

Plano de Teste - ServeRest (compassuol.serverest.dev)

🔗 Challenge - Sprint 4/Semana 8 Plano de Teste Versão 1.7

@Pedro Afonso de Alencar Silva

📜 Histórico de Revisões 🔗

Histórico de Revisões		
Versão	Data	Descrição
1.0	5 de mai. de 2025	Versão Inicial
1.1	5 de mai. de 2025	Preenchimento dos tópicos de 1 a 3.
1.2	6 de mai. de 2025	Preenchimento dos tópicos de 4 a 11.
1.3	7 de mai. de 2025	Realização dos testes referentes ao tópico 7 e sua rastreabilidade.
1.4	8 de mai. de 2025	Inclusão do tópico 10. Cobertura de Testes e 11. Artefatos Gerados. OBS: Tópico 11 da v1.2, agora é 9.
1.5	8 de mai. de 2025	Inclusão e reenchimento do Tópico 12.
1.6	8 de mai. de 2025	Inclusão do índice.
1.7	9 de mai. de 2025	Refinamento Geral

📋 Índice 🔗

1. 📄 Apresentação
2. 🎯 Objetivo
3. 💬 Escopo
4. 🔎 Análise
5. 🛠️ Técnicas Aplicadas
6. 🌐 Mapa Mental da API
7. 🌐 Cenários de Teste Planejados + Rastreabilidade

8. Priorização dos Cenários
 9. Matriz de Risco
 10. Cobertura de Testes
 11. Testes Candidatos à Automação
 12. Artefatos Gerados
 13. Considerações sobre Itens Omitidos
-

Plano de Teste

1. Apresentação

O **ServeRest** é uma API REST gratuita que simula as **funcionalidades de uma loja virtual**, tendo como **principal objetivo** servir como **ambiente de estudos para testes de APIs**. Ela disponibiliza rotas para gerenciamento de **usuários, autenticação, produtos e carrinhos**, permitindo a execução de testes funcionais, de negócio e exploratórios.

2. Objetivo

Este plano de testes visa assegurar a conformidade da API ServeRest com as **regras de negócio estabelecidas nas User Stories**, complementando as definições técnicas presentes na documentação Swagger. O foco está em validar cenários reais e alternativos, inclusive aqueles não explicitamente descritos, garantindo maior **confiabilidade, rastreabilidade e qualidade** da aplicação.

Adicionalmente, o plano visa identificar cenários candidatos à **automação de testes**, apoiar a cobertura de testes baseada em risco e gerar evidências claras de execução conforme critérios de aceite.

3. Escopo

Dentro do Escopo:

Este plano de testes contempla a validação funcional e de regras de negócios dos seguintes módulos da API ServeRest:

OBS: As US inseridas no JIRA, são as mesmas definidas no Learning.

Type	Key	Resumo	Responsável	Prioridade	Status	Atualizado(a)
	S8C0-8	BR004 - Ambiguidade na regra de negócio para ...	Pedro Afonso de...	Medi...	TAREFAS P...	9 de mai. de 2025
	S8C0-7	BR003 - API permite cadastro de usuário com se...	Pedro Afonso de...	High	TAREFAS P...	8 de mai. de 2025
	S8C0-6	BR002 - API permite cadastro de usuário com se...	Pedro Afonso de...	High	TAREFAS P...	8 de mai. de 2025
	S8C0-5	BR001 - API permite cadastro com e-mail de dom...	Pedro Afonso de...	High	TAREFAS P...	8 de mai. de 2025
	S8C0-4	Testes da API ServeRest - Validação de Regras d...	Pedro Afonso de...	Medi...	TAREFAS P...	5 de mai. de 2025
	S8C0-3	US003 - [API] Gerenciamento de Produtos	Pedro Afonso de...	Medi...	TAREFAS P...	8 de mai. de 2025
	S8C0-2	US002 - [API] Login	Pedro Afonso de...	Medi...	TAREFAS P...	5 de mai. de 2025
	S8C0-1	US001 - [API] Cadastro de Usuários	Pedro Afonso de...	Medi...	TAREFAS P...	5 de mai. de 2025

8 itens

Sincronizado agora

- **Módulo Carrinhos** (Mesmo não contemplado nas User Stories);
- **Evidências de execução e testes candidatos à automação**, conforme critérios definidos no plano.

Fora do Escopo: 🔗

Os seguintes itens não serão validados neste plano de testes:

- **Testes exploratórios:** não incluídos nesta fase do plano de testes, por foco exclusivo em **testes roteirizados** baseados nas User Stories e Swagger.
- **Testes de carga, desempenho ou segurança** da API;
- Execução e validação em diferentes ambientes (como mobile ou frontend);
- **Infraestrutura técnica**, como banco de dados, escalabilidade, configuração de rede ou servidores;
- Interações externas ou simulações reais de fluxo completo de compra.

4. 🔎 Análise 🔗

A documentação técnica da API ServeRest, acessada via Swagger, descreve claramente os endpoints disponíveis, seus parâmetros, respostas esperadas e status HTTP. No entanto, ao analisar as User Stories do desafio, foi possível identificar regras de negócio e critérios que **não estão documentados no Swagger**, sendo essenciais para garantir a conformidade funcional e comportamental da aplicação.

