

# Atualização do Plano de Testes – API ServeRest

Projeto: <https://compassuol.serverest.dev/>

Versão: 1.0

Autor: @Thais Nogueira

Data: 2 de jun. de 2025 - 6 de jun. de 2025

## 1. Introdução

A aplicação Serverest simula uma loja virtual oferecendo endpoints para cadastro de usuários, autenticação e gestão de produtos e gerenciamento de carrinhos.

Este plano documenta os testes realizados diretamente sobre a API pública, com base na documentação Swagger oficial

Ser

veRest

## 2. Objetivo

Garantir que a API ServeRest esteja de acordo com os requisitos funcionais e não funcionais esperados, por meio de testes baseados nas User Stories fornecidas e na documentação Swagger. Identificar inconsistências, bugs e oportunidades de melhoria.

## 3. Escopo dos Testes

Incluído:

- Usuários: cadastro, login,atualização e remoção
- Login: autenticação
- Produtos: Cadastro, listagem, atualização, remoção
- Carrinhos de compras: criação, listagem, finalização e cancelamento

Fora do escopo:

- Testes de performance
- Integrações externas

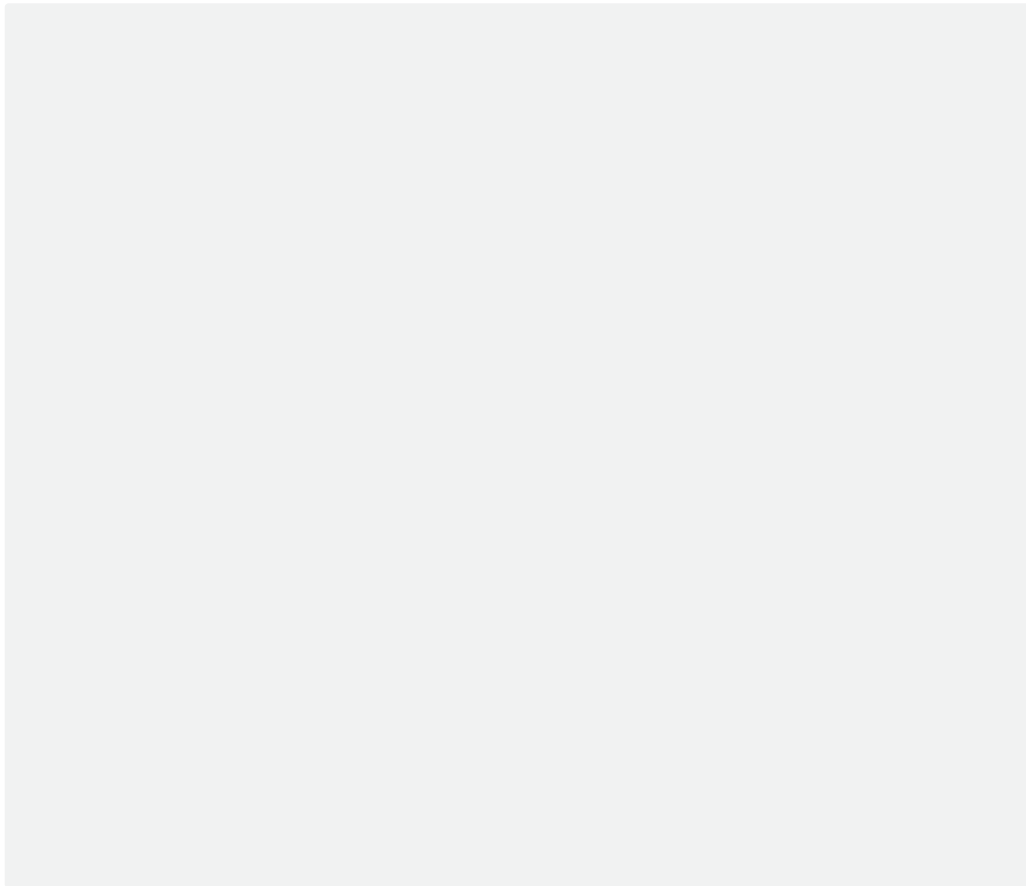
## 4. Referências

- Swagger: ServeRest
- User Stories: [US001 \(Usuários\)](#), [US002 \(Login\)](#), [US003 \(Produtos\)](#)
- Documentação interna: Confluence
- Gerenciamento de bugs e execução :Jira + Plugin QALity

