Plano de Teste - ServeRest [Robot Framework / AWS] (compassuol.serverest.dev)

Challenge 03 - Sprint 06/Semana 12
Plano de Teste
Versão 1.1.2

@Pedro Afonso de Alencar Silva

📜 Histórico de Revisões

Histórico de Revisões		
Versão	**Data**	**Descrição**
1.0	30 de mai. de 2025	• **Versão Inicial (Copia do Challenge da Semana 08)**

1.1	30 de mai. de 2025	Atualizações gerais no plano de teste:
		• + **Seção 5 – Técnicas Aplicadas**: Execução migrada para AWS (EC2), ajustes nas descrições técnicas unificando testes manuais e automatizados, inclusão de tópicos como uso de token, dados dinâmicos e geração de logs.
		• - **Seção 7 – Cenários de Teste Planejados + Rastreabilidade**: Remoção das colunas Resultado Esperado, Resultado, Bug Relacionado, Evidência.
		• + **Seção 8 – Priorização dos Cenários**: Reclassificação das prioridades, inclusão de novos cenários críticos (carrinho, compra, login) e remoção de cenários com menor relevância.
		• + **Seção 9 – Matriz de Risco**: Atualização dos riscos conforme nova priorização, adição de risco de falha na finalização de compra, remoção de riscos de baixo impacto e revisão da classificação de impacto/probabilidade.
		• + **Seção 12 – Cobertura de Testes:** Observação incluída indicando que a cobertura representa apenas o que foi testado ao menos uma vez e não cobre a execução final.
		• + **Seção 13 – Artefatos Gerados**: Inclusão das subseções:
		  ◦ Ciclos de Teste + Status (Apenas Testes no POSTMAN)
		  ◦ Relatório de Execução de Teste (Apenas Testes no POSTMAN)
		  ◦ Ciclos de Teste + Status (Robot Framework)
		  ◦ Issues abertas no Jira relacionadas a falhas encontradas:

1.1.1	3 de jun. de 2025	• + **Seção 14 - Interpretação dos Resultados**: Inclusão.
1.1.2	5 de jun. de 2025	• + **Reorganização das Seções**. • + **Seção 11 - Infraestrutura de Testes**: Inclusão.

📑 Índice

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 negócio dos seguintes módulos da API ServeRest:

OBS: As US inseridas no Jira são as mesmas definidas na plataforma Learning.

Type	Resumo	Responsável	Prioridade	Status
🔖	US003 - [API] Gerenciamento de Produtos	Pedro Afonso de Alencar Silva	= Medium	A FAZER
🔖	US002 - [API] Login	Pedro Afonso de Alencar Silva	= Medium	A FAZER
🔖	US001 - [API] Cadastro de Usuários	Pedro Afonso de Alencar Silva	= Medium	A FAZER

Sincronizado agora • 3 itens

- **Módulo Carrinhos**: Aapesar de não contemplado explicitamente nas User Stories, será testado devido à sua importância para o fluxo de negócio.
- **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:

- Identificada **ambiguidade na User Story US001** sobre o comportamento esperado ao realizar uma requisição PUT /usuarios/{id} com um ID inexistente. Foi aberta uma Issue no Jira.
- **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;
- Integrações 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**. Essas regras são essenciais para assegurar que a aplicação cumpra os requisitos funcionais e comportamentais esperados..

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 foi baseada em **testes manuais com Postman e automatizados com Robot Framework**, com foco na validação das **regras de negócio descritas nas User Stories** e no comportamento técnico documentado no **Swagger da API ServeRest**.
  Os testes seguiram **cenários roteirizados previamente (CT001 a CT024)**, sem execução de testes exploratórios livres.