Principais gaps identificados: 🔗

- **Restrições de e-mail:** O Swagger não menciona a proibição de e-mails dos domínios `gmail.com` e `hotmail.com`, mas essa regra aparece nos critérios de aceite da US001.
- **Validação de senhas:** O Swagger não impõe limites de 5 a 10 caracteres na senha, mas essa restrição é definida nas User Stories.
- **Comportamento do PUT em Usuários e Produtos:** Embora documentado parcialmente, as User Stories deixam claro que, se o ID não for encontrado, um novo cadastro deve ser feito — essa lógica não é explicitamente abordada no Swagger.
- **Autorização via token:** O Swagger apresenta o uso de token, mas não detalha o tempo de expiração (10 minutos), como descrito na US002.
- **Exclusão condicional de produtos:** A US003 define que não se pode excluir produtos vinculados a carrinhos, o que também não é detalhado tecnicamente no Swagger.
- **Módulo Carrinhos:** Está documentado no Swagger com rotas e respostas, mas **não está contemplado nas User Stories**.

Diante disso, a análise de testes será guiada tanto pelo Swagger quanto pelas User Stories, assegurando cobertura de requisitos técnicos e de negócio, inclusive para comportamentos não documentados.

5. 🔧 Técnicas Aplicadas 🔗

- A abordagem de testes adotada neste plano combinou técnicas funcionais **manuais e automatizadas**, com foco na validação das **regras de negócio descritas nas User Stories** e no comportamento técnico documentado no **Swagger da API ServeRest**. Os testes foram baseados em **cenários roteirizados planejados previamente (CT001 a CT025)**, sem execução de testes exploratórios livres.

📌 Técnicas Manuais Utilizadas 🔗

• Caixa Preta:

A validação foi realizada por meio da técnica de caixa preta, verificando o comportamento da API com base nas entradas e saídas documentadas, sem acesso ou análise do código-fonte.

• Particionamento de Equivalência (Parcial):

A técnica foi aplicada **apenas aos valores previstos nos cenários planejados**, como e-mails válidos e inválidos definidos nas User Stories. **Não foram criadas classes adicionais de equivalência ou valores além do roteiro**.

• Análise de Valor Limite (Parcial):

A técnica foi aplicada **apenas aos testes de senhas**, com validação de limites mínimo (5) e máximo (10 caracteres), incluindo tentativas fora dos limites (4 e 11 caracteres). **Não foi estendida a outros parâmetros como preço, quantidade ou IDs**.

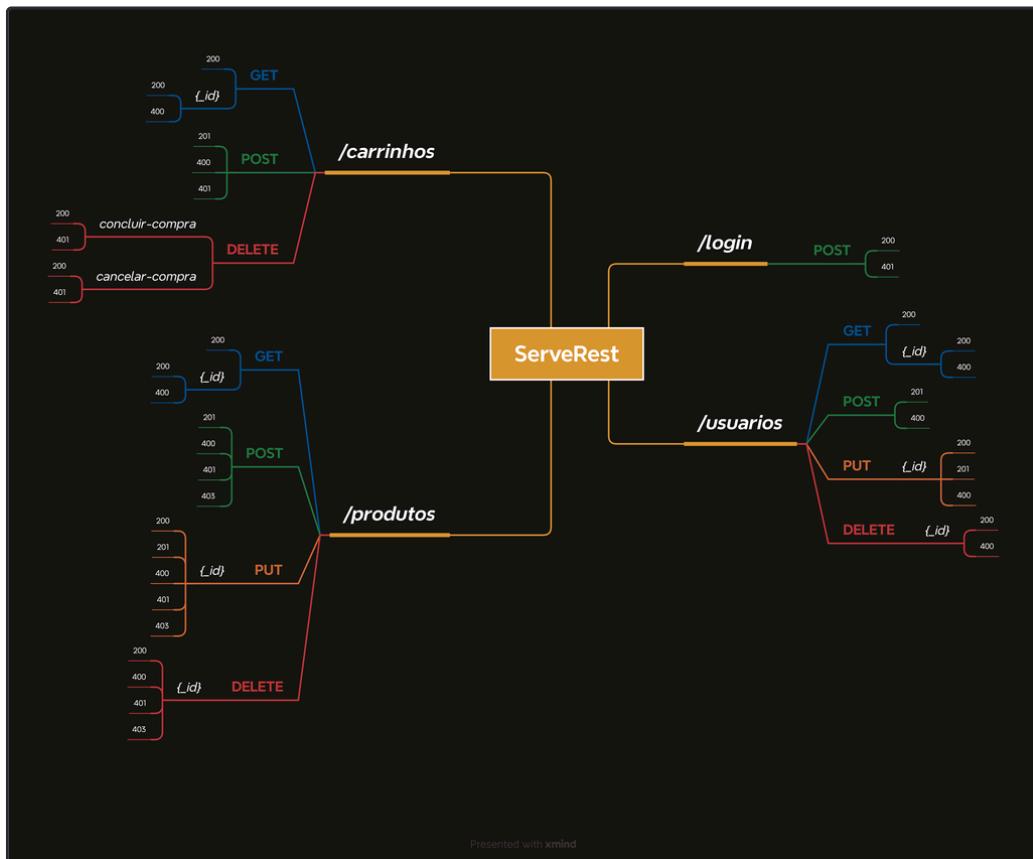
- Cobertura de Códigos de Resposta HTTP (Parcial):

Foram testados os status codes previstos em cada cenário planejado (ex: 200, 201, 400, 401). **Não foram criados testes adicionais para validar todos os status documentados no Swagger, nem cenários alternativos não previstos nas User Stories.**

Técnicas Planejadas para Automação (Postman): [🔗](#)

- Validação automática de **status codes, mensagens de resposta e estrutura do corpo JSON** nas abas de *Tests* do Postman.
- Testes automatizados com uso de **token Bearer**, garantindo simulação de usuários autenticados e não autenticados.
- Execução de rotinas básicas como **cadastro, autenticação, alteração e alteração de usuários e produtos** com scripts automatizados.
- Testes com **dados de massa** pré-carregados e **randomizados**, simulando múltiplas tentativas de ações inválidas (e-mails repetidos, senhas incorretas, etc.).