## 5. Estratégia de Teste

- **Nível:** Testes de API
- **Tipos:** Funcionais, exploratórios, regressão e testes de segurança
  - **Ferramentas:** Swagger, Postman, Robot Framework, Jira (QALity), Confluence

## 6. Mapa Mental da Aplicação



## 7. Técnicas Aplicadas

- Partição de equivalência (ex: e-mail válido x inválido)
- Valor limite (ex: senhas com 4, 5, 10 caracteres)
- Teste exploratório com entradas inválidas
- Testes baseados em requisitos documentados (Swagger + User Stories)
- Análise de regras de negócio

## 8. Análise de teste

### 1. Usuários

Funcionalidades: Cadastro, atualização, listagem e remoção de usuários via endpoints da API /usuarios

**Regras de Negócio:** (Origem: User Story US001)

- E-mail deve ser único, válido e de domínio permitido (não pode ser Gmail ou Hotmail) → Particionamento de Equivalência:
- Senha deve ter entre 5 e 10 caracteres. → Análise de Valor Limite
- Nome, E-mail, Senha e Administrador são campos obrigatórios. → testes negativos
- PUT em ID inexistente deve criar um novo usuário. → teste de fluxo
- Operações para IDs inexistentes devem retornar erro adequado. → testes negativos

**Fluxos Principais:** (Origem: Swagger)

1. Cadastro de usuário válido → Particionamento de Equivalência
2. Edição de usuário existente → Particionamento de Equivalência

3. Listagem de usuários cadastrados → Teste de Fluxo

4. Exclusão de usuário por ID. → Teste de Fluxo

**Cenários Alternativos e Inválidos:** (Origem: User Story US001)

- Tentativa de cadastro com e-mail já existente.
- Cadastro com e-mail inválido.
- Domínio não permitido (Gmail, Hotmail).
- Senha fora do limite de caracteres.
- PUT em ID inexistente (cria um novo usuário).
- Operações (GET, PUT, DELETE) em ID inválido ou inexistente.

## 2. Login

**Funcionalidades:** Autenticação de usuário (POST /login)

**Regras de Negócio:** (Origem: User Story US002)

- Apenas usuários cadastrados podem se autenticar. → Particionamento de Equivalência
- Token JWT deve ser válido por 10 minutos
- Senha incorreta retorna 401 Unauthorized. → Teste de Segurança
- Token JWT deve ser gerado e ter validade de 10 minutos. → Análise de Valor Limite

**Fluxos Principais:** Login com credenciais corretas

Cenários Alternativos e Inválidos: (Origem: User Story US002)

- Login com e-mail inexistente. → Testes Negativos
- Login com senha incorreta. → Testes Negativos
- Login com campos vazios. → Testes Negativos
- Login com e-mail malformatado. → Particionamento de Equivalência
- Expiração do token após 10 minutos. → Análise de Valor Limite

## 3. Produtos

**Funcionalidades:**

- Cadastrar produtos (POST /produtos)
- Atualizar produtos (PUT /produtos/:id)
- Listar produtos (GET /produtos)
- Excluir produtos (DELETE /produtos/:id)

**Regras de Negócio:** (Origem: User Story US003)

- Somente usuários autenticados podem gerenciar produtos. → teste de segurança
- Não é permitido cadastrar produtos com nomes duplicados. → Particionamento de Equivalência:
- Não é permitido excluir produtos que estejam em carrinhos. → Particionamento de Equivalência:
- PUT em ID inexistente deve criar um novo produto.  
CRUD

**Fluxos Principais:** (Origem: Swagger)

1. Cadastro de produto válido. → Particionamento de Equivalência
2. Edição de produto existente. → Particionamento de Equivalência
3. Listagem de produtos cadastrados. → Teste de Fluxo

4. Exclusão de produto por ID. → *Teste de Fluxo*

**Cenários Alternativos e Inválidos:** (*Origem: User Story US003*)

- Acesso sem autenticação.
- Cadastro com nome já existente.
- PUT com ID inexistente (deve criar).
- PUT com nome já existente (deve falhar).
- DELETE de produto que está em um carrinho.

