | Enviar POST /usuarios com | Retorno 400 com erro de validação | |

| | | | | | |
|---|---|---|---|---|---|
| | | | senha em branco | | |

**Priorização da Execução dos Cenários de Teste**

| ID | Prioridade | Justificativa |
|---|---|---|
| TC01 | Alta | Cadastro de usuários é a função principal e base para os demais testes. |
| TC02 | Alta | Prevenção de e-mails duplicados garante integridade dos dados. |
| TC03 | Alta | Validação de domínio é uma regra de negócio crítica. |
| TC05 | Alta | Atualização é parte essencial do fluxo CRUD. |
| TC04 | Média | Regras de senha são importantes, mas de menor impacto que e-mail. |
| TC06 | Média | Deletar ID inválido é um cenário de erro menos frequente. |
| TC07 | Baixa | Listar usuários é funcionalidade básica e pouco sensível a erros críticos. |
| TC09 | Média | Verificação de formato de e-mail é |

| | | |
|---|---|---|
| | | complementar às regras de domínio. |

**Matriz de Risco**

| Risco | Probabilidade | Impacto | Nível de Risco | Mitigação |
|---|---|---|---|---|
| Duplicidade de e-mails | Média | Alto | Alto | Execução dos testes TC02 e TC05. |
| Falha na validação de e-mail | Baixa | Médio | Médio | Testes TC03 e TC08 para cobrir formatos e domínios. |
| Falha na validação de senha | Baixa | Médio | Médio | Execução do TC04. |
| Comportamento inconsistente em PUT | Média | Médio | Médio | Cobertura pelo teste TC05 |
| Falha ao excluir usuário | Baixa | Baixo | Baixo | Execução do teste TC06 |

**Cobertura de Testes**

- **Requisitos Funcionais:** Todos os critérios definidos na história de usuário US001 estão mapeados para cenários de teste. A rastreabilidade será garantida por uma matriz.
- **Cobertura de Endpoints**: 100% dos endpoints da API de usuários serão testados (POST, GET, PUT, DELETE).
- **Cobertura de Cenários**: Os testes abrangem tanto os fluxos descritos na documentação (Swagger) quanto variações como:

- E-mails com caracteres especiais.
- Tamanhos-limite de senha (5 e 10 caracteres).
- Restrições de domínio (ex: Gmail, Hotmail).

- **Métrica de Qualidade:** Objetivo de 85% de cobertura de testes funcionais, com evidências registradas por ferramentas (como Postman, Newman ou relatórios automatizados).

**Testes Candidatos a Automação**

| ID | Descrição | Justificativa para Automação | Ferramenta Sugerida |
|---|---|---|---|
| TC01 | Criar usuário com dados válidos | Teste de base e altamente repetitivo | Postman |
| TC02 | Criar usuário com e-mail duplicado | Validação crítica com dados estáticos | Postman |
| TC03 | Criar usuário com e-mail Gmail | Regra de negócio que pode ser revalidada | Postman |
| TC05 | Atualizar usuário existente | Teste recorrente do fluxo CRUD | Postman |
| TC07 | Listar todos os usuários | Simples, ideal para smoke/regressão | Postman |

---

## Plano de Teste - US 002: API de Login

### Apresentação

Este plano de testes tem como objetivo validar a história de usuário US 002, responsável pela implementação da autenticação de vendedores no Marketplace do ServeRest através da API de login. A funcionalidade garante que usuários previamente cadastrados possam acessar o

sistema e gerenciar produtos, recebendo um token Bearer com validade de 10 minutos. Para este cenário, já estão disponíveis o ambiente de testes, o banco de dados e a API de cadastro de usuários (US 001), como foi definido no DoR (Definition of Ready).

### Objetivo

Confirmar que a API de login atende corretamente os critérios de aceitação, verificando:

- Autenticação bem-sucedida de usuários válidos, com a geração de um token Bearer.
- Rejeição de tentativas de login de usuários inexistentes ou senhas incorretas, retornando status 401.
- Expiração do token após 10 minutos.
- Cobertura de cenários positivos e negativos, com registros de evidências.
- Automação de casos repetitivos para otimizar os testes.