📌 Técnicas de Teste Aplicadas

- **Caixa Preta:**
  Todos os testes foram elaborados utilizando a técnica de caixa preta, **validando o comportamento da API com base nas entradas e saídas definidas nas User Stories** e documentação Swagger, sem acesso ao código-fonte da aplicação.
- **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

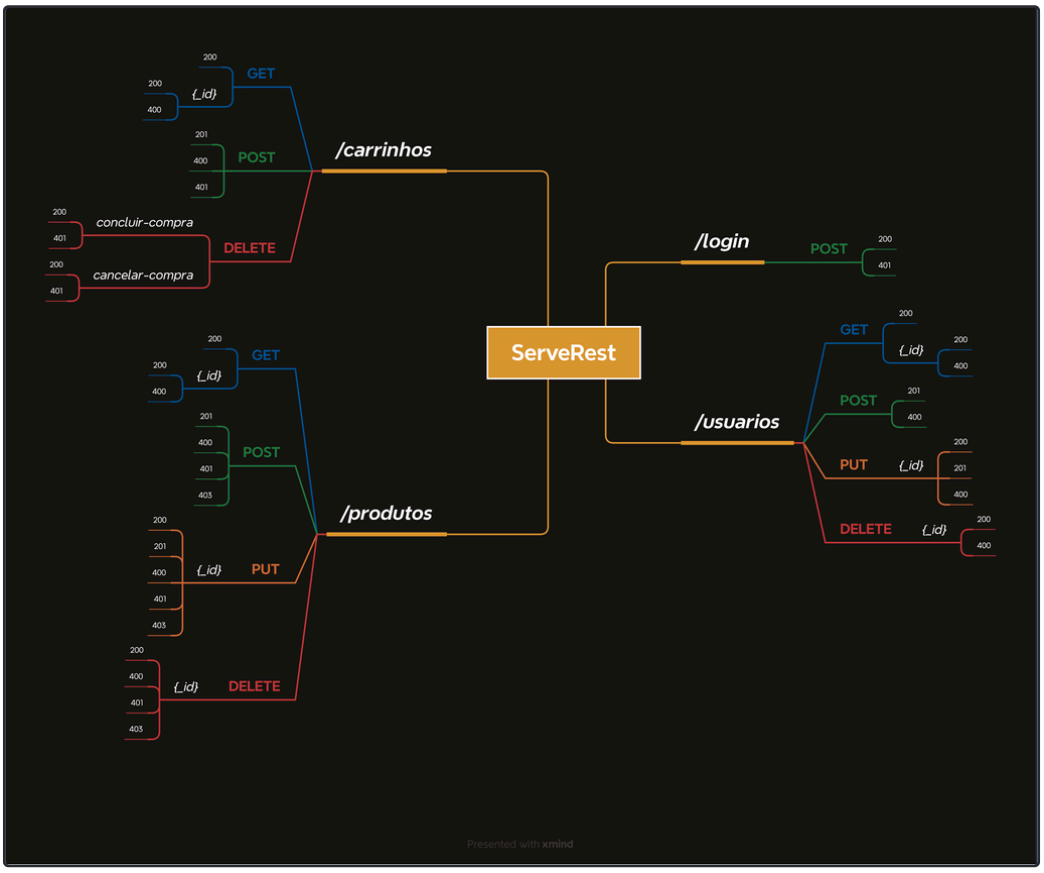
tentativas fora dos limites (4 e 11 caracteres). **Não foi estendida a outros parâmetros como preço, quantidade ou IDs.**

- **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:**

- **Validação de Respostas:**
  Validação automatizada de status codes, mensagens de retorno e estrutura do JSON com uso das `keywords` da `RequestsLibrary`.
- **Autenticação com Token:**
  Execução de cenários com e sem autenticação, utilizando o token Bearer nos headers para simular perfis de usuário.
- **Fluxos Automatizados:**
  Operações como cadastro, login, edição e exclusão de usuários e produtos foram roteirizadas com comandos automatizados, usando dados parametrizados.
- **Massa de Dados Dinâmica:**
  Utilização de dados randomizados e pré-definidos, incluindo cenários com e-mails duplicados e senhas inválidas, para cobrir variações negativas comuns.
- **Execução em Ambiente AWS:**
  Toda a automação será executada em instância EC2 configurada com Robot Framework, com coleta dos logs (`log.html`, `report.html` e `output.xml`) ao final de cada execução via terminal.

