# Raphael - Relatório de Issues

### Resumo Executivo

Este relatório documenta três issues identificadas durante os testes da API do sistema de e-commerce, abrangendo falhas funcionais, bugs críticos e sugestões de melhoria no sistema.

---

### [SVT-001] Falha no Retorno de Cadastro de Produtos

**Tipo:** Falha Funcional
**Prioridade:** Média
**Status:** Aberta
**ID:** SVT-001

#### Descrição

Identificada uma falha no endpoint de criação de produtos onde o response body não retorna informações completas sobre o produto cadastrado, apenas ID e mensagem genérica de sucesso.

#### Passos para Reprodução

1. Autenticar como vendedor e obter token válido
2. Realizar POST para `/produtos` com body:

```
{
  "nome": "Geladeira",
  "preco": 1800,
  "descricao": "500W eletrolux",
  "quantidade": 2
}
```

#### Resultados

- **Esperado:** Response body com todas informações do produto cadastrado
- **Obtido:** Apenas ID e mensagem "cadastro realizado"

#### Impacto

Vendedores não têm confirmação visual imediata dos dados cadastrados, necessitando de consulta adicional para verificação.

#### Evidência

```
// Response atual
{
  "id": "pqpq-02i2e",
  "message": "Cadastro realizado com sucesso"
}

// Response esperado
{
  "id": "pqpq-02i2e",
  "nome": "Geladeira",
  "preco": 1800,
  "descricao": "500W eletrolux",
  "quantidade": 2,
  "message": "Cadastro realizado com sucesso"
}
```

#### Recomendações

- Modificar endpoint POST `/produtos` para retornar objeto completo do produto criado
- Manter compatibilidade com consumers existentes

---

### [SVT-002] Validação Insuficiente no Cadastro de Produtos

**Tipo:** Bug
**Prioridade:** Alta
**Status:** Aberta
**ID:** SVT-002

#### Descrição

A API permite cadastro de produtos com dados aleatórios sem validações adequadas, possibilitando criação em massa de registros inválidos ou teste. Fora a ausência de validação de tipagem.

#### Passos para Reprodução

1. Configurar pre-request script no Postman:

```
const randomNum = Math.floor(Math.random() * 10000);
pm.environment.set("nomeProduto", `Produto Teste ${randomNum}`);
pm.environment.set("precoProduto", Math.floor(Math.random() * 200) + 1);
pm.environment.set("descricaoProduto", `Descrição automática ${randomNum
pm.environment.set("quantidadeProduto", Math.floor(Math.random() * 50) +
```

2. Executar POST para `/produtos` com body usando variáveis aleatórias

#### Resultados

- **Esperado:** Validação e rejeição de dados aleatórios/inválidos
- **Obtido:** Criação bem-sucedida de produtos com dados aleatórios e em validação entre Integer s e String

#### Impacto

- Base de dados poluída com registros inválidos
- Possível degradação de performance
- Risco de segurança por possível abuso do endpoint
- Uso indevido para SQL Injection

#### Evidência

Request com dados aleatórios retorna status 201 Created:

```
{
  "id": "def456",
  "message": "Cadastro realizado com sucesso"
}
```

#### Recomendações

- Implementar validações de schema no request body
- Adicionar rate limiting por usuário/vendedor
- Validar se dados fazem sentido contextualmente (ex: preço > 0)

---

### [SVT-003] Melhoria no Controle de Requisições Massivas

**Tipo:** Sugestão de Melhoria
**Prioridade:** Baixa
**Status:** Aberta
**ID:** SVT-003

#### Descrição

Sugestão para implementar mecanismos adicionais de proteção contra requisições massivas em endpoints de atualização.

#### Contexto

Testes com Postman Runner (100 iterações) para atualizar produtos com IDs inexistentes foram bloqueados, mas sistema pode beneficiar-se de proteções adicionais.

#### Cenário de Teste

- Requisições PUT para `/produtos/{{id}}` com IDs aleatórios
- Script de validação no pre-request:

```
let idUsado = pm.variables.get("produtoId");
let status = pm.response.code;

if(!isNaN(idUsado) && idUsado <= 5){
    pm.test("Deve atualizar com sucesso (200)", function () {
        pm.expect(status).to.eql(200);
    });
} else {
    pm.test("Deve retornar erro para ID inválido (404)", function () {
        pm.expect(status).to.eql(404);
    });
}
```

#### Resultados

- **Esperado:** Bloqueio de requisições massivas
- **Obtido:** Requisições recusadas individualmente (404)

#### Sugestão de Melhoria

Implementar rate limiting adicional para prevenir:

- Tentativas de força bruta
- Ataques de negação de serviço (DoS)
- Abuso de endpoints críticos

#### Recomendações

- Implementar limite de requisições por período (ex: 100 PUTs/hora por usuário)
- Adicionar mecanismo de timeout após múltiplas falhas
- Considerar uso de Redis para tracking de requisições

---

### [SVT-004] Melhoria no Retorno de Dados dos Usuários

**Tipo:** Sugestão de Melhoria
**Prioridade:** Alta

**Status:** Aberta
**ID:** SVT-003

#### Descrição

Sugestão para ocultar dados privados das conta dos usuários do sistema.

#### Contexto

Testes com Postman Flow para retornar os dados dos usuários terão senha. Mesmo que não foram bloqueados, o sistema deve  minimamente criptografar a senha para beneficiar-se de proteções adicionais.

