Por: João Pedro de Jesus Perin
Squad: 01 – Unit to Unity
Data: 06/02/2026

Versão do Plano:
2.1

Data de Criação:
02/02/2026

Data da Última Atualização:
06/02/2026

Responsável pela Elaboração:
João Pedro de Jesus Perin

Tempo Estimado de Execução:
Em torno de 16 horas (considerando o tempo integral do estágio durante a semana)

1. Escopo

1.1 Identificação do Projeto

Serão realizados testes exploratórios e funcionais na ServeRest API, uma API REST utilizada para simular um sistema de e-commerce, contemplando autenticação, usuários, produtos e carrinho de compras. O foco principal é validar a resiliência da API, a integridade dos dados, o tratamento de erros e o cumprimento do contrato HTTP.

1.2 Escopo do Teste

O escopo contempla os seguintes fluxos:

- Autenticação de usuários (Login)
- Cadastro, consulta, atualização e remoção de usuários
- Gerenciamento de produtos (operações administrativas)
- Criação, consulta e finalização/cancelamento de carrinho
- Validação de respostas HTTP e contratos

Fora do escopo:

- Testes de carga e performance
- Testes de segurança avançados (pentest)
- Interface gráfica (front-end)

2. Área Abrangida (Reprodução)

- Realizar requisições REST utilizando ferramentas como Postman
- Simular requisições válidas e inválidas
- Testar ausência, expiração e invalidez de token JWT

- Forçar erros de negócio (produto já existente, carrinho vazio, produto em carrinho)
- Validar códigos de status HTTP e mensagens retornadas

3. Mapeamento dos Endpoints

O mapeamento dos endpoints tem como objetivo identificar todas as rotas disponíveis na ServeRest API, seus métodos HTTP, finalidade, autenticação necessária e possíveis respostas esperadas. Esse mapeamento serve como base para a criação de cenários de teste funcionais, negativos e exploratórios.

3.1 Login

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/login	Autentica o usuário e retorna token JWT	Não	200 (Sucesso), 401 (Credenciais inválidas)

3.2 Usuário

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/usuario	Cadastro de novo usuário	Não	201 (Criado), 400 (Usuário já existente)
GET	/usuario	Lista usuários cadastrados	Não	200 (Sucesso)
GET	/usuario/{id}	Consulta usuário por ID	Não	200 (Encontrado), 400 (Não encontrado)
PUT	/usuario/{id}	Atualiza dados do usuário	Não	200 (Atualizado), 400 (Credencial duplicada)
DELETE	/usuario/{id}	Remove usuário	Não	200 (Deletado), 400 (Carrinho vinculado)

3.3 Produto

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/produto	Cria novo produto	Sim (ADM)	201 (Criado), 400 (Já existe), 401, 403
GET	/produto	Lista produtos	Não	200 (Sucesso)
GET	/produto/{id}	Consulta produto por ID	Não	200 (Encontrado), 400 (Não existe)
PUT	/produto/{id}	Atualiza produto	Sim (ADM)	200 (Atualizado), 400, 401, 403
DELETE	/produto/{id}	Remove produto	Sim (ADM)	200 (Deletado), 400 (Em carrinho), 401, 403

3.4 Carrinho

Método	Endpoint	Descrição	Autenticação	Respostas Esperadas
POST	/carrinho	Cria carrinho para o usuário	Sim	201 (Criado), 400 (Falha), 401
GET	/carrinho	Consulta carrinho do usuário	Sim	200 (Encontrado)
GET	/carrinho/{id}	Consulta carrinho por ID	Sim	200 (Encontrado), 400 (Não existe)
DELETE	/carrinho/concluir-compra	Finaliza compra	Sim	200 (Sucesso), 401

DELETE	/carrinho/ concluir-compra	Cancela compra	Sim	200 (Sucesso), 401

4. Ambiente de Testes

- Ferramenta de requisição: Postman
- Sistema Operacional: Windows 10/11
- API: ServeRest (ambiente público:)
- Protocolo: HTTP/HTTPS

5. Objetivos

5.1 Objetivo Geral

Explorar a ServeRest API por meio de testes funcionais e exploratórios, utilizando técnicas de sabotagem controlada para validar como a API reage a entradas inválidas, fluxos indevidos e falhas de autenticação, garantindo robustez e previsibilidade no consumo por aplicações clientes.

5.2 Objetivos Específicos

- Validar contratos de API (request/response)
- Garantir tratamento correto de erros HTTP (400, 401, 403, 404)
- Identificar riscos de inconsistência de dados
- Avaliar se a API responde de forma clara e padronizada a falhas

6. Riscos de Produto

Risco	Descrição	Impacto	Mitigação
Quebra de Autenticação	Acesso permitido sem token válido	Alto	Validação rigorosa de JWT em todos os endpoints protegidos
Inconsistência de Dados	Cadastro duplicado de usuários ou produtos	Alto	Validação rigorosa de JWT em todos os endpoints protegidos
Violação de Contrato	Campos ausentes ou tipos incorretos na resposta	Médio	Testes de contrato automatizados
Fluxo de Carrinho Inconsistente	Finalização ou exclusão indevida de carrinho	Médio	Validação de estado do carrinho antes das ações

7. Estratégia de Testes Exploratórios

Estratégia Escolhida: Testes Exploratórios Baseados em Sabotage Tours (API Sabotage)

Justificativa: A ServeRest API é amplamente utilizada como base para testes e estudos. A abordagem de Sabotage Tour permite explorar falhas além do caminho feliz, testando limites da

API, estados inválidos e sequências incorretas de chamadas, garantindo maior confiabilidade no consumo por aplicações externas.

8. Heurísticas Aplicadas (Adaptadas para API)

- Visibilidade do status do sistema: Retornos HTTP claros e consistentes
- Prevenção de erros: Bloqueio de dados inválidos no back-end
- Consistência: Padronização de mensagens e códigos de resposta
- Recuperação de erros: Mensagens claras ao consumidor da API

9.Cenários de Teste Funcionais

9.1 Cenários Positivos

- Realizar login com credenciais válidas e receber token JWT.
- Cadastrar usuário com dados válidos.
- Criar produto com token válido de administrador.
- Criar carrinho com produto existente.
- Concluir compra com carrinho válido.

9.2 Cenários Negativos

- Login com senha incorreta.
- Acesso a rota protegida sem token.
- Cadastro de usuário já existente.
- Criação de produto sem permissão de administrador.
- Exclusão de produto vinculado a carrinho.

9.3 Validação de Contrato

- Verificar obrigatoriedade de campos no request.
- Validar tipos de dados retornados no response.
- Conferir códigos HTTP conforme documentação (200, 201, 400, 401, 403, 404).

10.Tabela de Casos de Teste

10.1 Módulo: Login

ID	Cenário	Endpoint	Método	Pré-condição	Dados de Entrada	Resultado Esperado	Tipo	Situação
CT-LOGIN-01	Login com credenciais válidas	/login	POST	Usuário cadastrado	Email e senha válidos	Retornar token JWT e status 200	Positivo	Pass
CT-	Login com	/login	POST	Usuário	Email	Mensagem	Negativo	Pass

LOGIN-02	senha inválida			cadastrado	válido e senha incorreta	de erro e status 401		
CT-LOGIN-03	Login sem informar senha	/login	POST	Nenhuma	Email sem senha	Mensagem de campo obrigatório e status 400	Negativo	Pass
CT-LOGIN-04	Validar contrato do login	/login	POST	Usuário cadastrado	Credenciais válidas	Token no campo authorization	Contrato	Pass

10,**Módulo: Usuários**

ID	**Cenário**	**Endpoint**	**Método**	**Pré-condição**	**Dados de Entrada**	**Resultado Esperado**	**Tipo**	**Situação**
CT-USU-01	Criar usuário com dados válidos	/usuarios	POST	Nenhuma	Nome, email e senha válidos	Usuário criado com status 201	Positivo	Pass
CT-USU-02	Criar usuário com email duplicado	/usuarios	POST	Usuário já existente	Email já cadastrado	Mensagem de erro e status 400	Negativo	Pass
CT-USU-03	Consultar lista de usuários	/usuarios	GET	Nenhuma	N/A	Lista de usuários e status 200	Positivo	Pass
CT-USU-04	Validar contrato de usuário	/usuarios	GET	Nenhuma	N/A	Campos id, nome e email presentes	Contrato	Pass

10.3 Módulo: Produtos

ID	**Cenário**	**Endpoint**	**Método**	**Pré-condição**	**Dados de Entrada**	**Resultado Esperado**	**Tipo**	**Situação**
CT-PROD-01	Criar produto com token válido	/produtos	POST	Usuário autenticado	Nome, preço e quantidade	Produto criado com status 201	Positivo	Pass
CT-PROD-02	Criar produto sem token	/produtos	POST	Nenhuma	Dados válidos sem token	Erro de autenticação e status 401	Negativo	Pass
CT-PROD-03	Consultar produtos	/produtos	GET	Nenhuma	N/A	Lista de produtos e status 200	Positivo	Pass
CT-PROD-04	Validar contrato de produto	/produtos	GET	Nenhuma	N/A	Campos nome, preço e quantidade	Contrato	Pass

10.4 Módulo: Carrinho

ID	**Cenário**	**Endpoint**	**Método**	**Pré-condição**	**Dados de Entrada**	**Resultado Esperado**	**Tipo**	**Situação**
CT-	Criar	/carrinhos	POST	Usuário	Id do	Carrinho	Positivo	Pass

CAR-01	carrinho com produto válido			autenticado	produto e quantidade	criado e status 201		
CT-CAR-02	Criar carrinho sem autenticação	/carrinhos	POST	Nenhuma	Produto válido sem token	Erro de autenticação e status 401	Negativo	Pass
CT-CAR-03	Finalizar compra	/carrinhos /concluir-compra	DELETE	Carrinho existente	N/A	Compra finalizada e status 200	Positivo	Pass
CT-CAR-04	Validar contrato do carrinho	/carrinhos	GET	Usuário autenticado	N/A	Estrutura do carrinho válida	Contrato	Pass

11. Bugs e Isseus

Bug 01: Cadastro de usuário com nome e senha vazios

Descrição:

Durante a validação do endpoint de cadastro de usuários, foi identificado que a API permite a criação de usuário mesmo quando os campos obrigatórios nome e senha são enviados vazios ("").