6. 🗺️ Mapa Mental da API



7. 🌐 Cenários de Teste Planejados

Disponível no Jira: Test Cases

OBS: Dentro de cada Ticket, contém: *Passo(s) para o Teste, Resultado Esperado, Dados do Teste.*

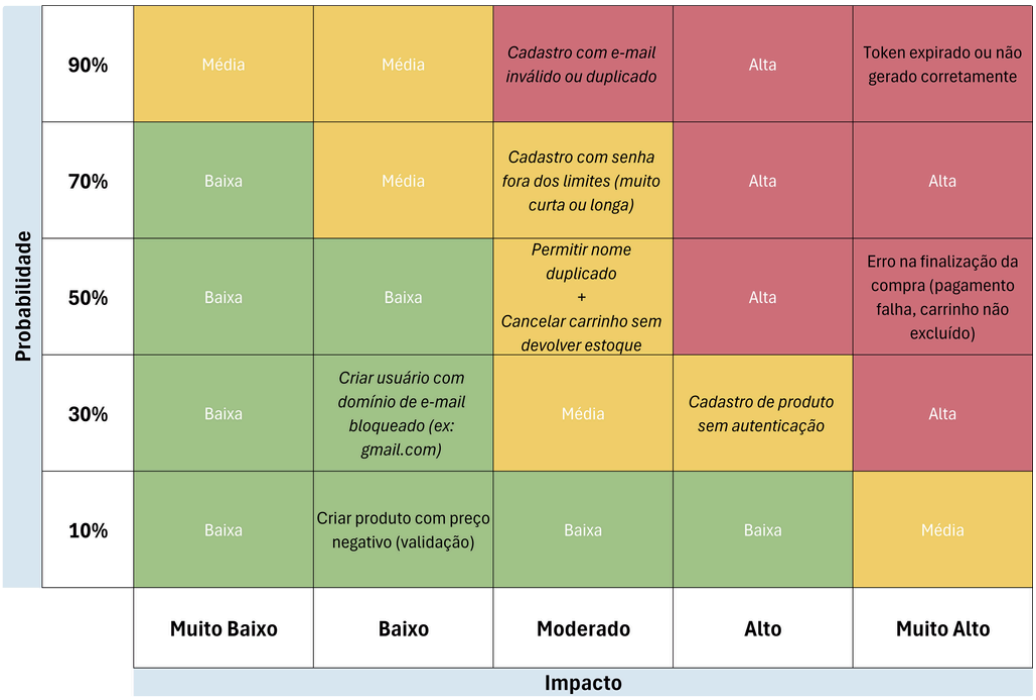
ID	Funcionalidade	User Story	Cenário
CT001	Usuários	US001	Criar usuário com dados válidos.
CT002	Usuários	US001	Criar usuário com e-mail já existente.
CT003	Usuários	US001	Criar usuário com e-mail inválido (formato incorreto)
CT004	Usuários	US001	Criar usuário com e-mail do domínio gmail.com e hotmail.com
CT005	Usuários	US001	Criar usuário com senha de 4 caracteres
CT006	Usuários	US001	Criar usuário com senha de 11 caracteres
CT007	Usuários	US001	Editar usuário existente com dados válidos
CT008	Usuários	US001	Editar usuário usando e-mail já existente
CT009	Login	US002	Login com dados válidos
CT010	Login	US002	Login com senha incorreta
CT011	Login	US002	Login com usuário inexistente
CT012	Login	US002	Login com token expirado após 10 minutos
CT013	Produtos	US003	Criar produto com token de admin
CT014	Produtos	US003	Criar produto sem token
CT015	Produtos	US003	Criar produto com nome já existente
CT016	Produtos	US003	PUT Produto com ID inexistente (criação via PUT)
CT017	Produtos	US003	PUT Produto com nome já existente
CT018	Produtos	US003	Excluir produto vinculado a carrinho
CT019	Carrinhos	-	Criar carrinho com token válido
CT020	Carrinhos	-	Criar carrinho com produto inexistente
CT021	Carrinhos	-	Criar carrinho com produto duplicado
CT022	Carrinhos	-	Criar segundo carrinho para o mesmo usuário
CT023	Carrinhos	-	Cancelar carrinho (estoque deve ser devolvido)
CT024	Carrinhos	-	Concluir compra (carrinho removido)

