

Kauã Raffaello (challenge 03) - Plano de testes do e-commerce ServeRest.

Apresentação

- Este documento descreve a estratégia de testes para a API ServeRest, no qual simula um e-commerce para fins de aprendizado.
-

Objetivo

- Garantir a qualidade do e-commerce ServeRest por meio de testes que garantem que os endpoints estejam funcionais garantindo a segurança da aplicação.
-

Resumo


- O ServeRest é um E-commerce genérico com os usuário de administrador, administrador do sistema, e usuário comum, usuário-alvo da plataforma. Esse planejamento de teste será realizado por conta de problemas de ferramentas do sistema. O planejamento de testes terão certas pessoas envolvidas, além do escopo explicado dos testes, estratégias, critérios, cronograma, recursos, ambiente de testes e possíveis riscos.
 - A hipótese deste teste é que os endpoints estejam todos corretos e funcionais.
-

Pessoas Envolvidas:

Neste planejamento teve apenas um total de 1 QA envolvido.

- Kauã Raffaello - QA
-

Ambientação que será testado

- URL do Swagger:  [ServeRest](#)
 - Ferramentas: Robot Framework, Jira, GitHub.
 - Dependências: Token de login.
-

Recursos e Ferramentas

- IDE: Vscode.

- Frameworks: Robot Framework.
 - Libraries Necessárias:
 - **RequestsLibrary** - Para requisições HTTP
 - **Collections** - Manipulação de dicionários e listas
 - **String** - Validações de strings
 - **BuiltIn** - Keywords nativas do Robot
 - GitHub para versionamento e controle CI/CD.
-

Escopo

- Endpoints de teste:

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

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

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

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

Fora do escopo:

- Integrações externas, conexões.
-

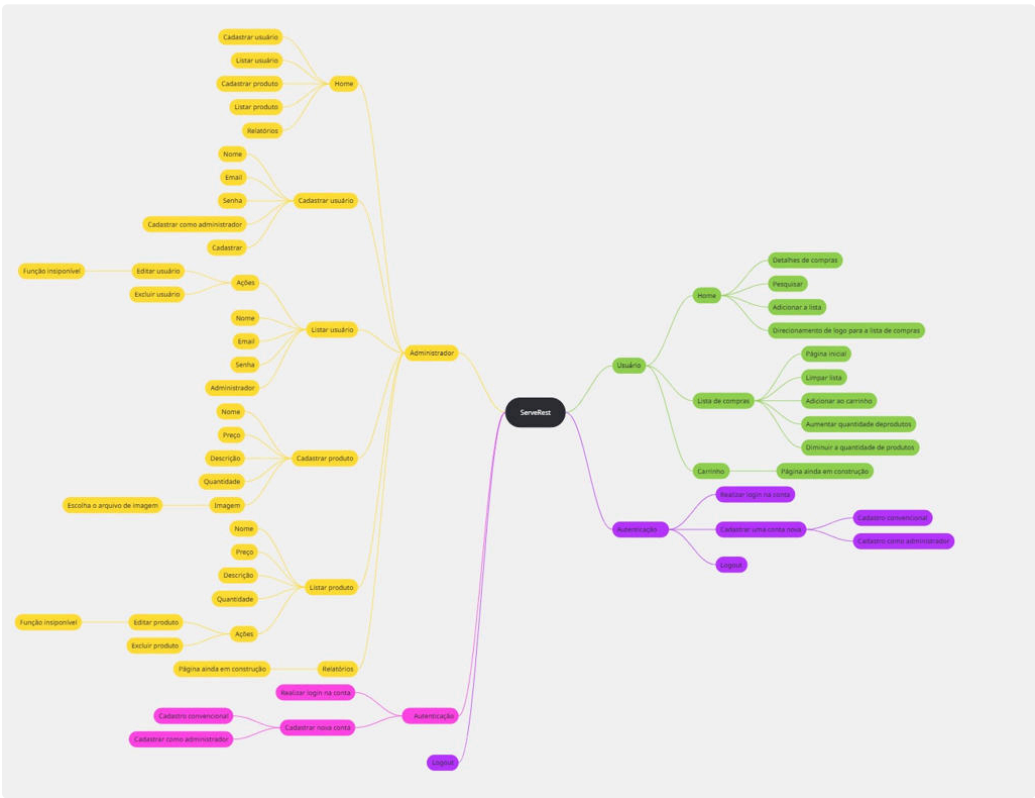
Análise

- Iremos focar em testes automatizados dos devidos endpoints.
 - Riscos: API ServeRest instável.
-

