Execução de Testes

Por: Daniela Siana Pabis | Squad 01: Unit to Unity | Data: 13/02/2026

IDENTIFICAÇÃO DO PROJETO

Nome do Projeto: Execução de Testes Funcionais e Exploratórios da API ServeRest

API Testada: ServeRest Compass UOL (https://compassuol.serverest.dev/)

Versão do documento: 1.0

Data de Criação: 09/02/2026

Data da Última Atualização: 12/02/2026

Responsável pela Elaboração: Daniela Siana Pabis

Responsável pela Aprovação: Avaliadores do programa de bolsas

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

ESCOPO

Este documento apresenta a execução dos testes funcionais e exploratórios definidos na versão 1.0 do Plano de Testes, entregue na semana anterior.

Foram realizados testes com foco em validação de entradas (Input Validation) e em fluxo de estados e regras de negócio (State Transition and Business Rules Testing) da API ServeRest.

O escopo contempla os principais recursos da API:

Usuários: validação de campos obrigatórios, unicidade de e-mail, criação, atualização e exclusão de registros, além do impacto da exclusão de usuários vinculados a carrinhos.

Autenticação: validação de credenciais, geração e uso de token e impacto do estado de autenticação nos fluxos dependentes.

Produtos: operações CRUD com foco na validação de payloads, permissões de acesso e impacto do estado do produto nos carrinhos.

Carrinhos: criação, finalização e cancelamento, avaliando dependências entre usuário, produtos e estoque, bem como o comportamento da API em fluxos inválidos.

Não fazem parte do escopo:

- Testes de performance ou carga;
- Testes de segurança avançados (pentest).

Ambiente de Testes:

- API Base URL: ServeRest;
- Ferramenta: Postman;
- Sistema Operacional: Windows 11.

OBJETIVOS

Geral

Validar, por meio da execução dos testes previamente planejados no Plano de Testes (versão 1.0), a qualidade funcional da API ServeRest, utilizando testes funcionais e exploratórios baseados em validação de entrada (Input Validation) e fluxo de estado e regras de negócio (State Transition and Business Rules Testing), assegurando que os endpoints tratem corretamente dados válidos e inválidos e mantenham a integridade da API ao longo das operações.

Específicos

- Verificar se a API valida corretamente os dados de entrada, rejeitando campos ausentes, inválidos ou fora dos limites esperados;
- Avaliar o comportamento da API diante de fluxos sequenciais, como criação, atualização e exclusão de recursos dependentes;
- Identificar falhas na aplicação das regras de negócio, como duplicidade de registros, restrições de exclusão e controle de estoque;
- Avaliar a consistência das respostas, incluindo status HTTP, mensagens de erro e estrutura dos payloads;
- Explorar cenários fora do fluxo esperado, analisando estados inválidos do sistema.

ESTRATÉGIA DE TESTES

Testes Funcionais

Os testes funcionais foram executados com base em cenários previamente definidos, cobrindo os fluxos principais da API, como cadastro de usuários, autenticação, gerenciamento de produtos e carrinhos.

Testes Exploratórios

Técnicas utilizadas: Input Validation e State Transition and Business Rules Testing.

A estratégia de testes exploratórios adotada consiste na execução de testes exploratórios baseados na validação de entrada (Input Validation) e no fluxo de estado e regras de negócio (State Transition and Business Rules Testing).

A exploração foi conduzida a partir de:

- Variações nos dados de entrada, incluindo ausência de campos obrigatórios, tipos inválidos e valores fora dos limites esperados;
- Execução de fluxos completos e incompletos, observando o impacto de cada operação no estado do sistema;
- Essa abordagem permite identificar falhas funcionais, inconsistências de negócio e comportamentos inesperados que não seriam facilmente detectados apenas por testes funcionais isolados.

MAPEAMENTO DE ENDPOINTS

Recurso	Descrição	Endpoint	Método	Regras	Respostas
Login	Autentica usuário e retorna token	/login	POST	Autenticação para acesso a rotas privadas	200, 401
Usuários	Cadastra um novo usuário	/usuarios	POST	Não é permitido email duplicado	201, 400
Usuários	Lista usuários cadastrados	/usuarios	GET	Permite filtragem de resultados	200
Usuários	Buscar usuário por ID	/usuarios/{_id}	GET	Localiza registro específico via identificador	200, 400
Usuários	Editar ou criar usuário	/usuarios/{_id}	PUT	Operação de UPSERT; valida e-mail único	200, 201, 400
Usuários	Exclui usuário	/usuarios/{_id}	DELETE	Proibida a exclusão de usuários com carrinho ativo	200, 400
Produtos	Cadastra novo produto	/produtos	POST	Restrito a administradores; nome deve ser único	201, 400, 401, 403
Produtos	Lista produtos cadastrados	/produtos	GET	Permite busca por atributos do produto	200, 400
Produtos	Retorna dados de um produto	/produtos/{_id}	GET	Localiza produto específico via identificador	200, 400
Produtos	Edita ou cria produto	/produtos/{_id}	PUT	Restrito a administradores; realiza UPSERT	200, 201, 400, 401, 403
Produtos	Exclui produto	/produtos/{_id}	DELETE	Restrito a administradores; proibido se houver carrinho vinculado	200, 400, 401, 403
Carrinhos	Cria carrinho	/carrinhos	POST	Apenas 1 carrinho por usuário; reduz estoque automaticamente	201, 400, 401

Carrinhos	Lista carrinhos cadastrados	/carrinhos	GET	Exibe todos os carrinhos ativos na base	200
Carrinhos	Buscar carrinho por ID	/carrinhos/{_id}	GET	Localiza carrinho específico via identificador	200, 400
Carrinhos	Finalizar Compra	/carrinhos/concluir-compra	DELETE	Exclui o carrinho vinculado ao token do usuário	200, 401
Carrinhos	Cancelar Carrinho	/carrinhos/cancelar-compra	DELETE	Exclui o carrinho e devolve os itens ao estoque	200, 401

