

# Plano de Testes 2 – ServerRest

## Informações Gerais

Apresentação

Objetivo

Escopo

Funcionalidades incluídas:

Fora de escopo:

Ambiente de Testes

Análise Inicial

Técnicas de Teste Aplicadas

Mapa Mental da Aplicação

Matriz de Rastreabilidade: Requisitos x Cenários de Teste

Cobertura de testes

Path Coverage (input)

Operator Coverage (input)

Análise de cobertura de testes com base em requisitos

Cenários de Teste Planejados com Priorização

Matriz de Risco

Testes Candidatos à Automação

Referências e Anexos

## Informações Gerais

### Apresentação

Este documento foi elaborado com o intuito de organizar e documentar o planejamento da rodada de testes referente à **API ServeRest**.

O plano serve como base para a execução controlada dos testes manuais e/ou automatizados, garantindo rastreabilidade entre requisitos e testes.


**Responsável pelo documento:** @Maria Eduarda Martins Rodrigues

**Data de criação:** 30 de mai. de 2025

**Última atualização:** 30 de mai. de 2025

**Sprint:** Sprint 06

### Objetivo

Validar o comportamento da API pública  **ServeRest**, assegurando o correto funcionamento dos fluxos principais — como autenticação, cadastro de usuários, gerenciamento de produtos e pedidos — conforme critérios definidos. Esta rodada de testes tem como foco verificar a estabilidade, a conformidade com as regras de negócio documentadas e a identificação de possíveis inconsistências funcionais nos endpoints disponíveis.

---

## Escopo [↗](#)

### Funcionalidades incluídas: [↗](#)


- **Cadastro e autenticação de usuários:** criação de novos usuários (com ou sem privilégios administrativos) e login via token JWT.
- **Gerenciamento de produtos:** criação, listagem, edição e exclusão de produtos cadastrados na base.
- **Gerenciamento de carrinho e pedidos:** adição de produtos ao carrinho, fechamento do pedido e verificação de status do mesmo.
- **Validação de regras de negócio:** testes com usuários comuns e administradores, controle de permissões e mensagens de erro esperadas.

### Fora de escopo: [↗](#)

- **Testes de performance e carga:** esta rodada não contempla validações de estresse, escalabilidade ou tempo de resposta sob alto volume de requisições.
- **Segurança avançada:** aspectos como injeção de SQL, XSS ou testes automatizados de vulnerabilidades.

---

## Ambiente de Testes [↗](#)

Item	Detalhes
Projeto	ServeRest
Versão	2.29.7
Tipo de Teste	<b>Teste Baseado em Requisitos</b> , particionamento de Equivalência, análise do Valor Limite, tabela de Decisão, transição de Estado, teste Baseado em Fluxo de Uso, teste Negativo, teste Exploratório e teste Automatizado
Período de Execução	30 de mai. de 2025 - 6 de jun. de 2025
Responsável(is)	Maria Eduarda Rodrigues
Ambiente	Testes
Acesso / URL	 <a href="#">ServeRest</a>

---

## Análise Inicial [↗](#)

Tipo de Análise	Detalhes
Requisitos	User Stories / Swagger (Os requisitos são bem detalhados e claros quanto aos cenários de teste, com excessão da área de carrinhos para a qual foi necessário extrair os requisitos do Swagger)
Riscos Identificados	Ambiente de testes instável, Falta de requisitos detalhados, Endpoints instáveis ou fora do ar durante a execução dos testes, Retornos genéricos em erros (sem mensagens claras ou status apropriado), Ambiguidade de permissões entre usuários comuns e administradores

