🚀 Plano de Testes – API ServeRest

1. Apresentação

O presente plano de testes tem como objetivo garantir a **qualidade da API ServeRest**, verificando se as regras de negócio das funcionalidades de **usuários, login e produtos** estão implementadas corretamente, seguindo as User Stories definidas. Serão testadas as operações CRUD, autenticação e restrições de negócio, utilizando técnicas manuais e automatizadas, com apoio do Postman e evidências documentadas.

---

2. Objetivo

- Validar a implementação da API conforme as User Stories US 001, 002 e 003.
- Garantir que as regras de negócio estão sendo respeitadas, como restrições de cadastro de usuários, autenticação e cadastro de produtos.
- Detectar inconsistências, erros e cenários alternativos não previstos.

---

3. Critérios

US 001 - [API] Usuários

- Os vendedores (usuários) deverão possuir os campos NOME, E-MAIL, PASSWORD e ADMINISTRADOR;
- Não deverá ser possível fazer ações e chamadas para usuários inexistentes;
- Não deve ser possível criar um usuário com e-mail já utilizado;
- Caso não seja encontrado usuário com o ID informado no PUT, um novo usuário deverá ser criado;
- Não deve ser possível cadastrar usuário com e-mail já utilizado utilizando PUT;
- Os testes executados deverão conter evidências;
- Não deverá ser possível cadastrar usuários com e-mails de provedor gmail e hotmail;
- Os e-mails devem seguir um padrão válido de e-mail para o cadastro;
- As senhas devem possuír no mínimo 5 caracteres e no máximo 10 caracteres;
- A cobertura de testes deve se basear no Swagger e ir além, cobrindo cenários alternativos.

US 002: [API] Login

- Usuários não cadastrados não deverão conseguir autenticar;
- Usuários com senha inválida não deverão conseguir autenticar;
- No caso de não autenticação, deverá ser retornado um status code 401 (Unauthorized);
- Usuários existentes e com a senha correta deverão ser autenticados;
- A autenticação deverá gerar um token Bearer;
- A duração da validade do token deverá ser de 10 minutos;
- Os testes executados deverão conter evidências;
- A cobertura de testes deve se basear no Swagger e ir além, cobrindo cenários alternativos.

US 003: [API] Produtos

- Usuários não autenticados não devem conseguir realizar ações na rota de Produtos;

- Não deve ser possível realizar o cadastro de produtos com nomes já utilizados;
- Não deve ser possível excluir produtos que estão dentro de carrinhos (dependência API Carrinhos);
- Caso não exista produto com o o ID informado na hora do UPDATE, um novo produto deverá ser criado;
- Produtos criados através do PUT não poderão ter nomes previamente cadastrados;
- Os testes executados deverão conter evidências;
- A cobertura de testes deve se basear no Swagger e ir além, cobrindo cenários alternativos.

---

4. Escopo

Incluído:

- Rotas de usuários (CRUD)
- Rota de login (autenticação com token Bearer)
- Rotas de produtos (CRUD)
- Validações de regras de negócio: e-mails válidos, duplicidade, senhas, token Bearer, restrições de exclusão.

Excluído:

- APIs externas não documentadas.
- Testes de performance ou carga neste escopo inicial.

---

5. Análise

- **Usuários:** Verificar criação, atualização, listagem e exclusão de usuários; validação de e-mail e senha; restrição de duplicidade; regra de PUT criar novo usuário se ID não existir.
- **Login:** Verificar autenticação de usuários válidos e inválidos, geração de token Bearer com validade de 10 minutos, retorno correto de status 401 para falhas.
- **Produtos:** Verificar CRUD de produtos, restrições de exclusão, prevenção de duplicidade, dependência com API de carrinhos, criação via PUT se ID inexistente.
- **Swagger:** Analisar rotas, parâmetros obrigatórios, tipos de dados, exemplos e documentação de retorno.