RISCOS E PRIORIDADES

Risco	Descrição	Impacto	Mitigação
Inconsistência no mecanismo de autenticação	O processo de emissão e validação de tokens pode aceitar credenciais inválidas ou rejeitar acessos legítimos	Alto	Executar testes de autenticação com tokens válidos, inválidos e expirados
Violação de controle de acesso	Regras de autorização podem não ser aplicadas corretamente aos endpoints restritos	Alto	Validar acesso a rotas protegidas com diferentes perfis de usuário
Falhas no saneamento de dados de entrada	Parâmetros podem aceitar tipos, formatos ou valores fora dos limites definidos	Alto	Aplicar Input Validation, incluindo testes de valores limite e tipos inválidos
Divergência de contrato da API	Estrutura de requests ou responses pode não seguir o contrato especificado	Alto	Validar schemas JSON conforme documentação da API
Estados inválidos entre operações	Sequências de operações podem gerar estados inconsistentes entre recursos relacionados	Alto	Executar State Transition and Business Rules Testing
Padronização inadequada de respostas de erro	Códigos HTTP ou payloads de erro podem ser inconsistentes entre endpoints	Médio	Validar status HTTP, mensagens e estrutura das respostas de erro

CENÁRIOS DE TESTES FUNCIONAIS

ID	Cenário	Endpoint	Tipo	Prior.	Status
CT01	Cadastrar usuário com dados válidos	POST /usuarios	Positivo	Alta	Passou
CT02	Cadastrar usuário com e-mail já existente	POST /usuarios	Negativo	Alta	Passou
CT03	Cadastrar usuário com credenciais inválidas	POST /usuarios	Negativo	Alta	Não passou
CT04	Validar schema de request e response do cadastro de usuário	POST /usuarios	Contrato	Média	Passou
CT05	Validar schema de request e response da busca de usuários	GET /usuarios	Contrato	Média	Passou
CT06	Realizar login com credenciais inválidas	POST /login	Negativo	Alta	Não passou
CT07	Validar schema da resposta de login (token)	POST /login	Contrato	Alta	Passou
CT08	Criar produto com usuário administrador	POST /produtos	Positivo	Alta	Passou
CT09	Criar produto com usuário não administrador	POST /produtos	Negativo	Alta	Passou
CT10	Criar produto com dados inválidos	POST /produtos	Negativo	Média	Não passou
CT11	Validar schema de request e response do produto	POST /produtos	Contrato	Média	Passou
CT12	Listar produtos cadastrados	GET /produtos	Positivo	Média	Passou
CT13	Buscar produto por ID válido	GET /produtos/{id}	Positivo	Média	Passou
CT14	Atualizar produto com usuário administrador	PUT /produtos/{id}	Positivo	Média	Passou
CT15	Excluir produto sem vínculo com carrinho	DELETE /produtos/{id}	Positivo	Média	Passou
CT16	Criar carrinho com produtos válidos	POST /carrinhos	Positivo	Alta	Passou
CT17	Criar carrinho sem autenticação	POST /carrinhos	Negativo	Alta	Passou
CT18	Criar carrinho com produto inexistente	POST /carrinhos	Negativo	Alta	Passou
CT19	Validar schema de request e response da criação de carrinho	POST /carrinhos	Contrato	Média	Passou
CT20	Listar carrinhos cadastrados	GET /carrinhos	Positivo	Média	Passou
CT21	Buscar carrinho por ID válido	GET /carrinhos/{id}	Positivo	Média	Passou
CT22	Finalizar carrinho com sucesso	DELETE /carrinhos/concluir-compra	Positivo	Alta	Passou
CT23	Finalizar carrinho inexistente	DELETE /carrinhos/concluir-compra	Negativo	Média	Passou
CT24	Validar schema da resposta de finalização de carrinho	DELETE /carrinhos/concluir-compra	Contrato	Média	Passou
CT25	Cancelar carrinho com sucesso	DELETE /carrinhos/cancelar-compra	Positivo	Média	Passou

CT26	Listar usuários cadastrados	GET /usuarios	Positivo	Média	Passou
CT27	Buscar usuário por ID válido	GET /usuarios/{id}	Positivo	Média	Passou
CT28	Realizar login com credenciais válidas	POST /login	Positivo	Alta	Passou

CENÁRIOS DE TESTES EXPLORATÓRIOS

Exploratory Testing based on Input Validation

ID	Cenário Exploratório	Endpoints Alvo	Prioridade	Foco da Exploração	Status
TE01	Submissão de payloads com tipos inválidos, campos ausentes e campos extras	POST /usuarios, POST /produtos	Alta	Validação de tipos, obrigatoriedade e saneamento de dados	Não passou
TE02	Envio de valores extremos e limites de tamanho nos campos de entrada	POST /usuarios, POST /produtos	Média	Boundary values e restrições de tamanho	Não passou
TE03	Uso de token ausente, malformado ou expirado em requisições autenticadas	POST /produtos, POST /carrinhos	Alta	Validação do token e consistência de respostas de erro	Passou

Exploratory Testing based on State Transition and Business Rules

ID	Cenário Exploratório	Endpoints Alvo	Prioridade	Foco da Exploração	Status
TE04	Execução de operações fora da ordem esperada do fluxo	POST /carrinhos, DELETE /carrinhos/concluir-compra	Alta	Transições de estado inválidas	Passou
TE05	Repetição de operações que deveriam ser bloqueadas ou idempotentes	POST /carrinhos, DELETE /carrinhos/concluir-compra DELETE /produtos	Média	Regras de negócio e proteção contra duplicidade	Não passou
TE06	Uso do mesmo token em operações sequenciais com perfis diferentes	POST /produtos, PUT /produtos/{id}	Alta	Autorização baseada em perfil e regras de acesso	Passou

REGISTRO DE BUGS E ISSUES

REGISTRO 01:

Caso de teste:

CT03 - Cadastrar usuário com credenciais inválidas.

Tipo de teste:

Teste Funcional

Título:

Validação permite que os campos obrigatórios ‘nome’ e ‘password’ sejam preenchidos por strings com espaços em branco ou muito longas, além do campo ‘email’ inválido.

Descrição:

A API aceita strings compostas apenas por espaços em branco (" ") nos campos ‘nome’ e ‘password’, indicando ausência de sanitização (trim()) e validação NotBlank. O campo ‘email’ não possui limite de caracteres nem validação de formato de domínio, permitindo endereços irreais e extremamente longos. Assim, o cadastro retorna HTTP 201 Created mesmo com dados inválidos.