6. Mapa Mental da API [🔗](#)



7. Cenários de Teste Planejados + Rastreabilidade [🔗](#)

OBS: Evidências com melhor qualidade no [Tópico 12](#).

ID	Funcionalidade	User Story	Cenário	Resultado Esperado	Executado?	Resultado	Evidência
CT001	Usuários	US001	Criar usuário com dados válidos.	201 Created	<input checked="" type="checkbox"/>	OK	
CT002	Usuários	US001	Criar usuário com e-mail já cadastrado.	400 E-mail já cadastrado	<input checked="" type="checkbox"/>	OK	

CT003	Usuários	US001	Criar usuário com e-mail inválido (formato incorreto)	400 E-mail inválido		OK	
CT004	Usuários	US001	Criar usuário com e-mail do domínio gmail.com e hotmail.com	400 E-mail não permitido		Falha	 Aceitou cadastramento de e-mail não permitido. gmail.com
							 Aceitou cadastramento de e-mail não permitido. hotmail.com
CT005	Usuários	US001	Criar usuário com senha de 4 caracteres	400 Senha fora dos limites		Falha	 Aceitou senha menor que 5 caracteres.
CT006	Usuários	US001	Criar usuário com senha de 11 caracteres	400 Senha fora dos limites		Falha	 Aceitou senha maior que 10 caracteres.
CT007	Usuários	US001	Editar usuário existente com dados válidos	200 Alterado com sucesso		OK	
CT008	Usuários	US001	Editar usuário usando e-mail já existente	400 E-mail já cadastrado		OK	
CT009	Login	US002	Login com dados válidos	200 OK + Token Bearer		OK	
CT010	Login	US002	Login com senha incorreta	401 Unauthorized		OK	
CT011	Login	US002	Login com usuário inexistente	401 Unauthorized		OK	
CT012	Login	US002	Login com token expirado após 10 minutos	401 Token expirado		OK	
CT013	Produtos	US003	Criar produto com token de admin	201 Created		OK	
CT014	Produtos	US003	Criar produto sem token	401 Token ausente		OK	
CT015	Produtos	US003	Criar produto com nome já existente	400 Nome duplicado		OK	
CT016	Produtos	US003	PUT com ID inexistente (criação via PUT)	201 Created		OK	

CT017	Produtos	US003	PUT com nome já existente	400 Nome duplicado		OK	
CT018	Produtos	US003	Excluir produto vinculado a carrinho	400 Não é permitido excluir		OK	
CT019	Carrinhos	-	Criar carrinho com token válido	201 Created		OK	
CT020	Carrinhos	-	Criar carrinho com produto inexistente	400 Produto não encontrado		OK	
CT021	Carrinhos	-	Criar carrinho com produto duplicado	400 Produto duplicado		OK	
CT022	Carrinhos	-	Criar segundo carrinho para o mesmo usuário	400 Apenas um carrinho permitido		OK	
CT023	Carrinhos	-	Cancelar carrinho (estoque deve ser devolvido)	200 Carrinho cancelado		OK	
CT024	Carrinhos	-	Concluir compra (carrinho removido)	200 Carrinho excluído com sucesso		OK	

8. Priorização dos Cenários

ID	Cenário	Prioridade	Justificativa
CT009	Login com dados válidos	Alta	Acesso à API depende da autenticação.
CT010	Login com senha incorreta	Alta	Validação de segurança essencial.
CT002	Criar usuário com e-mail já existente	Alta	Garante unicidade de dados.
CT004	Criar usuário com e-mail do domínio gmail.com e hotmail.com	Alta	Validação de regra de negócio não documentada no Swagger.
CT016	Criar produto com nome já existente	Alta	Evita conflito de dados e duplicidade.
CT001	Criar usuário com dados válidos	Média	Fluxo principal de criação de usuário.
CT005	Criar usuário com senha de 4 caracteres	Média	Validação de borda importante.
CT012	Login com token expirado após 10 minutos	Média	Confirma validade de autenticação temporária.
CT022	Criar segundo carrinho para o mesmo usuário	Baixa	Funcionalidade fora das User Stories.
CT023	Cancelar carrinho (estoque deve ser devolvido)	Baixa	Fluxo complementar, não prioritário no desafio.
CT024	Concluir compra (carrinho removido)	Baixa	Complemento do fluxo de carrinhos.