#### 4. Carrinho

**Funcionalidades:** (*Origem: Swagger*)

- Criar carrinho (POST /carrinhos)
- Listar carrinhos (GET /carrinhos)
- Adicionar e remover produtos em um carrinho (PUT /carrinhos/:id)
- Excluir carrinho (DELETE /carrinhos/:id)

**Regras de Negócio:** (*Origem: Swagger*)

- Apenas usuários autenticados podem criar carrinhos. → *Teste de Segurança*
- Só é possível adicionar produtos existentes. → *Particionamento de Equivalência*
- Produtos em carrinho não podem ser excluídos pela API de produtos. → *Teste de Fluxo*
- Finalizar carrinho só é permitido se ele tiver produtos válidos. → *Teste de Fluxo*

**Fluxos Principais:** (*Origem: Swagger*)

1. Criar carrinho com produtos válidos.
2. Adicionar e remover produtos.
3. Finalizar compra com sucesso.
4. Listar carrinho e verificar conteúdo.

**Cenários Alternativos e Inválidos:** (*Origem: Swagger*)

- Criar carrinho sem autenticação. → *Teste de Segurança*
- Adicionar produto inexistente. → *Particionamento de Equivalência*
- Tentar deletar produto em carrinho. → *Teste de Condição de Fronteira*
- Finalizar carrinho vazio. → *Testes Negativos*
- Enviar dados incompletos (campos obrigatórios ausentes). → *Testes Negativos*

## 9. Cenário de teste [↗](#)

**US001 Usuarios** [↗](#)

I D	Caso de Teste	Pré- condiç ão	Passos	Payload	Resultado Esperado	Pri ori
--------	---------------------	----------------------	--------	---------	-----------------------	------------

						da
C T - 0 0 1	Cadas tro de usuári o válido	API disponí vel e ambien te config urado	<ol style="list-style-type: none"> <li>1. Preparar dados válidos: nome, e-mail, senha e administrador</li> <li>2. Montar payload JSON com os dados</li> <li>3. Enviar requisição POST para /usuarios</li> <li>4. Validar status code 201</li> <li>5. Validar presença de _id e mensagem de sucesso na resposta</li> </ol>	{ "nome": "Ana Silva", "email": "Anateste@t este.com.br", "password": "teste", "administrad or": "true" }	201 {  "message": "Cadastro realizado com sucesso",  "_id": "" }	de a
C T - 0 0 2	Cadas tro com e-mail já existe nte	Usuári o já cadast rado no sistem a	<ol style="list-style-type: none"> <li>1. Cadastrar um usuário com um e-mail válido</li> <li>2. Tentar cadastrar outro usuário com o mesmo e-mail</li> <li>3. Enviar requisição POST para /usuarios</li> <li>4. Validar status code 400</li> <li>5. Validar mensagem de</li> </ol>	{ "nome": "João Silva", "email": "Anateste@t este.com.br m", "password": "teste", "administrad or": "true" }	Erro: "E- mail já cadastrado. "	Me dia

			erro sobre e-mail já existente			
C T - 0 0 3	Cadas tro com domíni o bloqu eado (Gmail , Hotma il)	API disponí vel e ambien te config urado	1. Montar payload com e-mail do tipo <a href="https://gmail.com">gmail.com</a> e hotmail.com 2. Enviar requisição POST para /usuarios 3. Validar status code 400 4. Validar mensagem de erro sobre domínio não permitido	{ "nome": "João Silva", "email": "joao@gmail. com",  "password": "teste", "administrad or": "true" }	400 (Bad Request)  "Domínio de e-mail não permitido."	Me dia

C T - 0 0 4	Cadas- tro sem camp- os obriga- tórios	API disponí- vel e ambien- te config- urado	<p>1. Montar payload com nome, e-mail, senha e administrador vazios</p> <p>2. Enviar requisição POST para /usuarios</p> <p>3. Validar status code 400</p> <p>4. Validar que todos os campos são indicados como obrigatórios na resposta</p>	{ "nome": "", "email": "", "password": "", "administrador": "" }	400 (Bad Request) "Campos obrigatórios não informados."	Alt a
C T - 0 0 5	Cadas- tro com senha fora do limite permitido	API disponí- vel e ambien- te config- urado	<p>1. Montar payload com senha menor que 5 ou maior que 10 caracteres</p> <p>2. Enviar requisição POST para /usuarios</p> <p>3. Validar status code 400</p> <p>4. Verificar mensagem de erro relacionada à senha</p>	{ "nome": "Maria", "email": "maria@teste.com", "password": "123", "administrador": "true" }	Erro: "A senha deve ter entre 5 e 10 caracteres."	Mé- dia