Passos para reprodução:

1. Acessar a collection da API via Postman
2. Acessar a pasta ‘usuários’
3. Acessar a request ‘POST Cadastrar usuário’
4. Clicar em ‘Body’ seguido de ‘raw’
5. Preencher os campos de ‘nome’ e ‘password’ com strings em branco e ‘email’ com valores irreais
6. Enviar request em ‘Send’
7. Observar a aprovação do cadastro

Resultado esperado:

POST /usuarios deve retornar HTTP 400 Bad Request quando:

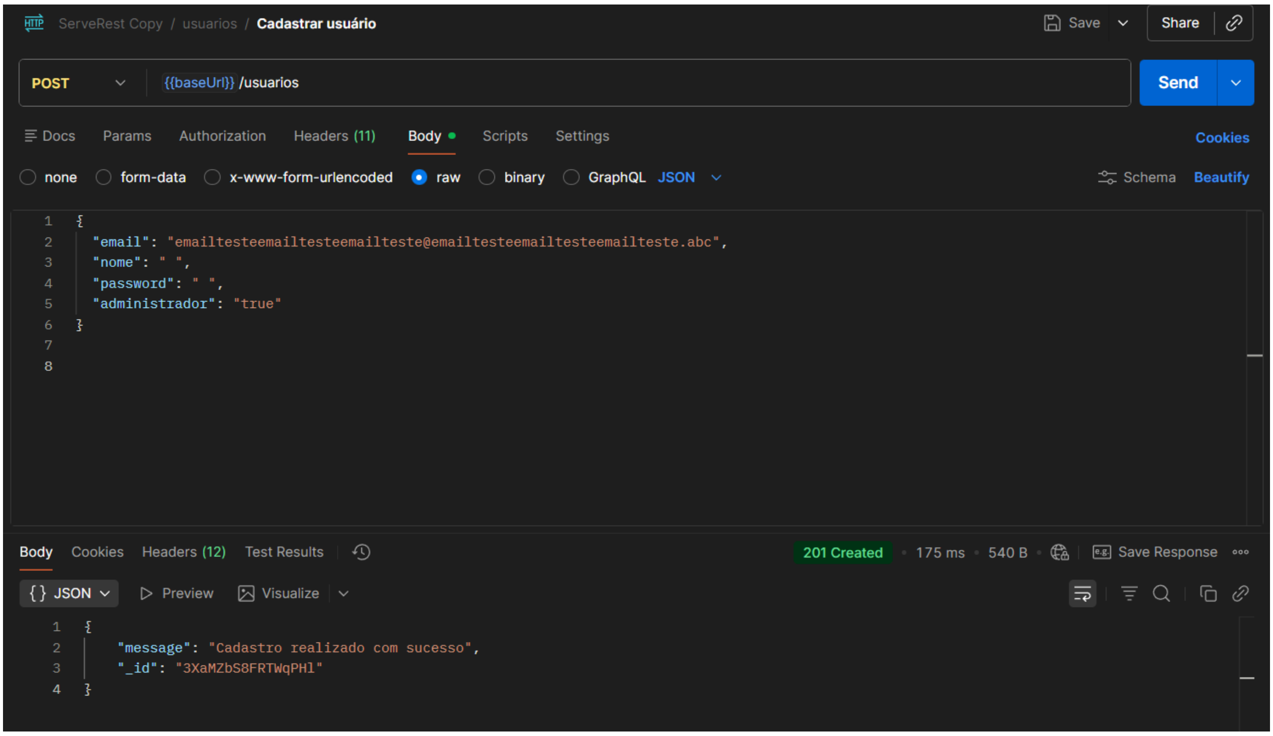
- Campos ‘nome’ ou ‘password’ contiverem apenas espaços após trim();
- Campo ‘email’ exceder limite (ex: 254 caracteres) ou não passar a validação de formato RFC 5322;
- E-mail possuir domínio inválido ou TLD inexistente.

A resposta deve especificar qual campo falhou e o motivo. POST /login deve retornar HTTP 400 Bad Request se a senha for inválida no momento da autenticação, impedindo um posterior login com credenciais malformadas/inválidas.

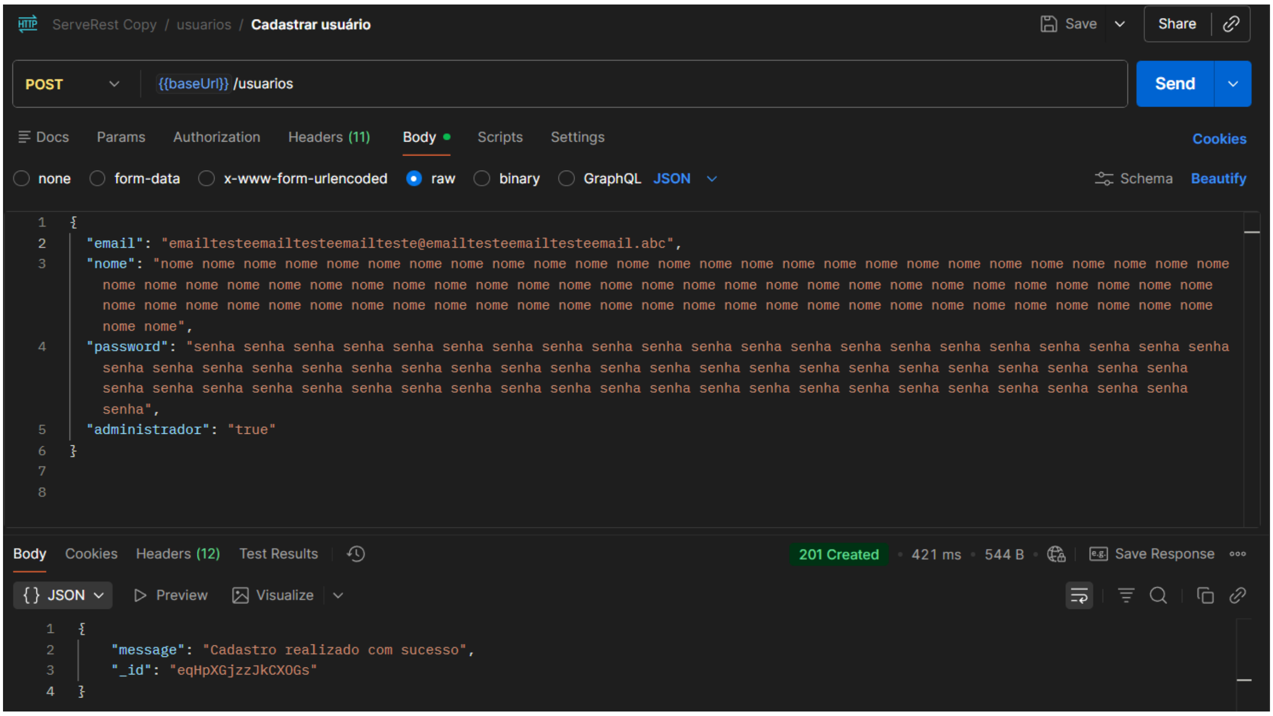
Resultado obtido:

POST /usuarios retornou HTTP 201 Created com nome=" ", password=" " e email inválido. A API não aplicou trim(), não validou comprimento de e-mail, não verificou formato de domínio, e armazenou dados tecnicamente errôneos como registros válidos.

