API ServerRest - Plano de Testes

✅ 1. Apresentação

Este documento apresenta o planejamento de testes para a API ServeRest (https://compassuol.serverest.dev/), visando garantir a qualidade e aderência às regras de negócio especificadas nas User Stories. O foco está nas rotas de Usuários, Login, Produtos e Carrinhos, cobrindo tanto testes manuais quanto candidatos à automação.

Visão Geral do Projeto

- **Nome do Projeto**: API ServerRest - Plano de Testes
- **Objetivo**: Documentar e executar testes de API na aplicação ServerRest
- **Responsáveis**: Douglas Paulo Cortes
- **Período**: 05/05/2025 - 09/05/2025
- Backlog: API ServerRest | Backlog
- Ferramentas: Jira, Confluence, Postman, Miro;

🎯 2. Objetivo

Assegurar que as funcionalidades da API ServeRest estejam de acordo com os requisitos definidos, identificando falhas, inconsistências e oportunidades de melhoria através de uma abordagem sistemática de planejamento, execução e análise de testes.

🗺️ 3. Escopo

O plano abrange:

- **Testes funcionais das rotas:**
  - **/usuarios**
  - **/login**
  - **/produtos**
  - **/carrinhos**
- **Validação de regras de negócio.**
- **Testes positivos, negativos e de borda.**
- **Planejamento e priorização de execução.**
- **Mapeamento de melhorias e issues.**
- **Identificação de cenários candidatos à automação.**
- **Geração de Collection Postman com scripts automatizados básicos.**

🔍 4. Análise(a melhorar e desenvolver)

User Stories e pontos-chave:

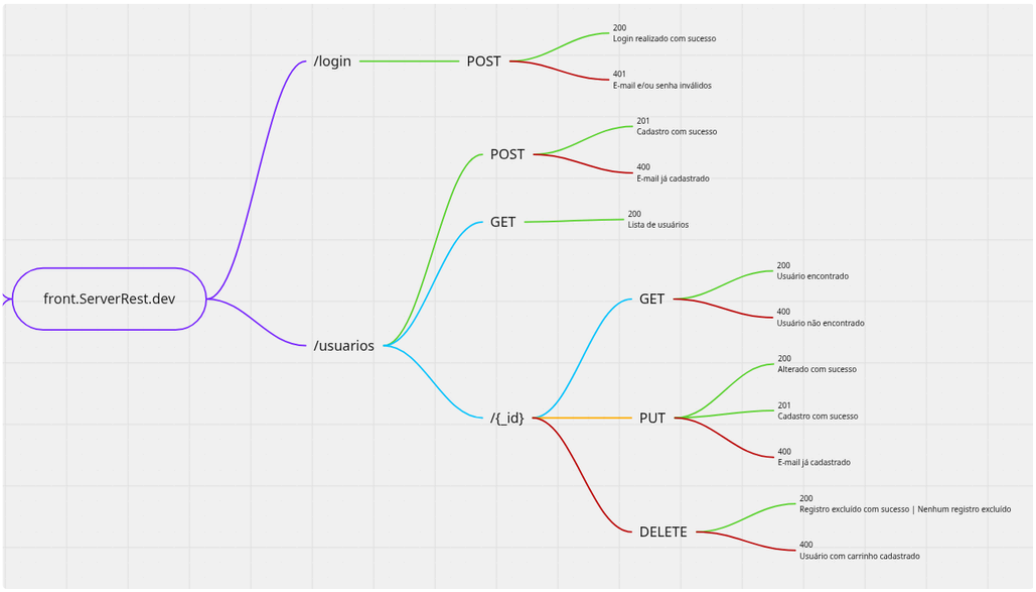
- **US 001 – Usuários: CRUD completo, evitar e-mails duplicados, validação de formato, restrição por domínio, senhas 5–10 caracteres.**
- **US 002 – Login: Autenticação correta, geração de token Bearer válido por 10 min, falhas para não cadastrados ou senha inválida.**