9. ! Matriz de Risco

Probabilidade	90%	Média	Média	Cadastro com e-mail inválido ou duplicado	Alta	Alta
	70%	Baixa	Média	Senha fora dos limites ser aceita	Alta	Token não gerado ou expirado incorretamente
	50%	Baixa	Baixa	Permitir nome duplicado + Cancelar carrinho sem devolver estoque	Alta	Alta
	30%	Baixa	Permitir segundo carrinho para mesmo usuário	Média	Cadastro sem token ou por usuário não admin	Alta
	10%	Baixa	Baixa	Baixa	Baixa	Média
		Muito Baixo	Baixo	Moderado	Alto	Muito Alto

10. 📊 Cobertura de Testes

A cobertura de testes foi calculada com base nos critérios definidos no artigo [Test Coverage Criteria for RESTful Web APIs](#). Os cálculos consideram o total de itens documentados no Swagger e User Stories, comparado ao total de itens efetivamente testados (**manuais + automatizados**). A tabela abaixo resume os percentuais obtidos.

Critério	Total Documentado	Testado	Cobertura (%)	Observações
Path Coverage (input)	8 paths	8	100%	Todos os caminhos únicos testados (login, usuários, produtos, carrinhos).
Operator Coverage (input)	16 operações	16	100%	Todos os métodos GET, POST, PUT, DELETE testados ao menos uma vez.
Parameter Coverage (input)	18 parâmetros	14	78%	Todos parâmetros de Body testados; faltaram alguns parâmetros de query (ex: GET /usuarios).
Parameter Value Coverage (input)	12 valores esperados	6	50%	Testados apenas os valores

				previstos no roteiro; faltaram equivalência e limites adicionais.
Content-Type Coverage (input/output)	1 content-type	1	100%	Único content-type <code>application/json</code> testado; nenhum outro necessário/documentado.
Operation Flow Coverage (input)	4 fluxos possíveis	3	75%	Testados fluxos principais (<code>POST → PUT</code> , <code>POST → DELETE</code> , <code>POST → GET</code>); faltou alguma combinação.
Response Properties Body Coverage (output)	8 propriedades totais	4	50%	Validado apenas <code>message</code> , <code>authorization</code> ; faltaram validações de todos os campos retornados.
Status Code Coverage (output)	40 status codes totais	20	50%	Testados status esperados de cada cenário; faltaram erros alternativos e status documentados adicionais.

11. Testes Candidatos à Automação

ID	Cenário	Justificativa
CT001	Criar usuário com dados válidos	Fluxo principal e repetitivo com validação simples (201).
CT002	Criar usuário com e-mail já existente	Validação objetiva de erro de negócio (400).
CT003	Criar usuário com e-mail inválido	Validação padronizada com resposta previsível.
CT005 CT006	Criar usuário com senha fora do limite	Teste de valor limite ideal para automatização.

CT009	Login com dados válidos	Essencial para quase todos os testes. Deve ser automatizado.
CT010	Login com senha incorreta	Teste negativo com status fixo (401).
CT013	Criar produto com token de admin	Fluxo padrão com autenticação, útil para testes de regressão.
CT015	Criar produto com nome já existente	Validação clara de regra de negócio duplicada (400).
CT019	Criar carrinho com token válido	Permite validação de dependência entre módulos.
CT023	Cancelar carrinho (estoque devolvido)	Fluxo pós-compra que pode ser validado via API.

12. Artefatos Gerados

Os seguintes artefatos foram produzidos e fazem parte do desafio:

- **Plano de Testes** (este documento)
- **Documento de Issues e Melhorias Identificadas:**

BR001 - API permite cadastro com e-mail de domínio proibido (gmail/hotmail)

Descrição
A API permite o cadastro de usuário utilizando e-mails dos domínios `gmail.com` e `hotmail.com`, contrariando a regra de negócio da User Story US001.

Dados para Teste

```
1 {
  "name": "Teste Gmail",
  "email": "usuario@gmail.com",
  "password": "12345",
  "administrador": "true"
}
```

Passos para Reprodução

- Realizar uma requisição `POST /usuarios`.
- Enviar o corpo JSON acima.
- Observar a resposta da API.

Resultado Esperado
A API deve rejeitar o cadastro com status `400` e mensagem informando que o domínio de e-mail não é permitido.

Resultado Atual
A API retorna `201 Created` e cadastra o usuário normalmente.

Observações
Testado também com `usuario@hotmail.com` e mesmo comportamento observado.

Evidências

Anexos

Atividades

Comentários

Informações

Associated User Story: US001 - [API] Cadastro de Usuários

Responsible: Pedro Afonso de Alencar Silva

Team: Nenhum

Poi: Nenhum

Categorias: Nenhum

Prioridade: High

Severity: Alto

Frequência: Recorrente

Desenvolvimento: Criar branch, Criar commit

Automação: Execuções de regras

Created 1 hour ago | Updated 5 minutes ago | Configure

BR001 - API permite cadastro com e-mail de domínio proibido (gmail/hotmail)

The screenshot shows a Jira issue page for a bug report titled "BR002 - API permite cadastro de usuário com senha menor que o mínimo permitido".

Informações:

- User Story associada: US001 - [API] Cadastro de Usuários
- Responsável: Pedro Alfonso de Alencar Silva
- Team: Nenhum
- Pai: Nenhum
- Categorias: Nenhum
- Prioridade: Alta
- Severidade: Alta
- Frequência: Recorrente

Dados para Teste:

```
1 {
  "name": "Teste Senha Curta",
  "email": "senha.curta@teste.com",
  "password": "1234",
  "administrador": "true"
}
```

Passos para Reprodução:

- Realizar uma requisição POST /usuarios.
- Enviar o corpo JSON acima.
- Observar o resposta da API.