O sistema retorna resposta de sucesso (201 Created ou 200 OK) e persiste o registro no banco de dados, sem aplicar validação mínima de obrigatoriedade ou integridade dos dados.

Esse comportamento compromete a consistência da base de usuários, permite criação de contas inválidas e pode gerar falhas posteriores no processo de autenticação, autorização e rastreabilidade de ações no sistema.

Reprodução:

1. Acessar o Postman ou ferramenta similar de testes de API;
2. Selecionar o endpoint POST /usuario;
3. Configurar o Body como raw → JSON;
4. Enviar a seguinte requisição:

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

5. Enviar a requisição;
6. Observar que a API retorna status de sucesso;
7. Confirmar que o usuário foi criado no sistema ou banco de dados mesmo com campos obrigatórios vazios.

Evidências:

- Status code retornado: 201 Created ;
- Registro persistido no banco de dados com campos vazios;
- Ausência de mensagem de erro de validação;
- Ausência de bloqueio da requisição;
- Usuário visível em listagem via GET /usuario.

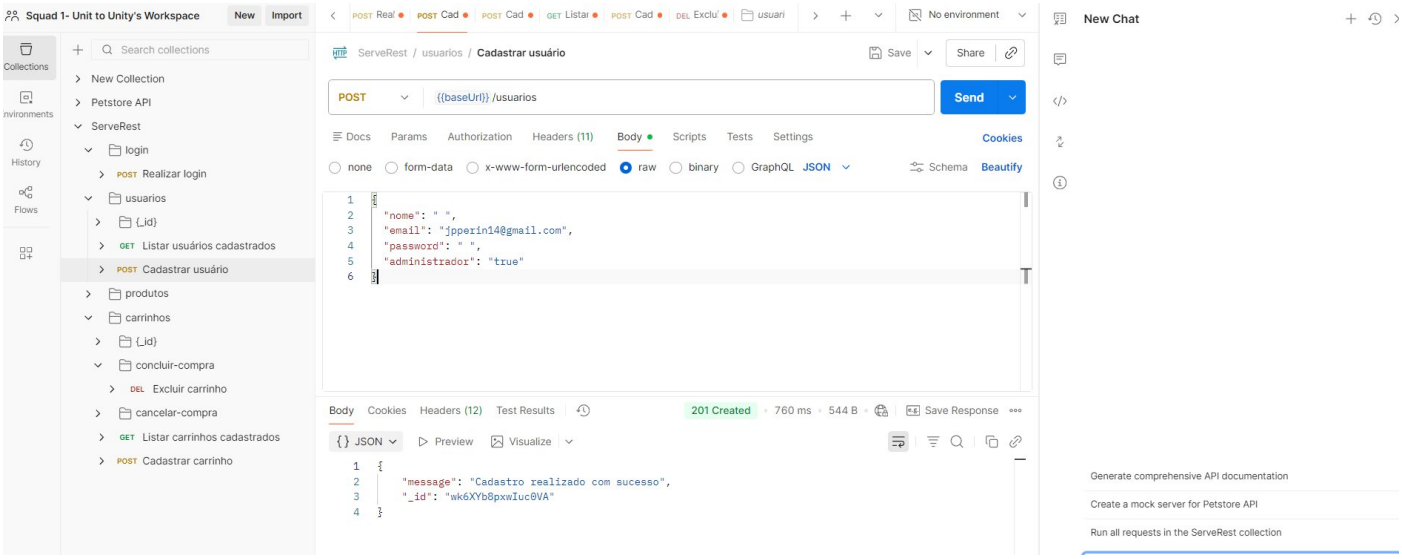

Figura 01 - Requisição enviada com nome e senha vazios

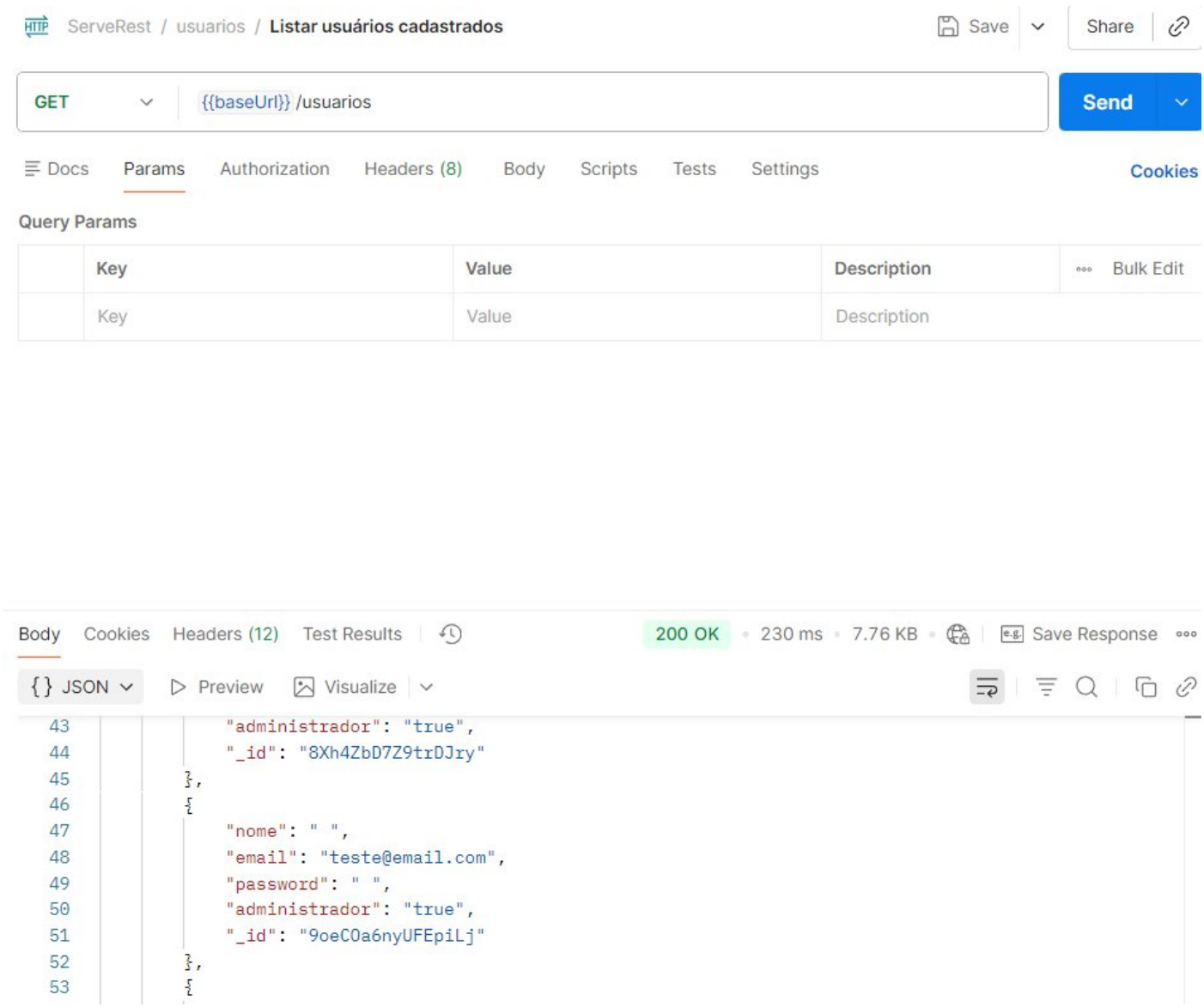

Figura 02 - Usuário inválido presente na listagem

Comportamento Esperado:

A API deve:

- Retornar 400 Bad Request;
- Informar mensagem clara como:
- "Nome é obrigatório"
- "Senha é obrigatória"
- Bloquear persistência no banco;
- Aplicar validação de campos obrigatórios antes de processar a requisição.