- **US 003 – Produtos: CRUD completo, ações apenas autenticadas, evitar nomes duplicados, bloqueios se vinculados a carrinhos.**
- **US 004 – Carrinhos (incluído adicionalmente):**
  - **Criar carrinho autenticado.**
  - **Obter carrinho.**
  - **Finalizar compra.**
  - **Cancelar compra.**
  - **Garantir integridade com produtos (ex.: não excluir produtos em carrinhos).**

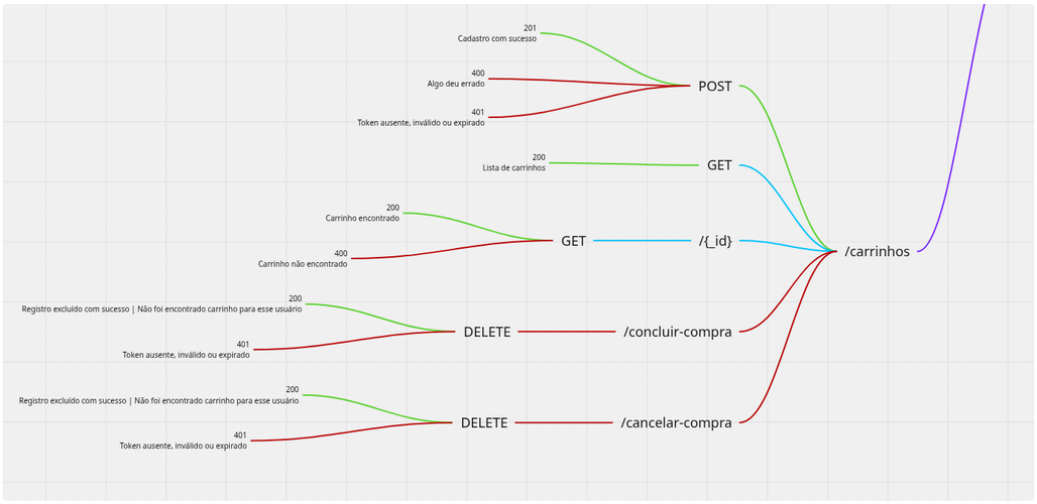
🛠️ 5. Técnicas Aplicadas(a melhorar e desenvolver)

- **Particionamento de equivalência.**
- **Análise de valor limite.**
- **Testes baseados em casos de uso.**
- **Testes negativos (erros esperados).**
- **Validação cruzada entre endpoints.**