Resultado Esperado:

A API deve rejeitar o cadastro com status 400 e mensagem informando senha inválida.

Observações:

Teste repetido várias vezes, sempre aceitando senha curta.

Evidências:

Shows a screenshot of a Postman interface with a successful "Created" response (status 201) for a user creation request.

Anexos:

Shows a screenshot of a file named "001.png" attached to the issue.

Atividade:

Shows the activity stream with the following entries:

- Adicionar comentário...
- Ficou bom! (green icon)
- Precisa de ajuda? (yellow icon)
- Este item está bloqueado... (red icon)
- Você pode escovorecer...? (blue icon)
- Este item está em... (green icon)

Ocultar de bora: aperte **M** para fazer comentários

BR002 - API permite cadastro de usuário com senha menor que o mínimo permitido

BR003 - API permite cadastro de usuário com senha maior que o máximo permitido

Descrição
A API permite o cadastro de usuário com senha maior que 10 caracteres, desrespeitando o critério de limite máximo da User Story US001.

Dados para Teste

```
1 {
  "name": "Teste Senha Longa",
  "email": "senha_longa@teste.com",
  "password": "12345678901",
  "administrador": "true"
}
```

Possos para Reprodução

- Realizar uma requisição POST /usuarios.
- Enviar o corpo JSON acima.
- Observar a resposta da API.

Resultado Esperado
A API deve rejeitar o cadastro com status 400 e mensagem informando senha inválida.

Resultado Atual
A API retorna 201 Created e cadastra o usuário.

Observações
Mesmo comportamento ocorre ao tentar senhas ainda maiores.

Evidências

Anexos

Atividade

Comentários

Adicionar comentário... ▶ Ficou bom! ▶ Preciso de ajuda! 🔴 Este item está bloqueado... 🔍 Você pode escanear...? ✅ Este item está em...

Ocultar bônus: clique em **A** para fazer comentários

BR003 - API permite cadastro de usuário com senha maior que o máximo permitido

BR004 - Ambiguidade na regra de negócio para PUT /usuarios com ID inexistente

Descrição:
Identificado ambiguidade no User Story US001 sobre o comportamento esperado ao realizar uma requisição PUT /usuarios/{id} com um ID inexistente.

A User Story contém duas afirmações conflitantes:

- “Não deverá ser possível fazer ações e chamadas para usuários inexistentes.”
- “Caso não seja encontrado usuário com o ID informado no PUT, um novo usuário deverá ser criado.”

Isso gera dúvida se o comportamento correto seria:

- Retornar erro (usuário inexistente)
- ou
- Criar um novo usuário (user)

Dados para teste

```

1 {
2   "name": "Teste PUT Inexistente",
3   "email": "put_inexistente@teste.com",
4   "password": "12345",
5   "administrador": "true"
6 }
    
```

Endpoint: PUT /usuarios/[Informar ID Não existente]

Passos para Reprodução:

1. Realizar uma requisição PUT /usuarios/[Informar ID Não Existente]
2. Envie o corpo JSON acima.
3. Observar o resultado da API.

Resultado Esperado:
De acordo com a User Story, espera-se uma definição clara sobre o comportamento:

- Se deve retornar erro (e qual código).
- Ou se deve criar um novo usuário com status 201 Created.

Resultado Atual:
User Story apresenta duas diretrizes contraditórias, dificultando a validação do comportamento correto da API.

Observações:
A ambiguidade impacta diretamente o planejamento e execução do teste CT008 (“Editor usuário com ID inexistente (criação via PUT)”), impossibilitando sua execução com critério de aprovação claro.

Evidências:
US 001.

Atividade:

Todo Comentários Histórico Registro de atividades

Comentários:

- Ficou bom!
- Preciso de ajuda?
- Este item está bloqueado...
- Você pode escrever...
- Este item está em...

