Plano de Teste - Challenge 03



Sumário | ☑ Apresentação | ⊚ Objetivo | ⊚ Escopo | 🋠 Ferramentas utilizadas | ◎ Mapa mental da aplicação | ℚ Análise | ♠ Requisitos | ☐ Cenários de teste planejados | Erros e Melhorias | ⚠ Matriz de risco | ் Testes automatizados | ☒ Cobertura de testes (Manuais e Automatizados) | ☐ Resultados finais | ☑ Conclusão

📝 Apresentação 🛭

A API REST ServeRest é uma ferramenta gratuita que foi desenvolvida para simular uma loja virtual com o principal objetivo de ser usada como material de estudo em testes de API. Em sua documentação (https://compassuol.serverest.dev/), conta com os endpoints /login , /usuarios , /produtos e /carrinhos . Esses recursos permitem que sejam realizadas explorações e análises em busca de possíveis erros, sendo possível que interessados na área de qualidade de software e teste de APIs REST apliquem seus conhecimentos de forma prática.

Objetivo Ø

O objetivo principal da realização dos testes realizados na API ServeRest é garantir a qualidade da aplicação por meio da validação das regras de negócio e da identificação erros e possíveis oportunidades de melhoria.

Escopo Ø

Serão testados os seguintes recursos e suas respectivas requisições, conforme documentado na API ServeRest:

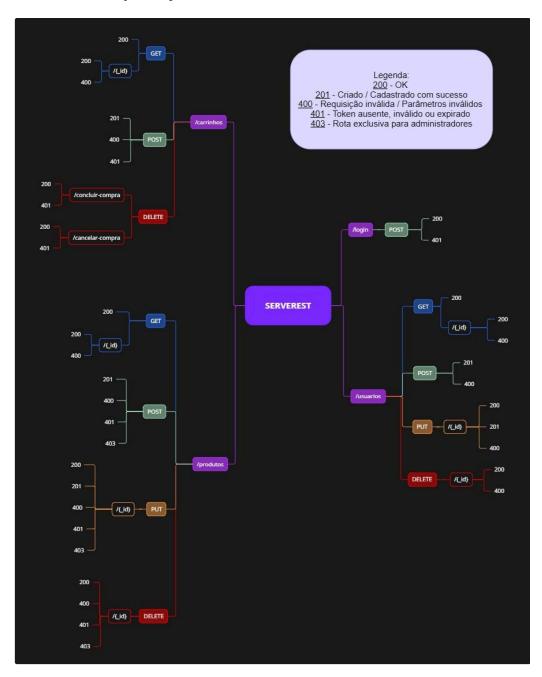
- /login
 - Realizar login
- /usuarios
 - Listar usuários cadastrados
 - o Cadastrar usuário
 - o Buscar usuário por ID
 - Excluir usuário
 - Atualizar usuário
- /produtos
 - o Listar produtos cadastrados
 - Cadastrar produto
 - Buscar produto por ID
 - Excluir produto
 - Atualizar produto
- /carrinhos
 - Listar carrinhos cadastrados
 - o Cadastrar carrinho
 - Buscar carrinho por ID
 - Excluir carrinho (quando compra foi concluída)

• Excluir carrinho e retornar produtos para estoque (quando compra foi cancelada)

💢 Ferramentas utilizadas 🔊

- Swagger da API ServeRest: https://compassuol.serverest.dev/
- Postman
- Jira
- Confluence
- Miro
- VS Code + Robot Framework
- QAlity no Jira

🧠 Mapa mental da aplicação 🖉



Análise 🕖

Cenários de teste @

Os cenários de teste foram elaborados com base nos requisitos definidos nas User Stories dos endpoints /login , /usuarios e /produtos , bem como na documentação Swagger referente ao endpoint /carrinhos . Além disso, foram incluídos cenários de "happy path" para garantir que os fluxos principais de cada recurso estejam funcionando corretamente.

Técnicas aplicadas 🕖

- Testes baseados em cenários
 - o Foi a principal técnica aplicada tendo como base os requisitos extraídos das User Stories e da documentação Swagger.

- Abordagem permitiu construir casos de uso realistas que cobrem tanto os fluxos principais quanto os alternativos.
- Testes exploratórios
 - Abordagem utilizada no endpoint /carrinhos analisando comportamentos através da documentação da API, já que ele que não possuía sua própria User Story.
- Partição de equivalência
 - Abordagem utilizada para validar entradas válidas e inválidas no campo de senha que precisava ter de 5 a 10 caracteres.
 Foram testadas diferentes classes de equivalência para garantir o comportamento correto com entradas dentro e fora do intervalo permitido.



/login ∂

Baseados nos critérios de aceite da US 002: [API] Login.

ID	Descrição
R1.1	Sistema não deve permitir autenticação de usuários não cadastrados
R1.2	Sistema não deve permitir autenticação de usuário com senha inválida
R1.3	Sistema deve retornar status code 401 (Unauthorized) ao falhar autenticação
R1.4	Ao autenticar com sucesso, sistema deve ser gerado um token Bearer
R1.5	Cada token Bearer gerado deve ter validade de 10 minutos

/usuarios ∂

Baseados nos critérios de aceite da US 001 - [API] Usuários.

ID	Descrição
R2.1	Cada vendedor (usuário) deve ter os campos NOME, E-MAIL, PASSWORD e ADMINISTRADOR
R2.2	Sistema não deve permitir que ações sejam executadas para usuários inexistentes (exceto em PUT)
R2.3	Se ID informado no PUT for de um usuário inexistente, um novo usuário deve ser criado pelo sistema
R2.4	Sistema não deve permitir que seja criado um novo usuário com e-mail já utilizado em POST e PUT
R2.5	Sistema não deve permitir que sejam cadastrados usuários com e-mail gmail e hotmail
R2.6	E-mails devem seguir padrão válido de e-mail para cadastro (ex: <u>nome@dominio.com</u>)
R2.7	Senhas devem ter de 5 a 10 caracteres
R2.8	Sistema não deve permitir que usuário com carrinho cadastrado seja excluído

R2.9	Sistema deve listar usuários cadastrados, buscar usuários por ID, excluir usuários e
	atualizar usuários corretamente

/produtos ∂

Baseados nos critérios de aceite da US 003: [API] Produtos.

ID	Descrição
R3.1	Sistema não deve permitir que usuários não autenticados consigam realizar ações na rota
R3.2	Sistema não deve permitir que produtos dentro do carrinho sejam excluídos
R3.3	Se ID informado no momento do UPDATE for de um produto inexistente, um novo produto deve ser criado pelo sistema
R3.4	Sistema não deve permitir o cadastro em POST e PUT de produtos com nomes já utilizado
R3.5	Sistema deve listar produtos cadastrados, cadastrar produtos, buscar produtos por ID, excluir produtos e atualizar produtos corretamente

/carrinhos ∂

Baseados na documentação Swagger.

ID	Descrição
R4.1	Sistema não deve permitir que usuários não autenticados consigam realizar ações na rota
R4.2	Sistema deve cadastrar apenas um carrinho por usuário
R4.3	Sistema deve reduzir a quantidade no cadastro do produto que foi inserido em um carrinho com cadastro bem sucedido
R4.4	Sistema deve excluir um carrinho quando uma compra for concluída
R4.5	Sistema deve excluir o carrinho e atualizar o estoque dos produtos contidos nele quando uma compra for cancelada
R4.6	Sistema deve listar carrinhos cadastrados e buscar carrinhos por ID corretamente

📋 Cenários de teste planejados 🛭

ID	Descrição	Pré-	Passo a passo	Resultad os	Resultad os	Status	Requisit os que	Priorida de 1
		S		esperad	obtidos		cobre	ue <u>i</u>
				os				

CT01	Login com usuário existente	Usuário cadastrad o	 Criar requisição POST para /login Incluir e-mail e senha válidos no corpo da requisição 	200 e Token Bearer do usuário	200 e Token Bearer do usuário	∨ Passou	R1.4	Alta
CT02	Login com usuário existente e senha incorreta	Usuário cadastrad o	1. Criar requisição POST para /login 2. Incluir e-mail válido e senha inválida no corpo da requisição	401	401	Passou	R1.2 e R1.3	Alta
CT03	Login com usuário inexistente	Nenhuma	 Criar requisição POST para /login Incluir e-mail e senha inválidos no corpo da requisição 	401	401	Passou	R1.1 e R1.3	Alta
CT04	Ação na API com token expirado	Token expirado	 Criar requisição POST para /login Incluir e-mail e senha válidos no corpo da requisição Usar token retornado Aguardar expiração do token 	401	401	V Passou	R1.5	Alta
CT05	Cadastrar usuário com os campos válidos	Dados de usuário válidos	 Criar requisição POST para /usuarios Incluir nome, e-mail, senha e administrador válidos no corpo da requisição 	201 e ID do novo usuário cadastra do	201 e ID do novo usuário cadastra do	✓ Passou	R2.1	Alto
CT06	Cadastrar usuário com e-mail já utilizado	E-mail de usuário cadastrad o	1. Criar requisição POST para /usuarios 2. Incluir e-mail já utilizado no corpo da requisição	400	400	✓ Passou	R2.4	Alta
СТ07	Cadastrar usuário com e-mail gmail e hotmail	E-mail gmail e e- mail hotmail	1. Criar requisição POST para /usuarios e incluir e-mail com @gmail.com no corpo da requisição 2. Criar requisição POST para /usuarios e	400	201	X Falhou	R2.5	Médi

			incluir e-mail com @hotmail.com no corpo da requisição					
СТ08	Cadastrar usuário com e-mail fora do padrão válido	Nenhuma	1. Criar requisição POST para /usuarios 2. Incluir e-mail fora do formato padrão nome@dominio.com no corpo da requisição	400	400	∨ Passou	R2.6	Média
CT09	Cadastrar usuário com senha válida	Senha com 5 a 10 caractere s	 Criar requisição POST para /usuarios Incluir password válida com 5 a 10 caracteres no corpo da requisição 	201 e ID do novo usuário cadastra do	201 e ID do novo usuário cadastra do	V Passou	R2.7	Alta
CT10	Cadastrar usuário com senha inválida curta	Senha com menos que 5 caractere s	1. Criar requisição POST para /usuarios 2. Incluir password inválida com menos de 5 caracteres no corpo da requisição	400	200	X Falhou	R2.7	Médio
CT11	Cadastrar usuário com senha inválida longa	Senha com mais que 10 caractere s	 Criar requisição POST para /usuarios Incluir senha inválida com mais de 10 caracteres no corpo da requisição 	400	200	X Falhou	R2.7	Médi
CT12	Listar usuários cadastrados	Parâmetr os válidos	 Criar requisição GET para /usuarios Incluir nome, e-mail, senha, administrador e ID do usuário válidos no parâmetros da requisição 	200	200	✓ Passou	R2.9	Médi
CT13	Buscar usuário com ID válido	Parâmetr os e dados válidos	 Criar requisição GET para /usuarios/{_id} Incluir ID do usuário válido no parâmetros da requisição 	200	200	∨ Passou	R2.9	Médi

CT14	Buscar usuário com ID inexistente	Parâmetr os e dados inválidos	1. Criar requisição GET para /usuarios/{_id} 2. Incluir ID do usuário que não existe no parâmetros da requisição	400	400	∨ Passou	R2.2	Baix
CT15	Atualizar usuário com ID válido	Parâmetr os e dados válidos	1. Criar requisição PUT para /usuarios/{_id} 2. Incluir ID do usuário válido no parâmetros e nome, e-mail, senha e administrador válidos no corpo da requisição	200	200	V Passou	R2.9	Méd
CT16	Atualizar usuário com ID inexistente	ID não válido	1. Criar requisição PUT para /usuarios/{_id} 2. Incluir ID do usuário que não existe no parâmetros e nome, e-mail, senha e administrador válidos no corpo da requisição	201 e ID do novo usuário cadastra do	201 e ID do novo usuário cadastra do	✓ Passou	R2.3	Méd
CT17	Excluir usuário com ID válido	Parâmetr os e dados válidos	1. Criar requisição DELETE para /usuarios/{_id} 2. Incluir ID do usuário válido no parâmetros da requisição	200	200	✓ Passou	R2.9	Méc
CT18	Excluir usuário com ID inexistente	ID inválido	1. Criar requisição DELETE para /usuarios/{_id} 2. Incluir ID do usuário que não existe no parâmetros da requisição	200 ("Nenhu m registro excluído"	200 ("Nenhu m registro excluído"	∨ Passou	R2.2	Baix
CT19	Excluir usuário com carrinho cadastrado	ID de usuário	1. Criar requisição DELETE para /usuarios/{_id}	400	400	✓ Passou	R2.8	Alte

		com	Incluir ID do usuário que possui um carrinho cadastrado no parâmetros da requisição					
CT20	Cadastrar produto em POST com dados válidos	Dados válidos e usuário autentica do	 Criar requisição POST para /produtos Incluir nome, preço, descrição e quantidade no corpo da requisição Adicionar token de autenticação no cabeçalho 	201 e ID do produto cadastra do	201 e ID do produto cadastra do	V Passou	R3.5	Alta
CT21	Cadastrar produto em POST com usuário não autenticado	Usuário não autentica do	 Criar requisição POST para /produtos Incluir nome, preço, descrição e quantidade no corpo da requisição Não adicionar token de autenticação no cabeçalho 	401	401	✓ Passou	R3.1	Alta
CT22	Cadastrar produto em POST com nome já utilizado	Produto já cadastrad o	1. Criar requisição POST para /produtos e incluir nome já utilizado no corpo da requisição 2. Adicionar token de autenticação no cabeçalho	400	400	V Passou	R3.4	Média
CT23	Cadastrar produto em PUT com nome já utilizado	Produto já cadastrad o	1. Criar requisição PUT para /produtos/{_id} e incluir nome já utilizado no corpo da requisição 2. Adicionar token de autenticação no cabeçalho	400	400	V Passou	R3.4	Média

CT24	Listar produtos cadastrados	Dados válidos	 Criar requisição GET para /produtos Incluir ID do produto, nome, preço, descrição e quantidade válidos no parâmetros da requisição 	200	200	✓ Passou	R3.5	Média
CT25	Buscar produto com ID válido	Parâmetr os e dados válidos e usuário autentica do	 Criar requisição GET para /produtos/{_id} Incluir ID do produto válido no parâmetros da requisição 	200	200	✓ Passou	R3.5	Média
CT26	Atualizar produto com ID válido	Parâmetr os e dados válidos e usuário autentica do	1. Criar requisição PUT para /pxodutos/{_id} 2. Incluir ID do produto válido no parâmetros e nome, preço, descrição e quantidade válidos no corpo da requisição 3. Adicionar token de autenticação no cabeçalho	200	200	V Passou	R3.5	Média
CT27	Atualizar produto com usuário não autenticado	Usuário não autentica do	1. Criar requisição PUT para /produtos/{_id} 2. Incluir ID do produto válido no parâmetros e nome, preço, descrição e quantidade válidos no corpo da requisição 3. Não adicionar token de autenticação no cabeçalho	401	401	V Passou	R3.1	Alta
CT28	Atualizar produto com ID inexistente	Usuário autentica do	 Criar requisição PUT para /produtos/{_id} Incluir ID do produto que não existe no parâmetros e nome, preço, descrição e quantidade válidos no corpo da requisição 	201 e ID do novo produto cadastra do	201 e ID do produto cadastra do	✓ Passou	R3.3	Alta

			Adicionar token de autenticação no cabeçalho					
CT29	Excluir produto com ID válido	Parâmetr os e dados válidos e usuário autentica do	 Criar requisição DELETE para /produtos/{_id} Incluir ID do produto válido no parâmetros da requisição Adicionar token de autenticação no cabeçalho 	200	200	∨ Passou	R3.5	Médio
СТ30	Excluir produto com usuário não autenticado	Usuário não autentica do	 Criar requisição DELETE para /produtos/{_id} Incluir ID do produto válido no parâmetros da requisição Não adicionar token de autenticação no cabeçalho 	401	401	∨ Passou	R3.1	Alta
CT31	Excluir produto dentro do carrinho	Carrinho cadastrad o e usuário autentica do	1. Criar requisição DELETE para /produtos/{_id} e incluir ID de um produto que esteja no carrinho do usuário no parâmetros da requisição 2. Adicionar token de autenticação no cabeçalho	400	400	✓ Passou	R3.2	Alta
CT32	Cadastrar carrinho com usuário válido	Usuário válido e autentica do	 Criar requisição POST para /carrinhos Incluir ID e quantidade dos produtos no corpo da requisição Adicionar token de autenticação no cabeçalho 	201 e redução da quantida de do produto que foi inserido	201 e redução da quantida de do produto que foi inserido	∨ Passou	R4.3	Alta
CT33	Cadastrar carrinho com usuário que já possui um carrinho	Usuário válido com carrinho e	 Criar requisição POST para /carrinhos Incluir ID de usuário que já possui um 	400	400	✓ Passou	R4.2	Alta

		autentica do	carrinho no corpo da requisição 3. Adicionar token de autenticação no cabeçalho					
CT34	Listar carrinhos cadastrados	Dados válidos	 Criar requisição GET para /carrinhos Incluir ID do carrinho, preço total, quantidade total e ID do usuário válidos no parâmetros da requisição 	200	200	✓ Passou	R4.6	Média
CT35	Buscar carrinho com ID válido	Parâmetr o válido	 Criar requisição GET para /carrinhos/{_id} Incluir ID de carrinho válido no parâmetros da requisição 	200	200	✓ Passou	R4.6	Média
CT36	Excluir carrinho ao concluir compra	Usuário válido com carrinho e autentica do	1. Criar requisição DELETE para /carrinhos/concluir -compra 2. Rodar requisição 3. Adicionar token de autenticação no cabeçalho	200 e carrinho excluído	200	✓ Passou	R4.4	Média
СТ37	Excluir carrinho ao concluir compra com usuário não autenticado	Usuário não autentica do	1. Criar requisição DELETE para /carrinhos/concluir -compra 2. Rodar requisição 3. Não adicionar token de autenticação no cabeçalho	401	401	V Passou	R4.1	Alta
CT38	Excluir carrinho ao cancelar compra	Usuário válido com carrinho e autentica do	1. Criar requisição DELETE para /carrinhos/cancelar -compra 2. Rodar requisição 3. Adicionar token de autenticação no	200, carrinho excluído e reabaste cimento do	200	✓ Passou	R4.5	Alta

			cabeçalho	estoque dos produtos contidos nele				
CT39	Excluir carrinho ao cancelar compra com usuário não autenticado	Usuário não autentica do	 Criar requisição DELETE para /carrinhos/cancelar -compra Rodar requisição Não adicionar token de autenticação no cabeçalho 	401	401	V Passou	R4.1	Alta

🚧 Erros e Melhorias 🛭

Os tickets do Jira abaixo representam falhas encontradas e oportunidades de melhoria levantadas durante o processo de testes da API. Cada ponto foi documentado no Jira, a fim de garantir maior controle e melhor acompanhamento.

Туре	Key	Resumo	Prioridade		
↑	CH-44	{US 001} Ajuste no segundo critério de aceitação. Ajuda no melhor entendimento do funci	= Medium		
↑	CH-43	{Mensagens de erro} API deveria enviar mensagens na resposta que deixem claro ao us	= Medium		
X	CH-42	{CT11} API cria usuário com senha inválida mais longa que o solicitado. Pode prejudicar l	= Medium		
X	CH-41	{CT10} API cria usuário com senha inválida mais curta que o solicitado. Pode prejudicar l	= Medium		
X	CH-40	{CT07} API cria usuário com e-mail gmail e hotmail. Prejudica usuário que será cadastra	= Medium		
	Sincronizado agora - 5 itens				

<u>∧</u> Matriz de risco *₀*

ID	Título	Probabilidade (1-3)	Impacto (1-3)	Risco
Risco 001	Falha na autenticação de usuários permite acessos não autorizados	2	3	6
Risco 002	Ausência de validação dos dados inseridos	2	3	6
Risco 003	Queda da API	2	3	6
Risco 004	Tempo insuficiente para execução de todos os testes planejados	3	2	6

Risco 005	Documentação da API desatualizada	2	2	4
Risco 006	Sistema não trata erros corretamente (mensagens genéricas)	2	2	4
Risco 007	Falha no ambiente de teste	2	2	4
Risco 008	Dados inconsistentes ou ausentes na base usada para testes	2	2	4
Risco 009	Mudanças na API sem aviso prévio	1	3	3
Risco 010	Token expirado permite continuar acessando a API	1	3	3

🔖 Testes automatizados ∅

Os principais testes que podem ser automatizados são os repetitivos e usados com frequência, os que possuem requisitos estáveis, os ligados a fluxos críticos do funcionamento da aplicação e os que apresentam resultados simples e previsíveis. Os cenários de teste criados que se encaixam nessa descrição e que foram automatizados são:

Critérios de automação	Cenários de teste	Descrição dos cenários	Justificativa para automação
Usados com frequência e fluxos críticos	CT01, CT02 e CT03	Relacionados ao login	Primeiro contato dos usuários com a aplicação e necessário para que loja possa ser acessada
Requisitos estáveis e fluxos críticos	CT07, CT08, CT09, CT10 e CT11	Cadastro de usuários com padrões específicos de e-mail e senha	Importante para validação de regras de negócio
Fluxos críticos	CT05, CT06, CT20, CT21, CT22 e CT32	Cadastro de usuários, produtos e carrinhos	Um dos principais fluxos do sistema, já que aplicação se trata de uma loja virtual que necessita de produtos para funcionar
Resultados simples e previsíveis	CT12, CT13, CT14, CT24, CT25, CT34 e CT35	Listagem e busca de usuários, produtos e carrinhos	Importantes para verificação de dados salvos e para garantir que estejam sendo corretamente retornados pelo sistema

Os testes ainda não automatizados poderão ser implementados futuramente, de acordo com a necessidade e prioridade do projeto.

Resultado no Robot Framework @



Observação: No arquivo do Robot Framework, os testes que falharam foram 4, diferente do reportado anteriormente nos manuais. Isso porque o CT07 foi dividido para realizar uma análise mais específica de cada domínio de e-mail (gmail e hotmail) e gerar resultados mais claros. Assim, os erros reportados nos testes manuais foram os mesmos que ocorreram nos testes automatizados. Essa alteração também explica por que há 22 testes automatizados, enquanto a tabela acima apresenta um total de 21 cenários de teste.



📈 Cobertura de testes (Manuais e Automatizados) 🛭

Path Coverage (input) @

Quantidade de endpoints na API REST: 9

Quantidade de endpoints testados de forma manual: 9

<u>Total manual</u> = 9/9 = 1 = 100%

Quantidade de endpoints testados de forma automatizada: 7

 $\underline{\text{Total automatizada}} = 7/9 = 0.78 = 78\%$

Operator Coverage (input) @

Quantidade de métodos na API REST: 16

Quantidade de métodos testados de forma manual: 16

<u>Total manual</u> = 16/16 = 1 = 100%

Quantidade de métodos testados de forma automatizada: 10

<u>Total automatizada</u> = 10/16 = 0,625 = 62,5%

Parameter Coverage (input) *⊘*

Quantidade de parâmetros na API REST: 27

Quantidade de parâmetros testados de forma manual: 27

<u>Total manual</u> = 27/27 = 1 = 100%

Quantidade de parâmetros testados de forma automatizada:

<u>Total automatizada</u> = 21/27 = 0,78 = 78%

Content-Type Coverage (input e output) *⊘*

Quantidade de métodos na API REST: 16 (há apenas a opção de JSON)

Quantidade de Content-Type testados de forma manual: 16

<u>Total manual</u> = 16/16 = 1 = 100%

Quantidade de Content-Type testados de forma automatizada:

<u>Total automatizada</u> = 10/16 = 0,625 = 62,5%

Status Code Coverage (Output) @

Quantidade de status code na API REST: 38

Quantidade de status code testados de forma manual: 33

<u>Total manual</u> = 33/38 = 0,87 = 87%

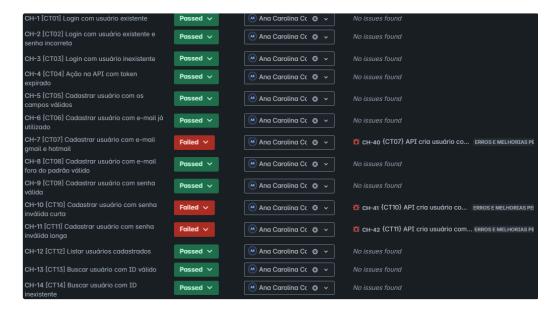
Quantidade de status code testados de forma automatizada:

 $\underline{\text{Total automatizada}} = 15/38 = 0,4 = 40\%$

📊 Resultados finais 🛭

Testes manuais 🕖

Ciclo de teste no plug-in QAlity no Jira:



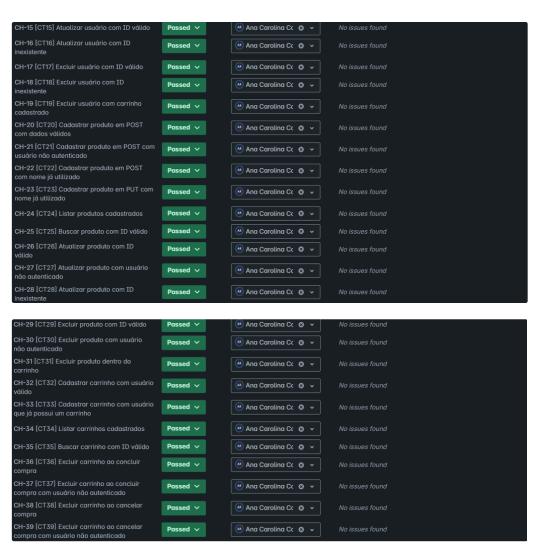
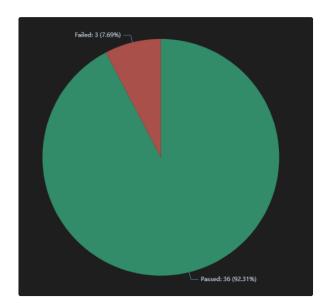


Gráfico gerado:



Testes automatizados @

Ciclo de teste no plug-in QAlity no Jira:

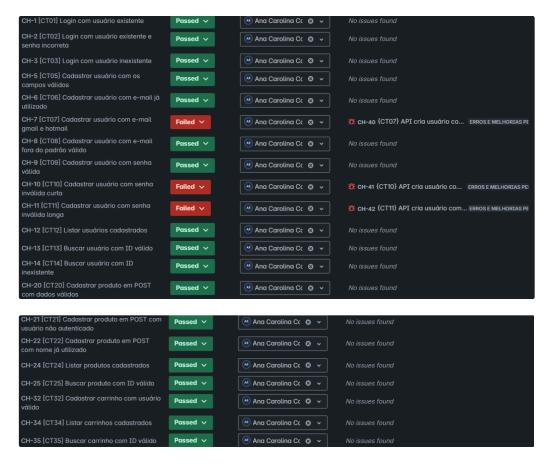
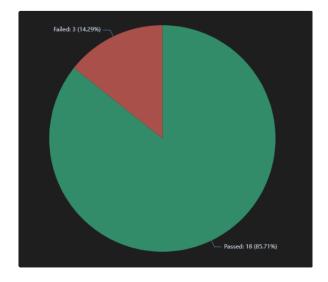


Gráfico gerado:



✓ Conclusão

De modo geral, a maioria dos testes foi bem-sucedida. No entanto, os poucos casos de falha revelaram pontos de atenção importantes, principalmente por envolverem regras de negócio ainda não contempladas pelo sistema. Destaca-se que os testes reprovados estão relacionados ao processo de login, que é uma funcionalidade essencial e a primeira interação do cliente com a aplicação. Problemas nessa rota da API podem comprometer a experiência do usuário logo no primeiro contato, afetando negativamente a percepção de confiabilidade do sistema.

Vale destacar que os testes manuais, elaborados inicialmente, priorizaram uma ampla cobertura dos requisitos, evidenciado pela análise de cobertura. O objetivo principal a partir deles foi garantir o alinhamento dos cenários de teste com as User Stories e a

documentação Swagger da API ServeRest. Já os testes automatizados focaram em cenários críticos, recorrentes e nos que possuíam resultados simples de avaliar. A verificação de respostas simples permite que o tempo de execução manual seja direcionado a casos que exigem maior atenção. Assim, os testes automatizados foram estruturados para permitir execuções frequentes e validação rápida de funcionalidades importantes para o sistema.

Em resumo, a execução dos testes contribuiu não apenas para a validação da aplicação, mas também para identificação de riscos, verificação de comportamentos esperados e definição de ações prioritárias.