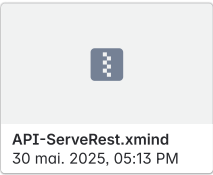
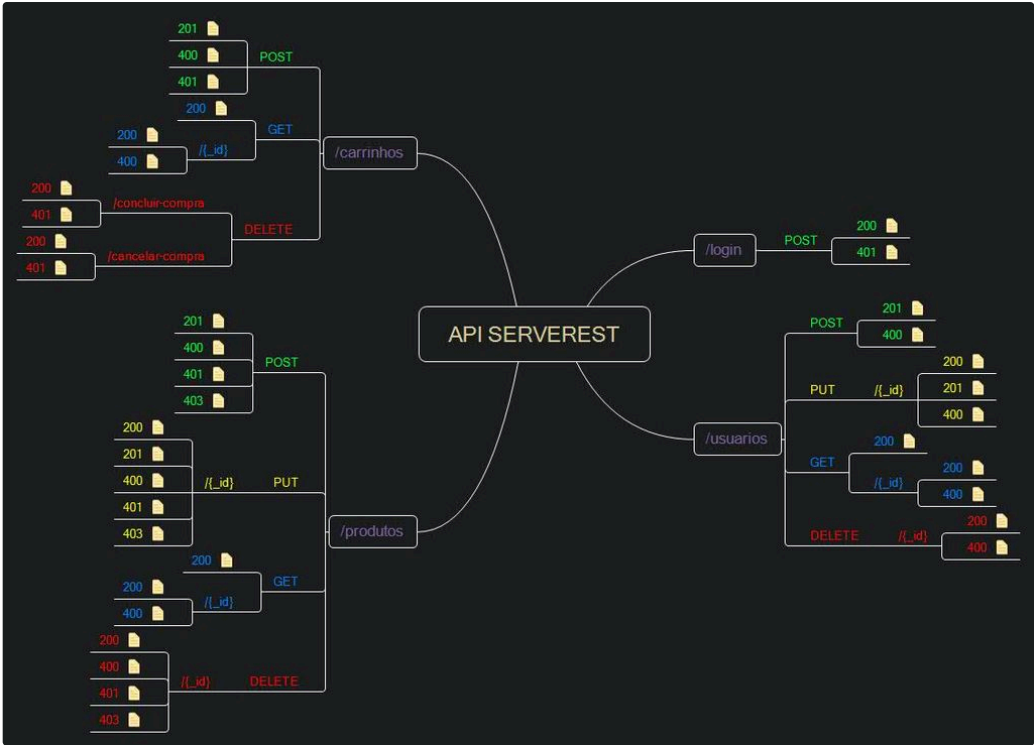
Histórico de Falhas	<a href="#">Link do relatório anterior.</a>
Apontamentos	A API do ServeRest foi analisada com base nos requisitos US001 a US004 extraídos das USs e do Swagger. A estrutura REST permitiu criar um plano de testes claro, com cenários baseados em CRUD, pré-requisitos bem definidos e rastreabilidade completa. O processo foi organizado, reproduzível e pronto para automação. Recomenda-se, futuramente, incluir testes não-funcionais em maior profundidade e explorar ferramentas além do Postman

---

## Técnicas de Teste Aplicadas

- **Teste Baseado em Requisitos (Requirements-Based Testing)** *[[Os testes são construídos com base direta nos critérios de aceitação e documentação Swagger.]]*
- **Particionamento de Equivalência (Equivalence Partitioning)** *[[Divide os dados de entrada em grupos válidos e inválidos para reduzir o número de casos de teste, mantendo a cobertura.]]*
- **Análise do Valor Limite (Boundary Value Analysis)** *[[Foca nos valores nos extremos (mínimos e máximos) permitidos, onde erros são mais comuns.]]*
- **Tabela de Decisão (Decision Table Testing)** *[[Combinações de condições de entrada são testadas para garantir a cobertura lógica do sistema.]]*
- **Transição de Estado (State Transition Testing)** *[[Valida o comportamento do sistema à medida que ele muda de estado com base em eventos ou condições.]]*
- **Teste Baseado em Fluxo de Uso (Use Case Testing)** *[[Verifica o comportamento do sistema a partir de fluxos reais de uso por parte do usuário.]]*
- **Teste Negativo (Negative Testing)** *[[Confirma que o sistema se comporta corretamente frente a dados inválidos ou ações proibidas.]]*
- **Teste Exploratório (Exploratory Testing)** *[[Executado sem scripts formais, com base na experiência do tester para descobrir falhas não previstas.]]*
- **Teste Automatizado (Automated Testing)** *[[Execução programática de testes para garantir repetibilidade, agilidade e regressão controlada.]]*

