📑 Plano de Testes – API ServeRest

---

1. Apresentação

O presente plano de testes foi elaborado para garantir a qualidade da API **ServeRest**, que simula um marketplace virtual. O objetivo é validar se as regras de negócio descritas nas **User Stories** estão corretamente implementadas, assegurando que o sistema atenda aos requisitos funcionais e não funcionais.

---

2. Objetivo

- Garantir que as rotas da API (Usuários, Login, Produtos e Carrinhos) funcionem conforme especificado.
- Identificar falhas e inconsistências em relação às regras de negócio.
- Definir cenários de teste que permitam validar tanto **fluxos positivos** quanto **negativos**.
- Selecionar cenários críticos para **automação de testes**.

---

3. Escopo

- **Incluso**:
  - Testes funcionais das rotas `usuarios`, `login`, `produtos` e `carrinhos`.
  - Testes de autenticação e autorização.
  - Testes de validação de dados (e-mails, senhas, campos obrigatórios).
  - Testes negativos (duplicidade, restrições, usuários/produtos inexistentes).
- **Excluído**:
  - Testes de performance, stress ou carga.
  - Testes de segurança avançados (SQL Injection, XSS).
  - Integrações externas não documentadas no Swagger.

---

4. Análise

Com base nas **User Stories (US001 a US003)** e no **Swagger** da aplicação, foram levantados os cenários necessários para validar:

- **CRUD de usuários** (restrições de e-mail, senha e provedores não permitidos).
- **Autenticação via login** (validação de token, expiração).
- **CRUD de produtos** (restrição para usuários autenticados e administradores, duplicidade de nomes, vínculo com carrinhos).
- **Carrinhos** (dependência de produtos, exclusão de carrinho devolvendo estoque).

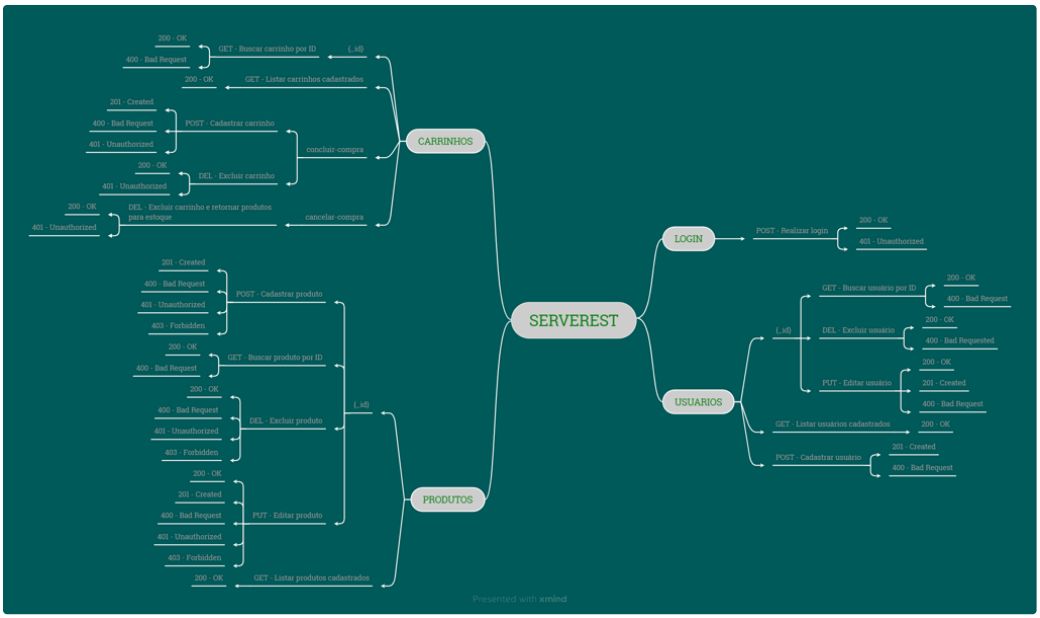
---

5. Técnicas aplicadas

- **Particionamento de equivalência** (e-mails válidos x inválidos, senha curta x longa).
- **Análise de valores-limite** (senhas de 4, 5, 10 e 11 caracteres).
- **Testes baseados em requisitos** (User Stories e Swagger).
- **Testes exploratórios** (cenários alternativos não documentados).