Ambiente:

Plataforma: API REST
Ferramenta de Teste: Postman
Sistema Operacional: Windows 11
Data e horário do teste: 08/01/2026 às 14:35h

Prioridade:

Alta

Justificativa:

O problema compromete a integridade da base de dados, permite criação de contas inválidas e pode impactar autenticação, autorização e segurança do sistema.

Além disso, facilita exploração automatizada (bots), geração de usuários fantasmas e inconsistências operacionais, podendo afetar métricas, relatórios e controles de acesso.

Bug 02: Autenticação permite login com usuário que possui senha vazia

Descrição:

Foi identificado que a API permite autenticação de usuário cuja senha foi cadastrada como vazia ("").

Durante o teste do endpoint de login (POST /login ou equivalente), ao informar um usuário previamente criado com campo **senha vazio**, o sistema retorna autenticação bem-sucedida (200 OK) e gera token válido de acesso, permitindo acesso completo ao sistema.

O comportamento indica ausência de validação adequada no fluxo de autenticação, comprometendo os controles de segurança e permitindo acesso indevido ao sistema com credenciais inválidas.

Reprodução:

1. Criar um usuário via POST /usuario com senha vazia:

```
{
  "nome": "usuario_teste",
  "email": "teste@email.com",
  "password": " "
 "administrador": "false "
}
```

2. Confirmar que o usuário foi criado com sucesso;
3. Acessar o endpoint POST /login;
4. Enviar as credenciais:

```
{
  "email": "teste@email.com",
  "senha": ""
}
```

5. Enviar a requisição;
6. Observar que a API retorna autenticação válida;

7. Confirmar que um token JWT (ou similar) é gerado e pode ser utilizado para acessar endpoints protegidos.

Evidências:

- Status code retornado: **200 OK**;
- Token de autenticação gerado;
- Possibilidade de acesso a endpoints protegidos utilizando o token;
- Ausência de mensagem de erro ou bloqueio de autenticação.

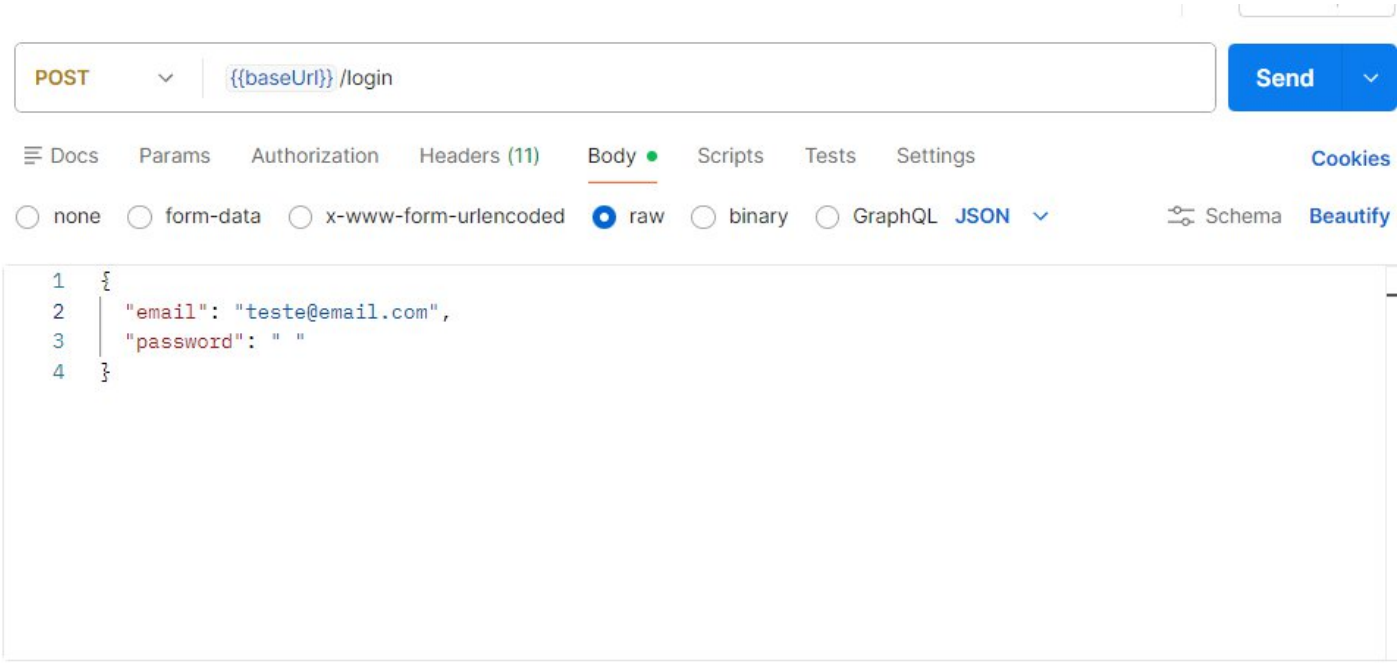

Figura 01 - Requisição de login com senha vazia

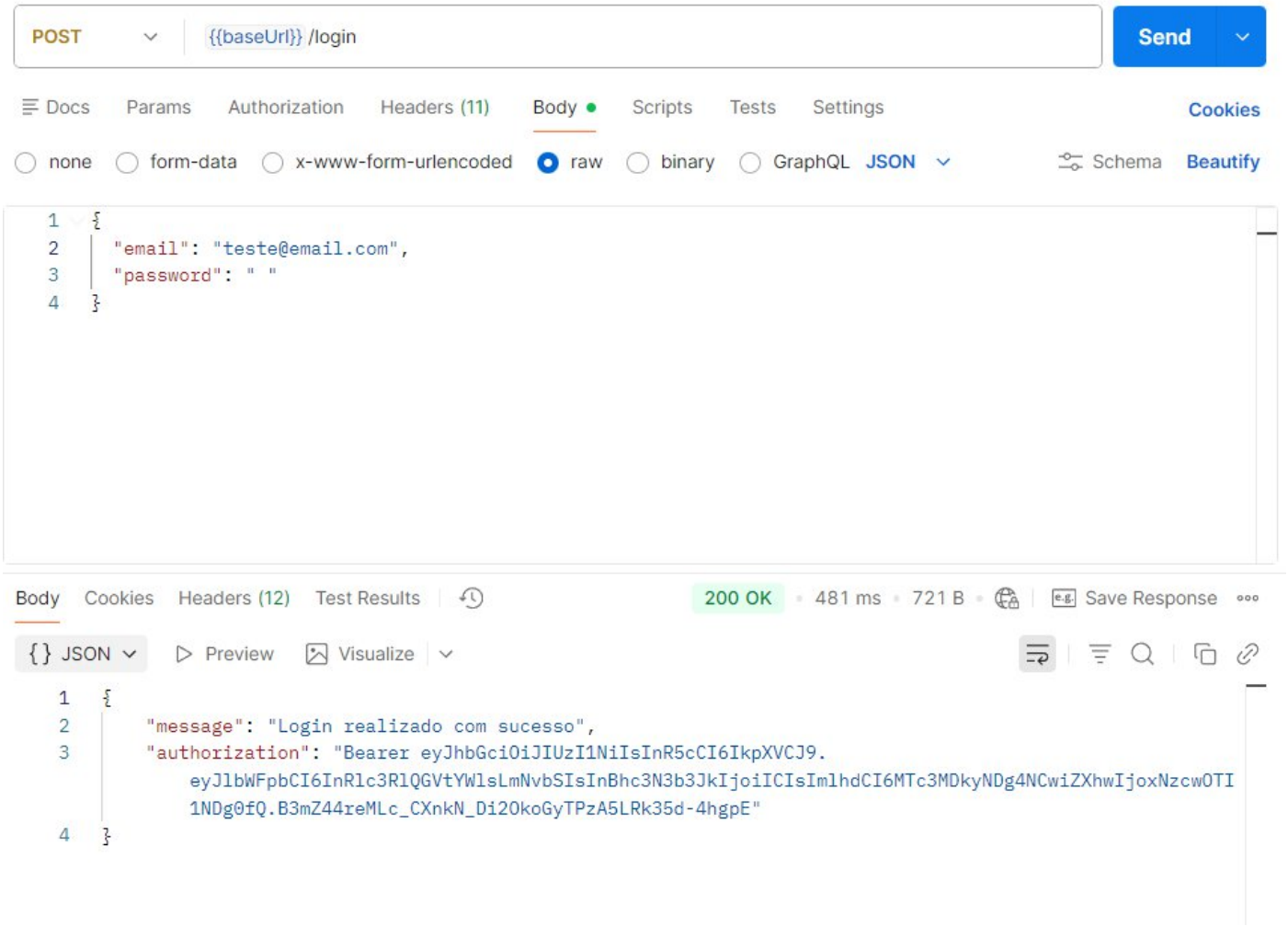

Figura 02 - Token retornado indevidamente

Comportamento Esperado:

A API deve:

- Retornar **400 Bad Request** (dados inválidos)
  ou
- Retornar **401 Unauthorized** (credenciais inválidas);
- Bloquear autenticação quando senha estiver vazia;
- Validar obrigatoriedade de senha antes de processar autenticação;
- Impedir geração de token em caso de credenciais inválidas.