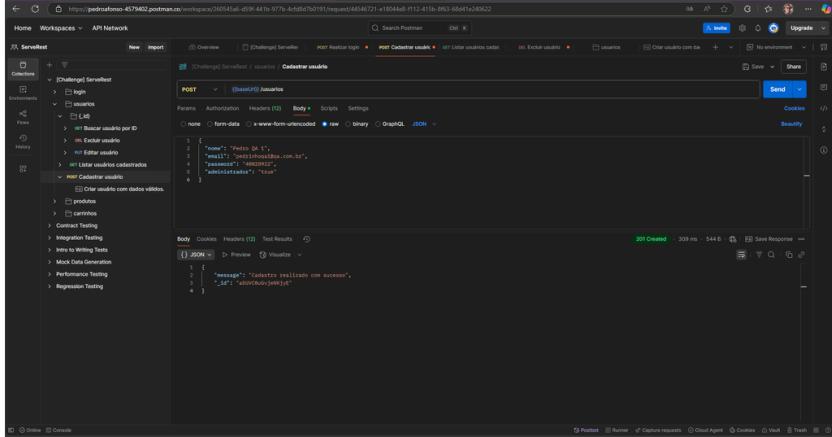
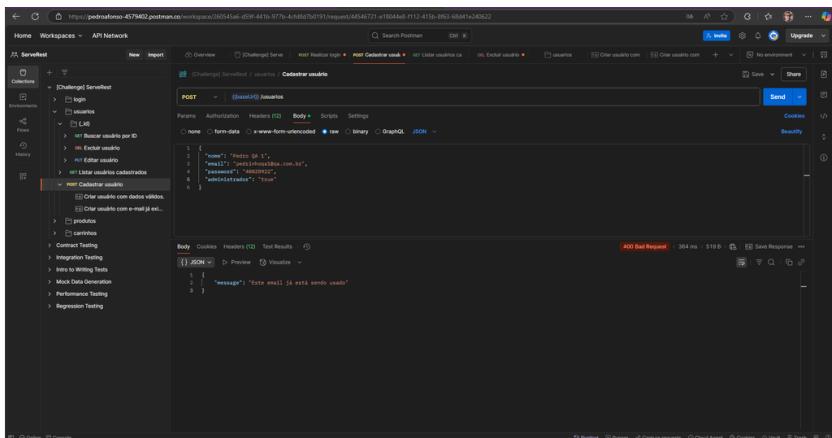
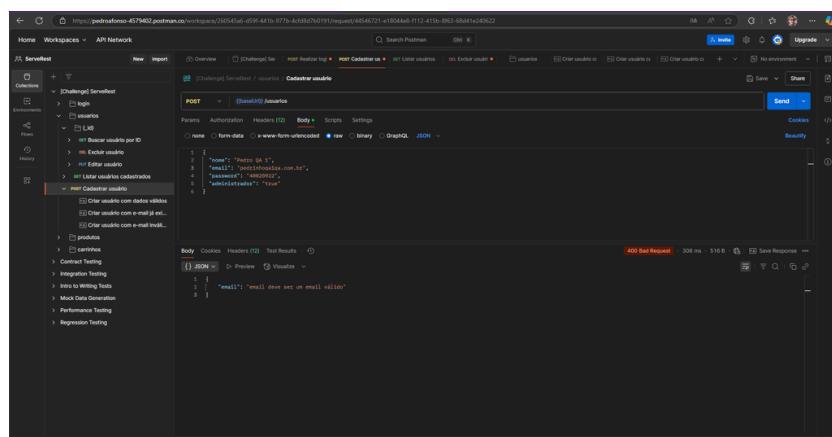
Dica de ouro: clique [] para fazer comentários

BR004 - Ambiguidade na regra de negócio para PUT /usuarios com ID inexistente

- Collection Postman com os cenários mapeados:

Swagger / User Stories	Automatização
[Challenge] ServeRest.postman_collection.json [Challenge] ServeRest <ul style="list-style-type: none"> ↓ [Challenge] ServeRest <ul style="list-style-type: none"> ↓ login <ul style="list-style-type: none"> > POST Realizar login ↓ usuarios <ul style="list-style-type: none"> ↓ {id} <ul style="list-style-type: none"> > GET Buscar usuário por ID > DEL Excluir usuário > PUT Editar usuário > GET Listar usuários cadastrados > POST Cadastrar usuário ↓ produtos <ul style="list-style-type: none"> ↓ {id} <ul style="list-style-type: none"> > GET Buscar produto por ID > DEL Excluir produto > PUT Editar produto > GET Listar produtos cadastrados > POST Cadastrar produto > DEL Excluir Produto ↓ carrinhos <ul style="list-style-type: none"> ↓ {id} <ul style="list-style-type: none"> > GET Buscar carrinho por ID > GET Concluir-compra > DEL Excluir carrinho > GET Cancelar-compra > DEL Excluir carrinho e retornar produt... > GET Listar carrinhos cadastrados > POST Cadastrar carrinho 	[Challenge] Extra.postman_collection.json [Challenge] Extra <ul style="list-style-type: none"> ↓ [Challenge] Extra <ul style="list-style-type: none"> ↓ Usuarios <ul style="list-style-type: none"> POST Criar usuário com dados válidos POST Criar usuário com e-mail já existente POST Criar usuário com e-mail inválido POST Criar usuário com senha fora do lim... ↓ Login <ul style="list-style-type: none"> POST Login com dados válidos POST Login com senha incorreta ↓ Produtos <ul style="list-style-type: none"> POST Criar produto com token de admin POST [DEPENDÊNCIA] Criar produto com... ↓ Carrinhos <ul style="list-style-type: none"> POST Criar carrinho com token válido DEL Cancelar carrinho (estoque devolvi...)

- Evidências com melhor qualidade:

ID	Evidência
CT00 1	
CT00 2	
CT00 3	

CT00

4

The screenshot shows the Postman application interface. The left sidebar displays a collection named 'Challenge| ServiTest' containing several items like 'login', 'L40', and 'POST Criar usuário'. The main workspace shows a POST request to 'http://pedroafonso-479402.postman.co/workspace/2054545d-d5f9-441b-977b-4100c70701/request/44546721-e18044e0-f112-411b-893-d8d41e240622'. The 'Body' tab is selected, showing a JSON payload:

```
[{"name": "Pedro G.", "email": "pedroafonso@gmail.com", "password": "12345678", "semelhante": "true"}]
```

The response status is 201 Created, with a response body indicating success:

```
{ "message": "Cadastro realizado com sucesso", "id": "4100c70701" }
```

This screenshot is identical to the one above, showing a successful POST request to 'Criar usuário' with a 201 Created response.

CT00

5

This screenshot is identical to the ones above, showing a successful POST request to 'Criar usuário' with a 201 Created response.

