

Relatório de Issues e Sugestões de Melhoria – API ServeRest

- **Projeto:** API ServeRest
- **Data da Análise:** 04 de setembro de 2025
- **Autor:** Guilherme Hepp da Fonseca
- **Versão da API Testada:** v1 (conforme ambiente de testes)

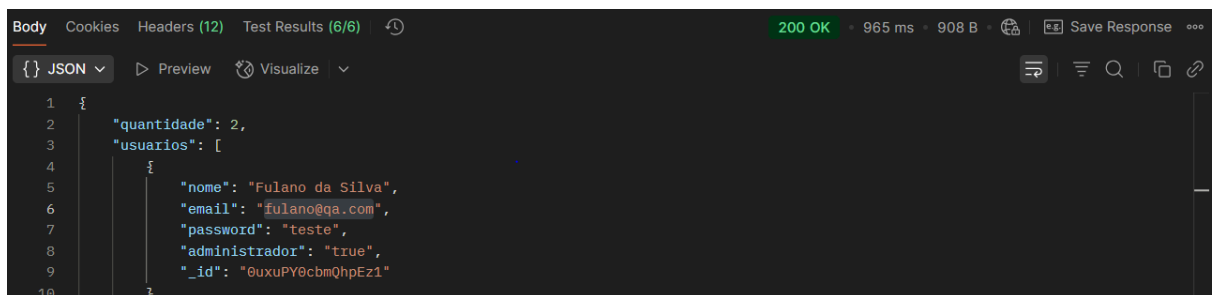
Resumo Executivo

Este documento apresenta os resultados da análise e execução de testes automatizados na API ServeRest. O processo de teste, guiado por User Stories e pela documentação da API, revelou não apenas o correto funcionamento das funcionalidades principais, mas também identificou **issues críticas de segurança** e oportunidades para **melhorias de usabilidade e padronização** da API. As seções a seguir detalham cada ponto encontrado.

1. Issues Encontradas (Bugs).

Issue: API-BUG-001

- **Título:** Vazamento de Dados Sensíveis na Listagem de Usuários
- **Gravidade:** Crítica
- **Endpoint Afetado:** GET /usuarios
- **Descrição:** A requisição para listar todos os usuários da base de dados retorna uma lista de objetos de usuário, onde cada objeto inclui o campo **password** em texto plano. Esta é uma vulnerabilidade de segurança grave que expõe as senhas de todos os usuários.
- **Passos para Reproduzir:**
 1. Realizar uma requisição **GET** para o endpoint /usuarios.
 2. Analisar o array **usuarios** no corpo da resposta.
- **Resultado Esperado:** A resposta deve conter uma lista de objetos de usuário **sem** o campo **password**.
- **Resultado Atual:** Cada objeto de usuário na lista **contém** o campo **password** com a senha do usuário.
- **Evidência:** Testes automatizados no Postman falham na asserção de segurança que verifica a ausência deste campo.
- **Anexos:**



```
1 {
2   "quantidade": 2,
3   "usuarios": [
4     {
5       "nome": "Fulano da Silva",
6       "email": "fulano@qa.com",
7       "password": "teste",
8       "administrador": "true",
9       "_id": "0uxuPY0cbmQhpEz1"
10    }
11  ]
12 }
```

Issue: API-BUG-002

- **Título:** Senha do Usuário Exposta em Texto Plano no Payload do Token JWT
- **Gravidade:** Crítica
- **Endpoint Afetado:** POST /login
- **Descrição:** O token de autenticação (JWT) gerado após um login bem-sucedido contém a senha do usuário em texto plano no seu payload. Como o payload de um JWT é apenas codificado em Base64 e não criptografado, qualquer pessoa com acesso ao token pode decodificá-lo e visualizar a senha do usuário.
- **Passos para Reproduzir:**
 1. Realizar uma requisição POST para /login com credenciais válidas.
 2. Copiar o token JWT da resposta.
 3. Decodificar a segunda parte do token (payload) usando um decodificador Base64.
- **Resultado Esperado:** O payload do token deve conter apenas informações não-sensíveis, como o ID do usuário, e-mail e nível de permissão (admin/comum).
- **Resultado Atual:** O payload do token contém o campo password com a senha do usuário.
- **Evidência:** Análise manual do token Bearer retornado pela API.
- **Anexos:**



Issue: API-BUG-003

- **Título:** Falha na Validação de Tamanho da Senha
- **Gravidade:** Alta
- **Endpoints Afetados:** `POST /usuarios`, `PUT /usuarios/{id}`
- **Descrição:** A API não está aplicando a regra de negócio de que as senhas devem conter entre 5 e 10 caracteres. Atualmente, o sistema permite o cadastro e a atualização de usuários com senhas fora deste intervalo (ex: "123" ou "1234567890abc"), o que viola os critérios de aceitação definidos.
- **Passos para Reproduzir:**
 1. Enviar uma requisição `POST` para `/usuarios` com um corpo (body) contendo uma senha com menos de 5 caracteres (ex: `"password": "abc"`).
 2. Enviar uma requisição `POST` para `/usuarios` com um corpo (body) contendo uma senha com mais de 10 caracteres (ex: `"password": "abcdefghijklm"`).
- **Resultado Esperado:** Para ambas as requisições, a API deveria retornar um status `400 Bad Request` com uma mensagem de erro indicando que a senha não atende aos requisitos de tamanho.
- **Resultado Atual:** A API retorna `201 Created`, cadastrando o usuário com a senha inválida.

- **Evidência:** Testes automatizados criados no Postman para validar os limites de tamanho da senha, que atualmente resultam em sucesso (201 Created) em vez do erro esperado (400).