### Escopo

**Incluso:**

- Testes da API REST no endpoint /login.
- Validação dos campos obrigatórios: email e password.
- Verificação da geração e do uso do token Bearer em rotas protegidas.
- Cobertura de cenários negativos, como: credenciais inválidas, campos vazios e tokens incorretos.
- Registro de evidências por meio de logs, capturas de tela e relatórios.

**Excluído:**

- Testes de carga e performance.
- Validação de interface gráfica (UI).
- Testes de segurança avançados, como brute force.

### Análise

A API de login é responsável por autenticar vendedores no Marketplace e liberar o uso das demais funcionalidades protegidas.

- **Endpoint principal**: POST /login
- **Campos obrigatórios:**
  - **Email**: string, deve ter formato válido e já existir no banco de dados.
  - **Passaword**: string, deve corresponder à senha cadastrada no usuário.
- **Saída esperada**:

    - Em caso de sucesso:
        - Código 200.
        - Mensagem de sucesso e token JWT válido.
    - Em caso de falha:
        - Código 401 ou 400 dependendo da natureza do erro.
        - Mensagem explicativa ("Email e/ou senha inválidos", "Campos obrigatórios").
- **Riscos identificados**:
    - Autenticação com credenciais inválidas, comprometendo a segurança.
    - Token gerado sem expiração correta, permitindo acessos indevidos.
    - Erros inconsistentes para cenários inválidos, dificultando o uso da API.
    - Possibilidade de falhas na automação devido a respostas dinâmicas (tokens).

### Técnicas Aplicadas

- **Teste de Contato:** garantir que a API esteja em conformidade com a especificação (Swagger).
- **Teste Baseado em Risco:** foco em cenários críticos, como autenticação inválida e expiração de token.
- **Teste Exploratório:** execução de cenários alternativos, como login com e-mail malformado ou campos vazios.
- **Teste de API Automatizado:** uso de Postman ou RestAssured para execução de testes repetitivos.
- **Teste de Validação de Token**: Verificação da geração e expiração do token Bearer.

### Mapa Mental da Aplicação

O mapa mental da API de login inclui:

**Nó principal:** API de Login

**Endpoint:**

- POST /login

**Campos de Entrada:**

- **E-MAIL**
- **PASSWORD**

**Saídas:**

- **Token Bearer** (sucesso)

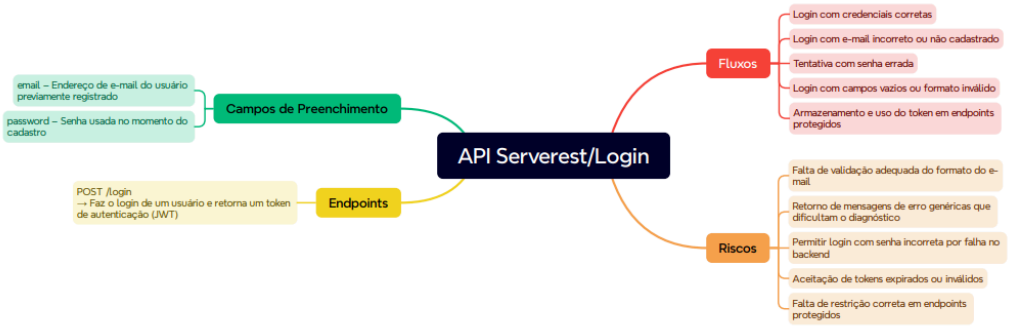
- **Erro 401** (falha)

**Fluxos:**

- Autenticação com credenciais válidas.
- Tentativa de login com usuário não cadastrado.
- Tentativa de login com senha inválida.
- Validação da expiração do token.

**Riscos:**

- Acesso indevido por falha de validação.
- Token com validade incorreta.
- Respostas inconsistentes em cenários de erro.




**Cenários de Teste Planejados**

| ID | Descrição do Cenário | Pré-condições | Passos | Resultado Esperado | Dados de Teste |
|---|---|---|---|---|---|
| TC01 | Login com credenciais válidas | Usuário cadastrado | Enviar POST /login com e-mail e senha corretos | Retorno 200 com token Bearer válido | {email: "carlos@exemplo.com", password: "ABC123"} |
| TC02 | Login com senha incorreta | Usuário cadastrado | Enviar POST /login com | Retorno 401 com mensagem "Email | {email: "carlos@exemplo.com", |