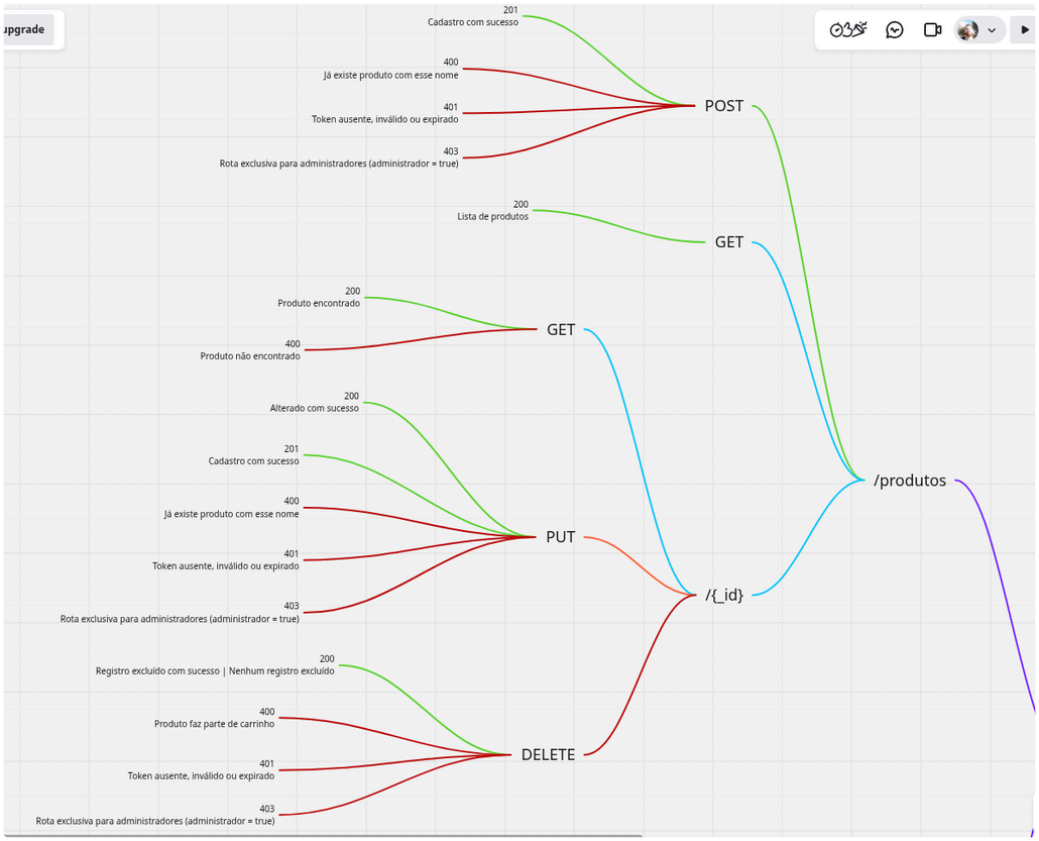
🧠 6. Mapa Mental da Aplicação



/login e /usuarios



/carrinhos



/produtos

Mapa Completo: link(https://miro.com/app/board/uXjVI87wAn8=/?share_link_id=841449426488 Conectar a conta do Miro)

📋 7. Cenários de Teste Planejados

ID	Endpoint	Cenário	Prioridade
U01	/usuarios	Criar usuário válido	Alta
U02	/usuarios	Criar usuário com e-mail duplicado	Alta

U03	/usuarios	Criar usuário com domínio gmail/hotmail	Alta
U04	/usuarios	Atualizar usuário inexistente (criar novo)	Média
U05	/usuarios	Validar senha fora do padrão (menor que 5, maior que 10)	Alta
L01	/login	Autenticar usuário válido	Alta
L02	/login	Autenticar usuário inexistente	Alta
L03	/login	Autenticar com senha inválida	Alta
P01	/produtos	Criar produto autenticado	Alta
P02	/produtos	Criar produto com nome duplicado	Alta
P03	/produtos	Atualizar produto inexistente (criar novo)	Média
P04	/produtos	Excluir produto fora de carrinho	Alta
P05	/produtos	Excluir produto dentro de carrinho	Alta
C01	/carrinhos	Criar carrinho autenticado	Alta
C02	/carrinhos	Criar carrinho sem autenticação	Alta
C03	/carrinhos	Adicionar produto inexistente	Média
C04	/carrinhos	Finalizar compra com carrinho válido	Alta
C05	/carrinhos	Cancelar compra com carrinho válido	Alta
C06	/carrinhos	Finalizar carrinho vazio	Média
C07	/carrinhos	Obter carrinho inexistente	Baixa

🏆 8. Priorização da Execução

1. Autenticação e permissões básicas (`/login` , `/usuarios`).

2. Operações críticas em produtos (`/produtos`), garantindo integridade.
3. Fluxo completo de carrinho (`/carrinhos`), validando impacto cruzado.
4. Cenários negativos e de borda.
5. Cenários complementares de atualização/criação indireta (PUT).

🧪 Bateria 1: Autenticação e Permissões Básicas (/usuarios, /login)

Nº	User Story	Caso de Teste	Esperado	Passo a Passo no Postman
1.1	US 001	Criar usuário válido (nome, e-mail não usado, senha 5-10 caracteres, não gmail/hotmail, admin opcional)	201 Created, usuário salvo corretamente	**POST** `https://compassuol.serverest.dev/usuarios` Body (JSON): `{"nome":"Teste", "email":"teste123@email.com","password":"123456","administrador":"true"}` Clique em **Send**
1.2	US 001	Criar usuário com e-mail já existente	400 Bad Request, mensagem de e-mail em uso	Use mesmo endpoint e body do 1.1, mas envie novamente para simular duplicação.
1.3	US 001	Criar usuário com senha fora do limite (ex.: 4 ou 11 caracteres)	400 Bad Request, erro de validação	Troque `"password":"123456"` por `"password":"1234"` ou `"password":"12345678910"` e envie.
1.4	US 001	Criar usuário com e-mail inválido	400 Bad Request, erro de validação	Troque `"email":"teste123@email.com"` por `"email":"email-invalido"` e envie.
1.5	US 002	Autenticar com usuário válido (gera API key válido por 10 min)	200 OK, token retornado	**POST** `https://compassuol.serverest.dev/login` Body: `{"email":"teste123@email.com","p`

				`assword":"123456"}` Verifique token no response.
1.6	US 002	Autenticar com senha inválida	401 Unauthorized, erro de autenticação	Mude `"password":"senhaerrada"` e envie o login.
1.7	US 002	Autenticar usuário não existente	401 Unauthorized, erro de autenticação	Use e-mail inexistente no body, ex.: `"email":"naoexiste@email.com"`.

🧪 Bateria 2: Operações Críticas em Produtos (/produtos)

Nº	User Story	Caso de Teste	Esperado	Passo a Passo no Postman
2.1	US 003	Criar produto válido (nome único, preço, descrição, quantidade)	201 Created, produto salvo corretamente	**POST** `https://compassuol.serverest.dev/produtos` Headers: `Authorization: API key` Body: `{"nome":"Produto Teste","preco":100,"descricao":"desc","quantidade":10}` Enviar.
2.2	US 003	Criar produto com nome já existente	400 Bad Request, erro de duplicação	Repita envio do 2.1 com mesmo `"nome"`.
2.3	US 003	Atualizar produto existente pelo ID com dados válidos	200 OK, produto atualizado	**PUT** `https://compassuol.serverest.dev/produtos/{id}` Headers: `Authorization: Bearer {token}` Body atualizado.
2.4	US 003	Atualizar produto com ID inexistente (PUT	201 Created, novo produto criado	Use ID inexistente no **PUT** e um body válido.

		→ cria novo produto válido)		
2.5	US 003	Deletar produto existente	200 OK, produto removido	**DELETE** `https://compassuol.serverest.dev/produtos/{id}` Headers: `Authorization: Bearer {token}`.
2.6	US 003	Deletar produto que está dentro de carrinho (bloqueado)	400 Bad Request, mensagem de bloqueio	1. Fazer uma requisição **POST /carrinhos** para criar um novo carrinho (se necessário). 2. Fazer uma requisição **POST /carrinhos/{id}/produtos** para **adicionar um produto bloqueado** ao carrinho (especificar o ID do produto bloqueado). 3. Fazer uma requisição **DELETE /carrinhos/{id}/produtos/{id Produto}** para tentar excluir o produto. 4. Verificar que o sistema retorna **HTTP 400 Bad Request** com a mensagem "Produto bloqueado para exclusão".
2.7	US 003	Operação em produto sem autenticação	401 Unauthorized	Faça **POST** ou **DELETE** sem

				header `Authorization`.

🧪 Bateria 3: Fluxo Completo de Carrinho (/carrinhos)

Nº	User Story	Caso de Teste	Esperado	Passo a Passo no Postman
3.1	US 004	Criar carrinho com produto válido, quantidade positiva	201 Created, carrinho criado	**POST** `https://compassuol.serverest.dev/carrinhos` Headers: `Authorization: Bearer {token}` Body: `{"produtos": [{"idProduto":"{id_produto}","quantidade":1}]}`
3.2	US 004	Criar carrinho com produto inexistente	400 Bad Request, erro de validação	Use `"idProduto":"inexistente"` no body.
3.3	US 004	Criar carrinho com quantidade zero ou negativa	400 Bad Request, erro de validação	Use `"quantidade":0` ou `"quantidade":-1`.
3.4	US 004	Atualizar carrinho existente (adicionar, remover item)	200 OK, carrinho atualizado	**PUT** `https://compassuol.serverest.dev/carrinhos/concluir-compra` (ou endpoint equivalente).
3.5	US 004	Listar carrinho do usuário autenticado	200 OK, retorna carrinho correto	**GET** `https://compassuol.serverest.dev/carrinhos` Headers: `Authorization: Bearer {token}`.
3.6	US 004	Deletar carrinho existente	200 OK, carrinho removido	**DELETE** `https://compassuol.serverest.dev/carrinhos/cancelar-compra`

				Headers: `Authorization: Bearer {token}`.
3.7	US 004	Tentar editar carrinho finalizado (status fechado)	400 Bad Request, operação bloqueada	Primeiro finalize compra, depois tente editar carrinho.
3.8	US 004	Usuário tentando acessar carrinho de outro usuário	403 Forbidden, acesso negado	Usar token de outro usuário para acessar **GET /carrinhos**.

🧪 Bateria 4: Cenários Negativos, Borda e PUT indireto

Nº	User Story	Caso de Teste	Esperado	Passo a Passo no Postman
4.1	US 001	Criar usuário usando PUT com e-mail já usado	400 Bad Request, erro de duplicação	**PUT** `https://compassuol.serverest.dev/usuarios/{id}` com body usando e-mail já existente.
4.2	US 001	PUT usuário com ID inexistente (criação de novo usuário)	201 Created, novo usuário criado	**PUT** `https://compassuol.serverest.dev/usuarios/{id_inexistente}` com body válido.
4.3	US 003	PUT produto com nome já usado	400 Bad Request, erro de duplicação	**PUT** `https://compassuol.serverest.dev/produtos/{id}` usando nome já existente no body.
4.4	US 002	Usar token expirado para acessar rota protegida	401 Unauthorized, sessão expirada	Use token expirado no header `Authorization` para **GET /produtos**.
4.5	US 003	Enviar requisição malformada (ex.: campos faltando) para /produtos	400 Bad Request, mensagem clara de erro	**POST** `https://compassuol.serverest.dev/produtos` com

				body incompleto (ex.: sem `nome`).
4.6	US 004	Forçar concorrência: dois usuários tentando alterar o mesmo carrinho	Um sucesso, outro erro (409 Conflict esperado, se implementado)	Simule duas requisições **PUT** simultâneas para o mesmo carrinho usando duas sessões Postman.

✅ Resumo das 4 Baterias

Bateria	Foco	Total de Testes
Bateria 1	Login, criação e permissões básicas	7
Bateria 2	Produtos: operações críticas	7
Bateria 3	Carrinhos: fluxo completo	8
Bateria 4	Cenários negativos e indiretos	6
Total		**28 testes**

🥔 9. Matriz de Risco

ID	Risco Identificado	Impacto	Probabilidade	Classificação (I x P)	Mitigação Proposta
R1	Falha no endpoint `/login` impedindo acesso de usuários	Alto	Médio	Alto	Implementar **testes automáticos diários** no `/login`; monitorar logs de autenticação.
R2	Cadastro de usuário duplicado no `/usuarios` sem bloqueio por email	Médio	Alto	Alto	Criar **testes automatizados de duplicidade**; validar unique constraint no banco de dados.
R3	Endpoint `/produtos` permitindo criação com preço ou quantidade inválida	Médio	Alto	Alto	Adicionar **validações de regras de negócio via testes de limite** e campos obrigatórios.

R4	`/carrinhos/concluir-compra` não invalida estoque corretamente	Alto	Médio	Alto	Realizar **testes integrados** validando a diminuição de estoque após compra; incluir testes de concorrência.
R5	Falha ao cancelar carrinho (`/carrinhos/cancelar-compra`) não desfaz reserva de estoque	Médio	Médio	Médio	Criar **testes de rollback** no cancelamento; revisar transações no backend.
R6	Deleção incorreta de produto ou usuário via `/produtos/{_id}` ou `/usuarios/{_id}`	Alto	Baixo	Médio	Implementar **testes de autorização e validação de ID existente antes de excluir**.
R7	Endpoint `/usuarios` ou `/produtos` aceitando `Content-Type` incorreto	Baixo	Alto	Médio	Criar **testes de headers obrigatórios** e validação de `Content-Type`.
R8	Retorno de **códigos HTTP incorretos** em falhas ou exceções	Médio	Médio	Médio	Criar **testes específicos para status codes esperados** em cada endpoint.
R9	Ausência de validação de campos obrigatórios no `/carrinhos`	Alto	Alto	Alto	Adicionar **testes negativos** para envio de payloads incompletos ou inválidos.
R10	API aceitando **valores nulos ou vazios** em campos obrigatórios	Médio	Alto	Alto	Incluir **testes de borda e negativos** para todos os campos obrigatórios no JSON de requisição.

📌 10. Cobertura de Testes

✅ Path Coverage (Cobertura de Endpoints)

A API ServeRest disponibiliza 9 endpoints principais:

- /usuarios
- /usuarios/{_id}
- /login
- /produtos
- /produtos/{_id}
- /carrinhos
- /carrinhos/{_id}
- /carrinhos/concluir-compra
- /carrinhos/cancelar-compra

Resultado:
Todos os endpoints foram contemplados nos cenários de teste planejados.
Cobertura estimada: `100%`

✅ Operator Coverage (Cobertura de Métodos HTTP)

Para cada endpoint, foram testados os seguintes métodos HTTP: `GET`, `POST`, `PUT` e `DELETE`.

Resultado:
Todos os métodos disponíveis na API foram abordados nos testes.
Cobertura estimada: `100%`

✅ Parameter Coverage (Cobertura de Parâmetros)

A cobertura dos parâmetros incluiu:

- Parâmetros de path, como :id para recursos específicos.
- Parâmetros de query, como filtros e paginação.
- Parâmetros de body, como nome, e-mail, senha, nome do produto, preço, etc.

Resultado:
Grande parte dos parâmetros documentados na API foi considerada nos cenários de teste, cobrindo entradas válidas e inválidas, mas ainda existem oportunidades de ampliação.
Cobertura estimada: `~85%`

✅ Parameter Value Coverage (Cobertura de Valores dos Parâmetros)

Foram testadas variações de valores, tais como:

- E-mails válidos e inválidos.
- Senhas curtas e longas.
- Campos obrigatórios e campos vazios.
- Produtos com estoque insuficiente ou preços zerados.

Resultado:
Os testes abordam uma variedade razoável de combinações, mas não exaustiva.
Cobertura estimada: `~75%`

✅ Content-Type Coverage

Todas as requisições foram enviadas com `Content-Type: application/json` e os testes validaram as respostas no mesmo formato.

Resultado:
A cobertura é satisfatória, porém limitada a apenas um tipo de content-type.
Cobertura estimada: `100% para application/json`

✅ Status Code Coverage

Foram mapeados e validados os principais códigos de status:

- 200 OK
- 201 Created
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 409 Conflict

Resultado:
Os testes abrangem os status codes esperados para fluxos positivos e negativos.
Cobertura estimada: `~90%`

🤖 11. Testes Candidatos à Automação (completo e desenvolvido)

🎯 Plano de Automação de Testes — Postman

Seção	Descrição
Ferramenta	🧰 **Postman + Newman** (execução automatizada via CLI ou pipelines CI/CD)
Escopo Inicial	🏆 **28 casos de teste priorizados**, divididos em 4 baterias: autenticação, produtos, carrinho, cenários negativos/borda
Abordagem de Automação	✔️ Automação de **cenários funcionais principais** ✔️ Scripts na aba **Tests** do Postman Collection ✔️ **Validações de status code, corpo da resposta, headers essenciais** ✔️ **Testes parametrizados com variáveis globais/ambiente**
Critério de Saída	✅ Todos os testes automatizados **executando com sucesso e assertivas cobrindo requisitos esperados**

	✅ Eventuais testes exploratórios serão documentados manualmente (fora da automação inicial)

📐 Roteiro para automação no Postman:

1. **Criar uma collection principal chamada** `ServeRest - API Tests`
2. Criar **pastas internas por endpoint**: `/usuarios`, `/produtos`, `/carrinhos`, `/login`
3. Em cada pasta, adicionar os métodos suportados (GET, POST, PUT, DELETE, etc)
4. Na aba **Pre-request Script**, incluir scripts de **autenticação** para gerar token automaticamente onde necessário
5. Na aba **Tests**, incluir **scripts de validação automática**, como:
   - **Asserção de status code esperado**
   - **Verificação de existência de campos obrigatórios na resposta**
   - **Validação de mensagens de erro e sucesso**
   - **Validação de schema básico da resposta (com pm.expect)**
6. Parametrizar variáveis globais/ambiente para URLs, tokens, IDs dinâmicos
7. Utilizar **Test Runner do Postman** para execução em lote
8. Exportar collection finalizada e também scripts Newman para execução CLI