## Mapa Mental da Aplicação [↗](#)



## Matriz de Rastreabilidade: Requisitos x Cenários de Teste [↗](#)

ID Requisito	Requisito (Acceptance Criteria)	ID Cenário de Teste
US001-R01	Usuário deve possuir os campos NOME, E-MAIL, PASSWORD e ADMINISTRADOR	CT001, CT006, CT033, CT034
US001-R02	Não deverá ser possível fazer ações e chamadas para usuários inexistentes	CT008
US001-R03	Não deve ser possível criar um usuário com e-mail já utilizado (POST)	CT002
US001-R04	Caso não seja encontrado usuário com o ID informado no PUT, um novo usuário deverá ser criado	CT006

US001-R05	Não deve ser possível cadastrar usuário com e-mail já utilizado utilizando PUT	CT007
US001-R06	Não deverá ser possível cadastrar usuários com e-mails de provedor gmail e hotmail	CT001, CT005, CT006, CT033, CT034
US001-R07	Os e-mails devem seguir um padrão válido	CT001, CT003, CT006, CT033, CT034
US001-R08	As senhas devem possuir no mínimo 5 caracteres e no máximo 10 caracteres	CT001, CT004, CT006, CT033, CT034
US001-R09	Deverá ser possível listar todos os usuários cadastrados	CT009
US001-R10	Não é permitido excluir usuário com carrinho	CT010

ID Requisito	Requisito (Acceptance Criteria)	ID Cenário de Teste
US002-R01	Usuários não cadastrados não deverão conseguir autenticar	CT011
US002-R02	Usuários com senha inválida não deverão conseguir autenticar	CT012
US002-R03	No caso de não autenticação, deverá ser retornado um status code 401 (Unauthorized)	CT011, CT012
US002-R04	Usuários existentes e com a senha correta deverão ser autenticados	CT013, CT035
US002-R05	A autenticação deverá gerar um token Bearer	CT013, CT035
US002-R06	A duração da validade do token deverá ser de 10 minutos	CT014

ID Requisito	Requisito (Acceptance Criteria)	ID Cenário de Teste
US003-R01	Usuários não autenticados não devem conseguir realizar ações na rota de Produtos	CT015
US003-R02	Não deve ser possível realizar o cadastro de produtos com nomes já utilizados	CT016

US003-R03	Não deve ser possível excluir produtos que estão dentro de carrinhos	CT017
US003-R04	Caso não exista produto com o ID informado na hora do UPDATE, um novo produto deverá ser criado	CT018a, CT018b
US003-R05	Produtos criados através do PUT não poderão ter nomes previamente cadastrados	CT019
US003-R06	Só os administradores podem cadastrar, excluir ou editar um produto	CT020a, CT020b, CT020c
US003-R07	O sistema deve ter a funcionalidade de listar todos os produtos	CT021
US003-R08	O sistema deve ter a funcionalidade de buscar um produto pelo seu ID	CT022

(Para o end-point carrinhos, não há US então os requisitos são com base na documentação do Swagger).

ID Requisito	Requisito (Swagger / Regras de Negócio)	ID Cenário de Teste
US004-R01	O sistema deve permitir buscar um carrinho pelo ID	CT023
US004-R02	Um usuário autenticado pode criar <b>um único carrinho</b>	CT024, CT025
US004-R03	O carrinho é vinculado ao token do usuário autenticado	CT024, CT026
US004-R04	Não é permitido cadastrar produtos duplicados no mesmo carrinho	CT027
US004-R05	Não é permitido adicionar produtos inexistentes ou com estoque insuficiente	CT028, CT029
US004-R06	Ao concluir a compra, o carrinho do usuário devidamente autenticado é excluído	CT030
US004-R07	Ao cancelar a compra, o carrinho do usuário devidamente autenticado é	CT031

	excluído e o estoque dos produtos é reabastecido	
US004-R08	O sistema deve ter a funcionalidade de listar todos os carrinhos	CT032