| | | | | | |
|---|---|---|---|---|---|
| | | | senha incorreta | e/ou senha inválidos" | password: "123456"} |
| TC03 | Login com usuário inexistente | Nenhum usuário com e-mail | Enviar POST /login com e-mail não cadastrado | Retorno 401 com mensagem "Email e/ou senha inválidos" | {email: "carlos.freitas@exemplo.com", password: "123456"} |
| TC04 | Login com e-mail de provedor bloqueado | Banco de dados limpo | Enviar POST /login com e-mail malformado | Retorno 400 com mensagem de formato inválido | {email: "maria.betania@gmail.com", password: "123ABC"} |
| TC05 | Login com campos vazios | Banco de dados limpo | Enviar POST /login sem e-mail e/ou senha | Retorno 400 com mensagem de campos obrigatórios | {email: "", password: ""} |
| TC06 | Uso de token inválido | Usuário autenticado | Acessar rota autenticada com token inválido | Retorno 401 com mensagem "Token inválido" | Token: "abc123" |
| TC07 | Uso de token expirado | Usuário autenticado | Acessar rota autenticada após tempo de | Retorno 401 com mensagem "Token expirado" | Token expirado |

| | | | | | |
|---|---|---|---|---|---|
| | | | expiração do token | | |

**Priorização da Execução dos Cenários de Teste**

| ID | Prioridade | Justificativa |
|---|---|---|
| TC01 | Alta | Fluxo principal de autenticação. |
| TC02 | Alta | Garante bloqueio de acessos indevidos. |
| TC03 | Alta | Previne login de usuários inexistentes. |
| TC06 | Alta | Validação crítica de segurança. |
| TC07 | Alta | Confirma expiração do token, requisito essencial. |
| TC04 | Média | Importante para validar formato, mas impacto menor. |
| TC05 | Média | Cenário alternativo comum, mas não crítico. |

**Matriz de Risco**

| Risco | Probabilidade | Impacto | Nível de Risco | Mitigação |
|---|---|---|---|---|
| Autenticação com senha incorreta | Alta | Alto | Alto | Execução do TC02 |
| Login com usuário | Média | Médio | Médio | Execução do TC03 |

| | | | | |
|---|---|---|---|---|
| inexistente | | | | |
| Uso de token expirado | Média | Alto | Alto | Execução do TC06 |
| Uso de token inválido | Média | Alto | Médio | Testes de autenticação válida (TC01). |
| Campos inválidos no login | Alta | Médio | Médio | Execução do TC04 e TC05 |

### Cobertura de Testes

- **Requisitos Funcionais:** Todos os critérios de aceitação da US 002 estão mapeados em cenários de teste.
- **Cobertura de Endpoint**: Endpoints: 100% do endpoint /login será testado.
- **Cobertura de Cenários**: Inclusão de casos de sucesso (login válido) e falhas (credenciais inválidas, tokens incorretos, campos vazios).
- **Métrica de Qualidade:** Objetivo de 90% de cobertura funcional, com evidências registradas via Postman ou RestAssured.

### Testes Candidatos a Automação

| ID | Descrição | Justificativa para Automação | Ferramenta Sugerida |
|---|---|---|---|
| TC01 | Login com credenciais válidas | Teste de base e recorrente | Postman ou RestAssured |
| TC02 | Login com senha incorreta | Cenário crítico de segurança | Postman ou RestAssured |
| TC03 | Login com usuário inexistente | Cenário de rejeição repetitivo | Postman |

| | | | |
|---|---|---|---|
| TC06 | Uso de token inválido | Teste repetitivo para garantir expiração correta | Postman |
| TC07 | Uso de token expirado | Garantia de expiração do token | Postman |

---

## Plano de Teste - US 003: API de Produtos

### Apresentação

Este plano de teste tem como objetivo validar a história de usuário US 003, responsável pela implementação do CRUD de produtos no Marketplace do ServeRest. A funcionalidade permite que vendedores autenticados realizem o gerenciamento completo de produtos (criação, atualização, listagem e exclusão) respeitando regras como a obrigatoriedade de autenticação e a exigência de nomes únicos para cada produto.

O ambiente de testes já está devidamente configurado e pronto para execução, incluindo o banco de dados, a API de cadastro de usuários (US 001) e a API de autenticação (US 002), em conformidade com os critérios estabelecidos no Definition of Ready (DoR).