CT00

6

The screenshot shows a Postman collection named '[Challenge] Serviços' with a 'users' folder containing several API endpoints. The current endpoint is 'Cadastrar usuário' (Create user), which is a POST request to 'https://pedrofonso-4579402.postman.co/workspaces/2604546-059-441b-977c-4c0886700f19/requests/14546721-e1854465-7112-4136-893-05d41a70022'. The request body is JSON:

```
{
  "name": "Pedro QA 1",
  "email": "pedrofonso1808.com.br",
  "password": "1234567890",
  "administrador": "true"
}
```

The response status is 201 Created, with a response body indicating success:

```
{
  "message": "Usuário registrado com sucesso",
  "id": "5e0039935317"
}
```

CT00

7

The screenshot shows the same Postman collection and 'users' folder. The current endpoint is 'Editar usuário' (Edit user), which is a PUT request to 'https://pedrofonso-4579402.postman.co/workspaces/2604546-059-441b-977c-4c0886700f19/requests/14546721-e1854465-7112-4136-893-05d41a70024'. The request body is JSON:

```
{
  "name": "Pedro QA 1",
  "email": "pedrofonso1808.com.br",
  "password": "1234567890",
  "administrador": "true"
}
```

The response status is 200 OK, with a response body indicating success:

```
{
  "message": "Registro alterado com sucesso"
}
```

CT00

8

The screenshot shows the same Postman collection and 'users' folder. The current endpoint is 'Editar usuário' (Edit user), which is a PUT request to 'https://pedrofonso-4579402.postman.co/workspaces/2604546-059-441b-977c-4c0886700f19/requests/14546721-e1854465-7112-4136-893-05d41a70024'. The request body is JSON:

```
{
  "name": "Pedro QA 1",
  "email": "pedrofonso1808.com.br",
  "password": "1234567890",
  "administrador": "true"
}
```

The response status is 400 Bad Request, with a response body indicating an error:

```
{
  "message": "Este e-mail já está sendo usado"
}
```

CT00

9

The screenshot shows a POST request to the '/login' endpoint of the 'Servelfest' collection. The request body contains the following JSON:

```
{
  "email": "pedrofonso@servelfest.com.br",
  "password": "4890992x2"
}
```

The response status is 200 OK, with a response time of 296 ms. The response body is as follows:

```
{
  "message": "Login realizada com sucesso",
  "authorization": "eyJhbGciOiJIUzI1NiJ9.eyJlbWFpbCI6ImV4ZG9ubm8xQmJyZW1vZG9tZWxpdCIsInVzZXIiOiJ1c2VycG9yY29sbGVjdG9yeSIiLCJpYXQiOjE21394MjUwNzAifQ.5uChDfWVwvL2RwLs0uKQjPjg"
}
```

CT01

0

The screenshot shows a POST request to the '/login' endpoint of the 'Servelfest' collection. The request body contains the following JSON:

```
{
  "email": "pedrofonso@servelfest.com.br",
  "password": "4890992x2"
}
```

The response status is 401 Unauthorized, with a response time of 403 ms. The response body is as follows:

```
{
  "message": "Email e/senha inválidos"
}
```

CT01

1

The screenshot shows a POST request to the '/login' endpoint of the 'Servelfest' collection. The request body contains the following JSON:

```
{
  "email": "pedrofonso@servelfest.com.br",
  "password": "4890992x2"
}
```

The response status is 401 Unauthorized, with a response time of 419 ms. The response body is as follows:

```
{
  "message": "Email e/senha inválidos"
}
```

CT01

2

The screenshot shows a Postman collection named "[Challenge] Serviços". A POST request is being made to the endpoint `(BaseURL)/produtos`. The body contains the following JSON:

```
[{"nome": "Televisão", "preco": "999", "descricao": "Televisor Games", "quantidade": "122"}]
```

The response status is 401 Unauthorized, with the message: "Token de acesso ausente, inválido, expirado ou usuário do token não existe mais".

CT01

3

The screenshot shows a Postman collection named "[Challenge] Serviços". A POST request is being made to the endpoint `(BaseURL)/produtos`. The body contains the same JSON as the previous screenshot. The response status is 201 Created, with the message: "Cadastro realizado com sucesso".

CT01

4

The screenshot shows a Postman collection named "[Challenge] Serviços". A POST request is being made to the endpoint `(BaseURL)/produtos`. The body contains the same JSON as the previous screenshots. The response status is 401 Unauthorized, with the message: "Token de acesso ausente, inválido, expirado ou usuário do token não existe mais".

CT01

5

POST /products

```
{
  "name": "Televisor 4K",
  "preco": 1999,
  "descricao": "Televisor 4K de 55 polegadas",
  "quantidade": 122
}
```

Body: JSON

Response: 400 Bad Request

CT01

6

PUT /products/99999999999999999999

```
{
  "name": "Produto Novo via PUT",
  "preco": 199,
  "descricao": "Produto criado via PUT com ID Inexistente",
  "quantidade": 19
}
```

Body: JSON

Response: 201 Created

CT01

7

PUT /products/99999999999999999999

```
{
  "name": "Produto Teste 3v",
  "preco": 299,
  "descricao": "Teste de PUT dualizado",
  "quantidade": 6
}
```

Body: JSON

Response: 400 Bad Request

CT01

8

The screenshot shows a Postman collection named "ServiRest". Under the "produtos" section, there is a "Excluir Produto" endpoint. A DELETE request is made to the URL `/produtos/wmZAAuLjwvGU`. The response status is 400 Bad Request, with the message: "{'message': 'Não é possível excluir produto que faz parte de carrinho', 'id': 'e6e6f000-0000-4000-8000-000000000000'}".

CT01

9

The screenshot shows a Postman collection named "ServiRest". Under the "carrinhos" section, there is a "Criar carrinho" endpoint. A POST request is made with the following JSON body:

```
[{"produtos": [{"idProduto": "11e6f000-0000-4000-8000-000000000000", "quantidade": 1}, {"idProduto": "11e6f000-0000-4000-8000-000000000001", "quantidade": 1}], "token": "e6e6f000-0000-4000-8000-000000000000"}]
```

The response status is 201 Created, with the message: "{'message': 'Carrinho criado com sucesso', 'id': 'e6e6f000-0000-4000-8000-000000000002'}".

CT02

0

The screenshot shows a Postman collection named "ServiRest". Under the "carrinhos" section, there is a "Criar carrinho" endpoint. A POST request is made with the following JSON body:

```
[{"produtos": [{"idProduto": "11e6f000-0000-4000-8000-000000000000", "quantidade": 1}, {"idProduto": "11e6f000-0000-4000-8000-000000000001", "quantidade": 1}], "token": "e6e6f000-0000-4000-8000-000000000000"}]
```

The response status is 400 Bad Request, with the message: "{'message': 'Produto não encontrado', 'id': 'e6e6f000-0000-4000-8000-000000000000'}".

CT02

1

POST https://pedrofonso-4579402.postman.co/workspace/ServerRest-260545d5-d59-441b-977b-4c65d7b0191/request/44540721-a052-515b-8dce-4f99-a030-54dcbcafe19

Body

```
[{"product": [{"idProduct": "1longuinhosTudoP", "quantity": 1}, {"idProduct": "1longuinhosTudoP", "quantity": 1}, {"idProduct": "1longuinhosTudoP", "quantity": 1}], "message": "Não é possível possuir produto duplicado"}]
```

400 Bad Request 284 ms - 595 B

CT02

2

POST https://pedrofonso-4579402.postman.co/workspace/ServerRest-260545d5-d59-441b-977b-4c65d7b0191/request/44540721-775a9bc-c1d3-4c1b-a979-7427b6632aef

Body

```
[{"product": [{"idProduct": "1longuinhosTudoP", "quantity": 1}, {"idProduct": "1longuinhosTudoP", "quantity": 1}, {"idProduct": "1longuinhosTudoP", "quantity": 1}], "message": "Não é possível tecer mais de 1 caximbo"}]
```

400 Bad Request 278 ms - 523 B

CT02

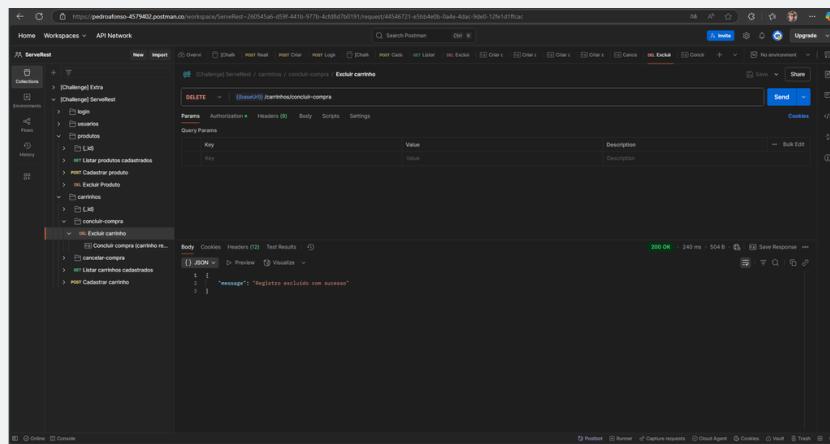
3

DELETE https://pedrofonso-4579402.postman.co/workspace/ServerRest-260545d5-d59-441b-977b-4c65d7b0191/request/44540721-775a9bc-c1d3-4c1b-a979-7427b6632aef

Body

```
[{"message": "Registro excluído com sucesso. Estoque dos produtos reabastecido"}]
```

200 OK 307 ms - 543 B



13. ? Considerações sobre Itens Omitidos ↗

Este plano foi adaptado para atender ao escopo e às exigências específicas do desafio proposto pela API ServeRest. Alguns tópicos tradicionalmente presentes em planos de teste mais robustos foram omitidos ou simplificados por não se aplicarem ao contexto deste projeto, conforme exemplos:

- **Glossário:** os termos utilizados são amplamente conhecidos no contexto de testes de APIs REST.
- **Infraestrutura e Responsabilidades da Equipe:** a infraestrutura já está fornecida e a execução será feita individualmente.
- **Fluxo de Trabalho, Comunicação e Cronograma:** não há múltiplos stakeholders, equipes envolvidas ou marcos formais no desafio.
- **Aprovações e Referências formais:** não há exigência de validações corporativas neste projeto prático.

Essa simplificação foi feita visando manter o plano de testes **focado, objetivo e funcional**, atendendo aos **entregáveis obrigatórios do desafio** e evitando complexidade desnecessária.