Técnicas Aplicadas

- Teste de caixa-branca.
 - Testes de API automatizado.
 - Teste funcional automatizado.
-

Mapa mental da aplicação



Cronograma do planejamento até a entrega do produto:

Dia	Atividade
Dia 1 (segunda)	Planejamento completo dos testes
Dia 2 (terça)	Adicionar no QALity Plus no Jira + Configuração do Robot Framework
Dia 3 (quarta)	Execução dos testes no Robot Framework
Dia 4 (quinta)	Report de issues encontradas
Dia 5 (sexta)	Entrega final do produto

Cenários de teste planejado

1. Produto

1.1 Criar produto com dados válidos (POST)

Pré-condição: Usuário autenticado como vendedor.

Ação: Enviar POST para `/produtos` com dados válidos.

Resultado esperado:

- Produto cadastrado com sucesso, HTTP 201.
- Resposta retorna ID e dados cadastrados.
- Campos obrigatórios preenchidos corretamente.

Cenários de borda:

- Preço = 0
- Nome muito longo
- Descrição vazia ou muito longa
- Tags ou categorias extras

Mensagens de erro reforçadas:

- “Preço deve ser maior que zero”
 - “Nome é obrigatório”
 - “Formato de campo inválido”
-

1.2 Criar produto com dados inválidos (POST)

Pré-condição: Usuário autenticado como vendedor.

Ação: Enviar POST com dados inválidos (preço negativo, nome vazio, campos ausentes).

Resultado esperado:

- Produto **não cadastrado**, HTTP 400.
- Mensagem de erro clara e específica para cada falha.

Cenários de borda:

- Preço como texto
 - Nome com caracteres especiais
 - Campos extras inesperados
-

1.3 Atualizar produto inexistente (PUT)

Pré-condição: Produto não existe.

Ação: Enviar PUT para `/produtos/{id}`.

Resultado esperado:

- Novo produto criado, HTTP 201.
- Resposta retorna ID e dados enviados.

Cenários de borda:

- ID inválido (texto, caracteres especiais)
 - Dados parcialmente válidos
-

2. Usuário

2.1 Criar usuário com dados válidos (POST)

- Pré-condição: Nenhum usuário com mesmo e-mail.
- Ação: POST `/usuarios` com dados válidos.
- Resultado esperado: Usuário cadastrado, HTTP 201, ID retornado.

2.2 Criar usuário com dados inválidos (POST)

- Dados duplicados ou obrigatórios faltando.
 - Resultado esperado: HTTP 400 com mensagem de erro específica.
 - Cenários de borda: e-mail inválido, senha curta, caracteres especiais no nome.
-

3. Carrinho

3.1 Adicionar produto ao carrinho (POST)

- Pré-condição: Usuário autenticado, produto existente.
- Ação: POST `/carrinhos/{id}/produtos`.
- Resultado esperado: Produto adicionado, HTTP 201, detalhes do carrinho atualizados.

3.2 Adicionar produto inexistente

- Resultado esperado: HTTP 404 com mensagem de erro clara: “Produto não encontrado”.

3.3 Remover produto do carrinho (DELETE)

- Pré-condição: Produto existe no carrinho.
- Resultado esperado: Produto removido, HTTP 200, carrinho atualizado.