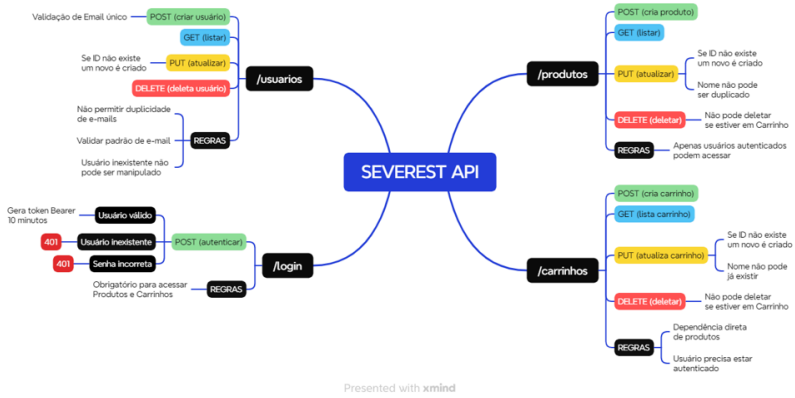
---

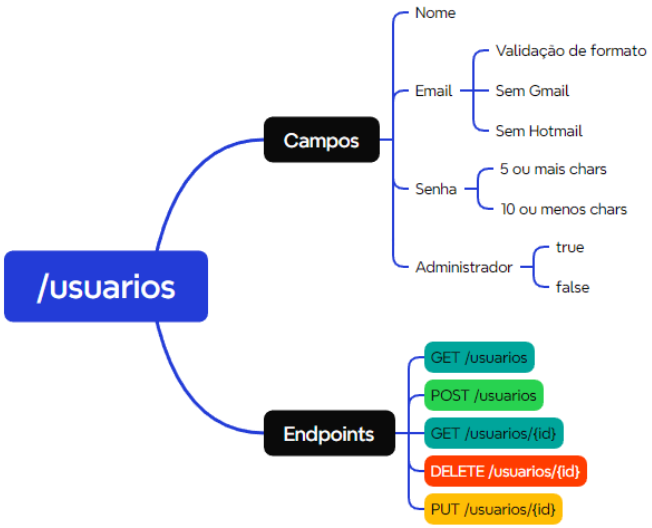
6. Técnicas Aplicadas

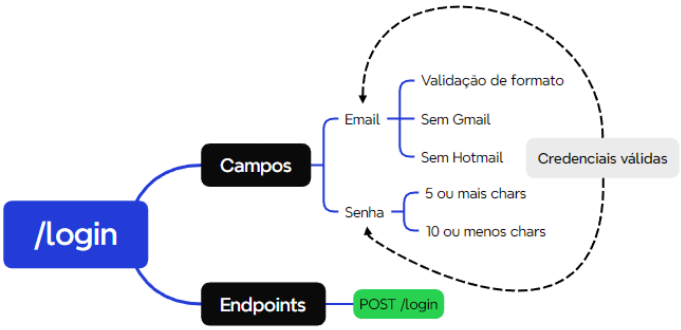
- **Testes Funcionais:** Validação das regras de negócio da API.
- **Testes Exploratório Goldilocks:** Testes além do esperado, como e-mails inválidos, senhas fora do padrão, nomes duplicados, usuários inexistentes.
- **Testes de Automação:** Scripts Postman na aba **Tests** para verificação automática de status codes, schema e retorno esperado.
- **Matriz de rastreabilidade:** Garantir que todas as User Stories possuem cobertura de testes.

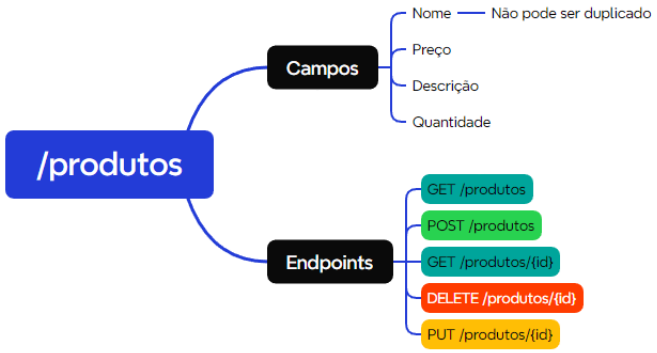
---

7. Mapa Mental da Aplicação









---

8. Cenários de Teste Planejados

ID	Funcionalidade	Cenário	Entrada	Resultado Esperado	Prioridade
TC001	Usuários	Criar usuário válido	Nome, e-mail válido, senha 5-10 chars	**201 Created**, usuário cadastrado	Alta
TC002	Usuários	Criar usuário com e-mail duplicado	E-mail já cadastrado	**400 Bad Request**	Alta
TC003	Usuários	Criar usuário com e-mail Gmail/Hotmail	Nome, e-mail @gmail.com ou @hotmail.com	**400 Bad Request**	Média
TC004	Usuários	Criar usuário com senha inválida	Senha < 5 ou > 10 chars	**400 Bad Request**	Alta
TC005	Usuários	Atualizar usuário existente	ID válido	**200 OK**, dados	Alta

				atualizados	
TC006	Usuários	Atualizar usuário inexistente via PUT	ID inexistente	**201 Created**, novo usuário criado	Média
TC007	Usuários	Atualizar usuário com e-mail já existente via PUT	ID válido, e-mail já cadastrado	**400 Bad Request**	Alta
TC008	Usuários	Tentar ação com usuário inexistente	ID inexistente em GET/DELETE	**400 Bad Request**	Alta
TC009	Login	Autenticar usuário válido	E-mail e senha corretos	**200 OK**, token Bearer	Alta
TC010	Login	Autenticar usuário inexistente	E-mail não cadastrado	**401 Unauthorized**	Alta
TC011	Login	Autenticar senha inválida	Senha incorreta	**401 Unauthorized**	Alta
TC012	Login	Token expira após 10 minutos	Login válido, aguardar 10min	Próximas requisições **401 Unauthorized**	Alta
TC013	Produtos	Criar produto válido	Nome, preço, quantidade	**201 Created**	Alta
TC014	Produtos	Criar produto com	Nome já cadastrado	**400 Bad Request**	Alta

		nome duplicado			
TC015	Produtos	Excluir produto dentro do carrinho	ID de produto vinculado	**400 Bad Request**	Média
TC016	Produtos	Atualizar produto inexistente via PUT	ID inexistente	**201 Created**, produto criado	Média
TC017	Produtos	Atualizar produto via PUT com nome já existente	Nome de outro produto já cadastrado	**400 Bad Request**	Alta
TC018	Produtos	Acesso sem token	Qualquer operação (GET/POST/PUT/DELETE)	**401 Unauthorized**	Alta

---

9. Priorização da Execução dos Cenários

- **Alta:** Cenários críticos para regras de negócio e segurança (usuários e login).
- **Média:** Cenários importantes mas com impacto menor imediato (atualização via PUT, restrições de e-mail/produto).
- **Baixa:** Cenários de exploração ou alternativos (e-mails inválidos não comuns, nomes alternativos de produtos).