#### Cenário de Teste

- Requisições GET para `/usuarios`
- Script de validação no pre-request:

```
let idUsado = pm.variables.get("UserId");
let status = pm.response.code;

if(!isNaN(idUsado) && idUsado <= 5){
    pm.test("Deve atualizar com sucesso (200)", function () {
        pm.expect(status).to.eql(200);
    });
} else {
    pm.test("Deve retornar erro para ID inválido (404)", function () {
        pm.expect(status).to.eql(404);
    });
}
```

#### Resultados

- **Esperado:** Bloqueio de requisições massivas
- **Obtido:** Requisições recusadas individualmente (404)

#### Sugestão de Melhoria

Implementar rate limiting adicional para prevenir:

- Tentativas de força bruta
- Ataques de negação de serviço (DoS)
- Abuso de endpoints críticos

#### Recomendações

- Implementar limite de requisições por período (ex: 100 PUTs/hora por usuário)
- Adicionar mecanismo de timeout após múltiplas falhas
- Considerar uso de Redis para tracking de requisições

## [SVT-CAR-001] Restrição de Criação de Múltiplos Carrinhos

**Tipo:** Bug
**Prioridade:** Alta
**Status:** Aberta
**ID:** SVT-CAR-001

#### Descrição

Usuário consegue criar mais de um carrinho ativo, quebrando a regra de negócio que permite apenas um por usuário.

#### Contexto

A API deveria bloquear a criação de um segundo carrinho para o mesmo usuário autenticado.

#### Cenário de Teste

```
POST /carrinhos
Content-Type: application/json

{
  "produtos": [
    {
      "idProduto": "produtoValidoId",
      "quantidade": 1
    }
  ]
}

```

#### Resultados

- **Esperado:** API retorna erro (ex: 400) informando que já existe carrinho ativo.
- **Obtido:** API cria um novo carrinho.

#### Sugestão de Melhoria

Implementar verificação de carrinho existente antes de criar um novo.

#### Recomendações

- Impedir criação de múltiplos carrinhos para o mesmo usuário.
- Retornar erro padronizado ( `400` com mensagem JSON clara).

---

## [SVT-CAR-002] Inclusão de Produtos Inexistentes

**Tipo:** Bug

**Prioridade:** Alta

**Status:** Aberta

**ID:** SVT-CAR-002

#### Descrição

API permite adicionar produtos inexistentes (IDs inválidos) no carrinho.

#### Contexto

IDs devem ser validados contra a base de produtos cadastrados.

#### Cenário de Teste

```
POST /carrinhos
Content-Type: application/json

{
  "produtos": [
    {
      "idProduto": "idInexistente123",
      "quantidade": 1
    }
  ]
}

```

#### Resultados

- **Esperado:** Erro `404` informando produto não encontrado.
- **Obtido:** Produto é adicionado ao carrinho.

#### Sugestão de Melhoria

Adicionar validação de `idProduto` no momento da inserção.

#### Recomendações

- Checar existência de produto antes de aceitar requisição.
- Padronizar resposta de erro.

---

## [SVT-CAR-003] Quantidade Inválida de Produtos

**Tipo:** Bug
**Prioridade:** Média
**Status:** Aberta
**ID:** SVT-CAR-003

#### Descrição

API aceita quantidade zero ou negativa ao adicionar produto no carrinho.

#### Contexto

Quantidade mínima deveria ser `1`.

#### Cenário de Teste

```
POST /carrinhos
Content-Type: application/json

{
  "produtos": [
    {
      "idProduto": "produtoValidoId",
      "quantidade": -3
    }
  ]
}

```

#### Resultados

- **Esperado:** Erro de validação (`400`).
- **Obtido:** Carrinho é criado com quantidade inválida.

#### Sugestão de Melhoria

Validar quantidade mínima de 1.

#### Recomendações

- Rejeitar requisições com `quantidade <= 0`.
- Mensagem clara de erro para o cliente.

---

## [SVT-CAR-004] Finalização de Compra com Carrinho Vazio

**Tipo:** Bug

**Prioridade:** Alta

**Status:** Aberta

**ID:** SVT-CAR-004

#### Descrição

API permite concluir compra mesmo sem itens no carrinho.

#### Contexto

Finalizar compra deveria exigir pelo menos um item válido.

#### Cenário de Teste

```
POST /carrinhos/concluir-compra
Content-Type: application/json
Authorization: Bearer {{token}}

```

#### Resultados

- **Esperado:** Erro `400`.
- **Obtido:** Compra finalizada com sucesso.

#### Sugestão de Melhoria

Adicionar verificação de itens antes de concluir compra.

#### Recomendações

- Impedir finalização se não houver produtos no carrinho.
- Garantir integridade do fluxo de compra.

---

## [SVT-CAR-005] Exclusão de Carrinho de Outro Usuário

**Tipo:** Bug de Segurança

**Prioridade:** Crítica

**Status:** Aberta

**ID:** SVT-CAR-005

#### Descrição

É possível deletar carrinho de outro usuário sem restrição adequada.

#### Contexto

Deleção deveria estar vinculada ao `idUsuario` autenticado.

#### Cenário de Teste

```
DELETE /carrinhos/{idCarrinho}
Authorization: Bearer {{tokenOutroUsuario}}

```

#### Resultados

- **Esperado:** Erro `403` (Forbidden).

- **Obtido:** Carrinho é deletado.

#### Sugestão de Melhoria

Aplicar checagem de usuário dono do carrinho.

#### Recomendações

- Associar carrinho ao usuário autenticado.
- Restringir operações de exclusão.

---

### Ambiente de Teste

- **Hardware:** Notebook Intel i5 11ª Geração
- **Software:** Postman v10.0+
- **Navegador:** Opera GX Browser
- **Executor de Testes:** Postman Flow

### Responsável

Raphael Rates - QA Analyst (NoBugs)
**Data:** 03/09/25
**Versão API Testada:** 1.24

---

*Este relatório contém informações confidenciais para uso interno da equipe de desenvolvimento e qualidade.*