C T - 0 0 6	Alterar usuári o válido	Usuári o cadast rado no sistem a	<ol style="list-style-type: none"> <li>1. Cadastrar um usuário previamente</li> <li>2. Montar payload com novo nome e dados válidos</li> <li>3. Enviar requisição PUT para /usuarios/{_id} com o ID do usuário</li> <li>4. Validar status code 200 Verificar mensagem de sucesso na resposta</li> </ol>	_id:qTQ4ick6 SPLCger5  { "nome": "Ana", "email": "Anateste@t este.com.br", "password": "teste", "administrad or": "true" }	Status code 200 OK  {"message": : "Registro alterado com sucesso"}	Alt a
C T - 0 0 7	Alterar usuári o com e-mail duplic ado	Outro usuári o já existe com o mesmo e-mail	<ol style="list-style-type: none"> <li>1. Cadastrar dois usuários distintos</li> <li>2. Tentar atualizar o segundo usuário com o e-mail do primeiro</li> <li>3. Enviar requisição PUT para /usuarios/{_id} com o ID do segundo usuário</li> <li>4. Validar status code 400</li> </ol>	{ "nome": "Ana Silva", "email": "Anateste@t este.com.br", "password": "teste", "administrad or": "true" }	status code 400 bad request  Erro: "E- mail já cadastrado. "	Alt a



			5. Verificar mensagem de erro sobre e-mail duplicado			
CT-008	Alterar usuário sem campos obrigatórios		1. Cadastrar um usuário previamente 2. Montar payload com nome, e-mail, senha e administrador vazios 3. Enviar requisição PUT para /usuarios/{_id} com o ID do usuário 4. Validar status code 400 5. Verificar mensagens de erro indicando os campos obrigatórios ausentes	{ "nome": "", "email": "", "password": "", "administrador": "" }	status code 400 bad request  Erro: "Campos obrigatórios não informados." "	Média
CT-009	Alterar usuário com e-mail inválido		1. Cadastrar um usuário previamente 2. Montar payload com e-mail inválido ex: sem @	{ "nome": "Ana", "email": "Anateste.com.br", "password": "teste",	status code 400 bad request  Erro: "Formato de e-mail inválido."	Média

			<p>3. Enviar requisição PUT para /usuarios/{_id} com o ID do usuário</p> <p>4. Validar status code 400</p> <p>5. Verificar mensagem de erro sobre e-mail inválido</p>	"administrador": "true" }		
CT-010	Listagem de usuários válida	Usuários cadastrados no sistema	<p>1. Cadastrar pelo menos um usuário previamente</p> <p>2. Enviar requisição GET para /usuarios</p> <p>3. Validar status code 200</p> <p>4. Verificar que a lista de usuários foi retornada com os dados esperados</p>	-	Lista de usuários cadastrados.	Média
CT-011	Listagem de usuários sem autent	-	<p>1. Enviar requisição GET para /usuarios sem token de autenticação</p> <p>2. Validar status code 401</p>	-	Erro: "Não autorizado."	Alta

	icação		3. Verificar se a listagem for permitida sem autenticação			
C T - 0 1 2	Exclusão de usuário válido	Usuário existente e autenticado	1. Cadastrar um usuário previamente 2. Enviar requisição DELETE para /usuarios/{_id} com o ID do usuário 3. Validar status code 200 4. Verificar mensagem de sucesso na resposta	ID: UQEPVoJ0rG5Jgj4E	Mensagem: Registro excluído com sucesso	Media

## US002 Login [🔗](#)

ID	Caso de Teste	Pré-condição	Passos	Payload	Resultado Esperado	Prioridade

CT-013	Login com credenciais válidas	Usuário existente no sistema	<ol style="list-style-type: none"> <li>1. Cadastrar usuário com e-mail e senha válidos</li> <li>2. Montar payload com as credenciais</li> <li>3. Enviar requisição POST para /login</li> <li>4. Validar status code 200</li> <li>5. Verificar presença do token JWT na resposta</li> </ol>	<pre>{ "email": "Anateste@teste.com.br",   "password": "teste" }</pre>	<p>status code 200</p> <pre>{"message": "Login realizado com sucesso",   "authorization": "Bearer "}</pre>	Alta
CT-014	Login com e-mail não cadastrado	Nenhum usuário cadastrado no sistema	<ol style="list-style-type: none"> <li>1. Montar payload com e-mail não cadastrado</li> <li>2. Enviar requisição POST para /login</li> <li>3. Validar status code 401</li> <li>4. Verificar mensagem de erro indicando credenciais inválidas</li> </ol>	<pre>{ "email": "naoexiste@teste.com",   "password": "teste" }</pre>	<p>status code 401</p> <p>Erro: "Credenciais inválidas."</p>	Alta