---

10. Matriz de Risco

ID	Funcionalidade	Cenário	Impacto	Probabilidade	Risco
TC001	Usuários	Criar usuário válido	Alto (base do sistema)	Baixa	**Médio**
TC002	Usuários	Criar usuário com e-mail duplicado	Médio	Alta	**Alto**

TC003	Usuários	Criar usuário com Gmail/Hotmail	Médio	Média	**Médio**
TC004	Usuários	Criar usuário com senha inválida	Alto (segurança)	Alta	**Crítico**
TC005	Usuários	Atualizar usuário existente	Alto	Média	**Alto**
TC006	Usuários	Atualizar usuário inexistente via PUT	Médio	Alta	**Alto**
TC007	Usuários	Atualizar usuário com e-mail já existente via PUT	Alto	Alta	**Crítico**
TC008	Usuários	Tentar ação com usuário inexistente	Médio	Média	**Médio**
TC009	Login	Autenticar usuário válido	Alto (acesso)	Baixa	**Alto**
TC010	Login	Autenticar usuário inexistente	Alto	Média	**Alto**
TC011	Login	Autenticar senha inválida	Alto (segurança)	Alta	**Crítico**
TC012	Login	Token expira	Alto (segurança)	Média	**Alto**

		após 10 minutos			
TC013	Produtos	Criar produto válido	Médio	Baixa	**Médio**
TC014	Produtos	Criar produto com nome duplicado	Médio	Alta	**Alto**
TC015	Produtos	Excluir produto dentro do carrinho	Alto (integridade)	Média	**Alto**
TC016	Produtos	Atualizar produto inexistente via PUT	Médio	Média	**Médio**
TC017	Produtos	Atualizar produto via PUT com nome já existente	Alto	Alta	**Crítico**
TC018	Produtos	Acesso sem token	Alto (segurança)	Alta	**Crítico**

---

11. Cobertura de Testes

- **Cobertura total de User Stories:** US 001, 002 e 003
- **Cobertura Swagger:** Todos os verbos e parâmetros obrigatórios
- **Cenários exploratórios:** Emails inválidos, senhas curtas/longas, produtos duplicados, usuários inexistentes.

---

12. Testes Automatizados Implementados

A suíte de testes automatizados foi desenvolvida utilizando Robot Framework com foco em validar a funcionalidade, as regras de negócio e a segurança da API ServeRest. Os seguintes cenários foram implementados e validados:

Validações Gerais e Autenticação

- **[SMOKE]** Verificação de disponibilidade e status online da API (Health Check).

- **[POSITIVO]** Geração de token de autenticação (Bearer Token) com credenciais de administrador e de usuário comum válidas.
- **[NEGATIVO]** Tentativa de autenticação com credenciais inválidas, validando a mensagem de erro "Bad credentials".

Módulo de Usuários (/usuarios)

- **[CRUD]** Cobertura do fluxo de criação (POST), consulta (GET por ID) e exclusão (DELETE) de usuários.
- **[NEGATIVO | REGRA DE NEGÓCIO]** Validação do bloqueio de cadastro para usuários com e-mail já existente, confirmando o retorno do status 400 Bad Request e a mensagem de erro específica.
- **[INVESTIGATIVO]** Confirmação de que a API não possui restrições para domínios de e-mail específicos (ex: @gmail.com, @hotmail.com), permitindo o cadastro.
- **[INVESTIGATIVO]** Confirmação de que a API não possui regras de validação para tamanho mínimo ou máximo de senhas.

Módulo de Produtos (/produtos)

- **[CRUD]** Cobertura do fluxo de criação (POST) e listagem (GET) de produtos.
- **[NEGATIVO | REGRA DE NEGÓCIO]** Validação do bloqueio de cadastro para produtos com nome já existente, confirmando o retorno do status 400 Bad Request e a mensagem de erro apropriada.
- **[SEGURANÇA]** Verificação de que a listagem de produtos (GET /produtos) é um endpoint público e não requer autenticação, ao contrário do que era esperado.

Módulo de Carrinhos (/carrinhos)

- **[FLUXO | END-TO-END]** Simulação de um fluxo de compra completo: autenticação de usuário, criação de produto, criação de carrinho e conclusão da compra.