📊 Tabela de Automação por Endpoint:

Nº	Caso de Teste	Endpoint	Método	Automação prevista no Postman
1.1	Criar usuário válido	/usuarios	POST	Assert status 201 + corpo JSON com id/email
1.2	Criar usuário com e-mail já existente	/usuarios	POST	Assert status 400 + msg de duplicação
1.3	Criar usuário com senha fora do limite	/usuarios	POST	Assert status 400 + msg de validação
1.4	Criar usuário com e-mail inválido	/usuarios	POST	Assert status 400 + msg de validação
1.5	Autenticar com usuário válido	/login	POST	Assert status 200 + salvar token Bearer
1.6	Autenticar com senha inválida	/login	POST	Assert status 401 + msg de erro
1.7	Autenticar usuário não existente	/login	POST	Assert status 401 + msg de erro

2.1	Criar produto válido	/produtos	POST	Assert status 201 + corpo JSON com id
2.2	Criar produto com nome já existente	/produtos	POST	Assert status 400 + msg de duplicação
2.3	Atualizar produto existente com dados válidos	/produtos/{id}	PUT	Assert status 200 + corpo atualizado
2.4	Atualizar produto com ID inexistente (cria novo)	/produtos/{id}	PUT	Assert status 201 + novo id retornado
2.5	Deletar produto existente	/produtos/{id}	DELETE	Assert status 200 + msg sucesso
2.6	Deletar produto que está em carrinho	/produtos/{id}	DELETE	Assert status 400 + msg bloqueio
2.7	Operação em produto sem autenticação	/produtos	POST	Assert status 401 + msg auth requerida
3.1	Criar carrinho com produto válido	/carrinhos	POST	Assert status 201 + id carrinho
3.2	Criar carrinho com produto inexistente	/carrinhos	POST	Assert status 400 + msg validação
3.3	Criar carrinho com quantidade zero/negativa	/carrinhos	POST	Assert status 400 + msg validação
3.4	Atualizar carrinho existente (adicionar/remover item)	/carrinhos/{id}	PUT	Assert status 200 + novo estado carrinho
3.5	Listar carrinho do usuário autenticado	/carrinhos	GET	Assert status 200 + corpo correto
3.6	Deletar carrinho existente	/carrinhos/{id}	DELETE	Assert status 200 + msg sucesso
3.7	Tentar editar carrinho finalizado	/carrinhos/{id}	PUT	Assert status 400 + msg bloqueio
3.8	Usuário acessando carrinho de outro usuário	/carrinhos/{id}	GET	Assert status 403 + msg acesso negado
4.1	Criar usuário via PUT com e-mail já usado	/usuarios/{id}	PUT	Assert status 400 + msg duplicação

4.2	PUT usuário com ID inexistente (criação de novo usuário)	/usuarios/{id}	PUT	Assert status 201 + novo id retornado
4.3	PUT produto com nome já usado	/produtos/{id}	PUT	Assert status 400 + msg duplicação
4.4	Usar token expirado para acessar rota protegida	/produtos	GET	Assert status 401 + msg token expirado
4.5	Enviar requisição malformada (ex.: campos faltando) para /produtos	/produtos	POST	Assert status 400 + msg erro campos faltam
4.6	Forçar concorrência: dois usuários tentando editar o mesmo carrinho simultaneamente	/carrinhos/{id}	PUT	Assert status 200 um, 409 Conflict outro

✍️ **Validações automatizadas específicas:**

Em **todos os endpoints**, serão aplicadas **asserções automáticas na aba Tests** para:

- ✅ Status code esperado (ex: `pm.response.to.have.status(200)`)
- ✅ Presença de campos essenciais no JSON (`pm.expect(json).to.have.property("nome")`)
- ✅ Conteúdo da mensagem de sucesso/erro (`pm.expect(json.message).to.eql("Usuário criado com sucesso")`)
- ✅ Validação de listas não vazias (`pm.expect(json.length).to.be.above(0)`)
- ✅ Validação de schema básico manual ou via plugin (opcional)