CT-015	Login com senha incorreta	Usuário existente no sistema	<ol style="list-style-type: none"> <li>1. Cadastrar usuário com senha válida</li> <li>2. Montar payload com senha incorreta</li> <li>3. Enviar requisição POST para /login</li> <li>4. Validar status code 401</li> <li>5. Verificar mensagem de erro indicando falha de autenticação</li> </ol>	<pre>{   "email": "Anateste@teste.com.br",   "password": "senhaincorreta"} </pre>	<p>status code 401</p> <p>Erro: "Credenciais inválidas."</p>	Alta
CT-016	Expiração de token após 10 minutos	Token gerado no login	<ol style="list-style-type: none"> <li>1. Realizar login e salvar o token gerado</li> <li>2. Aguardar 10 minutos</li> <li>3. Tentar acessar um endpoint protegido com o token expirado</li> <li>4. Validar status code 401</li> <li>5. Verificar mensagem de erro</li> </ol>	<pre>{   "Authorization": "Bearer &lt;token_gerado&gt;" } </pre>	<p>Erro: "Token expirado."</p>	Média

			informando que o token expirou			
--	--	--	--------------------------------------	--	--	--

**US003 Produto** [🔗](#)

<b>I D</b>	<b>Caso de Teste</b>	<b>Pré- condi ção</b>	<b>Passos</b>	<b>Payload</b>	<b>Resultado Esperado</b>	<b>Pr io ri d a d e</b>
C T - 0 17	Cadas tro de produt o válido	Usuári o autenti cado no sistem a	1. Autenticar usuário e obter token JWT 2. Montar payload com nome, preço, descrição e quantidade válidos 3. Enviar requisição POST para /produtos com token no header 4. Validar status code 201	{ "nome": "Celular Samsung", "preco": 1500.00, "descricao": "Smartphone Samsung Galaxy", "quantidade": 10 }	"Cadastro realizado com sucesso"	Alt a

			5. Verificar mensagem de sucesso e presença do ID do produto			
CT-018	Cadastro de produto com nome duplicado	Produto já existente com o mesmo nome	<ol style="list-style-type: none"> <li>1. Cadastrar um produto com nome específico</li> <li>2. Montar payload com o mesmo nome e outros dados válidos</li> <li>3. Enviar requisição POST para /produtos</li> <li>4. Validar status code 400</li> <li>5. Verificar mensagem de erro informando que o produto já existe</li> </ol>	{ "nome": "Celular Samsung", "preco": 1500.00, "descricao": "Smartphone Samsung Galaxy", "quantidade": 10 }	Erro: "Produto já cadastrado."	Média
CT-019	Cadastro de produto com preço	API disponível e ambiente de teste	<ol style="list-style-type: none"> <li>1. Montar payload com preço negativo</li> <li>2. Enviar requisição</li> </ol>	{ "nome": "Celular Samsung", "preco": -1500.00, "descricao":	Erro: "Preço inválido."	Média

	negativo	configurado	POST para /produtos 3. Validar status code 400 4. Verificar mensagem de erro para o campo preço	"Smartphone Samsung Galaxy", "quantidade": 10 }		
C T - 0 2 0	Cadastro de produto com quantidade negativa	API disponível e ambiente de teste configurado	1. Montar payload com quantidade negativa 2. Enviar requisição POST para /produtos 3. Validar status code 400 4. Verificar mensagem de erro indicando campo quantidade inválido	{ "nome": "Celular Samsung", "preco": 1500.00, "descricao": "Smartphone Samsung Galaxy", "quantidade": -10 }	Erro: "Quantidade inválida."	Média
C T - 0 2 1	Atualização de produto válido	Produto existente no sistema	1. Cadastrar um produto previamente 2. Montar payload com dados atualizados	{ "nome": "Celular Samsung Atualizado", "preco": 1500.00, "descricao": "Smartphone	JSON contendo: nome , preco , descricao , quantidade	Alt



			<p>3. Enviar requisição PUT para /produtos/{_id} com token JWT</p> <p>4. Validar status code 200</p> <p>5. Verificar mensagem de sucesso e alterações aplicadas</p>	<p>Samsung Galaxy Atualizado", "quantidade": 10 }</p>		
C T - 0 2 2	Atualização de produto com nome duplicado	Produto existente com o mesmo nome	<p>1. Cadastrar dois produtos distintos</p> <p>2. Tentar atualizar o segundo com o nome do primeiro</p> <p>3. Enviar requisição PUT para /produtos/{_id}</p> <p>4. Validar status code 400</p> <p>5. Verificar mensagem de erro sobre nome duplicado</p>	<p>{ "nome": "Celular Samsung", "preco": 1500.00, "descricao": "Smartphone Galaxy", "quantidade": 10 }</p>	<p>Erro: "Nome do produto já cadastrado ."</p>	M e d i a

C T - 0 2 3	Listagem de produtos válida	Produtos cadastrados e usuário autenticado	<ol style="list-style-type: none"> <li>1. Autenticar usuário e obter token JWT</li> <li>2. Enviar requisição GET para /produtos com token JWT</li> <li>3. Validar status code 200</li> <li>4. Verificar que a lista contém produtos válidos</li> </ol>	-	Lista de produtos gerados	Alta
C T - 0 2 4	Listagem sem autenticação	Sem autenticação	<ol style="list-style-type: none"> <li>1. Enviar requisição GET para /produtos sem autenticação</li> <li>2. Validar status code 401</li> <li>3. Verificar mensagem de erro indicando falta de autenticação</li> </ol>	-	Erro: "Não autorizado."	Média
C T - 0	Exclusão de produto válido	1. Cadastrar produto	1. Enviar requisição para excluir o produto.	_id: jcAMnCpGiqEF AE7S	Produto removido do sistema.	Alta

2  
5

o  
se  
m  
vín  
cul  
o  
co  
m  
car  
rinh  
o

2. Validar retorno  
de sucesso.

2. Env  
iar  
req  
uisi  
ção  
DE  
LET  
E  
par  
a  
/pr  
odu  
tos/  
{\_id  
}  
co  
m  
aut  
enti  
caç  
ão

3. Vali  
dar

		stat us cod e 200 4. Veri fica r me nsa ge m de suc ess o				
C T - 0 2 6	Exclus ão de produt o vincul ado a um carrinh o	Produt o associ ado a um carrinh o	1. Criar carrinho com um produto 2. Tentar excluir esse produto com DELETE /produtos/{_id} 3. Validar status code 400 4. Verificar mensagem indicando vínculo com carrinho	Id: lJRn5q3HI8djD 3gTlJRn5q3HI8 djD3gT	Erro: "Produto vinculado a um carrinho. Não pode ser excluído."	Alt a

ID	Caso de Teste	Pré-condição	Passos	Payload	Resultado Esperado	Prioridade
CT-027	Criação de carrinho válido	Usuário autenticado e produto existente	<ol style="list-style-type: none"><li>1. Cadastrar produto e obter ID</li><li>2. Autenticar usuário</li><li>3. Montar payload com idProduto e quantidade</li><li>4. Enviar requisição POST para /carrinhos</li><li>5. Validar status code 201</li><li>6. Verificar ID do carrinho na resposta</li></ol>	<pre>{   "produtos": [     {       "idProduto":       "lJRn5q3HI8djD3gT",       "quantidade":       1     }   ] }</pre>	produto adicionado ao carrinho	Alta
CT-028	Criação de carrinho sem autenticação		<ol style="list-style-type: none"><li>1. Montar payload com produto válido</li><li>2. Enviar requisição POST para /carrinhos sem autenticação</li><li>3. Validar status code 401</li></ol>	<pre>{   "produtos": [     {       "idProduto":       "lJRn5q3HI8djD3gT",       "quantidade":       1     }   ] }</pre>	Erro: "Não autorizado."	Média

			4. Verificar mensagem de erro de autenticação	<pre> } ] } </pre>		
CT-029	Criação de carrinho com produto inexistente	Produto não está cadastrado no sistema	1. Montar payload com ID de produto inexistente 2. Enviar requisição POST para /carrinhos 3. Validar status code 400 4. Verificar mensagem de erro	<pre> { "produtos": [ { "idProduto": "produtoInvalido", "quantidade": 2 } ] } </pre>	Erro: "Produto não encontrado."	Média
CT-030	Criação de carrinho com produto duplicado	Produto duplicado no mesmo carrinho	1. Montar payload com dois itens com mesmo idProduto 2. Enviar requisição POST para /carrinhos 3. Validar status code 400 4. Verificar mensagem de erro sobre produto duplicado	<pre> { "produtos": [ { "idProduto": "umproduto", "quantidade": 2 }, { "idProduto": "umproduto", "quantidade": 2 } ] } </pre>	Erro: "Produto duplicado no carrinho."	Média
CT-031	Visualização de carrinho válido	Carrinho criado previamente	1. Criar carrinho e obter ID 2. Autenticar usuário 3. Enviar requisição GET para /carrinhos/{_id}	<pre> "_id": "Qwieyo2PAo zIXS7Y" </pre>	visualizar carrinho	Alta

			4. Validar status code 200 5. Verificar se produtos estão presentes na resposta			
CT-032	Finalização de compra com carrinho válido	Carrinho criado previamente e autenticado	1. Criar carrinho com produtos válidos 2. Autenticar usuário 3. Enviar requisição DELETE para /carrinhos/concluir-compra 4. Validar status code 200 5. Verificar mensagem de sucesso	-	"Carrinho excluído com sucesso"	Alta
CT-033	Finalização de compra e validação de esvaziamento	Compra finalizada previamente	1. Finalizar compra de carrinho existente 2. Enviar GET para /carrinhos/{_id} 3. Validar que carrinho não existe mais	-	Retorno de carrinho vazio após finalizar compra.	Alta

CT-034	Cancelamento de compra com carrinho válido	Carrinho criado previamente e autenticado	<ol style="list-style-type: none"> <li>1. Enviar requisição DELETE para /carrinhos/cancelar-compra</li> <li>2. Validar status code 200</li> <li>3. Verificar mensagem de cancelamento e estoque reabastecido</li> </ol>		"Compra cancelada com sucesso e produtos retornados ao estoque"	Alta
CT-035	Cancelamento de compra sem autenticação		<ol style="list-style-type: none"> <li>1. Criar carrinho com autenticação</li> <li>2. Enviar requisição DELETE para /carrinhos/cancelar-compra sem autenticação</li> <li>3. Validar status code 401</li> <li>4. Verificar mensagem de erro</li> </ol>		Erro: "Não autorizado."	Alta

## 10. Matriz de Rastreabilidade [🔗](#)

User Story	Casos de Teste Relacionados	Justificativa
------------	-----------------------------	---------------



<b>US001 – Usuários</b>	CT-001, CT-002, CT-004, CT-005, CT-006, CT-007, CT-008, CT-009, CT-010, CT-011, CT-012	Cobertura completa das regras de negócio sobre cadastro, atualização, listagem e exclusão de usuários, incluindo validações de formato, obrigatoriedade e unicidade.
<b>US002 – Login</b>	CT-013, CT-014, CT-015, CT-016	Garante os testes essenciais de autenticação, tratamento de credenciais inválidas e expiração do token JWT.
<b>US003 – Produtos</b>	CT-017, CT-018, CT-019, CT-020, CT-021, CT-022, CT-023, CT-024, CT-025, CT-026	Validações relacionadas ao ciclo de vida dos produtos: cadastro, atualização, listagem e exclusão, com foco em dados válidos e regras de negócio como nome duplicado ou vínculo com carrinho.
<b>US004 – Carrinho</b>	CT-027, CT-028, CT-029, CT-030, CT-031, CT-032, CT-033, CT-034, CT-035	Abrange todo o processo de criação, visualização, finalização e cancelamento de carrinho, incluindo segurança, duplicidade, validações de produtos e integridade dos dados.

## 11. Matriz de Risco [🔗](#)

Caso de Teste	Nível de Risco	Justificativa
CT-001	Alta	Fluxo base de cadastro; falhas podem impedir novos usuários.
CT-002	Média	Erro esperado por duplicidade de e-mail, não bloqueia sistema.
CT-003	Média	Validação importante, mas não impede funcionalidades críticas.
CT-004	Alta	Campos obrigatórios são essenciais para integridade dos dados.
CT-005	Média	Validação de campo individual, não impede operações principais.
CT-006	Alta	Atualização correta garante consistência de dados do usuário.
CT-007	Alta	Pode causar inconsistência em dados se e-mail duplicado for aceito.
CT-008	Média	Falha esperada, mas não crítica. Garante feedback apropriado.

CT-009	Média	Validação de formato de e-mail, importante mas não crítica.
CT-010	Média	Listagem de dados; falha impacta visualização, não operação.
CT-011	Alta	Segurança – acesso indevido a dados confidenciais.
CT-012	Média	Remoção de usuário; falhas não impedem fluxo geral.
CT-013	Alta	Login é essencial para autenticação e segurança.
CT-014	Alta	Segurança – autenticação incorreta deve ser rejeitada.
CT-015	Alta	Validação de senha protege contra acesso indevido.
CT-016	Média	Testa segurança temporal; importante, mas não impeditivo imediato.
CT-017	Alta	Cadastro de produto é fundamental para

		operação da loja.
CT-018	Média	Validação de duplicidade de nome é importante, mas contornável.
CT-019	Média	Preço negativo deve ser tratado, mas não bloqueia sistema.
CT-020	Média	Validação de estoque; relevante, mas não crítica.
CT-021	Alta	Edição correta garante integridade do catálogo.
CT-022	Média	Reforça controle de duplicidade, sem impacto direto.
CT-023	Alta	Listagem é fundamental para exibição do catálogo.
CT-024	Média	Validação de segurança; falha não afeta funcionalidade do backend.
CT-025	Alta	Exclusão de item pode afetar vendas e estoque.

CT-026	Alta	Evita falha de integridade por remover item em uso.
CT-027	Alta	Criação de carrinho é essencial para conversão de vendas.
CT-028	Média	Segurança de acesso, importante mas com fallback possível.
CT-029	Média	Garante consistência de dados na seleção de produtos.
CT-030	Média	Previne duplicidade em pedidos, relevante mas não crítica.
CT-031	Alta	Visualização do carrinho é essencial para finalizar compra.
CT-032	Alta	Conclusão de compra representa a etapa final de conversão.
CT-033	Alta	Valida limpeza de dados após a conclusão de compra.
CT-034	Alta	Garante controle do processo de cancelamento.

CT-035	Alta	Segurança no cancelamento; operação sem autenticação é risco.
--------	------	---

## 12. Priorização da execução dos cenários de teste



Prioridade	Casos de Teste	Justificativa
<b>Alta</b>	CT-001, CT-004, CT-006, CT-007, CT-011, CT-013, CT-014, CT-015, CT-017, CT-021, CT-023, CT-025, CT-026, CT-027, CT-031, CT-032, CT-033, CT-034, CT-035	Casos que afetam diretamente as funcionalidades principais da aplicação (autenticação, cadastro, produtos, finalização de compra). Erros nesses pontos comprometem o uso do sistema e a segurança.
<b>Média</b>	CT-002, CT-003, CT-005, CT-008, CT-009, CT-010, CT-012, CT-016, CT-018, CT-019, CT-020, CT-022, CT-024, CT-028, CT-029, CT-030	Casos que reforçam regras de negócio e consistência dos dados, mas que, se falharem, não impedem o funcionamento geral da aplicação.

## 13. Cobertura de Testes

Cobertura (%) = (Caso de testes mapeados : / Casos de teste planejado:) × 100

Total de Casos de planejado:35

✔ Cobertura de Testes: 100%

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

Os testes selecionados como candidatos à automação têm como foco garantir a consistência, estabilidade e validação contínua das funcionalidades essenciais da aplicação. São fluxos que percorrem ações críticas do sistema e que são ideais para automação. Esses testes cobrem desde o cadastro de usuário, passando por autenticação, cadastro de produtos, adição ao carrinho, até a exclusão e cancelamento de carrinhos. Ao automatizá-los, assegura-se que regras de negócio fundamentais sejam continuamente validadas, principalmente em ciclos de regressão.

Etapa	Casos de Teste	Objetivo
1. Cadastro de Usuário	CT-001, CT-006,CT-010,CT-012	Validar criação e atualização com dados válidos
2. Login de Usuário	CT-013, CT-014, CT-015, CT-016	Verificar autenticação, erros de login e validade do token
3. Cadastro de Produto	CT-017,CT-021,CT-023,CT-025,CT-026	Garantir que produtos válidos são criados e atualizados
4. Criação de Carrinho	CT-027, CT-029,	Testar criação com produtos válidos e inválidos.
5. Visualização do Carrinho	CT-031	Confirmar dados no carrinho antes da compra
6. Finalização da Compra	CT-032, CT-033	Validar o sucesso da compra e o

		esvaziamento do carrinho
<b>7. Cancelamento da Compra</b>	CT-034, CT-035	Testar reversão da compra com autenticação

### Observações Finais: [🔗](#)

- Os testes manuais serão executados via QAlity para evidência e acompanhamento
- A automação utilizará o Robot Framework, com keywords reutilizáveis e estrutura modular
- Repositório Git será mantido atualizado diariamente
- Tarefa extra com EC2 será executada conforme disponibilidade.