Evidências:



Cadastro de usuário com e-mail inválido e credenciais em branco



Cadastro de usuário com nome e senha muito longas

REGISTRO 02:

Caso de teste:

CT06 - Realizar login com credenciais inválidas.

Título:

Validação permite login com senha composta apenas por espaços em branco ou demasiadamente longa e e-mail com credenciais irreais.

Descrição:

O endpoint POST /login aceita senhas em branco ou muito longas, além de e-mails mal formatados (já previamente cadastrados), retornando HTTP 200 OK com token válido. A API não implementa trim(), validação NotBlank, nem requisitos mínimos de complexidade de senha e e-mail (comprimento, caracteres especiais, etc.). Assim, não há proteção contra entradas triviais ou vazias.

Passos para reprodução:

1. Acessar a collection da API via Postman
2. Acessar a pasta ‘usuários’
3. Realizar cadastro de usuário com e-mail potencialmente inválido e campos de nome e password muito longos ou em branco
4. Acessar a pasta ‘login’
5. Acessar a request ‘POST Realizar login’
6. Clicar em ‘Body’ seguido de ‘raw’
7. Preencher os campos ‘email’ e ‘password’
8. Enviar request em ‘Send’
9. Observar a aprovação do login

Resultado esperado:

O endpoint POST /usuarios deve retornar HTTP 400 – Bad Request quando:

- O campo password, após aplicação de trim(), estiver vazio;
- A senha não atender aos requisitos mínimos de segurança (ex: tamanho mínimo, presença de caracteres obrigatórios, etc.);
- O campo email estiver em formato inválido ou fora do padrão esperado (validação de formato, tamanho e estrutura).

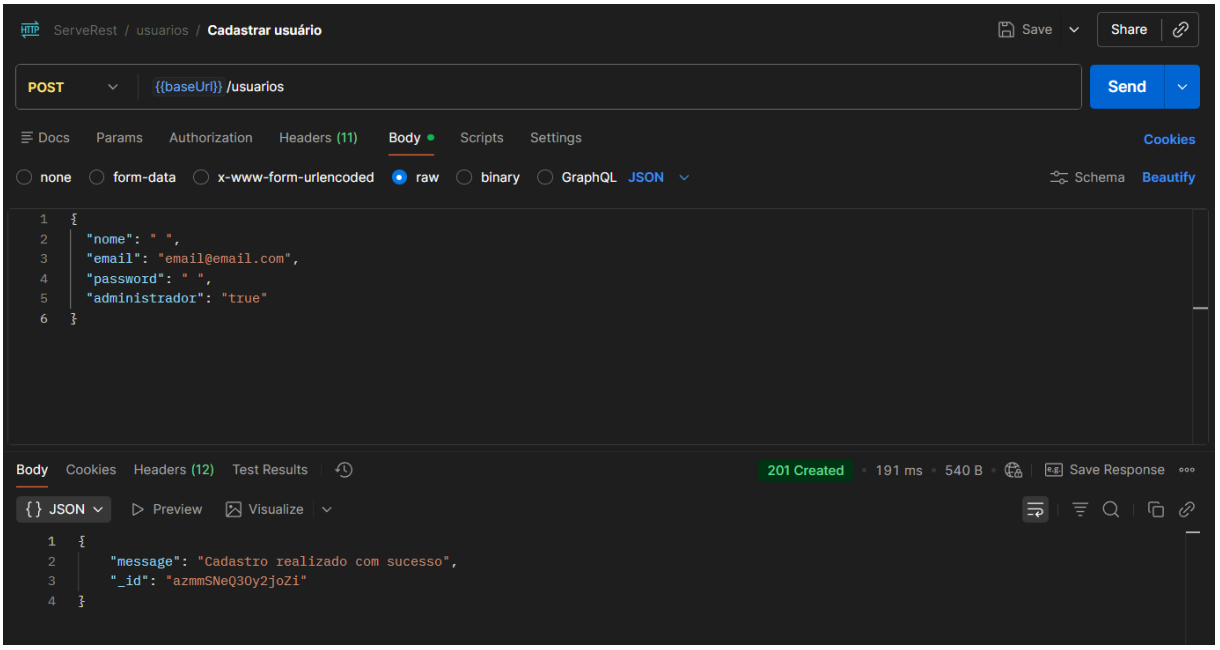
A resposta deve indicar claramente quais critérios de validação não foram atendidos. O endpoint POST /login deve retornar HTTP 401 – Unauthorized quando as credenciais forem inválidas ou não atenderem aos critérios de validação. Ambos os endpoints devem implementar mecanismos de proteção, como validação adequada de entrada para mitigar riscos como brute force e autenticação com dados inválidos.

Resultado obtido:

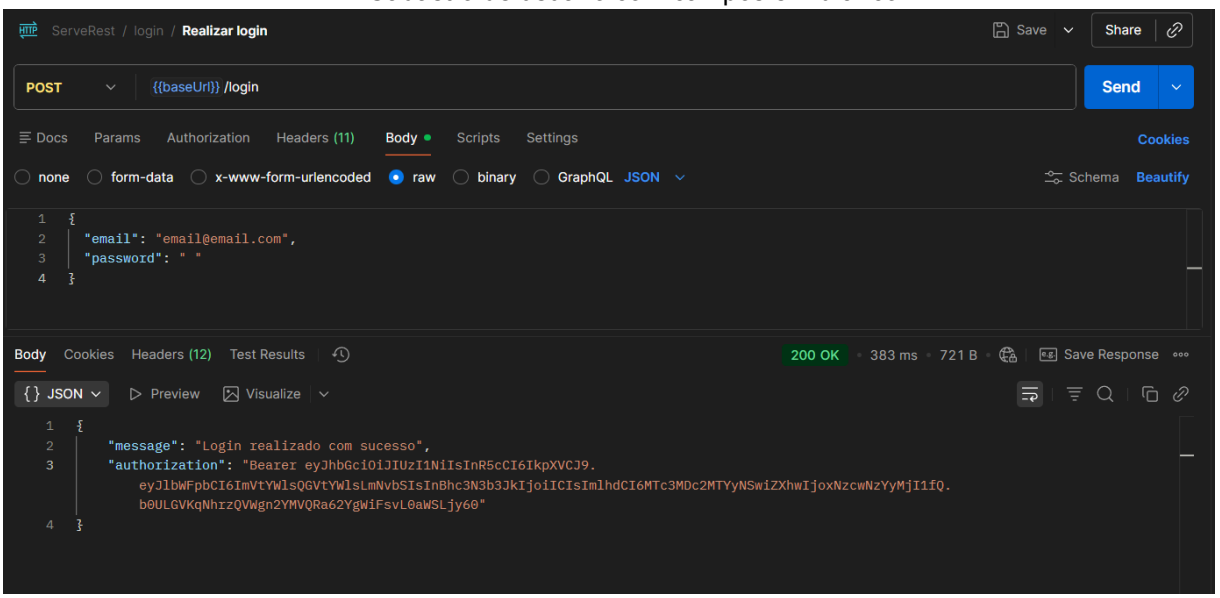
O endpoint POST /usuarios aceitou password=" " (composta apenas por espaço em branco) e e-mail potencialmente inválido, retornando HTTP 201 – Created, e criando o registro normalmente.