Ambiente:

Plataforma: API REST
Ferramenta de Teste: Postman
Sistema Operacional: Windows 11
Data e horário do teste: 08/01/2026 às 15:10h

Prioridade:

Crítica

Justificativa:

A falha compromete diretamente o mecanismo de autenticação do sistema, permitindo acesso não autorizado.

O problema representa risco elevado de segurança, podendo possibilitar:

- Acesso indevido a dados sensíveis;
- Escalada de privilégios;
- Exploração automatizada;
- Comprometimento da integridade do sistema.

Trata-se de vulnerabilidade de segurança com impacto direto na confidencialidade e integridade das informações.

Bug 03: Cadastro de produto permite nome e descrição vazios

Descrição:

Durante os testes do endpoint POST /produto, foi identificado que a API permite a criação de produtos com os campos **nome** e **descrição** vazios (""); **preço** e **quantidade** sendo igual ou maior que zero(0).

Mesmo com ausência de informações essenciais para identificação e comercialização do item, o sistema retorna resposta de sucesso (201 Created ou 200 OK) e persiste o produto na base de dados.

O comportamento indica ausência de validação adequada de campos obrigatórios, comprometendo a qualidade dos dados e podendo gerar impactos operacionais, comerciais e de usabilidade no sistema.

Reprodução:

1. Acessar o Postman;
2. Selecionar o endpoint POST /produto;
3. Configurar o Body como raw → JSON;
4. Enviar a seguinte requisição:

```
{
  "nome": " ",
  "preco": 1,
  "descricao": " ",
  "quantidade": 0
}
```

5. Enviar a requisição;
6. Observar que a API retorna sucesso;
7. Confirmar que o produto foi criado e está disponível via GET /produto.

Evidências:

- Status code retornado: **201 Created** (ou 200 OK);
- Produto persistido no banco com nome e descrição vazios;
- Produto visível na listagem;
- Ausência de mensagem de erro de validação;
- Nenhum bloqueio aplicado durante o cadastro.

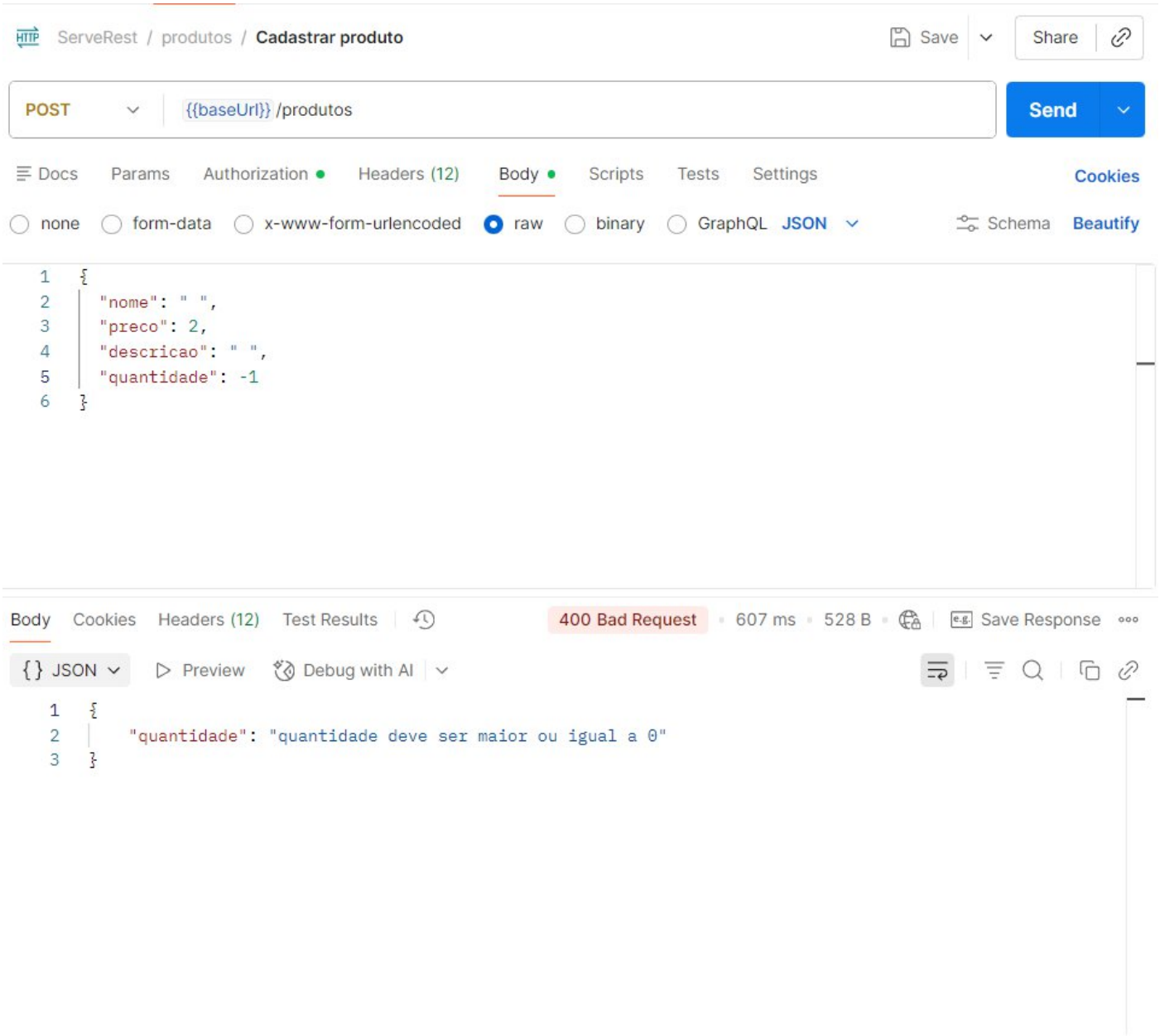

Figura 01 - Requisição com nome e descrição vazios

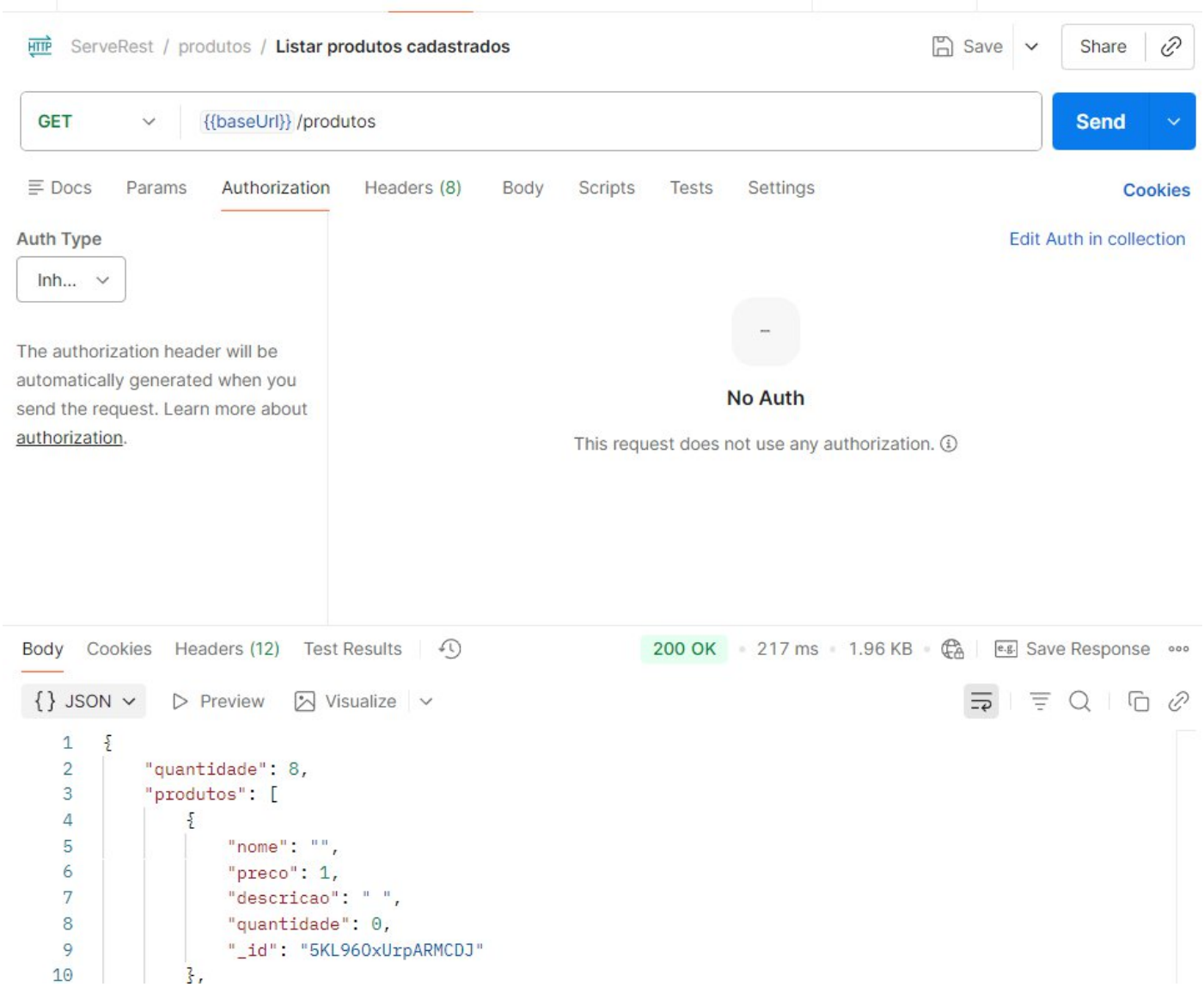

Figura 02 - Produto inválido presente na listagem

Comportamento Esperado:

A API deve:

- Retornar **400 Bad Request**;
- Informar mensagens claras como:
    - "Nome é obrigatório"
    - "Descrição é obrigatória"
    - “Preço deve ser maior que 1”
    - “Quantidade deve ser maior que 1”
- Impedir persistência de dados incompletos;
- Validar tamanho mínimo dos campos de texto;
- Garantir regras de integridade antes da gravação no banco.

Ambiente:

Plataforma: API REST
Ferramenta de Teste: Postman
Sistema Operacional: Windows 11
Data e horário do teste: 08/01/2026 às 15:40h

Prioridade:

Alta

Justificativa:

O problema compromete a integridade da base de produtos, pode impactar exibição em catálogo, relatórios financeiros e experiência do usuário.

Permitir cadastro de produtos sem identificação adequada pode gerar inconsistências operacionais, erros em pedidos e prejuízos comerciais.

Bug 04: Inconsistência entre cadastro de produto e adição ao carrinho com quantidade zero

Descrição:

Durante os testes funcionais da API, foi identificado que o sistema permite o cadastro de um produto com **quantidade igual a zero** via POST /produto.

Entretanto, ao tentar adicionar esse mesmo produto ao carrinho via endpoint de carrinho (ex: POST /carrinho), o sistema bloqueia a operação informando indisponibilidade de estoque.

Isso evidencia uma inconsistência nas regras de negócio: o sistema aceita o cadastro de produto sem estoque disponível, mas posteriormente impede sua utilização no fluxo de compra, gerando comportamento incoerente entre os módulos de catálogo e carrinho.

Reprodução:

1. Acessar o endpoint POST /produto;
2. Enviar requisição com:

```
{
  "nome": "  ",
  "preco": 1,
  "descricao": "  ",
  "quantidade": 0
}
```

3. Confirmar que o produto é criado com sucesso (201 Created);
4. Acessar o endpoint de adicionar ao carrinho (ex: POST /carrinho);
5. Enviar requisição contendo o ID do produto criado;
6. Observar que o sistema bloqueia a adição informando estoque insuficiente ou indisponível.

Evidências:

- Produto criado com sucesso com quantidade = 0;
- Produto visível na listagem de produtos;
- Mensagem de erro ao tentar adicionar ao carrinho;
- Status code retornado indicando falha (ex: 400 ou 409);
- Inconsistência entre regra de cadastro e regra de compra.

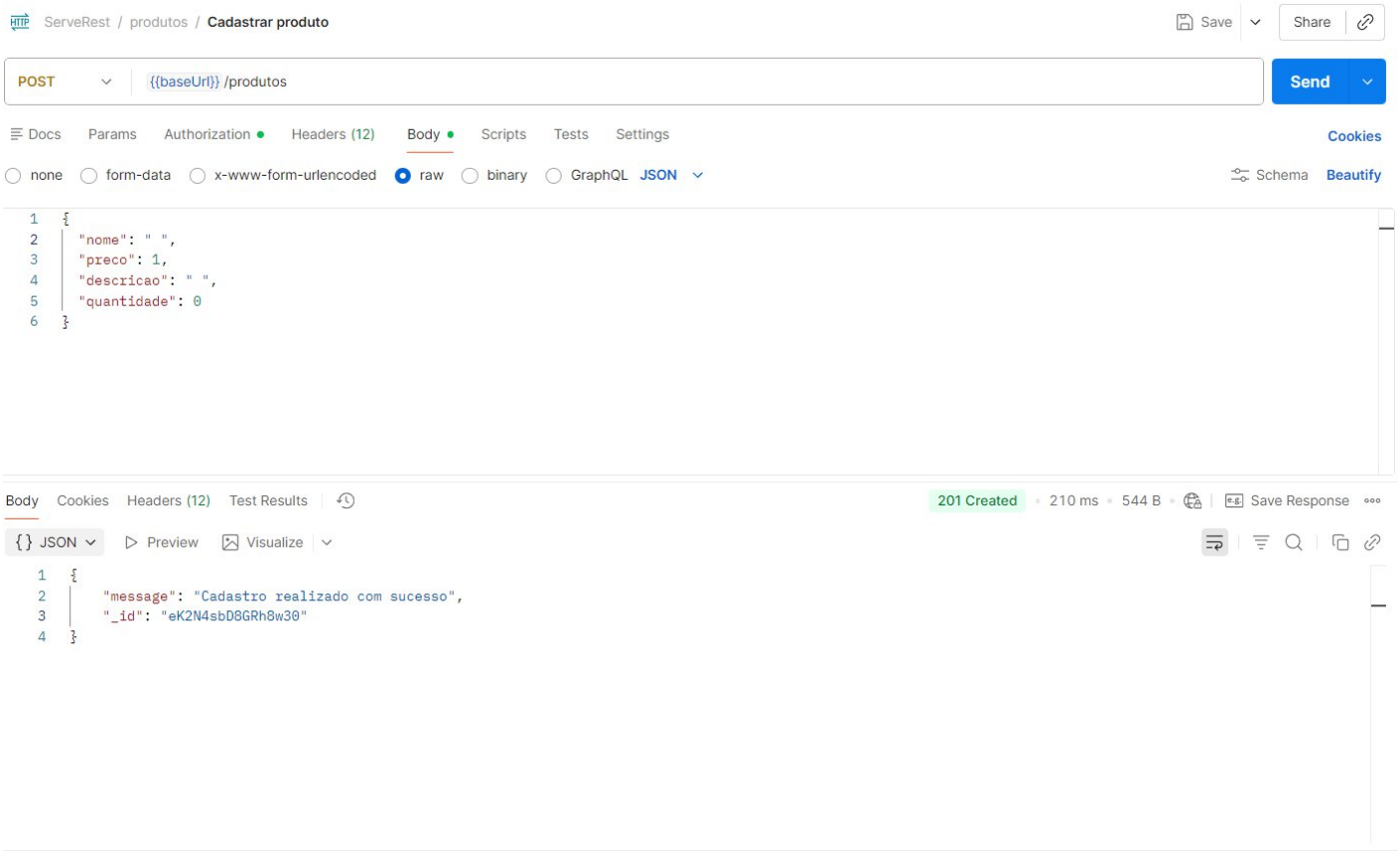

Figura 01 – Produto criado com estoque zero

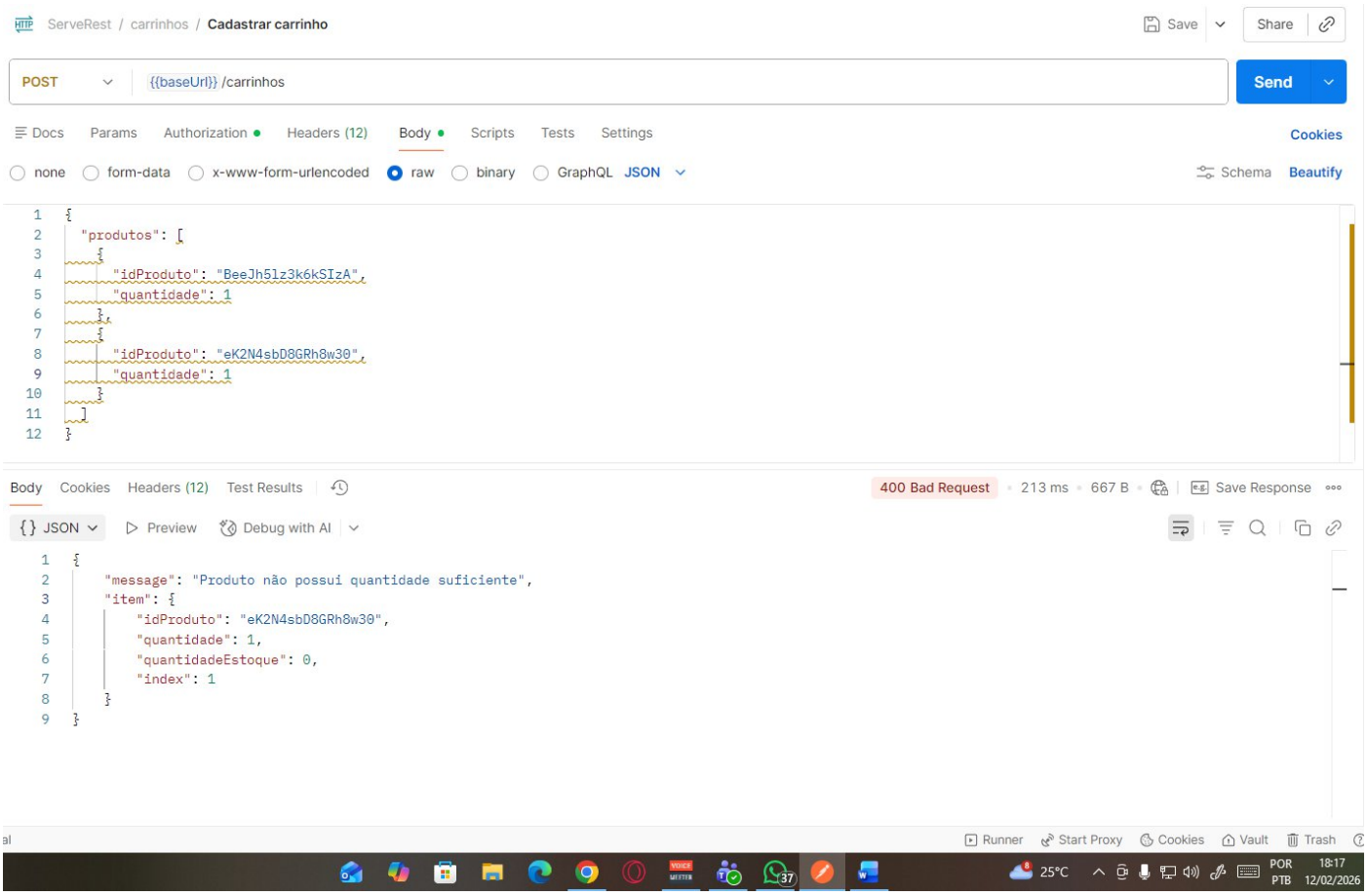

Figura 02 – Bloqueio ao adicionar ao carrinho

Comportamento Esperado:

Uma das seguintes regras deveria ser aplicada de forma consistente:

Opção A – Regra Restritiva

- O sistema não deve permitir cadastro de produto com quantidade menor ou igual a zero;
- Retornar 400 Bad Request no cadastro.

Ou

Opção B – Regra Permissiva Controlada

- Permitir cadastro com quantidade zero;
- Exibir claramente como "Produto indisponível";
- Impedir exibição para compra;
- Garantir coerência entre catálogo e carrinho.

Atualmente, há inconsistência entre os módulos.

Ambiente:

Plataforma: API REST
Ferramenta: Postman
Sistema Operacional: Windows 11
Data do teste: 08/01/2026

Prioridade:

Média

Justificativa:

O problema não compromete segurança diretamente, mas causa inconsistência funcional e pode gerar:

- Confusão para administradores;
- Cadastro de produtos inutilizáveis;
- Inconsistência de regras de negócio;
- Falhas na lógica de estoque.

Bug 05: API permite cadastro de produto com nome inválido (null e caracteres especiais) e permite adicioná-lo ao carrinho

Descrição:

Durante os testes do endpoint POST /produto, foi identificado que a API permite o cadastro de produtos com nome inválido, incluindo:

- Valor null;
- Strings compostas apenas por caracteres especiais, como:
- ()()()()()

Mesmo com nome inválido ou sem significado semântico, o sistema aceita o cadastro (201 Created) e persiste o produto no banco de dados.

Além disso, o produto pode ser adicionado normalmente ao carrinho, indicando ausência de validação tanto no cadastro quanto no fluxo de compra.

Esse comportamento demonstra falha na validação de dados obrigatórios e ausência de regras mínimas de integridade textual.

Reprodução:

1. Acessar o endpoint POST /produto;
2. Enviar requisição com:

Caso 1 – Nome null:

```
{
  "nome": null,
  "preco": 1,
  "descricao": " ",
  "quantidade": 1
}
```

Caso 2 – Nome com caracteres inválidos:

{
"nome": "()()()()()",
"preco": 1,
"descricao": " ",
"quantidade": 1
}

3. Confirmar que a API retorna sucesso (201 Created);
4. Verificar que o produto aparece na listagem;
5. Acessar endpoint de carrinho (ex: POST /carrinho);
6. Adicionar o produto criado;
7. Observar que o sistema permite a adição normalmente.

Evidências:

- Status code: 201 Created;
- Produto persistido com nome inválido;
- Produto visível via GET /produto;
- Produto adicionado com sucesso ao carrinho;
- Ausência de validação de formato mínimo do campo nome.

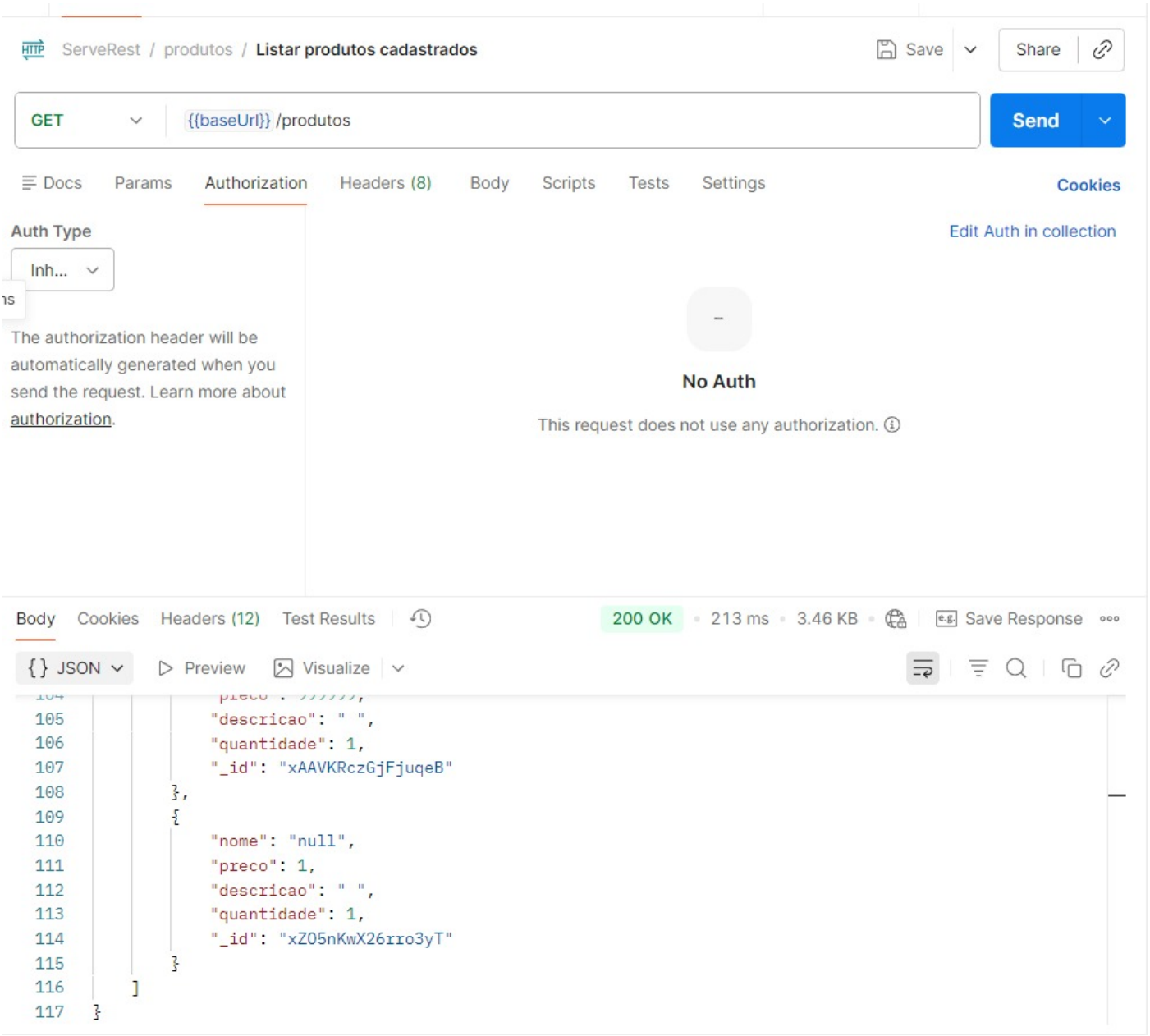

Figura 01 – Cadastro com nome null

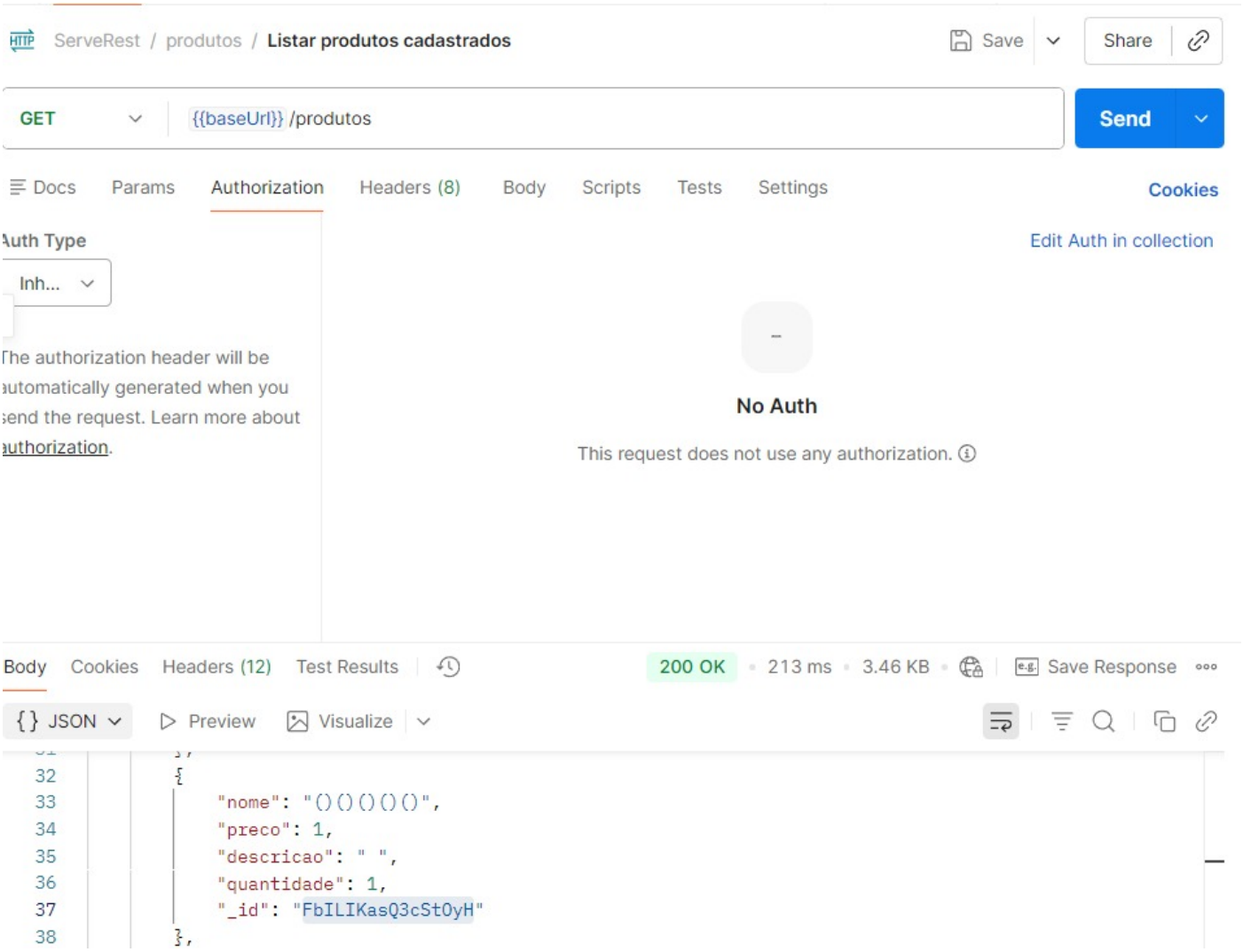

Figura 02 – Cadastro com caracteres inválidos

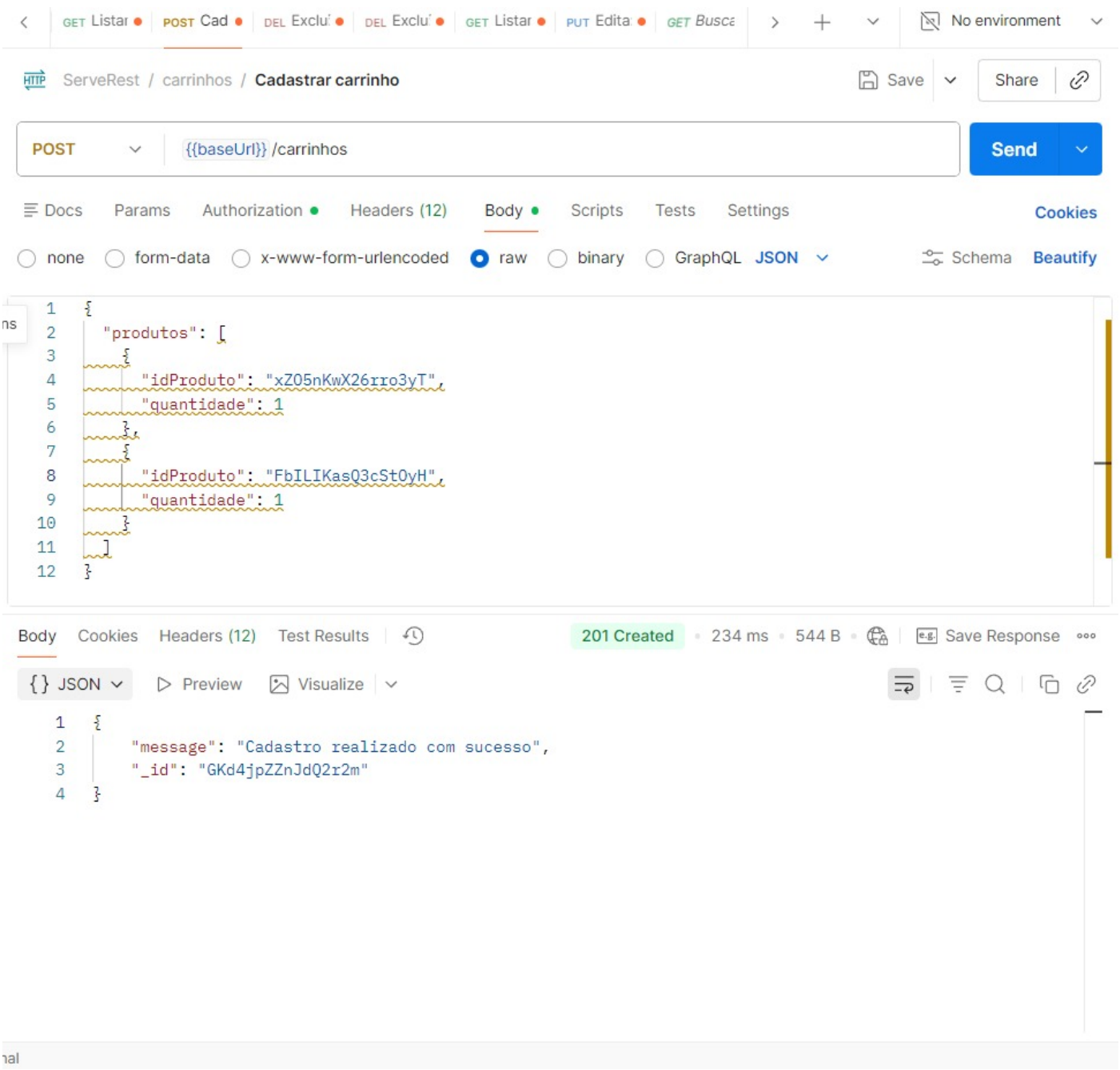

Figura 03 – Produto inválido adicionado ao carrinho

Comportamento Esperado:

A API deve:

- Bloquear cadastro com null;
- Bloquear nomes compostos apenas por caracteres especiais;
- Validar tamanho mínimo;
- Validar padrão aceitável (ex: regex alfanumérica);
- Retornar 400 Bad Request com mensagem clara:
    - "Nome é obrigatório"
    - "Nome inválido"

Ambiente:

Plataforma: API REST
Ferramenta: Postman
Sistema Operacional: Windows 11
Data do teste: 08/01/2026

Prioridade:

Alta

Justificativa:

O problema compromete a integridade da base de produtos e pode gerar:

- Dados inconsistentes no catálogo;
- Problemas de exibição no frontend;
- Impacto em relatórios e buscas;
- Risco de exploração por entradas maliciosas (input injection).

A ausência de validação de entrada é uma falha estrutural que pode indicar vulnerabilidades mais graves.

12. Critérios de Entrada, Saída e Definição de Pronto

12.1 Critérios de Entrada

- API disponível e acessível.
- Documentação da ServeRest consultável.
- Ferramenta de testes configurada (Postman).

12.2 Critérios de Saída

- Todos os endpoints mapeados testados.
- Cenários críticos executados.
- Não existência de falhas bloqueadoras abertas.

12.3 Definição de Pronto (Definition of Done)

- Endpoints atendem ao contrato definido.
- Respostas retornam status HTTP corretos.
- Tratamento adequado de erros e exceções.

13. Evidências – Exemplos de Requisições (Modelo)

As evidências a seguir apresentam **modelos de requisições** baseadas na documentação Swagger oficial da ServeRest API. As requisições são ilustrativas e têm como objetivo demonstrar o preparo dos cenários de teste, não sendo necessária sua execução.

13.1 Login

13.1.1Login – POST /login

Cenário Positivo – Credenciais válidas

POST /login
Content-Type: application/json
{ **"email": "usuario@email.com",** **"password": "123456"** **}**

Resposta esperada: 200 OK + token JWT

Cenário Negativo – Senha inválida

POST /login
Content-Type: application/json
{ **"email": "usuario@email.com",** **"password": "123456"** **}**

Resposta esperada: 401 Unauthorized

Validação de Contrato

- Campo email obrigatório (string)
- Campo password obrigatório (string)
- Resposta deve conter apenas o atributo authorization

13.2 Usuário

13.2.1Criar Usuário – POST /usuário

Criar Usuário – Cenário Positivo

POST /usuario
Content-Type: application/json
{ **"nome": "Usuário Teste",** **"email": "teste@email.com",** **"password": "123456",** **"administrador": "false"**

```
}
```

Resposta esperada: 201 Created

Criar Usuário – Cenário Negativo (Usuário duplicado)

```
POST /usuario
Content-Type: application/json
{
"nome": "Usuário Teste",
"email": "teste@email.com",
"password": "123456",
"administrador": "false"
}
```

Resposta esperada: 400 Bad Request

13.2.2 Listar Usuários – GET /usuario

```
GET /usuario
```

Resposta esperada:200 OK

Cenário Negativo – Método inválido

```
POST /usuario
```

Resposta esperada: 405 Method Not Allowed

Validação de Contrato

- Retorno deve ser uma lista de usuários
- Cada usuário deve conter: id, nome, email, password, administrador
- Estrutura da resposta deve ser JSON

13.2.3 Consultar Usuário por ID – GET /usuario/{id}

Cenário Positivo – ID existente

```
GET /usuario/{id}
```

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

```
GET /usuario/id_invalido
```

Resposta esperada: 400 Bad Request

Validação de Contrato

- Parâmetro id é obrigatório
- Retorno deve conter os dados completos do usuário
- Estrutura JSON padronizada

13.2.4 Atualizar Usuário – PUT /usuario/{id}

Cenário Positivo – Atualização válida

PUT /usuario/{id}
Content-Type: application/json
{ **"nome": "Usuário Atualizado",** **"email": "novo@email.com",** **"password": "123456",** **"administrador": "false"** **}**

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

PUT /usuario/id_invalido

Resposta esperada: 400 Bad Request

Validação de Contrato

- id obrigatório
- Campos obrigatórios: nome, email, password, administrador
- Email deve ser válido
- administrador aceita apenas "true" ou "false"

13.2.5 Remover Usuário – DELETE /usuario/{id}

Cenário Positivo – Exclusão válida

DELETE /usuario/{id}

Resposta esperada: 200 OK

Cenário Negativo – Usuário inexistente

DELETE /usuario/id_invalido

Resposta esperada: 400 Bad Request

Validação de Contrato

- id obrigatório
- Usuário deve existir para ser removido
- Retorno deve confirmar a exclusão

13.3 Produtos

13.3.1 Criar Produto – POST /produto

Cenário Positivo – Produto válido

POST /produto
Authorization: Bearer <token_jwt>
Content-Type: application/json
{ **"nome": "Produto Teste",** **"preco": 100,** **"descricao": "Produto para testes",** **"quantidade": 10** **}**

Resposta esperada: 201 Created

Cenário Negativo – Sem autenticação

POST /produto

Resposta esperada: 401 Unauthorized

Validação de Contrato

- Token JWT obrigatório
- Campos obrigatórios: nome, preco, descricao, quantidade
- preco e quantidade devem ser numéricos e positivos

13.3.2 Listar Produtos – GET /produto

Cenário Positivo – Listagem de produtos

GET /produto

Resposta esperada: 200 OK

Cenário Negativo – Endpoint inválido

GET /produtox

Resposta esperada: 404 Not Found

Validação de Contrato

- Retorno deve ser uma lista de produtos
- Estrutura JSON consistente

13.3.3 Consultar Produto por ID – GET /produto/{id}

Cenário Positivo – Produto existente

GET /produto/{id}

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

GET /produto/id_invalido

Resposta esperada: 400 Bad Request

Validação de Contrato

- Parâmetro id obrigatório
- Retorno deve conter os dados completos do produto

13.3.4 Atualizar Produto – PUT /produto/{id}

Cenário Positivo – Atualização válida

PUT /produto/{id}
Authorization: Bearer <token_jwt>

Content-Type: application/json

```
{
"nome": "Produto Atualizado",
"preco": 120,
"descricao": "Descrição atualizada",
"quantidade": 5
}
```

Resposta esperada: 200 OK

Cenário Negativo – Produto em carrinho

PUT /produto/{id}

Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Validação de Contrato

- Token JWT obrigatório
- Produto não pode estar vinculado a um carrinho
- Campos devem respeitar os tipos definidos

13.3.5 Remover Produto – DELETE /produto/{id}

Cenário Positivo – Exclusão válida

DELETE

/produto/{id}

Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – Produto inexistente

DELETE /produto/id_invalido

Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Validação de Contrato

- Token JWT obrigatório
- Produto deve existir para exclusão
- Retorno deve confirmar a remoção

13.4 Carrinho

13.4.1 Criar Carrinho – POST /carrinho

POST /carrinho
Authorization: Bearer <token_jwt>
Content-Type: application/json
{ **"produtos": [** **{** **"idProduto": "id_produto_exemplo",** **"quantidade": 1** **}** **]** **}**

Resposta esperada: 201 Created

Cenário Negativo – Produto inexistente

POST /carrinho
Authorization: Bearer <token_jwt>
Content-Type: application/json
{ **"produtos": [** **{** **"idProduto": "id_invalido",** **"quantidade": 1** **}** **]** **}**

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

POST /carrinho
Content-Type: application/json

```
{
  "produtos": [
    {
      "idProduto": "id_produto_exemplo",
      "quantidade": 1
    }
  ]
}
```

Resposta esperada: 401 Unauthorized

Validação de Contrato

- Token JWT é obrigatório
- Campo produtos deve ser uma lista não vazia
- Campo idProduto deve ser um identificador válido
- Campo quantidade deve ser numérico e maior que zero
- A resposta deve conter mensagem de sucesso ou erro conforme o status HTTP

13.4.2 Consultar Carrinho do Usuário – GET /carrinho

Cenário Positivo – Carrinho existente

GET /carrinho
Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – Usuário sem carrinho ativo

GET /carrinho
Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

GET /carrinho

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **Token JWT é obrigatório**
- **Resposta deve conter os dados do carrinho**
- **Estrutura do JSON deve seguir o padrão definido no Swagger**

13.4.3 Consultar Carrinho por ID – GET /carrinho/{id}

Cenário Positivo – Carrinho existente

GET /carrinho/{id}
Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – ID inexistente

GET /carrinho/id_invalido
Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token inválido

GET /carrinho/{id}

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **ID deve ser válido**
- **Token JWT obrigatório**
- **Retorno deve conter lista de produtos e totais do carrinho**

13.4.4 Finalizar Compra – DELETE /carrinho/concluir-compra

Cenário Positivo – Carrinho válido

DELETE /carrinho/concluir-compra

Authorization: Bearer <token_jwt>

Resposta esperada: 200 OK

Cenário Negativo – Usuário sem carrinho ativo

DELETE /carrinho/concluir-compra

Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

DELETE /carrinho/concluir-compra

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **Carrinho deve existir e estar ativo**
- **Token JWT obrigatório**
- **Resposta deve conter mensagem de confirmação**

13.4.5 Cancelar Compra – DELETE /carrinho/cancelar-compra

Cenário Positivo – Carrinho válido

DELETE /carrinho/cancelar-compra

Authorization: Bearer <token_jwt>

Resposta esperada: 200 Sucesso

Cenário Negativo – Usuário sem carrinho ativo

DELETE /carrinho/cancelar-compra

Authorization: Bearer <token_jwt>

Resposta esperada: 400 Bad Request

Cenário Negativo – Token ausente ou inválido

DELETE /carrinho/cancelar-compra

Resposta esperada: 401 Unauthorized

Validação de Contrato

- **Token JWT obrigatório**
- **Carrinho deve estar ativo**
- **Resposta deve confirmar o cancelamento da operação**

14. Gerenciamento de Defeitos

Os defeitos serão registrados contendo:

- Endpoint afetado
- Método HTTP
- Payload enviado
- Resultado esperado vs obtido
- Evidências da resposta

15. Cronograma de Execução

- Dia 1: Login e Usuários
- Dia 2: Produtos
- Dia 3: Carrinho e retestes

16. Conclusão Técnica

Com base nos testes planejados, espera-se validar que a ServeRest API apresenta comportamento previsível, robusto e aderente aos contratos REST, sendo adequada para uso em ambientes de estudo, automação e integração de sistemas.