---

6. Mapa Mental da Aplicação



---

7. Cenários de Teste Planejados

Usuários (US001)

- Criar usuário válido ✅
- Criar usuário com e-mail duplicado ❌
- Criar usuário com e-mail inválido ❌
- Criar usuário com e-mail do Gmail/Hotmail ❌

- Criar usuário com senha < 5 ou > 10 ❌
- Atualizar usuário existente ✅
- Atualizar usuário inexistente → cria novo ✅
- Atualizar usuário com e-mail duplicado ❌
- Deletar usuário sem carrinho ✅
- Deletar usuário com carrinho ❌

Login (US002)

- Login com usuário válido (gera token) ✅
- Login com senha incorreta ❌
- Login com usuário inexistente ❌
- Login com token expirado ❌

Produtos (US003)

- Criar produto válido ✅
- Criar produto com nome duplicado ❌
- Criar produto sem autenticação ❌
- Atualizar produto válido ✅
- Atualizar produto inexistente → cria novo ✅
- Atualizar produto com nome duplicado ❌
- Excluir produto válido ✅
- Excluir produto em carrinho ❌
- Excluir produto sem autenticação ❌

Carrinhos

- Criar carrinho válido ✅
- Criar carrinho com produto inexistente ❌
- Buscar carrinho por ID ✅
- Deletar carrinho concluindo compra ✅
- Deletar carrinho retornando produtos ao estoque ✅

---

8. Priorização dos Testes

- **Alta prioridade**: Login, Cadastro de Usuário, Cadastro de Produto, Exclusão de Produto com Carrinho.
- **Média prioridade**: Atualização de registros, Cenários alternativos de erro.

- **Baixa prioridade**: Testes exploratórios não documentados no Swagger.

---

9. Matriz de Risco

Risco	Probabilidade	Impacto	Mitigação
Cadastro de usuário duplicado	Alta	Alta	Testes automatizados de duplicidade
Token inválido/expirado	Média	Alta	Testes de autenticação recorrentes
Exclusão de produto em carrinho	Alta	Alta	Testes automatizados de dependência
E-mails inválidos ou bloqueados aceitos	Média	Média	Validação em automação + evidências
Senha fora do padrão aceita	Baixa	Média	Testes de valores-limite

---

10. Cobertura de Testes

- **Rotas cobertas**: 100% de usuários, login, produtos e carrinhos.
- **Cenários positivos e negativos** contemplados.
- **Cobertura além do Swagger**: restrições de provedores de e-mail, senhas inválidas, expiração de token.

---

11. Testes candidatos à automação

- Cadastro de usuário válido e inválido.
- Login válido e inválido.
- Cadastro de produto (válido, duplicado, não autenticado).
- Exclusão de produto em carrinho.

- Fluxo completo de carrinho (criar → concluir → devolver estoque).

A **collection Postman já contempla scripts automatizados** para essas validações, utilizando inclusive dados dinâmicos do arquivo `usuario.JSON`.

📑 Documento de Issues e Melhorias

Issues encontradas (exemplo baseado na execução da collection):

1. **[BUG] E-mails de Gmail/Hotmail ainda são aceitos** – não atende US001.
2. **[BUG] Senhas maiores que 10 caracteres são aceitas** – violação da regra de negócio.
3. **[BUG] PUT de usuário inexistente cria novo registro, mas sem validar restrição de provedores de e-mail**.
4. **[BUG] Exclusão de usuário com carrinho não retorna erro consistente (às vezes 200 ao invés de 400)**.
5. **[MELHORIA] Mensagens de erro inconsistentes (ora "Usuário não encontrado", ora "Bad Request")**.

Melhorias sugeridas:

- Padronizar mensagens de erro em todas as rotas.
- Implementar bloqueio efetivo de provedores `gmail.com` e `hotmail.com`.
- Melhorar retorno do PUT de usuário inexistente (informar que foi criado novo).
- Adicionar no Swagger exemplos de respostas negativas (401, 403, 404).