### Objetivo

Assegurar que a API de produtos atenda corretamente os critérios definidos, garantindo:

- Suporte completo ao CRUD de produtos.
- Restrição de acesso para usuários não autenticados.
- Validação de unicidade no nome do produto.
- Bloqueio de exclusão de produtos vinculados a carrinhos.
- Comportamento da API ao receber uma requisição PUT com ID inexistente.
- Cobertura de testes positivos e negativos, com evidências e relatórios.
- Aplicação de automação para fluxos recorrentes.

### Escopo

**Incluso:**

- Testes da API REST nos endpoints de produtos(/produtos).

- Validação de regras de negócio (ex.: não permitir duplicidade de nomes).
- Verificação de autorização com token válido e inválido.
- Testes de cenários negativos (ex.: produto inexistente, dados incompletos).
- Evidências de execução (logs, relatórios e capturas de tela).

**Excluído:**

- Testes de performance (carga, stress).
- Testes de interface gráfica.
- Testes de integração com meios de pagamento.

### Análise

A API de produtos gerencia o cadastro e manipulação de produtos no Marketplace, utilizando das seguintes caracteristicas:

- **Endpoints principais**:
  - **POST /produtos**: Cadastrar produto
  - **GET /produtos**: Listar produtos
  - **GET /produtos/:id**: Consultar produto por ID
  - **PUT /produtos/:id**: Atualizar produto
  - **DELETE /produtos/:id**: Excluir produto
- Campos obrigatórios (POST / PUT)
  - **nome:** (string) – obrigatório, único.
  - **preço:** (número > 0).
  - **descrição:** (string).
  - **quantidade:** (inteiro ≥ 0).
- **Riscos identificados**:
  - Produto duplicado ser aceito (falha de regra de negócio).
  - Falha de autenticação permitindo manipulação sem token.
  - Respostas inconsistentes em casos de erro (ex.: mensagens diferentes para o mesmo tipo de problema).
  - Token expirado não ser validado corretamente.

### Técnicas Aplicadas

- **Teste de Caixa Preta**: validar entradas e saídas conforme o Swagger.
- **Teste Baseado em Risco**: priorizar cenários críticos como duplicidade de produto e autorização.

- **Teste Exploratório**: verificar campos vazios, tipos de dados incorretos.
- **Teste de API Automatizado**: Postman/RestAssured para cenários repetitivos.

### Mapa Mental da Aplicação

O mapa mental da API de produtos inclui:

**Nó principal:** API de Produtos

**Endpoints:**

- GET /produtos
- POST /produtos
- GET /produtos/{_id}
- PUT /produtos/{_id}
- DELETE /produtos/{_id}

**Campos de Entrada:**

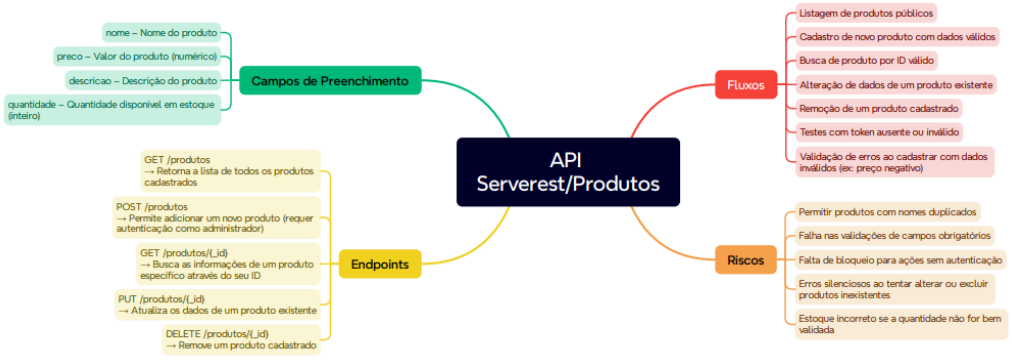
- **NOME**
- **PRECO** (valor numérico)
- **DESCRICAO**
- **QUANTIDADE** (inteiro positivo)

**Fluxos:**

- Listagem geral de produtos.
- Criação de novo produto com token de admin.
- Consulta de produto por ID.
- Atualização de produto existente.
- Exclusão de produto por ID válido.