Cenários de borda:

- Carrinho vazio ao tentar remover
 - Adicionar produto duplicado
 - IDs inválidos de carrinho ou produto
-

Priorização da execução dos cenários de teste

Cenário	Prioridade	Justificativa
Criar produto com dados válidos (POST)	Alta	Fluxo principal da API de produtos; falha aqui compromete todo o cadastro.
Criar produto com dados inválidos (POST)	Alta	Garante validação de dados; falha pode permitir produtos inválidos.
Atualizar produto inexistente (PUT)	Média	Exceção importante, mas não impede funcionamento normal.
Criar usuário com dados válidos (POST)	Alta	Fluxo crítico de registro de usuário; impede cadastro duplicado.
Criar usuário com dados inválidos (POST)	Alta	Garantia de integridade de dados do usuário.
Adicionar produto ao carrinho (POST)	Alta	Fluxo principal de uso do carrinho; falha impacta experiência do usuário.
Adicionar produto inexistente ao carrinho	Média	Exceção; importante validar mensagens de erro.
Remover produto do carrinho (DELETE)	Média	Garantir consistência do carrinho; falha não

		bloqueia outros fluxos.
--	--	-------------------------

Matriz de risco

Cenário	Impacto	Probabilidade	Classificação
Criar produto com dados válidos	Alto	Alta	Crítico
Criar produto com dados inválidos	Alto	Alta	Crítico
Atualizar produto inexistente	Médio	Média	Alto
Criar usuário com dados válidos	Alto	Alta	Crítico
Criar usuário com dados inválidos	Alto	Alta	Crítico
Adicionar produto ao carrinho	Alto	Alta	Crítico
Adicionar produto inexistente ao carrinho	Médio	Média	Alto
Remover produto do carrinho	Médio	Média	Alto

Cobertura de testes

Cenário	Endpoint	Cobertura API
---------	----------	---------------

Criar produto com dados válidos	<code>/produtos</code>	Cadastro completo de produto; valida campos obrigatórios.
Criar produto com dados inválidos	<code>/produtos</code>	Validação de regras e campos obrigatórios; automatização via pre-request para dados aleatórios.
Atualizar produto inexistente	<code>/produtos/{id}</code>	Criação de produto quando ID não existe; IDs gerados dinamicamente.
Criar usuário com dados válidos	<code>/usuarios</code>	Cadastro de usuário; valida campos obrigatórios e duplicidade.
Criar usuário com dados inválidos	<code>/usuarios</code>	Validação de campos e regras de negócio; mensagens de erro específicas.
Adicionar produto ao carrinho	<code>/carrinhos/{id}/produtos</code>	Adição de produtos; valida carrinho e quantidade de itens.
Adicionar produto inexistente ao carrinho	<code>/carrinhos/{id}/produtos</code>	Validação de erro quando produto não existe; mensagem clara.
Remover produto do carrinho	<code>/carrinhos/{id}/produtos/{idProduto}</code>	Remoção de produtos; valida atualização do carrinho.

Testes candidatos a automação

Cenário	Endpoint	Justificativa para automação
Criar produto com dados válidos (POST)	<code>/produtos</code>	Fluxo principal de cadastro; precisa ser validado sempre em regressão.
Criar produto com dados inválidos (POST)	<code>/produtos</code>	Cenário repetitivo que valida regras e mensagens de erro; ideal para execução automática em cada build.
Atualizar produto inexistente (PUT cria novo)	<code>/produtos/{id}</code>	Cenário de exceção; importante para validar comportamento de criação dinâmica.
Criar usuário com dados válidos (POST)	<code>/usuarios</code>	Fluxo crítico para onboarding; automatizado garante que cadastro nunca quebre.
Criar usuário com dados inválidos (POST)	<code>/usuarios</code>	Teste repetitivo de borda e validação de mensagens de erro; automatização assegura integridade de dados.
Adicionar produto ao carrinho (POST)	<code>/carrinhos/{id}/produtos</code>	Fluxo essencial do e-commerce; automação garante que adições e quantidades funcionem sempre.

Adicionar produto inexistente ao carrinho	<code>/carrinhos/{id}</code> <code>/produtos</code>	Cenário de borda que valida mensagens de erro; automatizado evita falsos positivos na API.
Remover produto do carrinho (DELETE)	<code>/carrinhos/{id}</code> <code>/produtos/{id}</code> <code>Produto</code>	Cenário previsível e repetitivo; automatização garante que a API mantenha consistência dos carrinhos.