8. 🔢 Priorização dos Cenários

ID	Cenário	Prioridade	Justificativa

CT019	Criar carrinho com token válido	Alta	Funcionalidade principal de compra, diretamente ligada à receita.
CT023	Cancelar carrinho (estoque deve ser devolvido)	Alta	Impacta a experiência do usuário e a gestão de estoque, crucial para a disponibilidade de produtos.
CT024	Concluir compra (carrinho removido)	Alta	Finalização do fluxo de compra, essencial para a receita e satisfação do cliente.
CT001	Criar usuário com dados válidos	Alta	Fluxo essencial para novos usuários acessarem a plataforma.
CT009	Login com dados válidos	Alta	Funcionalidade básica para acesso de usuários existentes.
CT013	Criar produto com token de admin	Alta	Essencial para a gestão do catálogo de produtos por administradores.
CT002	Criar usuário com e-mail já existente	Média	Validação importante para integridade dos dados e experiência do usuário.
CT007	Editar usuário existente com dados válidos	Média	Permite ao usuário manter seus dados atualizados.
CT010	Login com senha incorreta	Média	Cenário comum, importante para a segurança e feedback ao usuário.
CT018	Excluir produto vinculado a carrinho	Média	Regra de negócio importante para evitar inconsistências em carrinhos ativos.
CT004	Criar usuário com e-mail do domínio gmail.com e hotmail.com	Baixa	Teste de regra de negócio específica (domínios não permitidos), menor impacto geral.
CT005 CT006	Criar usuário com senha de 4 e 11 caracteres	Baixa	Validação de limite de senha, importante, mas menos crítico que falhas de fluxo principal.

9. ⚠️ Matriz de Risco



Probabilidade	Muito Baixo	Baixo	Moderado	Alto	Muito Alto
90%	Média	Média	*Cadastro com e-mail inválido ou duplicado*	Alta	Token expirado ou não gerado corretamente
70%	Baixa	Média	*Cadastro com senha fora dos limites (muito curta ou longa)*	Alta	Alta
50%	Baixa	Baixa	*Permitir nome duplicado* + *Cancelar carrinho sem devolver estoque*	Alta	Erro na finalização da compra (pagamento falha, carrinho não excluído)
30%	Baixa	*Criar usuário com domínio de e-mail bloqueado (ex: gmail.com)*	Média	*Cadastro de produto sem autenticação*	Alta
10%	Baixa	Criar produto com preço negativo (validação)	Baixa	Baixa	Média

Impacto

10. ⚙️ Testes Candidatos à Automação

ID	Cenário	Justificativa
CT001	Criar usuário com dados válidos	Fluxo base de entrada no sistema, necessário para demais funcionalidades.
CT002	Criar usuário com e-mail já existente	Fluxo base de entrada no sistema, necessário para demais funcionalidades.
CT003	Criar usuário com e-mail inválido	Fluxo base de entrada no sistema, necessário para demais funcionalidades.
CT004	Criar usuário com e-mail do domínio gmail.com e hotmail.com	Fluxo base de entrada no sistema, necessário para demais funcionalidades.
CT005	Criar usuário com senha de 4 caracteres	Fluxo base de entrada no sistema, necessário para demais funcionalidades.
CT006	Criar usuário com senha de 11 caracteres	Fluxo base de entrada no sistema, necessário para demais funcionalidades.
CT007	Editar usuário existente com dados válidos	Manutenção de dados — importante mas não crítico.
CT008	Editar usuário usando e-mail já existente	Manutenção de dados — importante mas não crítico.
CT009	Login com dados válidos	Autenticação básica — fluxo essencial para acesso ao sistema.
CT010	Login com senha incorreta	Validação de segurança e feedback ao usuário.

CT011	Listar todos os usuários	Cenário de apoio à navegação — menor impacto direto na receita.
CT012	Atualizar usuário autenticado	Manutenção de dados — importante mas não crítico.
CT013	Criar produto com token de admin	Fluxo comum de cadastro de produtos — relevante para o catálogo.
CT014	Excluir usuário autenticado	Controle de remoção de dados — relevante para segurança e integridade.
CT015	Criar produto com nome já existente	Fluxo comum de cadastro de produtos — relevante para o catálogo.
CT016	Listar todos os produtos	Cenário de apoio à navegação — menor impacto direto na receita.
CT017	Buscar produto por ID	Cenário de apoio à navegação — menor impacto direto na receita.
CT018	Excluir produto vinculado a carrinho	Controle de remoção de dados — relevante para segurança e integridade.
CT019	Criar carrinho com token válido	Fluxo de compra, diretamente ligado à receita e integração entre módulos.
CT020	Adicionar produtos ao carrinho	Fluxo de compra, diretamente ligado à receita e integração entre módulos.
CT021	Cadastrar produto com sucesso	Fluxo comum de cadastro de produtos — relevante para o catálogo.
CT022	Cadastrar produto com preço negativo	Validação de regra técnica que evita falhas financeiras.
CT023	Cancelar carrinho (estoque deve ser devolvido)	Fluxo de compra, diretamente ligado à receita e integração entre módulos.
CT024	Concluir compra (carrinho removido)	Finalização do processo de venda — essencial para receita.

11. 🖥️ Infraestrutura de Testes

A aplicação ServeRest foi hospedada em uma **instância EC2 da AWS**, com IP público e configuração padrão para execução.

A automação dos testes foi desenvolvida com **Robot Framework**, estruturada localmente conforme padrão de diretórios proposto no desafio. Para execução dos testes automatizados, utilizou-se o comando `robot -d reports tests/`, com testes executados via terminal em ambiente local.

Para testes manuais exploratórios, foi utilizado o Postman, importando a coleção a partir do Swagger oficial da API.

Ferramentas utilizadas:

- Robot Framework (execução local e AWS EC2)
- Postman (testes manuais)
- AWS EC2 (hospedagem da API ServeRest)

- FakerLibrary (geração dinâmica de dados de teste)
- VS Code (desenvolvimento dos testes)

12. 📊 Cobertura de Testes

A cobertura de testes apresentada nesta seção foi calculada com base nos critérios definidos no artigo ***Test Coverage Criteria for RESTful Web APIs***. Ela representa os **itens técnicos testados ao menos uma vez** durante a elaboração dos cenários e execução parcial da suíte.

Todos os **24 cenários previstos no plano foram**:

- Executados manualmente no **Postman**.
- Automatizados com **Robot Framework**, utilizando dados dinâmicos e validações detalhadas.

A tabela abaixo consolida a cobertura funcional e técnica obtida até o momento:

Critério	Total Documentado	Testado	Cobertura (%)	Observações
Path Coverage (input)	16 paths	11	69%	Faltou testar: GET /usuarios, GET /usuarios/{id}, GET /produtos, GET /produtos/{id}, GET /carrinhos, GET /carrinhos/{id} e DELETE /usuarios/{id}.
Operator Coverage (input)	16 operações	11	69%	Métodos executados: POST, PUT e DELETE. Faltou testar GET em todos os recursos.
Parameter Coverage (input)	13 parâmetros	6	46%	Foram cobertos todos os parâmetros de body. Nenhum parâmetro de path foi testado (ex.: `{id}` em PUT/DELETE). Swagger não define query params.
Parameter Value Coverage (input)	24 valores esperados	12	50%	Foram testados apenas valores válidos. Faltam casos negativos (ex.: email inválido, senha curta, quantidade negativa, preço inválido).
Content-Type Coverage (input/output)	1 content-type	1	100%	`application/json` (e `application/x-www-form-urlencoded` quando aplicável) testados, conforme especificação.
Operation Flow Coverage (input)	4 fluxos possíveis	3	75%	Faltou fluxo que envolve GET (ex.: listar usuários/produtos antes de criar ou editar) e outros cenários alternativos.
Response Properties Body Coverage (output)	8 propriedades totais	4	50%	Validado apenas `message`, `authorization`, `item` e `index`. Faltam verificar campos detalhados de usuário, produto e carrinho (como `nome`, `email`, `preco`, `quantidade`, `idUsuario` etc.).
Status Code Coverage (output)	5 códigos totais	4	80%	Testado: 200, 201, 400 e 401. Faltou validar 403 (Forbidden).

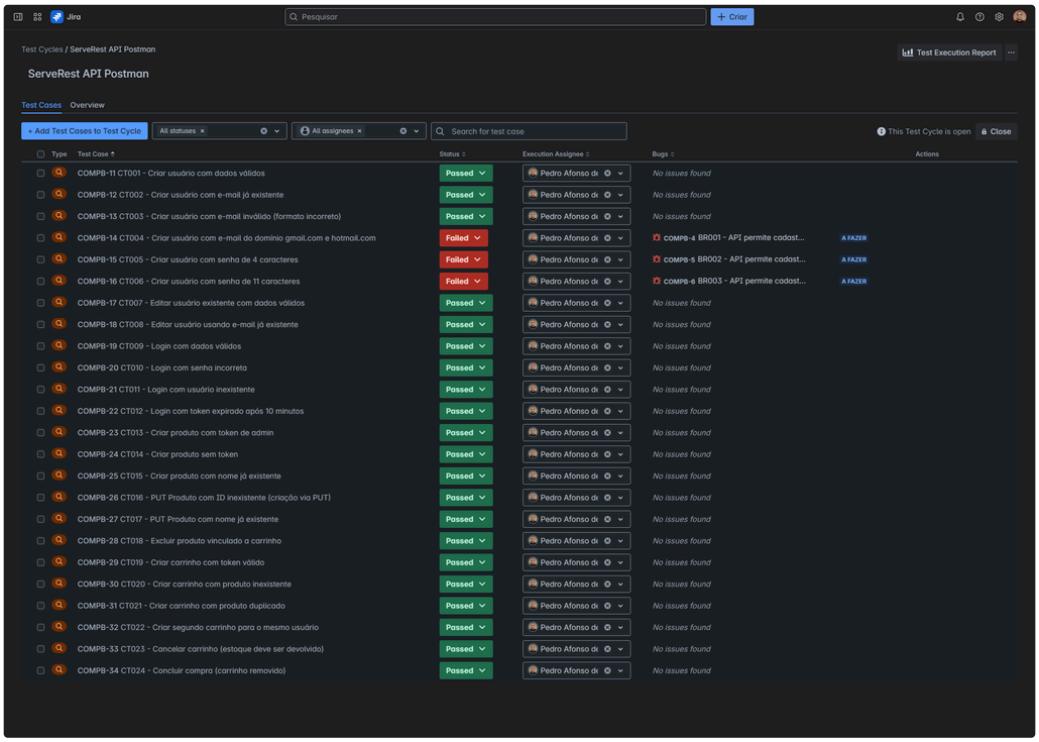
13. 📦 Artefatos Gerados

Os seguintes artefatos foram produzidos e fazem parte do desafio:

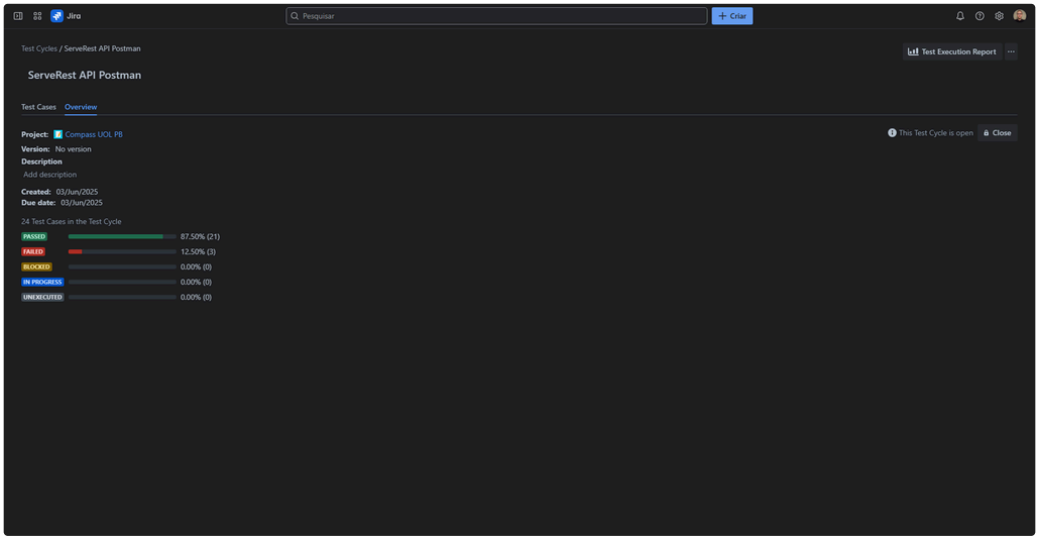
- **Plano de Testes** (este documento)
- **Ciclos de Teste + Status (Apenas Testes no POSTMAN):**

Disponível no Jira: [Test Cycles / ServeRest API Postman]

OBS: Dentro de cada Ticket, contém: *Passo(s) para o Teste, Resultado Esperado, Resultado Obtido, Dados do Teste, Evidência, Bug Relacionado*



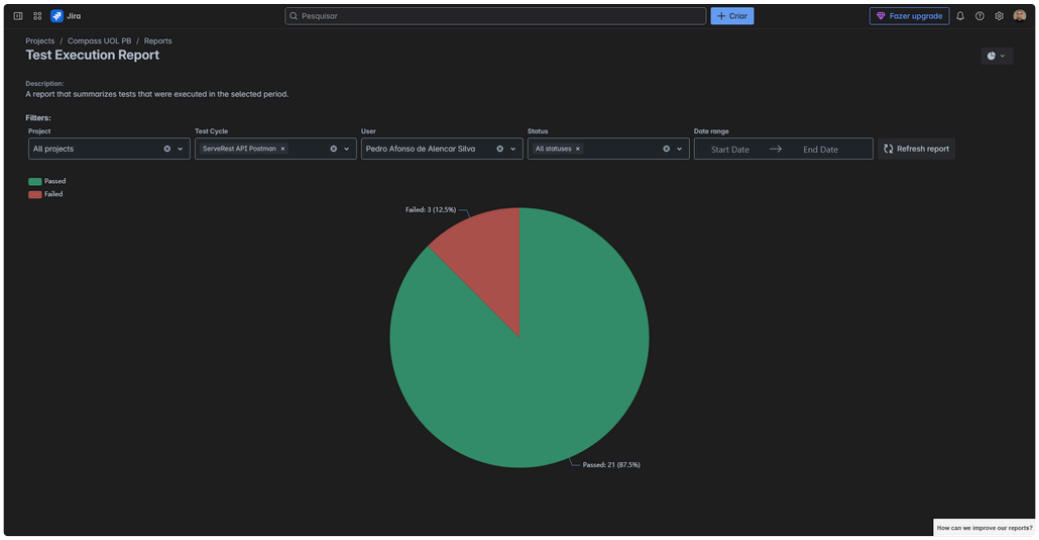
Apenas Testes Manuais realizados no Postman



Apenas Testes Manuais realizados no Postman

- **Relatório de Execução de Teste (Apenas Testes no POSTMAN):**

Disponível no Jira: [Test Execution Report / ServeRest API Postman]



Apenas Testes Manuais realizados no Postman

- **Ciclos de Teste + Status (Robot Framework):**

Disponível no GitHub: [ServeRest-Tests / reports]

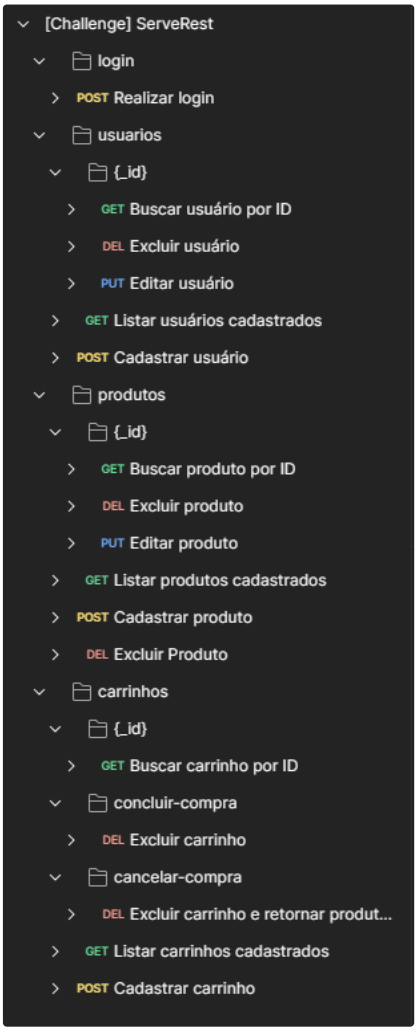
Apenas Testes Automatizados realizados com Robot Framework

- **Collection Postman com os cenários mapeados:**

Swagger / User Sotires	Automatização

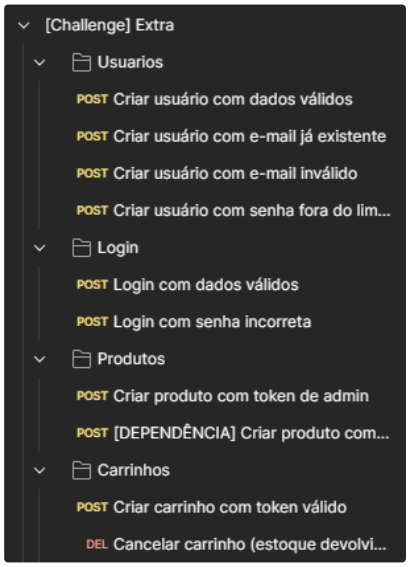
[Challenge] ServeRest.postman_collection.json

[Challenge] ServeRest



[Challenge] Extra.postman_collection.json

[Challenge] Extra



- **Issues abertas no Jira relacionadas a falhas encontradas:**

Disponível no Jira: ServeRest-Tests / reports

Type	Resumo	Responsável	Prioridade	Status
	BR004 - Ambiguidade na regra de negócio para PUT /usuarios com ID inexistente	Pedro Afonso d...	High	A FAZER
	BR003 - API permite cadastro de usuário com senha maior que o máximo permitido	Pedro Afonso d...	High	A FAZER
	BR002 - API permite cadastro de usuário com senha menor que o mínimo permitido	Pedro Afonso d...	High	A FAZER
	BR001 - API permite cadastro com e-mail de domínio proibido (gmail/hotmail)	Pedro Afonso d...	High	A FAZER

Sincronizado agora • 4 itens

14. 📄 Interpretação dos Resultados

Durante a execução dos testes planejados, a análise foi além do simples resultado de sucesso ou falha. O foco esteve em identificar o que esses resultados representam para a **qualidade do sistema**, **a experiência do usuário** e **os riscos para o**

negócio.

Resumo dos Resultados:

- Dos 24 cenários executados:
  - **21 testes passaram com sucesso**, cobrindo os principais fluxos de negócio da aplicação (cadastro de usuário, login, produtos, carrinho).
  - **3 testes apresentaram falhas**, CT004, CT005 e CT006.

Impacto para a Aplicação:

- A aplicação está **estável** nos fluxos principais e adequada para o ambiente de homologação.
- As falhas encontradas **não comprometem a operação da loja virtual**, mas apontam pontos a serem reforçados:
  - Validação de campos obrigatórios
  - Cobertura de cenários negativos e limites

Recomendações:

- Corrigir as 3 falhas reportadas, apesar de serem de baixa prioridade, pois não têm impacto significativo na aplicação.
- Considerar a ampliação dos **testes negativos**, especialmente com foco em formatos inválidos e dados ausentes.
- Avaliar a real necessidade de bloquear domínios de e-mail como `gmail.com/hotmail.com`, conforme o CT004.
- **Revisão da ambiguidade Identificada** na User Story US001. Foi aberta uma Issue no Jira.

Conclusão:
A rodada de testes demonstrou que o sistema ServeRest atende aos principais critérios funcionais com **baixo risco residual**. As correções indicadas são pontuais, de fácil implementação, aumentam ainda mais a confiabilidade da API, e podem ser implementadas em curto prazo.

15. ❓ 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.
- **Responsabilidades da Equipe**: 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.