Documento de Melhorias e Issues

Relatório de Bug - US 001: API de Usuários

Título: Cadastro com Gmail

ID: BUG-US001-001

Data: 04/09/2025

Prioridade: Alta

Severidade: Média

Ambiente:

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

Passos para Reproduzir:

1. Enviar POST /usuarios com payload:

```
{
  "nome": "Fulano da Silva",
  "email": "beltrano@gmail.com",
```

```
  "password": "teste",
  "administrador": "true"
}
```

2. Executar a requisição.

Comportamento Observado:

Status 201, usuário criado:

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

Comportamento Esperado:

Status 400, mensagem:

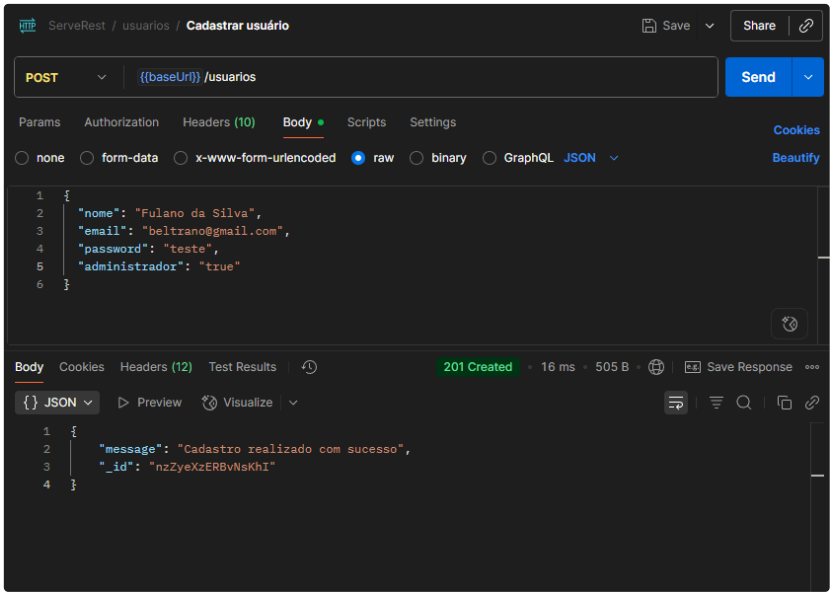
```
{
    "message": "Provedor de e-mail inválido"
}

```

Critério Violado:

"Não deverá ser possível cadastrar usuários com e-mails de provedor Gmail e Hotmail."

Evidências:

- **Captura de Tela**:



Cenário de Teste: TC03 - Criar usuário com e-mail Gmail/Hotmail

Sugestão de Correção:

Adicionar validação em `usuarios-controller.js`:

```
const email = req.body.email;
if (email.endsWith('@gmail.com') || email.endsWith('@hotmail.com')) {
  return res.status(400).json({ message: 'Provedor de e-mail inválido' }
}

```

Status: Aberto

Título: Cadastro com Hotmail

ID: BUG-US001-002

Data: 04/09/2025

Prioridade: Alta

Severidade: Média

Ambiente:

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

Passos para Reproduzir:

1. Enviar POST /usuarios com payload:

```
{
  "nome": "teste",
  "email": "ana@hotmail.com",
  "password": "teste",
  "administrador": "true"
}
```

2. Executar a requisição.

Comportamento Observado:

Status 201, usuário criado:

```
{
    "message": "Cadastro realizado com sucesso",
    "_id": "ndULl7ITa0xXloji"
}
```

Comportamento Esperado:

Status 400, mensagem:

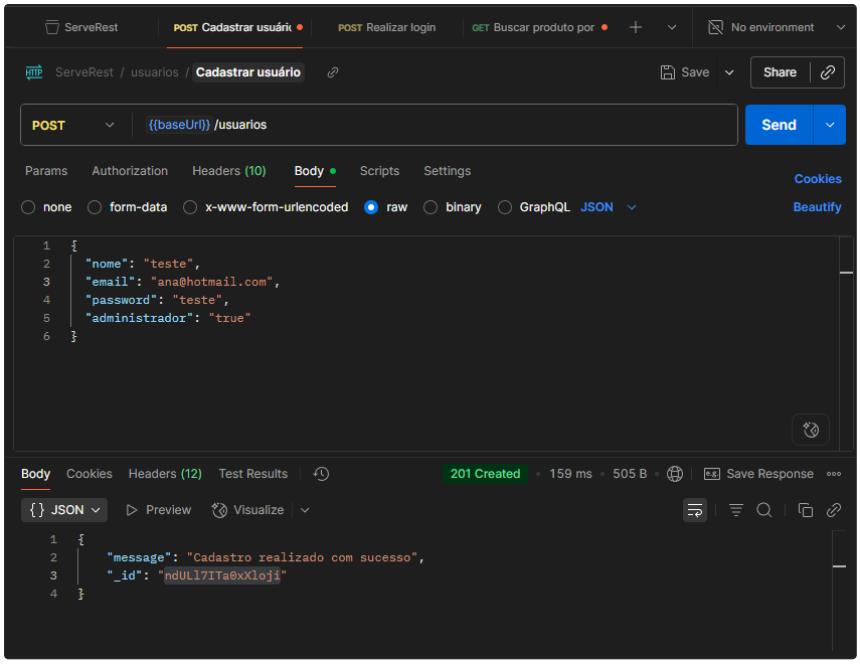
```
{
    "message": "Provedor de e-mail inválido"
}

```

Critério Violado:

"Não deverá ser possível cadastrar usuários com e-mails de provedor Gmail e Hotmail."

Evidências:

- **Captura de Tela**:



Cenário de Teste: TC03 - Criar usuário com e-mail Gmail/Hotmail

Sugestão de Correção:

Adicionar validação em `usuarios-controller.js`:

```
const email = req.body.email;
if (email.endsWith('@gmail.com') || email.endsWith('@hotmail.com')) {
  return res.status(400).json({ message: 'Provedor de e-mail inválido' }
}

```

Status: Aberto

Título: Senha menor que 5 chars

ID: BUG-US001-003

Data: 04/09/2025

Prioridade: Alta

Severidade: Alta

Ambiente:

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

Passos para Reproduzir:

1. Enviar POST /usuarios com payload:

```
{
  "nome": "teste",
  "email": "ana@teste.com",
  "password": "test",
  "administrador": "true"
}
```

2. Executar a requisição.

Comportamento Observado:

Status 201, usuário criado:

```
{
    "message": "Cadastro realizado com sucesso",
    "_id": "7A1vpkshGGAEeWrz"
}
```

Comportamento Esperado:

Status 400, mensagem:

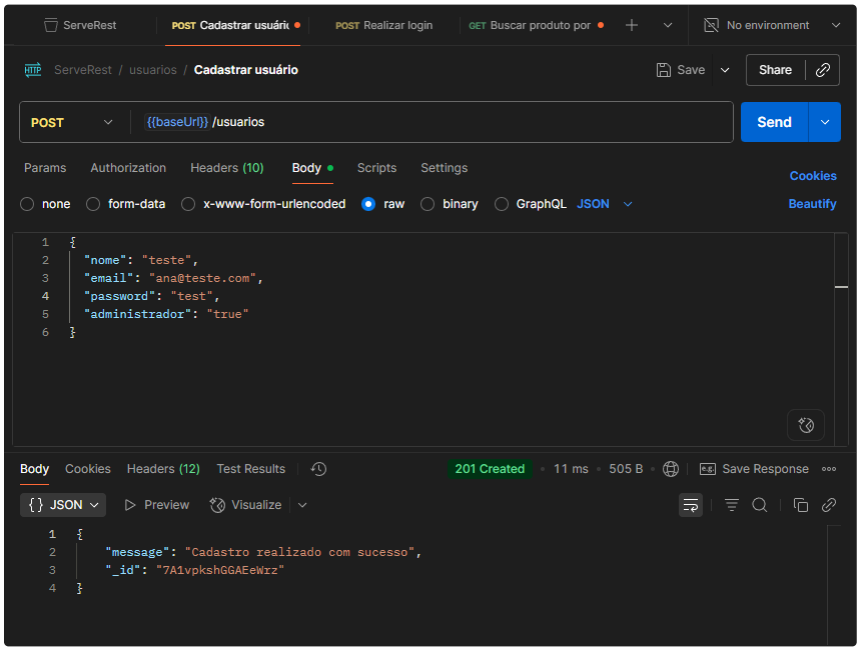
```
{
    "message": "Senha deve conter entre 5 e 10 caracteres"
}

```

Critério Violado:

"As senhas devem possuír no mínimo 5 caracteres e no máximo 10 caracteres."

Evidências:

- **Captura de Tela**:



Cenário de Teste: TC04 - Criar usuário com senha inválida

Sugestão de Correção:

Adicionar validação em `usuarios-controller.js`:

```
const senha = req.body.password;
if (senha.length < 5 || senha.length > 10) {
  return res.status(400).json({ message: 'Senha deve conter entre 5 e 10
}
```

Status: Aberto

Título: Senha maior que 10 chars

ID: BUG-US001-004

Data: 04/09/2025

Prioridade: Alta

Severidade: Alta

Ambiente:

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

Passos para Reproduzir:

1. Enviar POST /usuarios com payload:

```
{
  "nome": "teste",
  "email": "ana1@teste.com",
  "password": "testetestet",
  "administrador": "true"
}
```

2. Executar a requisição.

Comportamento Observado:

Status 201, usuário criado:

```
{
    "message": "Cadastro realizado com sucesso",
    "_id": "1w0gK7Ic5YKyorvt"
}
```

Comportamento Esperado:

Status 400, mensagem:

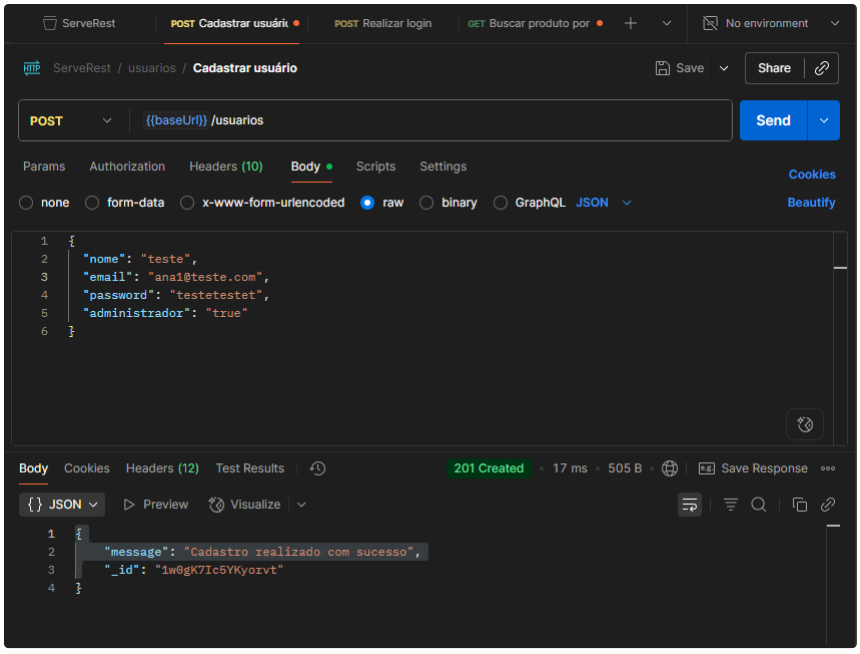
```
{
    "message": "Senha deve conter entre 5 e 10 caracteres"
}

```

Critério Violado:

"As senhas devem possuir no mínimo 5 caracteres e no máximo 10 caracteres."

Evidências:

- **Captura de Tela**:



Cenário de Teste: TC04 - Criar usuário com senha inválida

Sugestão de Correção:

Adicionar validação em `usuarios-controller.js`:

```
const senha = req.body.password;
if (senha.length < 5 || senha.length > 10) {
  return res.status(400).json({ message: 'Senha deve conter entre 5 e 10
}
```

Status: Aberto

Título: Buscar usuários sem autenticação

ID: BUG-US001-005

Data: 04/09/2025

Prioridade: Alta

Severidade: Alta

Ambiente:

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

Passos para Reproduzir:

1. Enviar Get /usuarios com email:

```
ana@teste.com
```

2. Executar a requisição.

Comportamento Observado:

Status 201, usuário encontrado:

```
{
    "quantidade": 1,
    "usuarios": [
        {
            "nome": "teste",
            "email": "ana@teste.com",
            "password": "test",
            "administrador": "true",
            "_id": "7A1vpkshGGAEeWrz"
        }
    ]
}
```

```
test",
```

```
            "administrador": "tru
```

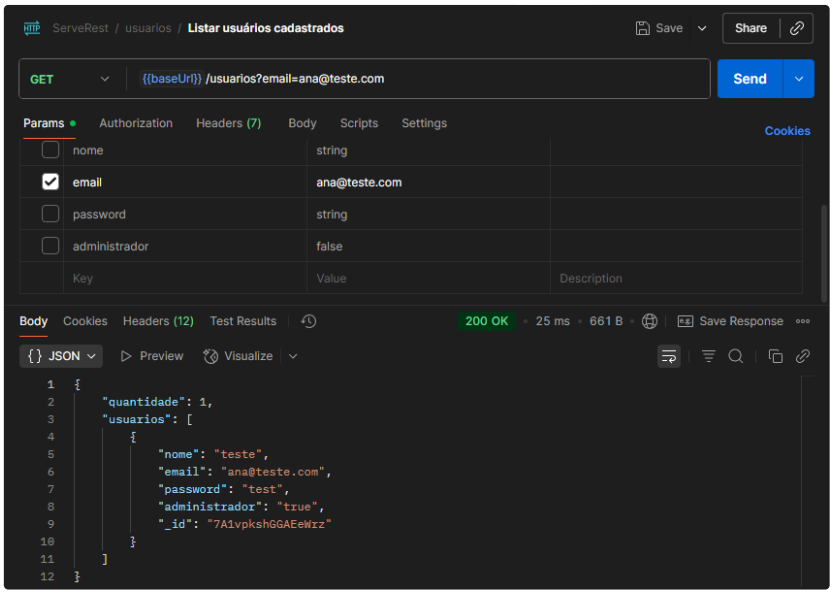
Comportamento Esperado:

Status 401 Unauthorized, mensagem:

```
{
    "message": "Ação precisa de administrador"
}

```

Evidências:

- **Captura de Tela**:



Cenário de Teste: Bug alternativo encontrado durante teste exploratório.

Sugestão de Correção: Mudar regra de Site para que seja necessário um Bearer Token para essa ação, visto que Email e Senha visíveis para todos os usuários do Site é um grande problema de segurança.

Status: Aberto

Relatório de Bug - US 002: API de Login

Título: Login com Gmail

ID: BUG-US002-001

Data: 05/09/2025

Prioridade: Alta

Severidade: Média

Ambiente:

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

Passos para Reproduzir:

1. Enviar POST /login com payload:

```
{
  "email": "teste@gmail.com",
  "password": "teste"
}
```

2. Executar a requisição.

Comportamento Observado:
Status 201, login realizado com sucesso:

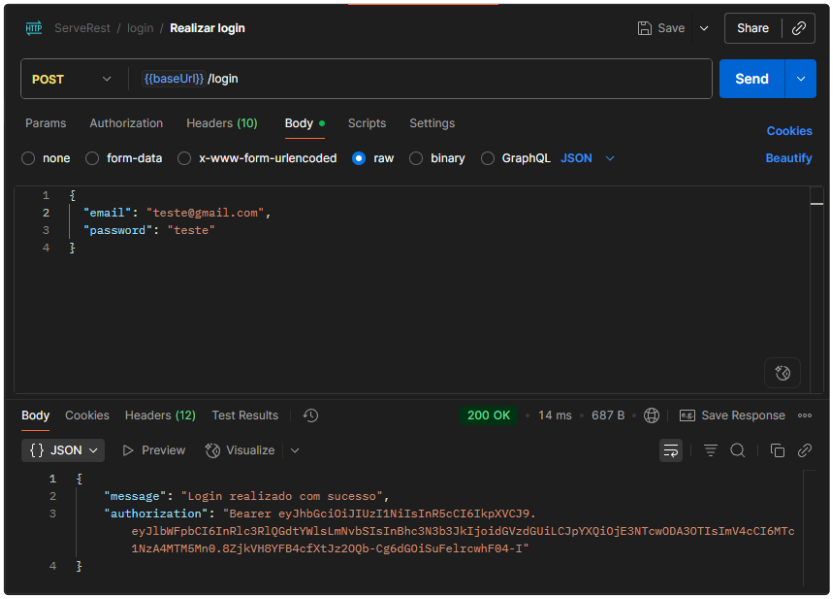
```
{
    "message": "Login realizado com sucesso",
    "authorization": "Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbW
}
```