**Riscos:**

- Criação de produtos com nomes duplicados.
- Falha nas validações de campos (ex: preço negativo).
- Permissões incorretas (ações sem token admin).
- Comportamento inesperado em operações com ID inexistente.




## Cenários de Teste Planejados

| ID | Descrição do Cenário | Pré-condições | Passos | Resultado Esperado | Dados de Teste |
|---|---|---|---|---|---|
| TC01 | Criar produto válido | Usuário autenticado | POST /produtos com todos os campos corretos | 201 Created, produto registrado com ID | {nome:"Notebook Dell", preco:4500, descricao: "i7 16GB RAM", quantidade:10} |
| TC02 | Criar produto duplicado | Produto já existente | POST /produtos com mesmo nome | 400 Conflict, mensagem "Já existe produto com esse nome" | Mesmo JSON de TC01 |
| TC03 | Criar produto sem token | Nenhum | POST /produtos sem | 401 Unauthorized, mensage | Dados válidos |

| | | | | | |
|---|---|---|---|---|---|
| | | | Authorization | m "Token ausente, inválido ou expirado" | |
| TC04 | Listar produtos | Produtos cadastrados | GET /produtos | 200 OK, lista JSON de produtos | N/A |
| TC05 | Consultar produto existente | Produto cadastrado | GET /produtos/:id válido | 200 OK, retorna dados do produto | id=abc123 |
| TC06 | Consultar produto inexistente | Produto não cadastrado | GET /produtos/:id inválido | 404 Not Found, mensagem "Produto não encontrado" | id=xyz999 |
| TC07 | Atualizar produto válido | Produto cadastrado | PUT /produtos/:id com novos dados | 200 OK, produto atualizado | {nome:"Notebook Dell", preco:4800, descricao: "i7 16GB RAM", quantidade:11} |
| TC08 | Atualizar produto inexistente | Nenhum | PUT /produtos/:id inexistente | 404 Not Found, mensagem "Produto | id=asxyz999ewqasdasdasdasdasdas |

| | | | | | |
|---|---|---|---|---|---|
| | | | | não encontrado" | |
| TC09 | Excluir produto válido | Produto cadastrado | DELETE /produtos/:id existente | 200 OK, mensagem "Registro excluído com sucesso" | id=abc123 |
| TC10 | Excluir produto inexistente | Nenhum | DELETE /produtos/:id inexistente | 404 Not Found | id=xyz789 |

**Priorização da Execução dos Cenários de Teste**

| ID | Prioridade | Justificativa |
|---|---|---|
| TC01 | Alta | Cadastro válido é a base da funcionalidade. |
| TC02 | Alta | Evitar nomes duplicados é essencial para integridade. |
| TC03 | Alta | Testar acesso sem token garante segurança. |
| TC05 | Alta | Consulta individual é requisito essencial. |
| TC09 | Alta | Exclusão é um dos principais fluxos do CRUD. |

| | | |
|---|---|---|
| TC04 | Média | Listagem é importante, mas menos crítica que manipulação. |
| TC06 | Baixa | Consulta inexistente cobre erro comum. |
| TC07 | Baixa | Atualização válida é importante, mas complementar. |
| TC08 | Baixa | Atualização inexistente é cenário alternativo. |
| TC10 | Baixa | Exclusão inexistente é cenário alternativo. |

**Matriz de Risco**

| Risco | Probabilidade | Impacto | Nível de Risco | Mitigação |
|---|---|---|---|---|
| Produto duplicado | Alto | Média | Alto | Testar TC02. |
| Acesso sem token | Média | Alto | Alto | Testar TC03. |
| Token expirado | Média | Alto | Alto | Reaproveitar cenários da US002. |
| Produto inexistente | Média | Baixa | Médio | Testar TC06, TC08, TC10. |
| Respostas inconsistentes | Média | Médio | Médio | Revisar mensagens de erro. |

### Cobertura de Testes

- **Cobertura de Requisitos**: Todos os critérios de aceitação serão mapeados aos cenários de teste.
- **Cobertura de Endpoints:** 100% dos métodos (POST, GET, PUT, DELETE).
- **Cobertura de Cenários:** positivos, negativos, alternativos (campos inválidos, ausência de token).
- **Métrica**: 90% de cobertura funcional mínima garantida, com evidências documentadas.