## Cobertura de testes [↗](#)

### Path Coverage (input) [↗](#)

✔ 9 / 9 = 1 = 100%

### Operator Coverage (input) [↗](#)

✔ 16 / 16 = 1 = 100%

### Análise de cobertura de testes com base em requisitos [↗](#)

✔ 32 / 32 = 1 = 100%

## Cenários de Teste Planejados com Priorização [↗](#)

ID Cenário	Prioridade	Cenário de Teste	Pré-condições	Passos	Resultado Esperado
CT001	Alta	Criar usuário com dados válidos	Nenhuma	POST /usuarios JSON: { "nome": "Valido", "email": "valido@empresa.com", "password": "123456", "administrador": "true" }	201 { "message": "Cadastro realizado com sucesso", "_id": "<gerado>" }
CT002	Média	Criar usuário com e-mail duplicado	CT001 executado	POST /usuarios com mesmo e-mail	400 { "message": "Este email já está sendo usado" }
CT003	Média	Criar usuário com e-mail inválido	Nenhuma	POST /usuarios JSON: { "nome": "Invalido", "email": "email.com", "password": "123456", "administrador": "true" }	400 { "email": "email deve ser um email válido" }
CT004	Alta	Criar usuário com senha	Nenhuma	POST /usuarios JSON: { "nome": "SenhaCurta", "email": "valido2@empresa.com", "password": "123", "administrador": "true" }	400 { "password": "password deve ter entre 5 e

		fora do padrão			10 caracteres" } }
CT005	Alta	Criar usuário com e-mail de provedor proibido	Nenhuma	POST /usuarios JSON: { "nome": "Bloqueado", "email": "exemplo@gmail.com", "password": "123456", "administrador": "true" }	400 { "message": "Cadastro com email de provedor não permitido" } }
CT006	Alta	Criar novo usuário com PUT (ID inexistente)	Nenhuma	PUT /usuarios/999xyz JSON: { "nome": "NovoPUT", "email": "put@empresa.com", "password": "senha10", "administrador": "true" }	201 { "message": "Cadastro realizado com sucesso", "_id": "999xyz" } }
CT007	Média	PUT em usuário existente com e-mail de outro	CT001 + novo usuário com e-mail diferente	PUT /usuarios/{id} com JSON de outro e-mail existente	400 { "message": "Este email já está sendo usado" } }
CT008	Baixa	Ação em usuário inexistente (GET)	Nenhuma	GET /usuarios/999inexistente	400 ou 404 { "message": "Usuário não encontrado" } }
CT009	Baixa	Listar todos os usuários cadastrados	CT001 e CT006 executados	GET /usuarios	200 { "quantidade": ≥ 2, "usuarios": [...] } }
CT010	Baixa	Impedir exclusão de usuário com carrinho	Criar carrinho vinculado ao ID do CT001	DELETE /usuarios/{id}	400 { "message": "Não é permitido excluir usuário com carrinho ativo" } }
CT033	Baixa	Criar usuário com dados válidos (verificar formato de dados aceito)	Nenhuma	POST /usuarios JSON: { "nome": Valido, "email": valido@empresa.com, "password": 123456, "administrador": true } }	<p>Mas de acordo com o histórico (estava com inconsistência em relação à documentação):</p> <p>400 { "message": "Adicione</p>



					aspas em todos os valores. Para mais informações acesse a <a href="https://github.com/ServerRest/ServerRest/issues/225">issuehttps://github.com/ServerRest/ServerRest/issues/225</a>
<b>CT034</b>	Baixa	Criar usuário comum com dados válidos	Nenhuma	POST /usuarios <b>JSON:</b> { "nome": "Comum", "email": "comum@empresa.com", "password": "123456", "administrador": "false" }	201 { "message": "Cadastro realizado com sucesso", "_id": "<gerado>" }