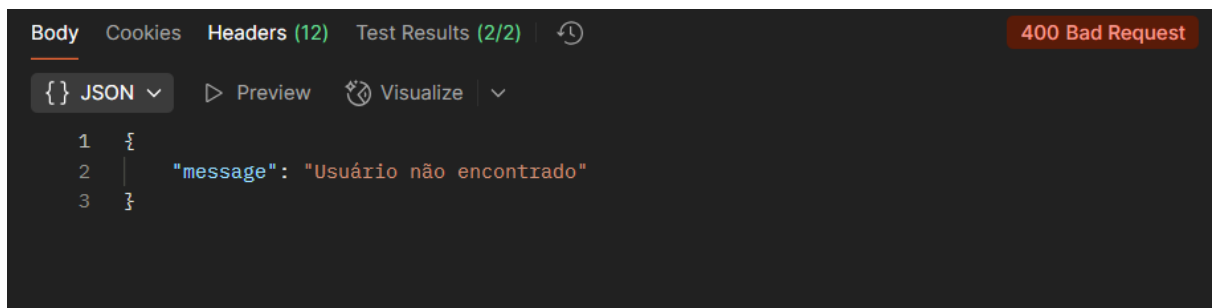
Issue: API-BUG-004

- **Título:** Falha na Validação de Restrição de Domínio de E-mail
- **Gravidade:** Média
- **Endpoints Afetados:** POST /usuarios, PUT /usuarios/{id}
- **Descrição:** A API não está aplicando a regra de negócio que proíbe o cadastro de usuários com e-mails dos provedores "@gmail.com" e "@hotmail.com". O sistema está aceitando o cadastro, violando os critérios de aceitação.
- **Passos para Reproduzir:**
 1. Enviar uma requisição POST para /usuarios com um corpo (body) contendo um e-mail com domínio @gmail.com (ex: "email": "teste.usuario@gmail.com").
 2. Enviar uma requisição POST para /usuarios com um corpo (body) contendo um e-mail com domínio @hotmail.com.
- **Resultado Esperado:** Para ambas as requisições, a API deveria retornar um status 400 Bad Request com uma mensagem de erro indicando que o provedor de e-mail não é permitido.
- **Resultado Atual:** A API retorna 201 Created, cadastrando o usuário com o e-mail de um provedor restrito.
- **Evidência:** Testes automatizados no Postman que tentam cadastrar usuários com e-mails @gmail.com e @hotmail.com, que atualmente resultam em sucesso (201 Created) em vez do erro esperado (400).

2. Sugestões de Melhoria

Melhoria: API-001

- **Título:** Padronizar Código de Erro para Recursos Não Encontrados para **404 Not Found**
- **Prioridade:** Média
- **Endpoints Afetados:** **GET /usuarios/{id}**, **GET /produtos/{id}**, **GET /carrinhos/{id}**
- **Descrição da Melhoria Proposta:** Atualmente, ao tentar buscar um recurso por um ID que não existe, a API retorna o status **400 Bad Request**. A recomendação é alterar este comportamento para retornar ao status **404 Not Found**.
- **Justificativa:** O uso do **404 Not Found** é a convenção padrão do protocolo HTTP e do padrão REST para indicar que o recurso solicitado não pôde ser encontrado. Isso torna a API mais previsível e intuitiva para os desenvolvedores que a consomem, alinhando-a com as melhores práticas de mercado.
- **Anexos:**



Melhoria: API-IMP-002

- **Título:** Melhorar Mensagens de Erro para Requisições com Dados Inválidos (400)
- **Prioridade:** Média
- **Endpoints Afetados:** Todos os endpoints que recebem um corpo (body), como **POST** e **PUT** para usuários, produtos e carrinhos.
- **Descrição da Melhoria Proposta:** Em alguns cenários de falha (ex: tentar cadastrar um carrinho com um produto inexistente), a API retorna uma mensagem de erro genérica como **"Algo deu errado"**. A sugestão é padronizar todas as respostas de erro **400 Bad Request** para que retorne uma mensagem específica, indicando qual campo causou o problema e por quê.
- **Justificativa:** Mensagens de erro específicas e descritivas (**"Produto com ID 'XYZ' não existe"**, por exemplo) facilitam drasticamente o processo de debugging e desenvolvimento para quem está integrando com a API, economizando tempo e reduzindo a frustração. A API já faz isso bem em alguns endpoints (como na validação de senhas), e a sugestão é aplicar essa qualidade de forma consistente.

Melhoria: API-IMP-003:

- **Título:** Padronizar a Resposta do Token de Login para Evitar Erros de Integração
- **Prioridade:** Média
- **Endpoint Afetado:** POST /login
- **Descrição do Problema Identificado:** Durante a implementação dos testes automatizados no Postman, foi identificado que a requisição de POST /produtos falhava com erro 401 Unauthorized. A causa raiz foi a duplicação do prefixo "Bearer" no cabeçalho de autorização (ex: Authorization: Bearer Bearer ey...). Isso ocorreu porque a resposta do endpoint de login já inclui o prefixo ("authorization": "Bearer ey..."), e a ferramenta cliente (Postman) também adiciona o mesmo prefixo por padrão, resultando em um token malformatado.
- **Sugestão de Melhoria Proposta:** Modificar a resposta do endpoint POST /login para separar o tipo de token do token em si. Isso segue as melhores práticas de mercado (como o padrão OAuth 2.0).
- **Justificativa:** Separar o token do seu tipo torna a API **mais clara, menos suscetível a erros e mais fácil de ser consumida** por diferentes clientes e bibliotecas. Essa abordagem elimina a ambiguidade e a necessidade de o cliente tratar a string para extrair o token puro, prevenindo erros comuns de integração, como o que foi encontrado em nosso cenário de teste.