O endpoint POST /login, utilizando as mesmas credenciais, retornou HTTP 200 – OK, gerando token Bearer válido. A API não aplicou trim() no campo senha, não validou complexidade mínima, não bloqueou senha vazia após normalização e não validou adequadamente o formato do e-mail. Como consequência, permitiu autenticação completa com credenciais tecnicamente inválidas, gerando token funcional e acesso a endpoints protegidos.

Evidências:



Cadastro de usuário com campos em branco



Login do usuário com senha em branco

REGISTRO 03:

Caso de teste:

CT10 - Criar produtos com dados inválidos.

Título:

Criação de produto permite dados inválidos e inconsistentes com a documentação.

Descrição:

POST /produtos aceita strings ilimitadamente longas nos campos ‘nome’ e ‘descricao’ (testado com 1000+ caracteres), sem retornar erro ou limite máximo. Os campos ‘preco’ e ‘quantidade’, documentados como integer, aceitam valores como string ("254", "1000") e realizam coerção automática para número. A documentação não menciona essa conversão, criando discrepância entre contrato e comportamento real.

Passos para reprodução:

1. Acessar a collection da API via Postman
2. Acessar a pasta ‘usuários’
3. Realizar cadastro de usuário (‘administrador’ deve ser ‘true’)
4. Acessar a pasta ‘login’
5. Realizar login
6. Copiar token (string de ‘authorization’)
7. Acessar a pasta ‘produtos’
8. Acessar a request ‘POST Cadastrar produto’
9. Em ‘Authorization’ informe o token copiado, em {{apikey}}
10. Clicar em ‘Body’ seguido de ‘raw’
11. Preencher os campos ‘nome’, ‘descricao’ com strings longas e ‘preco’, ‘quantidade’ com inteiros dentro de aspas
12. Enviar request em ‘Send’
13. Observar a aprovação do cadastro de produto

Resultado esperado:

POST /produtos deve retornar HTTP 400 Bad Request quando:

- campo ‘nome’ exceder limite (ex: 50 caracteres);
- campo ‘descricao’ exceder limite (ex: 100 caracteres);
- campos ‘preco’ ou ‘quantidade’ não forem enviados como integer/number.

Alternativamente, se aceitar type coercion, a documentação deve especificar esse comportamento explicitamente. A resposta deve indicar qual campo violou qual regra.

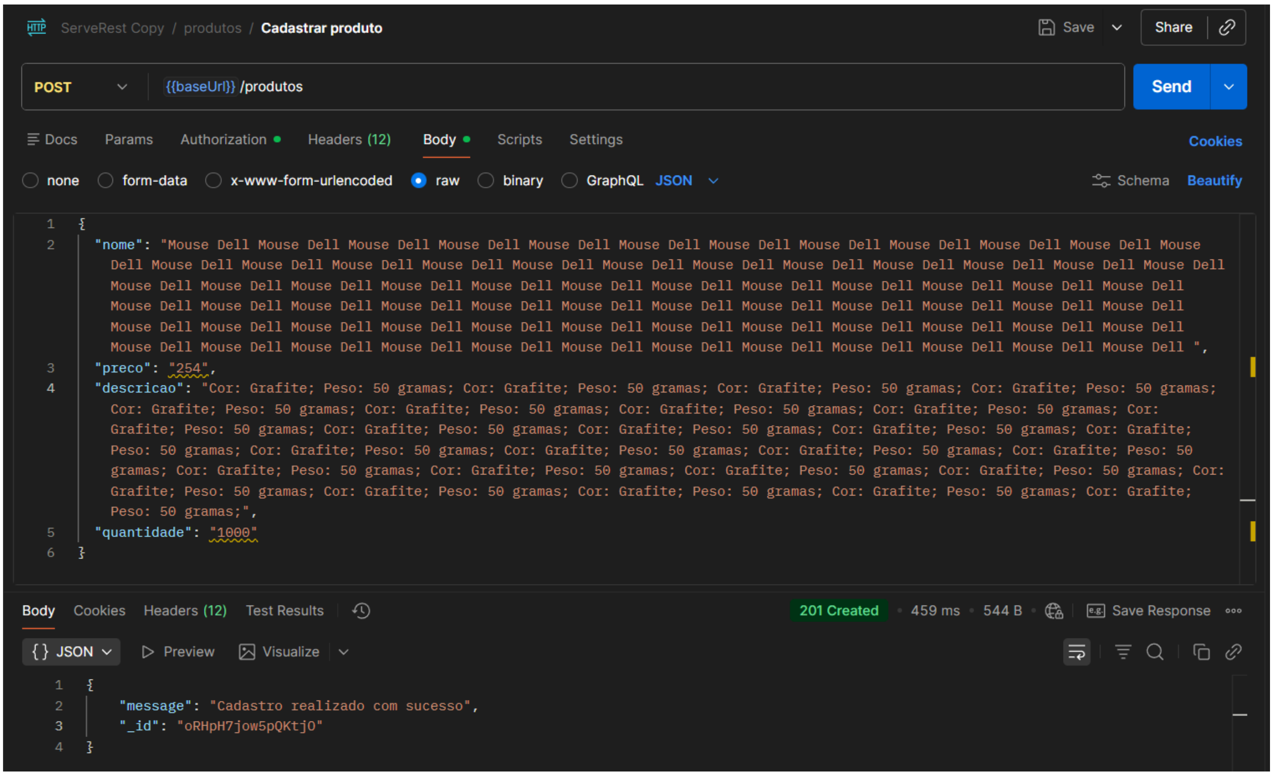
Resultado obtido:

POST /produtos retornou HTTP 201 Created com:

- nome e descrição contendo 1000+ caracteres cada;
- preco="254" e quantidade="1000" (strings ao invés de integers).