### Testes Candidatos a Automação

| ID | Descrição | Justificativa para Automação | Ferramenta Sugerida |
|---|---|---|---|
| TC01 | Criar produto válido | Repetitivo e base para outros testes | Postman |
| TC02 | Criar produto duplicado | Cenário crítico para regra de negócio | Postman |
| TC03 | Criar produto sem token | Cenário crítico de segurança | Postman |
| TC05 | Consultar produto existente | Cenário base e repetitivo | Postman |
| TC09 | Excluir produto válido | Fluxo essencial e recorrente | Postman |

---

---

## Relatório de Bug 1 - US 001: API de Usuários

**Título**: Cadastro de usuário com e-mail de provedor bloqueado

**ID**: BUG-US001-001

**Data**: 03/09/2025

**Prioridade**: Alta

**Severidade**: Média

**Status**: Aberto

**Ambiente**:

- **Servidor:** ServeRest (teste)
- **Endpoint:** POST /usuarios
- **Ferramenta:** Postman

**Descrição:** Durante os testes realizados foi identificado a possiblidade de realizar o cadastro de usuário com o uso de provedores de e-mail que deveriam estar bloqueados. Ferindo assim a regra de negócio “Não deverá ser possível cadastrar usuários com e-mails de provedor gmail e hotmail”.

**Cenário de Teste**: TC03 - Cadastro com e-mail de provedor bloqueado (Gmail)

**Passos para Reproduzir**:
Enviar a requisição **POST /**usuarios com payload:
`{nome: "Maria Betania", email: "maria.betania@gmail.com", password: "123ABC", administrador: "true"}`.

**Comportamento Observado**:
Status 201, usuário criado:

```
{
    "message": "Cadastro realizado com sucesso",
    "_id": "paZZu8WCOTi33osS"
}

```

**Comportamento Esperado**:
Status 400, mensagem:

```
{
    "message": "Provedor de e-mail inválido"
}

```

**Evidências**:

- **Log**:

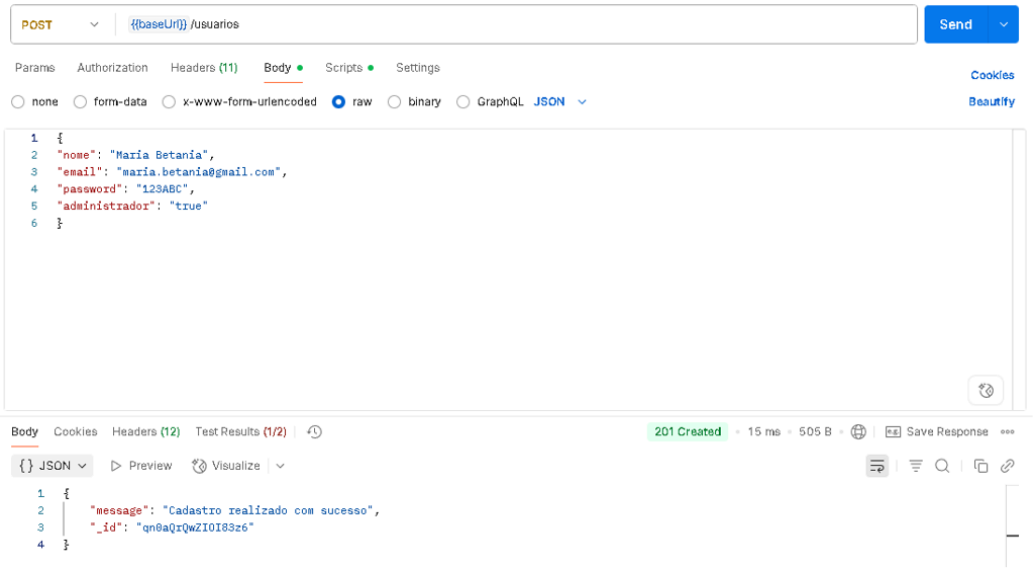
```
Request: POST /usuarios

Payload: { "nome": "Ana", "email": "ana@gmail.com", "password": "123456", "administrador": 

Response: { "message": "Cadastro realizado com sucesso", "_id": "paZZu8WCOTi33osS" }

```

- **Captura de Tela**:




---

## Relatório de Bug 2 - US 001: API de Usuários

**Título**: Cadastro de usuário com senha menor que 5 dígitos

**ID**: BUG-US001-002

**Data**: 03/09/2025

**Prioridade**: Alta

**Severidade**: Média

**Status**: Aberto

**Ambiente**:

- **Servidor:** ServeRest (teste)
- **Endpoint:** POST /usuarios
- **Ferramenta:** Postman

**Descrição:** Durante os testes realizados foi identificado a possiblidade de realizar o cadastro com o uso de senhas menor que 5 dígitos. Ferindo assim a regra de negócios “As senhas devem possuír no mínimo 5 caracteres e no máximo 10 caracteres”.