ID Cenário	Prioridade	Cenário de Teste	Pré-condições	Passos	Resultado Esperado
<b>CT011</b>	Alta	Impedir autenticação de usuário inexistente	Nenhuma	POST /login <b>JSON:</b> { "email": "naoexiste@empresa.com", "password": "123456" }	401 { "message": "Email e/ou senha inválidos" }
<b>CT012</b>	Alta	Impedir autenticação com senha inválida	CT001 (usuário válido criado com senha "123456")	POST /login <b>JSON:</b> { "email": "valido@empresa.com", "password": "errada123" }	401 { "message": "Email e/ou senha inválidos" }
<b>CT013</b>	Alta	Autenticação bem-sucedida com usuário válido	CT001 (usuário válido criado com senha "123456")	POST /login <b>JSON:</b> { "email": "valido@empresa.com", "password": "123456" }	200 { "message": "Login realizado com sucesso", "authorization": "Bearer <token>" }
<b>CT014</b>	Baixa	Verificar validade do token por 10 minutos	CT013 (obter token válido)	1. Enviar requisição com Authorization: Bearer <token> dentro de 10 min. 2. Aguardar 11 min e repetir a requisição.	<b>Passo 1:</b> 200 { "message": "Acesso autorizado" }  <b>Passo 2:</b>

					401 { "message": "Token expirado ou inválido" }
<b>CT035</b>	Alta	Autenticação bem-sucedida com usuário comum válido	CT001 (usuário comum válido criado com senha "123456")	POST /login <b>JSON:</b> { "email": "comum@empresa.com", "password": "123456" }	200 { "message": "Login realizado com sucesso", "authorization": "Bearer <token>" }

ID Cenário	Prioridade	Cenário de Teste	Pré-condições	Passos	Resultado Esperado
<b>CT015</b>	Alta	Impedir acesso à rota de produtos sem autenticação	Nenhuma	GET /produtos sem token	401 { "message": "Token de acesso ausente, inválido ou expirado" }
<b>CT016</b>	Baixa	Criar produto com nome já existente	Produto criado com nome "Produto X"	POST /produtos <b>JSON:</b> { "nome": "Produto X", "preco": 99.9, "descricao": "Novo", "quantidade": 10 } com token admin	400 { "message": "Já existe produto com esse nome" }
<b>CT017</b>	Média	Impedir exclusão de produto presente em carrinho	Produto adicionado previamente a um carrinho	DELETE /produtos/{id} com token admin	400 { "message": "Produto não pode ser excluído, pois está em um carrinho" }
<b>CT018a</b>	Alta	PUT cria novo produto com ID inexistente e pequeno e nome único	Nenhum produto com ID "p999" nem com nome "Produto Y"	PUT /produtos/p999 <b>JSON:</b> { "nome": "Produto Y", "preco": "33", "descricao": "Novo produto", "quantidade": "5" } com token admin	201 { "message": "Cadastro realizado com sucesso", "_id": "p999" }  Mas de acordo com o histórico (estava com inconsistência em relação à

					documentação): 404 { "id": "id deve ter exatamente 16 caracteres alfanuméricos" }
CT018b	Alta	PUT cria novo produto com ID inexistente e nome único	Nenhum produto com ID "p999p999p999p999" nem com nome "Produto Y"	PUT /produtos/p999p999p999p999 <b>JSON:</b> { "nome": "Produto Y", "preco": 33.3, "descricao": "Novo produto", "quantidade": 5 } com token admin	201 { "message": "Cadastro realizado com sucesso", "_id": "..." }
CT019	Média	PUT falha se nome já existe para outro produto	Produto existente com nome "Produto X"	PUT /produtos/{id} <b>JSON:</b> { "nome": "Produto X", "preco": 10, "descricao": "Teste", "quantidade": 1 } com token admin	400 { "message": "Já existe produto com esse nome" }
CT020a	Alta	Impedir usuário comum de <b>cadastrar</b> produto	Usuário comum autenticado (não-admin)	POST /produtos com token de usuário comum	403 { "message": "Permissão negada: apenas administradores podem realizar esta ação" }
CT020b	Baixa	Impedir usuário comum de <b>excluir</b> produto	Usuário comum autenticado (não-admin)	DELETE /produtos/{id} com token de usuário comum	403 { "message": "Permissão negada: apenas administradores podem realizar esta ação" }
CT020c	Baixa	Impedir usuário comum de <b>editar</b> produto	Usuário comum autenticado (não-admin)	PUT /produtos/{id} com token de usuário comum	403 { "message": "Permissão negada: apenas administradores podem realizar esta ação" }
CT021	Baixa	Listar todos os	Pelo menos 2 produtos	GET /produtos com token válido	200 { "quantidade": ≥2, }

		produtos existentes	cadastrados		"produtos": [...] }
<b>CT022</b>	Baixa	Buscar produto pelo ID	Produto criado previamente	GET /produtos/{id} com token válido	200 { "nome": "...", "descricao": "...", ... }
CT036	Alta	POST de novo produto com valor quebrado com sucesso	Usuário autenticado	Enviar requisição POST { "nome": "Produto Teste", "preco": 99.90, "descricao": "Produto criado para fins de validação", "quantidade": 1 }	201 { "message": "Cadastro realizado com sucesso", "_id": "..." }
CT037	Alta	POST de novo produto com valor inteiro com sucesso	Usuário autenticado	Enviar requisição { "nome": "Produto Teste", "preco": "99", "descricao": "Produto criado para fins de validação", "quantidade": "1" }	201 { "message": "Cadastro realizado com sucesso", "_id": "..." }

ID Cenário	Prioridade	Cenário de Teste	Pré-condições	Passos	Resultado Esperado
<b>CT023</b>	Baixa	Buscar um carrinho pelo ID	Carrinho criado com ID conhecido	GET /carrinhos/{id} com token do dono	200 { "idCarrinho": "...", "produtos": [...], ... }
<b>CT024</b>	Média	Criar um único carrinho por usuário autenticado	Usuário autenticado sem carrinho	POST /carrinhos JSON: { "produtos": [{ "idProduto": "abc123", "quantidade": 1 } ] }	201 { "message": "Cadastro realizado com sucesso", "idCarrinho": "xyz" }
<b>CT025</b>	Média	Impedir criação de múltiplos carrinhos para o mesmo usuário	Usuário já possui carrinho criado (CT024)	POST /carrinhos novamente com mesmo token	400 { "message": "Não é permitido possuir mais de um carrinho ativo" }
<b>CT026</b>	Baixa	Carrinho é vinculado ao token	Carrinho criado por usuário A	GET /carrinhos/{id} com token de usuário B	403 { "message": "Acesso negado" }

		do usuário			ao carrinho solicitado" }
CT027	Média	Impedir produtos duplicados no carrinho	Produto A já adicionado ao carrinho	POST /carrinhos com o mesmo produto novamente	400 { "message": "Produto já adicionado ao carrinho" }
CT028	Alta	Impedir adicionar produto inexistente	Produto com ID inválido	POST /carrinhos{ "produtos": [{"idProduto": "inexistente", "quantidade": 1 }] }	404 { "message": "Produto não encontrado" }
CT029	Alta	Impedir adicionar produto com estoque insuficiente	Produto com estoque = 1	POST /carrinhos{ "produtos": [{"idProduto": "abc123", "quantidade": 5 }] }	400 { "message": "Estoque insuficiente para o produto solicitado" }
CT030	Alta	Concluir compra exclui o carrinho	Carrinho criado previamente	POST /carrinhos/concluir-compra com token do dono	200 { "message": "Registro excluído com sucesso   Não foi encontrado carrinho para esse usuário" }
CT031	Alta	Cancelar compra exclui o carrinho e reabastecer o estoque	Produto X no carrinho antes do cancelamento	DELETE /carrinhos/cancelar-compra com token do dono	200 { "message": "Registro excluído com sucesso   Não foi encontrado carrinho para esse usuário" }
CT032	Baixa	Listar todos os carrinhos existentes	Ao menos dois carrinhos existentes	GET /carrinhos com token admin	200 { "quantidade": ≥2, "carrinhos": [...] }

## Matriz de Risco

Risco	Impacto	Probabilidade	Ação de Mitigação
-------	---------	---------------	-------------------

Ambiente instável durante a execução	Média	Alta	Check diário com replanejamento
Falta de requisitos detalhados	Baixo	Média	Aceitar passivamente
Endpoints instáveis ou fora do ar durante a execução dos testes	Alta	Baixa	Reexecutar em horários alternativos
Retornos genéricos em erros (sem mensagens claras ou status apropriado)	Médio	Baixa	Documentar os comportamentos e reportar inconsistências
Ambiguidade de permissões entre usuários comuns e administradores	Médio	Baixa	Verificar todos os fluxos com diferentes perfis de usuário

## Testes Candidatos à Automação [↗](#)

ID	Candidato à Automação?	Justificativa para Automatizar	Como Automatizar com Robot Framework
CT001	✅ Sim	Fluxo fundamental e recorrente. Automação garante validação constante do cadastro.	<code>POST /usuarios</code> , validar status 201 e presença do <code>_id</code>
CT002	✅ Sim	Alta probabilidade de regressão. Erro de e-mail duplicado precisa ser testado sempre.	Executar CT001, depois novo <code>POST</code> com mesmo e-mail. Validar status 400
CT003	✅ Sim	Validações de campos são críticas e devem ser testadas automaticamente para evitar falhas básicas.	<code>POST</code> com e-mail inválido, validar mensagem de erro
CT004	✅ Sim	Verifica regra de negócio sensível (limite de senha). Rápida de automatizar, útil em cada release.	<code>POST</code> com senha curta, verificar resposta 400 e mensagem específica
CT005	✅ Sim	Regra de provedor bloqueado pode mudar. Ideal para regressão automática.	Enviar e-mail do tipo <code>@gmail.com</code> , validar erro 400

CT006	✔ Sim	Permite validar lógica de PUT com criação implícita, que costuma causar confusão.	PUT com ID inexistente, validar criação e retorno 201
CT007	✔ Sim	Evita bugs de conflito de dados na edição. Automatização evita quebras em cenários reais.	Criar dois usuários, fazer PUT com e-mail duplicado. Validar erro 400
CT008	✔ Sim	Cenário negativo recorrente. Ideal para ser validado sempre que houver refatorações.	GET com ID inválido, validar 404 ou 400
CT009	✔ Sim	Consulta simples e frequente. Importante garantir que listagem funcione em qualquer versão da API.	GET /usuarios, validar campo quantidade e estrutura da resposta
CT010	✔ Sim	Regras de negócio importantes devem ser validadas automaticamente para evitar falhas em exclusões.	Criar carrinho vinculado a usuário e tentar deletar
CT011	✔ Sim	Cenário básico de segurança. Ideal para smoke tests automatizados.	POST /login com usuário inexistente, validar 401
CT012	✔ Sim	Complementar ao anterior. Garante que autenticação falha com senha inválida.	Usar usuário válido com senha errada, validar erro
CT013	✔ Sim	Login é essencial em qualquer fluxo. Teste precisa estar automatizado.	POST /login, validar sucesso e token retornado
CT014	✔ Sim	Controle de expiração é crítico. Automatização pode validar tempo de vida com precisão e monitorar falhas.	Testar dentro e após 10 minutos com Sleep ou Wait Until Keyword

CT015	✓ Sim	Segurança da API depende da proteção de rotas. Ideal para teste contínuo.	GET /produtos sem token, validar 401
CT016	✓ Sim	Conflitos de nome são frequentes. Teste automatizado evita duplicidades acidentais.	Criar produto, tentar recriar com mesmo nome
CT017	✓ Sim	Exclusão de recursos sensíveis é um ponto crítico. Automatização previne erros graves.	Adicionar produto a carrinho e tentar deletar
CT018a	✓ Sim	Confirma o comportamento correto em falhas por ID inválido. Fácil de validar com automação.	PUT com ID inválido, validar erro 404
CT018b	✓ Sim	Criação via PUT deve funcionar com ID válido. Bom para regressão.	PUT com ID correto, validar retorno 201
CT019	✓ Sim	Nome duplicado em PUT é um erro comum. Ideal para teste de estabilidade da API.	PUT com nome já usado. Validar erro 400
CT020a	✓ Sim	Controle de permissões deve ser testado sempre. Automatizar evita falhas de segurança.	User comum tenta criar produto. Validar 403
CT020b	✓ Sim	Idem anterior. Exclusão por usuários indevidos não pode passar despercebida.	User comum tenta deletar produto. Validar 403
CT020c	✓ Sim	Idem anterior. Tentativa de edição deve ser bloqueada.	User comum tenta editar produto. Validar 403
CT021	✓ Sim	Listagem de produtos é base para vários testes. Ideal para health checks.	GET e validar estrutura do JSON e quantidade
CT022	✓ Sim	Busca por ID é comum e precisa funcionar	Criar produto, buscar pelo ID, validar conteúdo



		mesmo após múltiplas mudanças.	
CT023	✓ Sim	Busca de carrinho por ID é chave para operações do fluxo de compra.	Criar carrinho, obter ID e fazer GET
CT024	✓ Sim	Fluxo principal de compra. Cadastro de carrinho deve funcionar sempre.	POST com produtos válidos. Validar sucesso e ID
CT025	✓ Sim	Limite de carrinho por usuário evita bugs críticos. Automação garante consistência.	Criar carrinho, tentar criar outro com mesmo token. Validar erro 400
CT026	✓ Sim	Segurança por token é central. Ideal para automação com múltiplos perfis.	Criar com user A, acessar com user B. Validar 403
CT027	✓ Sim	Impede erro funcional frequente: duplicidade de itens.	Tentar adicionar mesmo item duas vezes. Validar erro 400
CT028	✓ Sim	Impede carrinhos com produtos inválidos. Teste automatizado garante integridade.	POST com ID inexistente. Validar 404
CT029	✓ Sim	Estoque insuficiente é uma regra crítica de negócio. Deve ser validada a cada entrega.	Criar produto com estoque 1. Tentar adicionar 5. Validar erro
CT030	✓ Sim	Conclusão de compra é o fim do fluxo. Crítico para automação.	POST para concluir, validar retorno e exclusão
CT031	✓ Sim	Cancelamento é reversão complexa. Automação garante retorno correto de estoque.	Criar carrinho, cancelar. Validar mensagem e reestoque
CT032	✓ Sim	Listagem de carrinhos é simples, útil em testes de visualização e auditoria.	GET /carrinhos , validar estrutura e quantidade

CT033	✔ Sim	Validações de payload são importantes. Mas precisa atenção com formato exigido (sem aspas).	POST com payload formatado corretamente. Validar resposta
CT034	✔ Sim	Cadastro de usuário comum é fluxo padrão. Automatização assegura estabilidade.	POST com "administrador": "false", validar resposta
CT035	✔ Sim	Login de usuário comum precisa estar no pipeline de testes de autenticação.	Login com dados válidos. Validar token
CT036	✔ Sim	Teste de tipos numéricos garante integridade dos dados. Automatizar previne erros de conversão.	POST com preco decimal. Validar criação
CT037	✔ Sim	Complementar ao anterior. Garante suporte a valores inteiros também.	

---

## Referências e Anexos [↗](#)

- Quadro de histórias no Jira ([Link](#))
- Documentação do Swager ([Link](#))
- Relatório de testes ([Link](#))