Comportamento Esperado:
Status 400, mensagem:

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

```

Evidências:

- **Captura de Tela**:



Cenário de Teste: Bug alternativo encontrado durante teste exploratório.

Sugestão de Correção:

Adicionar validação durante cadastro `usuarios-controller.js`:

```
const email = req.body.email;
if (email.endsWith('@gmail.com') || email.endsWith('@hotmail.com')) {
  return res.status(400).json({ message: 'Provedor de e-mail inválido' }
}

```

Status: Aberto

Título: Login com Hotmail

ID: BUG-US002-002

Data: 05/09/2025

Prioridade: Alta

Severidade: Média

Ambiente:

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

Passos para Reproduzir:

1. Enviar POST /login com payload:

```
{
  "email": "teste@hotmail.com",
  "password": "teste"
}
```

2. Executar a requisição.

Comportamento Observado:

Status 201, login realizado com sucesso:

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

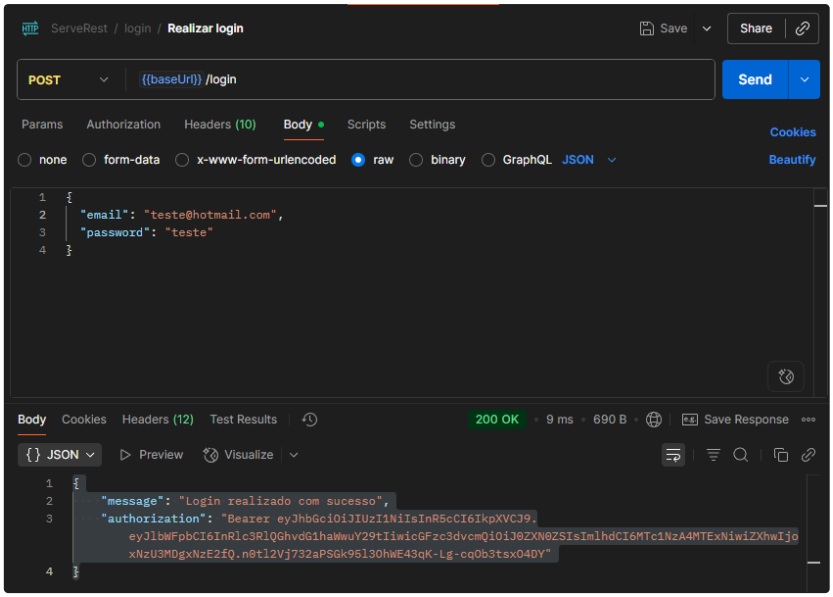
Comportamento Esperado:

Status 400, mensagem:

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

```

Evidências:

- **Captura de Tela**:



Cenário de Teste: Bug alternativo encontrado durante teste exploratório.

Sugestão de Correção:

Adicionar validação durante cadastro `usuarios-controller.js`:

```
const email = req.body.email;
if (email.endsWith('@gmail.com') || email.endsWith('@hotmail.com')) {
  return res.status(400).json({ message: 'Provedor de e-mail inválido' }
}

```

Status: Aberto

Relatório de Bug - US 003: API de Produtos

Título: Usuário não autenticado busca produto por ID

ID: BUG-US003-001

Data: 04/09/2025

Prioridade: Alta

Severidade: Alta

Ambiente:

- Servidor: ServeRest (teste)
- Endpoint: GET /produtos/{id}
- Ferramenta: Postman

Passos para Reproduzir:

1. Enviar GET /produtos com ID:

```
BeeJh5lz3k6kSIzA
```

2. Executar a requisição.

Comportamento Observado:

Status 200, produto buscado:

```
{
    "nome": "Logitech MX Vertical",
    "preco": 470,
    "descricao": "Mouse",
    "quantidade": 382,
    "_id": "BeeJh5lz3k6kSIzA"
}
```

Comportamento Esperado:

Status 401 Unauthorized, mensagem:

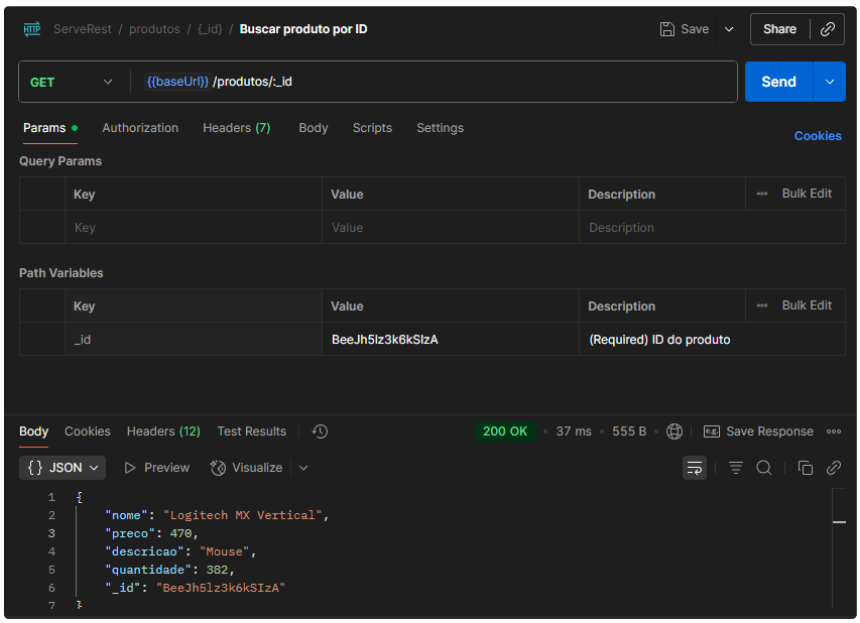
```
{
    "message": "Usuário deve ser autenticado"
}

```

Critério Violado:

"Usuários não autenticados não devem conseguir realizar ações na rota de Produtos."

Evidências:

- **Captura de Tela**:



Sugestão de Correção:

Adicionar validação para verificação de token bearer de usuário.

Status: Aberto

Título: Usuário não autenticado lista produtos cadastrados

ID: BUG-US003-002

Data: 04/09/2025

Prioridade: Alta

Severidade: Alta

Ambiente:

- Servidor: ServeRest (teste)

- Endpoint: GET /produtos
- Ferramenta: Postman

Passos para Reproduzir:

1. Enviar GET /produtos com ID:

```
BeeJh5lz3k6kSIzA
```

2. Executar a requisição.

Comportamento Observado:
Status 200, produto buscado:


```
{
    "quantidade": 1,
    "produtos": [
        {
            "nome": "Logitech MX Vertical",
            "preco": 470,
            "descricao": "Mouse",
            "quantidade": 382,
            "_id": "BeeJh5lz3k6kSIzA"
        }
    ]
}
```

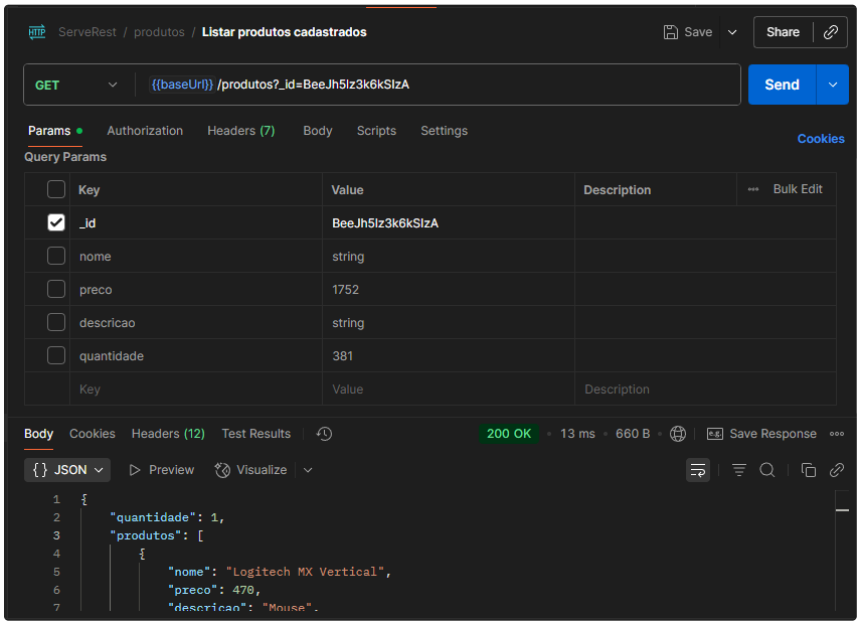

Comportamento Esperado:
Status 401 Unauthorized, mensagem:


```
{
    "message": "Usuário deve ser autenticado"
}
```


Critério Violado:
"Usuários não autenticados não devem conseguir realizar ações na rota de Produtos."

Evidências:

- **Captura de Tela**:



Sugestão de Correção:
Adicionar validação para verificação de token bearer de usuário.

Status: Aberto

Relatório de Bug - API de Carrinho

Título: Produto que não existe mais em carrinho é excluído

ID: BUG-US004-001

Data: 03/10/2025

Prioridade: Média

Severidade: Alta

Ambiente:

- Servidor: ServeRest (teste)
- Endpoint: DELETE /carrinhos/cancelar-compra
- Ferramenta: RobotFramework

Passos para Reproduzir:

1. Criar carrinho de compras
2. Concluir a compra (isso deleta o carrinho)
3. Tentar cancelar o carrinho que não existe mais (Gatilho do Bug)

Comportamento Observado:

status `200 OK`

Comportamento Esperado:

`404 Not Found`

Evidências:

- **Log:**