**Cenário de Teste**: TC04 - Cadastro com senha inválida (menos de 5 caracteres)

**Passos para Reproduzir**:

Enviar a requisição POST /usuarios com payload: `{nome: "Gustavo Martins", email: "gustavo.m@exemplo.com", password: "EGO", administrador: "true"}`.

**Comportamento Observado**:
Status 201, usuário criado:

```
{
    "message": "Cadastro realizado com sucesso",
    "_id": "zkaaxNYWaq06z"
}

```

**Comportamento Esperado**:
Status 400, mensagem:

```
{
    "message": "Provedor de e-mail inválido"
}

```

**Evidências**:

- **Log**:

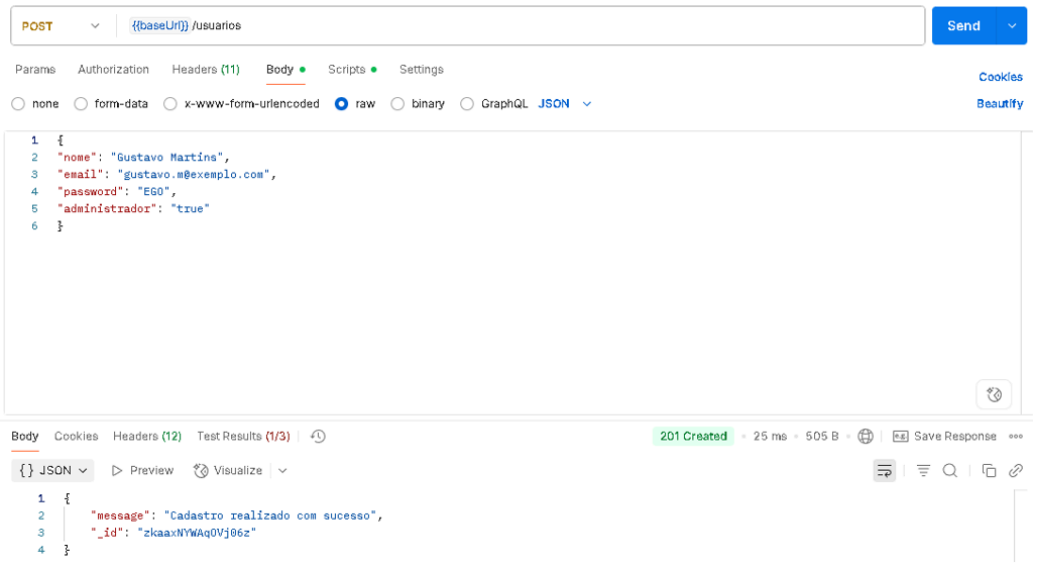
```
Request: POST /usuarios

Payload: {nome: "Gustavo Martins", email: "gustavo.m@exemplo.com", password: "EGO", administ

Response: { "message": "Cadastro realizado com sucesso", "_id": "zkaaxNYWaq06z" }

```

- **Captura de Tela**:




---

## Relatório de Bug 3 - US 002: API de Usuários

**Título**: Loguin de usuário com e-mail de provedor bloqueado

**ID**: BUG-US002-001

**Data**: 03/09/2025

**Prioridade**: Alta

**Severidade**: Média

**Status**: Aberto

**Ambiente**:

- **Servidor:** ServeRest (teste)
- **Endpoint:** POST /usuarios
- **Ferramenta:** Postman

**Descrição:** Durante os testes realizados foi identificado a possiblidade de realizar o cadastro de usuário com o uso de provedores de e-mail que deveriam estar bloqueados, sendo assim foi constatado a possibilidade de realizar loguincom este usuário. Ferindo a mesma regra de negócio, “Não deverá ser possível cadastrar usuários com e-mails de provedor gmail e hotmail”.