A API não validou o comprimento máximo de strings, realizou type coercion automática de string para número sem documentação, e armazenou dados sem limite. Nenhum erro ou aviso foi gerado.

Evidências:



Cadastro de produto com valores inadequados

<u>REGISTRO 04</u>

Caso de teste:

ET05 - Repetição de operações que deveriam ser bloqueadas ou idempotentes.

Título:

Exclusão de produto inexistente retorna 200 OK indevidamente através de ID inválido.

Descrição:

DELETE /produtos/:id com ID inexistente retorna HTTP 200 OK acompanhado de mensagem "Nenhum registro encontrado". Porém, segundo RFC 7231, status 200 OK indica sucesso na operação sobre um recurso existente. A resposta cria contradição, onde o código sugere sucesso, mas o corpo indica que nenhuma ação foi realizada. Isso viola a semântica HTTP e pode causar interpretação incorreta em sistemas automatizados.

Passos para reprodução:

1. Acessar a collection da API via Postman
2. Acessar a pasta ‘usuários’
3. Realizar cadastro de usuário (‘administrador’ deve ser ‘true’)
4. Acessar a pasta ‘login’
5. Realizar login
6. Copiar token (string de ‘authorization’)
7. Acessar a pasta ‘produtos’
8. Acessar a request ‘POST Cadastrar produto’
9. Realizar cadastro de produto

10. Acessar a request ‘DELETE Excluir produto ’
11. Informar um valor inválido para ID no endpoint
12. Informar o token do usuário administrador em {{apikey}}
13. Enviar request em ‘Send’
14. Observar a aprovação da requisição

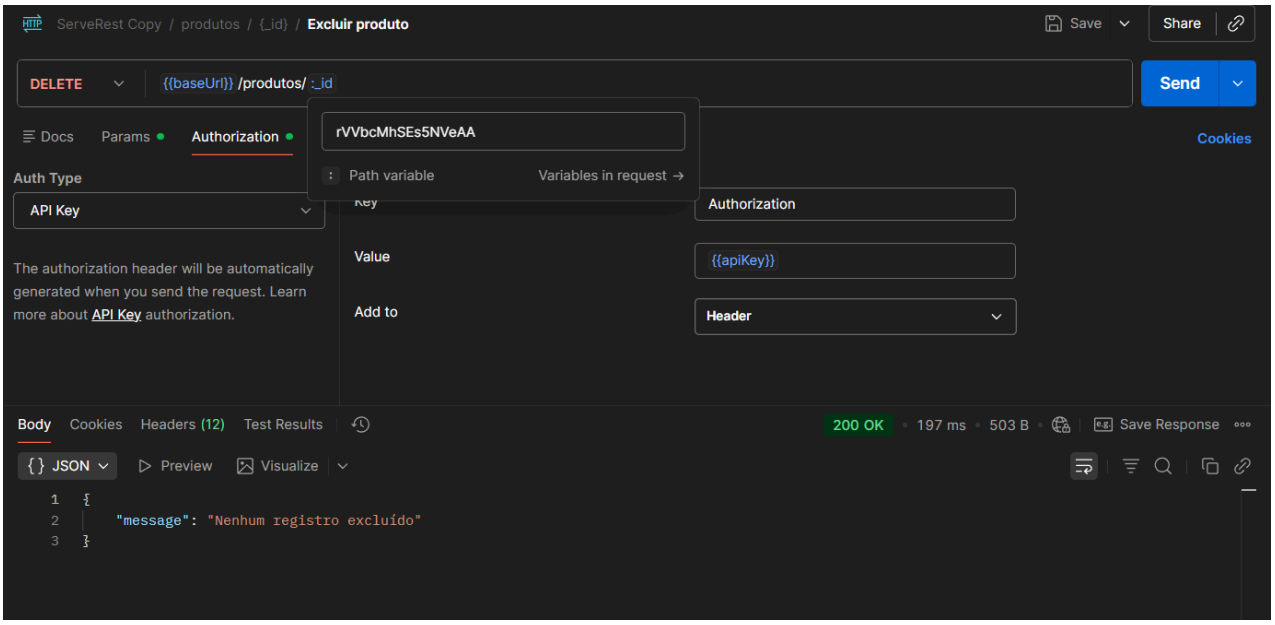
Resultado esperado:

DELETE /produtos/:id com ID inexistente deve retornar HTTP 404 Not Found com mensagem clara (ex: {"message": "Produto não encontrado", "id": "xyz123"}). Alternativamente, se adotar semântica idempotente, pode retornar HTTP 204 No Content, mas isso deve estar documentado explicitamente. DELETE com ID existente deve retornar HTTP 200 OK ou 204 No Content após remoção bem-sucedida.

Resultado obtido:

DELETE /produtos/:id com IDs inexistentes (aleatórios e previamente excluídos) retornou consistentemente HTTP 200 OK com corpo {"message": "Nenhum registro encontrado"}. Essa contradição entre status code (sucesso) e mensagem (falha) pode causar: sistemas automatizados interpretarem 200 OK como confirmação de exclusão; falhas em sincronização de dados entre sistemas; dificuldade em auditoria (não fica claro se recurso foi excluído ou nunca existiu); necessidade de parsing do corpo da resposta além do status code para determinar resultado real da operação.

Evidências:



Deletar produto com ID inexistente

CHECKS AUTOMATIZADOS

Foram realizados três testes automatizados utilizando JavaScript no Postman, com validações de status HTTP, estrutura de resposta e tipos de dados.

CHECK 01 - CADASTRO DE USUÁRIO (CENÁRIO POSITIVO):

Cenário: Cadastrar um novo usuário com dados válidos.

Endpoint: POST /usuarios

Request Body:

```
{
        "nome": "Fulano da Silva",
        "email": "teste@qa.com.br",
        "password": "senha123",
        "administrador": "true"
}
```

Validações Automatizadas:

- Status Code: deve retornar 201 (Created)
- Campo "message": deve existir e conter texto
- Campo "_id": deve existir e ser uma string
- Estrutura da resposta: deve corresponder ao esperado

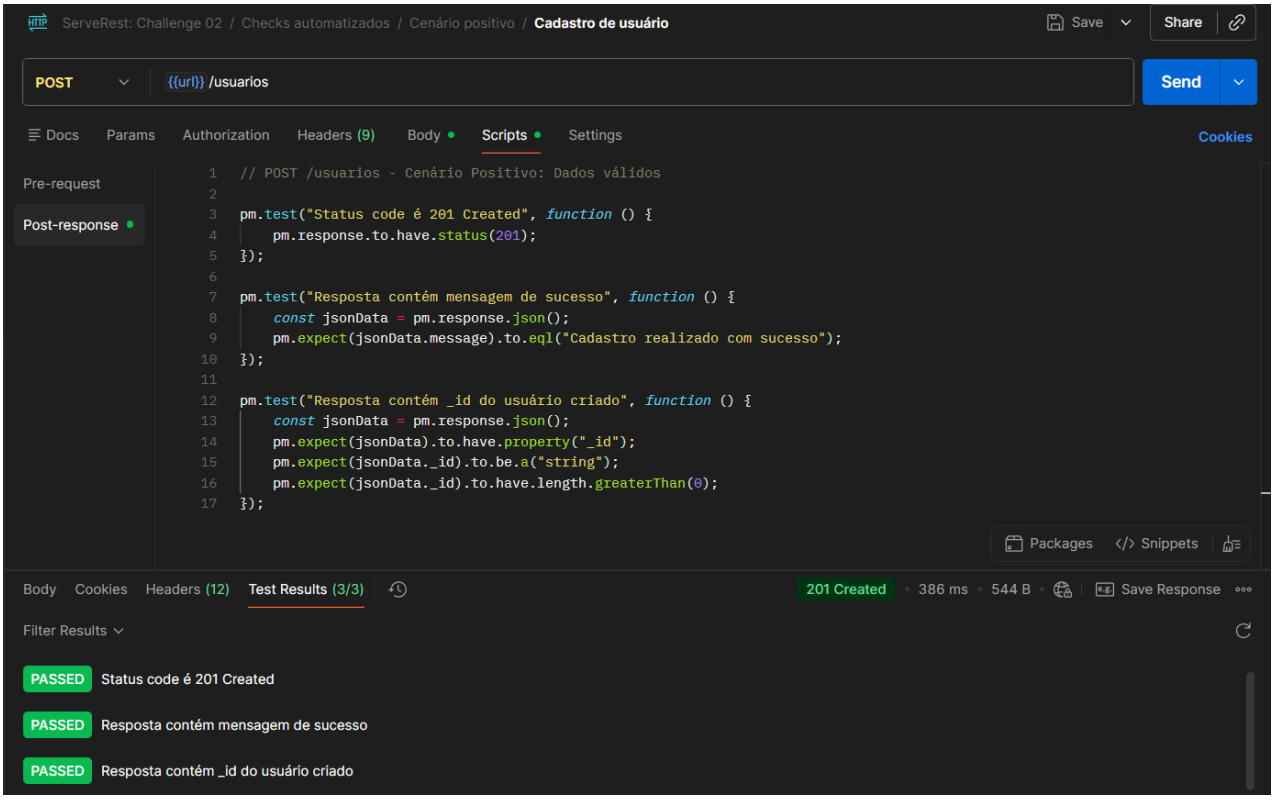
Script de Teste:

```
pm.test("Status code é 201 Created", function () {
    pm.response.to.have.status(201);
});

pm.test("Resposta contém mensagem de sucesso", function () {
    const jsonData = pm.response.json();
    pm.expect(jsonData.message).to.eql("Cadastro realizado com sucesso");
});

pm.test("Resposta contém _id do usuário criado", function () {
    const jsonData = pm.response.json();
    pm.expect(jsonData).to.have.property("_id");
    pm.expect(jsonData._id).to.be.a("string");
    pm.expect(jsonData._id).to.have.length.greaterThan(0);
```

```
});
```

Resultado: Passou

Evidência:



CHECK 02 - CADASTRO DE USUÁRIO (CENÁRIO NEGATIVO):

Cenário: Tentar cadastrar usuário com e-mail já existente.

Endpoint: POST /usuarios

Request Body:

```
{
  "nome": "Usuário Duplicado",
  "email": "email_ja_cadastrado@qa.com",
  "password": "senha123",
  "administrador": "false"
}
```

Validações Automatizadas:

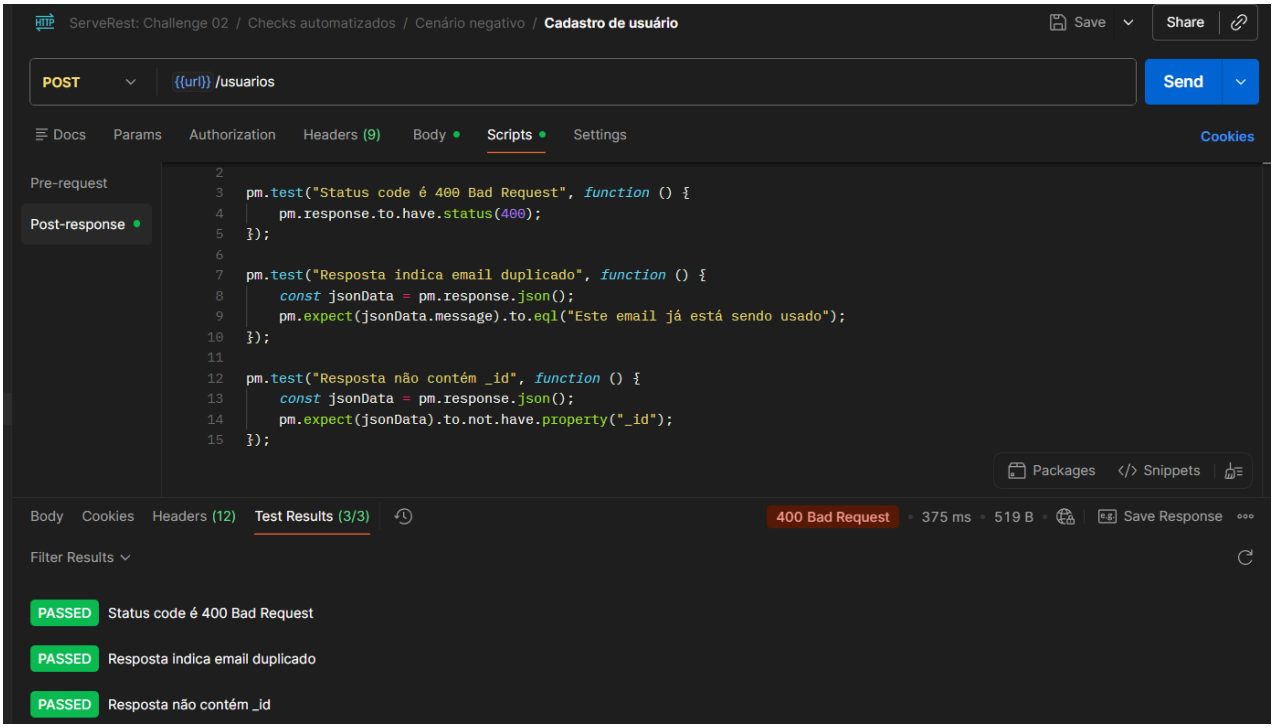
- Status Code: deve retornar 400 (Bad Request)
- Mensagem: deve informar que o e-mail já está cadastrado
- Campo "message": deve existir

Script de Teste:

```
pm.test("Status code é 400", function () {
    pm.response.to.have.status(400);
});

pm.test("Message informa email duplicado", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.message).to.include("já está sendo usado");
});
```

Resultado: Passou

Evidência:



CHECK 03 - LISTAGEM DE USUÁRIOS (VALIDAÇÃO DE CONTRATO):

Cenário: Listar usuários cadastrados e validar a estrutura do contrato da API.

Endpoint: GET /usuarios

Validações Automatizadas:

- Status Code: deve retornar 200 (OK)
- Campo "quantidade": deve existir e ser um número
- Campo "usuarios": deve existir e ser um array
- Cada usuário deve conter: _id, nome, email, password, administrador
- Tipos de dados: _id, nome, email, password e administrador devem ser strings
- Campo "email": deve ser um formato de e-mail válido

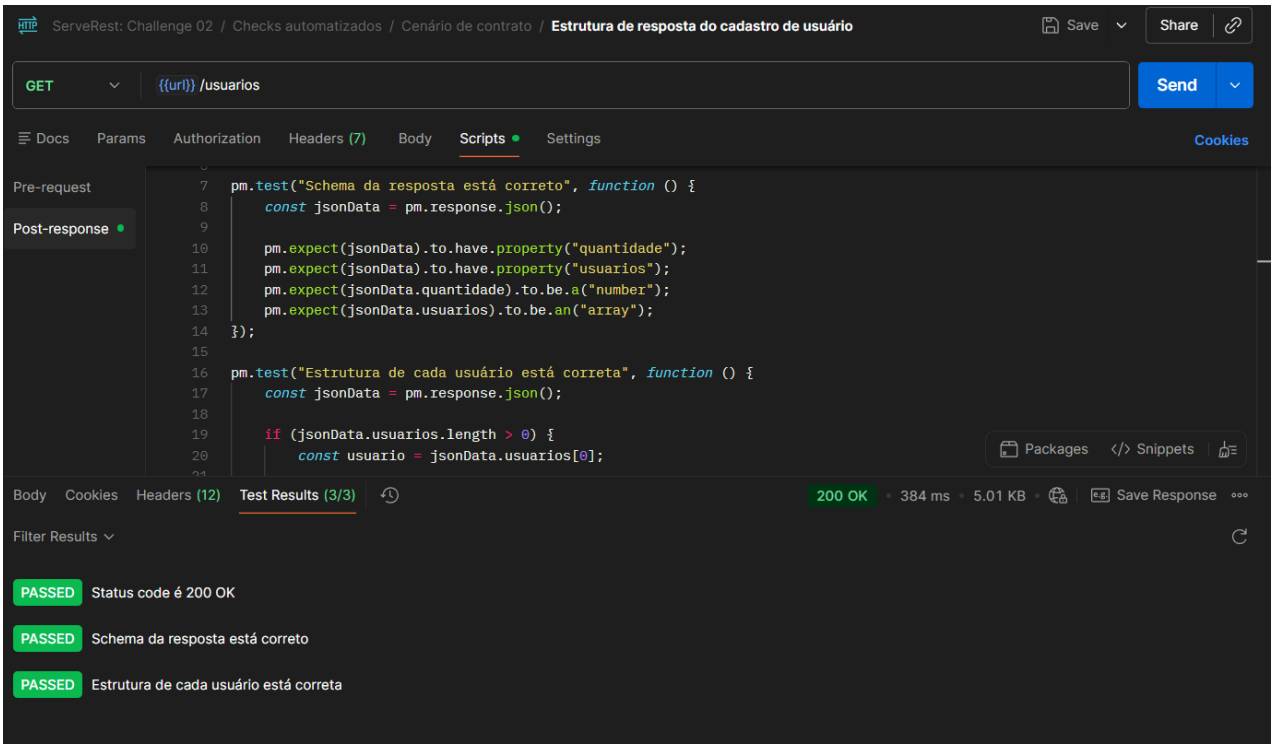
- Campo "administrador": deve ser "true" ou "false"

Script de Teste:

```
pm.test("Status code é 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Valida estrutura do contrato", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.usuarios).to.be.an("array");

    if (jsonData.usuarios.length > 0) {
        jsonData.usuarios.forEach(function(usuario) {
            pm.expect(usuario).to.have.property("_id");
            pm.expect(usuario).to.have.property("nome");
            pm.expect(usuario).to.have.property("email");
            pm.expect(usuario).to.have.property("password");
            pm.expect(usuario).to.have.property("administrador");

            pm.expect(usuario._id).to.be.a("string");
            pm.expect(usuario.nome).to.be.a("string");
            pm.expect(usuario.email).to.be.a("string");
            pm.expect(usuario.password).to.be.a("string");
            pm.expect(usuario.administrador).to.be.a("string");

            pm.expect(usuario.email).to.match(/^[\w-.]+@([\w-]+\.)+[\w-]{2,4}$/);
            pm.expect(usuario.administrador).to.be.oneOf(["true", "false"]);
        });
    }
});
```

Resultado: Passou

Evidência:



MATRIZ DE TÉCNICAS VS BUGS

Esta matriz apresenta os cenários de teste que falharam durante a execução e que foram descritos em “Registro de Bugs”, incluindo tanto testes funcionais quanto exploratórios. Essa tabela tem o objetivo de tornar mais visual a técnica aplicada para a descoberta de cada inconformidade (bug).

Técnica	Registro 1	Registro 2	Registro 3	Registro 4
Teste Funcional (Cenário positivo)	-	-	-	-
Teste Funcional (Cenário negativo)	X	X	X	X
Teste Exploratório (Input Validation)	X	X	X	X
Teste Exploratório (State Transition)	-	-	-	X
Teste Exploratório (Business Rules)	-	-	X	X