**Cenário de Teste**: TC04 - Login com e-mail inválido

**Passos para Reproduzir**:

Enviar a requisição POST /usuarios com payload: `{email: "maria.betania@gmail.com", password: "123ABC"}`.

**Comportamento Observado**:

Status 400, usuário criado:

```
{
  "message": "Login realizado com sucesso",
  "authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}

```

**Comportamento Esperado**:

Status 400, mensagem:

```
{
    "message": "Provedor de e-mail inválido"
}

```

**Evidências**:

- **Log**:

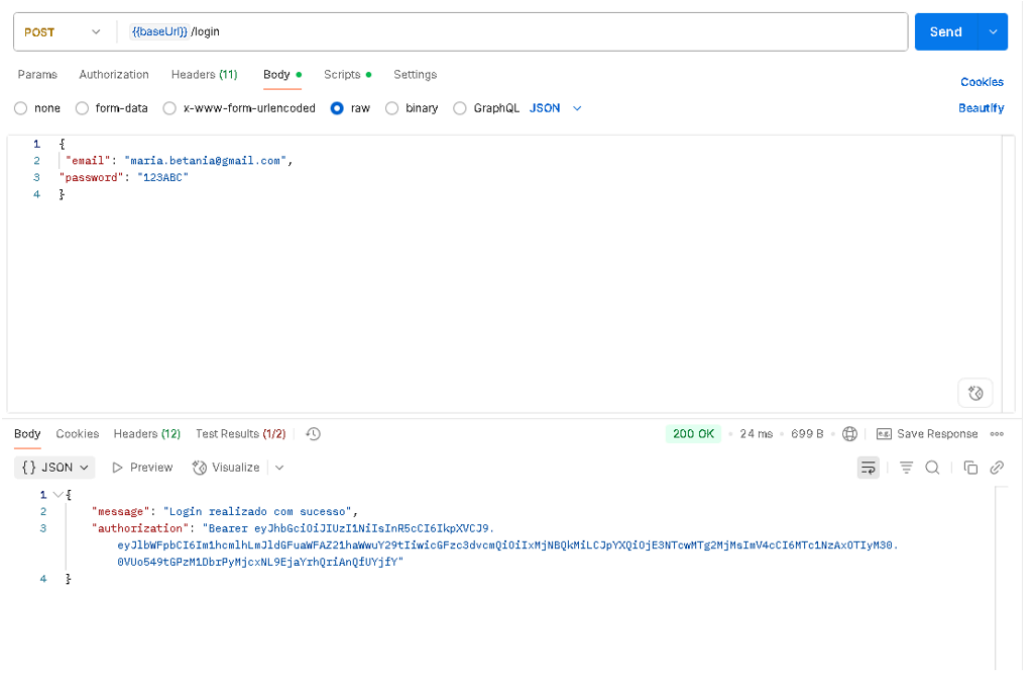
```
Request: POST /login

Payload: {email: "maria.betania@gmail.com", password: "123ABC"}

Response: {"message": "Login realizado com sucesso",
  "authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."}

```

- **Captura de Tela**:




---

## Relatório de Bug 4 - US 003: API de Usuários

**Título**: Exclusão de item inexistente

**ID**: BUG-US003-001

**Data**: 04/09/2025

**Prioridade**: Baixa

**Severidade**: Baixa

**Status**: Aberto

**Ambiente**:

- **Servidor:** ServeRest (teste)
- **Endpoint:** PUT /produto/:_id
- **Ferramenta:** Postman

**Descrição:** Durante os testes foi realizado a manutenção de um item inexistente na loja, onde ele retornou a necessidade de um id de 16 caracteres, sendo que o id apresentado ja cumpria com essa regra.

**Cenário de Teste**: TC08 - Atualizar produto inexistente

**Passos para Reproduzir**:

Enviar a requisição PUT /usuarios preenchido a key _id em Path Variables com um valor de id inexistente para produtos.

**Comportamento Observado**:

Status 400, usuário criado:

```
{
  "message":"Produto não encontrado",
}

```

**Comportamento Esperado**:

Status 400, mensagem:

```
{
    "id": "id deve ter exatamente 16 caracteres alfanuméricos"
}

```

**Evidências**:

- **Log**:

```
Request: PUT /login

Response: {"id": "id deve ter exatamente 16 caracteres alfanuméricos"}

```

- **Captura de Tela**: