ENGENHEIRO DE QUALIDADE DE SOFTWARE

Renan Yassumoto Ferreira

Trabalho de Conclusão de Curso
Análise de Qualidade

Barueri
2026

Resumo

Este Trabalho de Conclusão de Curso tem como objetivo aplicar, de forma prática e estruturada, os conhecimentos adquiridos ao longo da formação em Engenharia de Qualidade de Software, por meio da elaboração e implementação de uma estratégia completa de testes para validação do e-commerce EBAC Shop.

O projeto contemplou a definição de critérios de aceitação, elaboração de casos de teste e automação das camadas de Interface Web, API e Mobile. Foram utilizadas ferramentas modernas amplamente adotadas no mercado, como Cypress para testes End-to-End, Supertest para testes de API, WebdriverIO com Appium para automação mobile, k6 para testes de performance e GitHub Actions para implementação de Integração Contínua.

A estratégia adotada combinou testes funcionais e não funcionais, incluindo validação de regras de negócio, testes negativos, verificação de contratos de API, testes sob carga e execução automatizada em pipeline. Durante a execução das automações, foram identificadas inconsistências relacionadas a regras de negócio não implementadas, demonstrando a efetividade do processo de qualidade aplicado.

Os resultados obtidos evidenciam que a aplicação apresenta estabilidade sob carga moderada, comportamento consistente em fluxos críticos e estrutura adequada para execução contínua de testes. O projeto demonstra aplicação prática dos conceitos de Quality Engineering, integrando planejamento, automação, análise técnica e integração contínua em um ambiente controlado e reproduzível.

Sumário

1. Introdução

A crescente complexidade dos sistemas de software e a necessidade de entregas frequentes e confiáveis tornam a Engenharia de Qualidade um elemento essencial no ciclo de desenvolvimento. Em ambientes modernos, caracterizados por metodologias ágeis e integração contínua, o papel do Quality Engineer vai além da simples execução de testes, abrangendo planejamento estratégico, automação, análise de riscos e validação contínua do produto.

Este Trabalho de Conclusão de Curso tem como propósito consolidar os conhecimentos adquiridos na formação em Engenharia de Qualidade de Software por meio da aplicação prática em um cenário realista: a validação de um e-commerce baseado em WordPress e WooCommerce.

O projeto foi estruturado considerando histórias de usuário previamente refinadas, simulando a atuação em um time ágil. A abordagem adotada envolveu definição de estratégia de testes, modelagem de critérios de aceitação em Gherkin, criação de casos de teste, automação de fluxos críticos, implementação de testes de performance e configuração de pipeline de Integração Contínua.

Foram aplicadas técnicas clássicas de teste, como Partição de Equivalência, Análise de Valor Limite e Tabela de Decisão, além da utilização de ferramentas modernas amplamente empregadas no mercado, garantindo alinhamento entre teoria e prática.

A proposta deste trabalho não se limita à validação funcional, mas busca demonstrar visão sistêmica, maturidade técnica e capacidade analítica, evidenciando como a automação estruturada e a integração contínua contribuem para a confiabilidade e sustentabilidade de sistemas de software.

2. O projeto

O presente Trabalho de Conclusão de Curso tem como objetivo aplicar, de forma prática e estruturada, os conhecimentos adquiridos ao longo da formação em Engenharia de Qualidade de Software, por meio da elaboração e execução de uma estratégia completa de testes para validação do e-commerce EBAC Shop.

O sistema avaliado consiste em uma aplicação web baseada em WordPress com WooCommerce, representando um cenário realista de comércio eletrônico. O projeto foi conduzido considerando as histórias de usuário previamente refinadas, simulando a atuação dentro de um time ágil, no qual o Quality Engineer participa ativamente desde o planejamento até a validação final das entregas.

A proposta não se limita à execução de testes, mas contempla todo o ciclo de atuação de um QE, incluindo:

- Análise de requisitos e critérios de aceitação;
- Definição de estratégia de testes;
- Elaboração de casos de teste;
- Automação de testes End-to-End (UI);
- Automação de testes de API;
- Testes mobile;
- Implementação de Integração Contínua;
- Testes de performance;
- Análise de resultados e evidências técnicas.

Para garantir reprodutibilidade, isolamento e padronização do ambiente, optou-se pela utilização de containers Docker, permitindo a execução local da aplicação de forma controlada e independente de infraestrutura externa. Foram utilizadas as seguintes imagens disponíveis no Docker Hub:

- Banco de Dados: ernestosbarbosa/lojaebacdb
- Aplicação Web: ernestosbarbosa/lojaebac

A inicialização do ambiente é realizada por meio da criação de uma rede Docker dedicada e execução dos containers da aplicação e do banco de dados. Após a inicialização, a loja torna-se acessível localmente via http://localhost:80.

Essa abordagem garante:

- Ambiente isolado e controlado;
- Independência de instabilidades externas;
- Base adequada para execução de testes automatizados e pipelines de Integração Contínua.

Dessa forma, o projeto consolida a aplicação prática das competências de um Engenheiro de Qualidade, demonstrando domínio técnico, visão estratégica e capacidade de análise crítica sobre a qualidade de software.

2.1 Estratégia de Teste

2.1.1 Objetivo

A estratégia de testes definida para o projeto tem como finalidade garantir a qualidade funcional e não funcional do e-commerce EBAC Shop, reduzindo riscos de falhas em ambiente produtivo e assegurando aderência às regras de negócio estabelecidas nas histórias de usuário.

Os principais objetivos são:

- Validar as funcionalidades críticas da aplicação;
- Assegurar conformidade com critérios de aceitação;
- Mitigar riscos técnicos e de negócio;
- Implementar automação de testes para cenários prioritários;
- Estabelecer integração contínua para validação automatizada;
- Avaliar desempenho sob carga controlada.

As histórias de usuário priorizadas neste projeto foram:

- US-0001 – Adicionar item ao carrinho
- US-0002 – Login na plataforma
- US-0003 – API de Cupons

2.1.2 Modelo de Trabalho

O projeto foi conduzido seguindo os princípios da metodologia ágil Scrum, simulando um ambiente real de desenvolvimento incremental e colaborativo.

Metodologia

- Scrum

Artefatos Utilizados

- Product Backlog
- Histórias de Usuário
- Critérios de Aceitação em Gherkin
- Planejamento de Sprint (Sprint Planning)
- Revisão de Sprint (Sprint Review)

Papel do Quality Engineer (QE)

Durante o ciclo de desenvolvimento, o QE atua de forma estratégica e preventiva, assumindo as seguintes responsabilidades:

- Participação ativa no refinamento das histórias;
- Escrita e revisão dos critérios de aceitação;
- Criação de casos de teste baseados em risco;
- Implementação de automação para cenários críticos;
- Garantia de cobertura mínima adequada para funcionalidades prioritárias;
- Análise de resultados e evidências técnicas.

2.1.3 Abordagem de Testes

Foi adotada uma abordagem híbrida, combinando testes manuais e automatizados, visando equilíbrio entre profundidade exploratória e eficiência operacional.

Testes Automatizados

- Caminho feliz (Happy Path)
- Fluxos alternativos
- Fluxos negativos
- Testes de regressão
- Testes de API
- Testes de performance
- Execução em pipeline de Integração Contínua

Testes Manuais

- Testes exploratórios
- Validação visual da interface
- Análise de comportamento inesperado
- Casos não críticos

Essa abordagem permitiu maximizar a cobertura mantendo controle sobre custo e tempo de execução.

2.1.4 Tipos de Testes

Testes Funcionais

- Testes de Interface (UI)
- Testes de API REST
- Testes Mobile (Android)
- Testes End-to-End

Testes Não Funcionais

- Testes de Performance (K6)
- Testes de Regressão Automatizada
- Testes de Integração entre camadas

2.1.5 Técnicas de Teste Aplicadas

Foram utilizadas técnicas clássicas de engenharia de testes para garantir qualidade na modelagem dos cenários:

- Partição de Equivalência
- Análise de Valor Limite
- Tabela de Decisão
- Teste Baseado em Risco
- Caminho Feliz
- Fluxos Alternativos
- Fluxos Negativos

A aplicação dessas técnicas reduziu redundância e aumentou a eficácia dos casos de teste.

2.1.6 Níveis de Teste

A estratégia contemplou diferentes níveis de validação:

- Teste de Sistema
- Teste de Integração
- Teste End-to-End
- Teste de API

Essa distribuição permitiu validar desde regras isoladas até fluxos completos da aplicação.

2.1.7 Ambiente de Testes

O ambiente principal foi configurado localmente utilizando Docker, garantindo isolamento e reprodutibilidade.

Infraestrutura

- Docker local
- Container WordPress (Aplicação)
- Container MySQL (Banco de Dados)
- Rede dedicada para comunicação entre containers

A aplicação foi disponibilizada em: http://localhost:80

O uso de containers reduziu riscos de instabilidade e inconsistência ambiental.

2.1.8 Ferramentas Utilizadas

UI

- Cypress
- Mochawesome Reporter

API

- Supertest
- Node.js

Mobile

- Appium

Performance

- K6

CI/CD

- GitHub Actions

Documentação

- Miro (Mapeamento estratégico)
- Microsoft Word (Relatório formal)

2.1.9 Plataformas Testadas

- Web (Interface do e-commerce)
- API REST (Cupons)
- Aplicação Mobile (Android)

2.1.10 Critérios de Entrada

Os testes foram iniciados somente após:

- História refinada
- Critérios de aceitação definidos
- Ambiente disponível e funcional
- Massa de dados preparada

2.1.11 Critérios de Saída

A etapa de testes foi considerada concluída quando:

- Todos os casos planejados foram executados;
- Defeitos identificados foram registrados;
- Testes automatizados executaram com sucesso;
- Performance validada conforme thresholds;
- Pipeline de CI com status verde.

2.1.12 Riscos Identificados

- Instabilidade do ambiente
- Inconsistência de dados
- Dependência de APIs externas
- Mudanças de requisitos durante execução

Estratégias de Mitigação

- Uso de ambiente Docker isolado
- Massa de dados controlada
- Execução automatizada via CI
- Monitoramento contínuo de falhas

2.1.13 Métricas de Qualidade

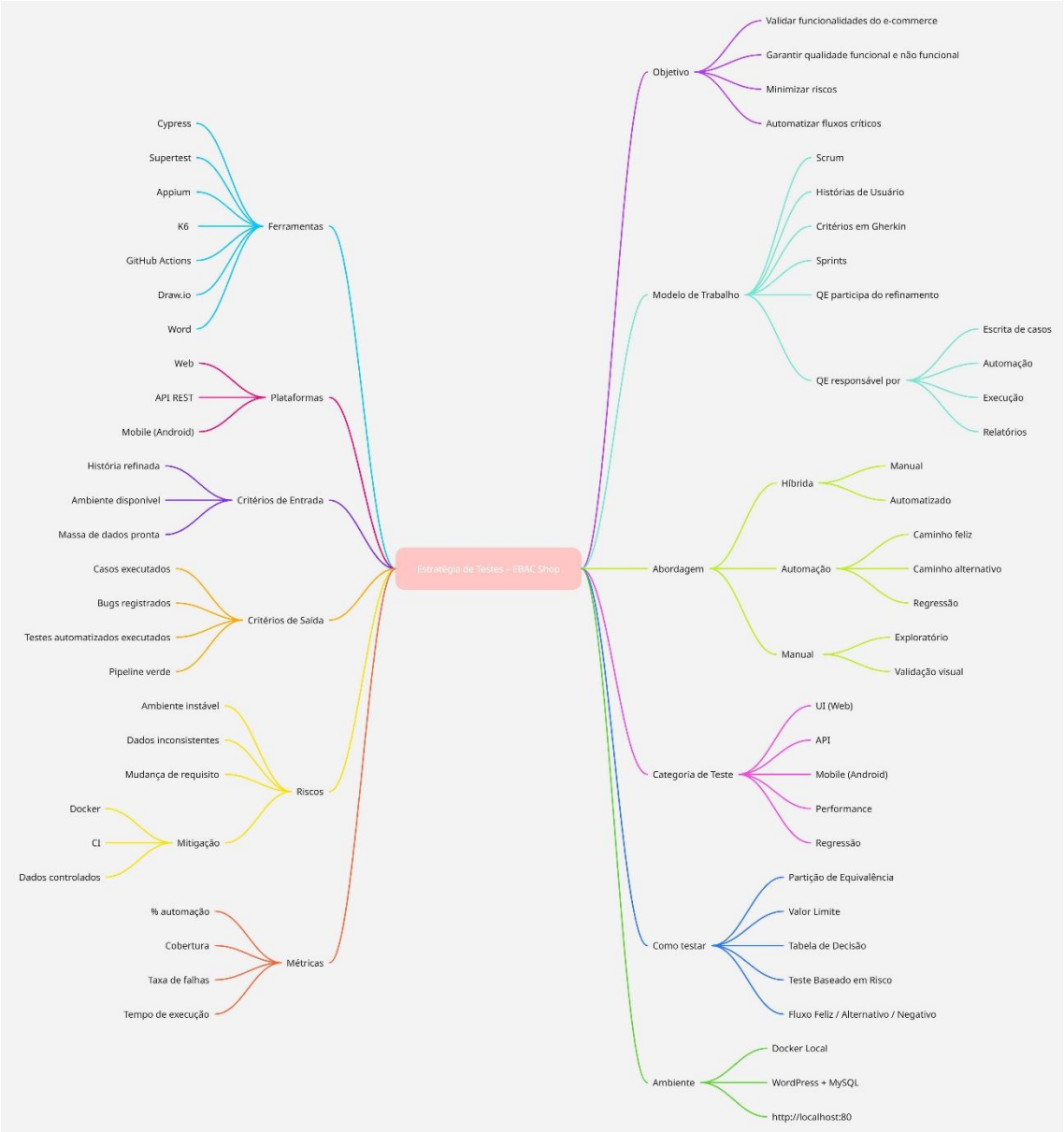
Foram monitoradas as seguintes métricas:

- Cobertura de testes
- Percentual de casos automatizados
- Taxa de falha
- Tempo médio de execução
- Resultado do pipeline de CI
- Métricas de performance (p95, taxa de erro)

2.1.14 Mapa Mental

O planejamento estratégico do projeto foi estruturado por meio de um mapa mental colaborativo, disponível no seguinte link:
https://miro.com/app/board/uXjVGAUfrLA=/?share_link_id=882713391531

O mapa mental auxiliou na visualização da arquitetura de testes, priorização de cenários e organização das entregas.



2.2 Critérios de aceitação

2.2.1 Histórias de Usuário

Foram selecionadas três histórias de usuário consideradas críticas para o funcionamento do e-commerce EBAC Shop. Essas histórias representam funcionalidades centrais da aplicação e impactam diretamente a experiência do usuário e a gestão administrativa da loja.

As histórias contempladas neste projeto são:

- [US-0001] – Adicionar item ao carrinho
- [US-0002] – Login na plataforma
- [US-0003] – API de Cupons

Cada história foi detalhada previamente por meio de critérios de aceitação escritos no padrão Gherkin, garantindo clareza, rastreabilidade e alinhamento entre negócio e validação técnica.

2.2.2 Critérios de Aceitação em Gherkin

Os critérios de aceitação foram modelados utilizando a linguagem Gherkin, no formato Given–When–Then, permitindo:

- Clareza na definição do comportamento esperado;
- Facilitação da automação de testes;
- Comunicação objetiva entre áreas técnicas e de negócio.

1 - Feature: Adicionar item ao carrinho
Como cliente da EBAC-SHOP
Quero adicionar produtos ao carrinho
Para realizar a compra dos itens

Scenario: Adicionar produto ao carrinho com sucesso
Given que o cliente está na página de um produto disponível
When o cliente seleciona a quantidade menor ou igual a 10
And clica no botão "Adicionar ao carrinho"
Then o produto deve ser adicionado ao carrinho
And o carrinho deve exibir o produto com a quantidade selecionada

Scenario: Não permitir adicionar mais de 10 itens do mesmo produto
Given que o cliente está na página de um produto disponível
When o cliente seleciona quantidade maior que 10
And tenta adicionar ao carrinho
Then o sistema deve exibir mensagem de erro informando limite máximo de 10 itens

Scenario: Não permitir valor total acima de R$ 990,00
Given que o cliente possui produtos no carrinho
When o valor total ultrapassar R$ 990,00
Then o sistema deve impedir a adição de novos itens
And deve exibir mensagem informando valor máximo permitido

Scenario: Aplicar desconto de 10% para compras entre R$ 200 e R$ 600
Given que o cliente adicionou produtos ao carrinho
When o valor total estiver entre R$ 200 e R$ 600
Then o sistema deve aplicar automaticamente desconto de 10%

Scenario: Aplicar desconto de 15% para compras acima de R$ 600
Given que o cliente adicionou produtos ao carrinho
When o valor total for superior a R$ 600
Then o sistema deve aplicar automaticamente desconto de 15%

Observação técnica: Durante a execução dos testes automatizados, identificou-se que algumas regras de negócio não estavam implementadas no sistema base, caracterizando evidência válida de inconsistência funcional.

2 - Feature: Login na plataforma
Como cliente da EBAC-SHOP
Quero realizar login na plataforma
Para visualizar meus pedidos

Scenario: Login com credenciais válidas
Given que o usuário está na página de login
And possui cadastro ativo na plataforma
When informa usuário e senha válidos
And clica no botão "Entrar"
Then o sistema deve autenticar o usuário
And redirecionar para a página Minha Conta

Scenario: Exibir erro para login inválido
Given que o usuário está na página de login
When informa usuário ou senha inválidos
And clica no botão "Entrar"
Then o sistema deve exibir mensagem de erro informando credenciais inválidas

Scenario: Não permitir login para usuário inativo
Given que o usuário possui cadastro inativo
When informa credenciais válidas
And tenta realizar login
Then o sistema deve impedir o acesso
And exibir mensagem informando que o usuário está inativo

Scenario: Bloquear usuário após 3 tentativas inválidas
Given que o usuário informou senha incorreta por 3 vezes consecutivas
When tenta realizar novo login
Then o sistema deve bloquear o acesso por 15 minutos
And exibir mensagem informando bloqueio temporário

Scenario: Permitir login após período de bloqueio
Given que o usuário foi bloqueado por 15 minutos
When o tempo de bloqueio expirar
And o usuário informar credenciais válidas
Then o sistema deve permitir o login normalmente

Observação técnica: Algumas regras de bloqueio não estavam implementadas na aplicação base, o que foi devidamente evidenciado nos testes automatizados.

3 - Feature: API de Cupons
Como administrador da EBAC-SHOP
Quero criar e listar cupons
Para gerenciar descontos da loja

Scenario: Listar todos os cupons com autenticação válida
Given que o administrador possui credenciais válidas
When realiza requisição GET para /wc/v3/coupons
Then a API deve retornar status 200
And deve retornar lista de cupons cadastrados

Scenario: Não permitir acesso com autenticação inválida
Given que o usuário informa credenciais inválidas
When realiza requisição GET para /wc/v3/coupons
Then a API deve retornar status 401
And deve exibir mensagem de não autorizado

Scenario: Buscar cupom por ID válido
Given que existe um cupom previamente cadastrado
When realiza requisição GET para /wc/v3/coupons/{id}
Then a API deve retornar status 200
And deve retornar os dados do cupom correspondente

Scenario: Retornar erro ao buscar cupom com ID inexistente
Given que não existe cupom com determinado ID
When realiza requisição GET para /wc/v3/coupons/{id}
Then a API deve retornar status 404

Scenario: Cadastrar cupom com dados obrigatórios válidos
Given que o administrador possui autenticação válida
When envia requisição POST com campos obrigatórios preenchidos
code	Ganhe10
amount	10
discount_type	fixed_product
description	Cupom de teste
Then a API deve retornar status 201
And deve cadastrar o cupom com sucesso

Scenario: Não permitir cadastro de cupom com código duplicado
Given que já existe um cupom com código "Ganhe10"
When envia requisição POST com o mesmo código
Then a API deve retornar erro
And não deve cadastrar o cupom novamente

Scenario: Não permitir cadastro sem campos obrigatórios
Given que o administrador possui autenticação válida
When envia requisição POST sem o campo "code"
Then a API deve retornar erro de validação
And deve informar campo obrigatório ausente

2.2.3 Histórias de Usuário Complementares

Além das histórias principais (US-0001 a US-0003), foram identificadas funcionalidades complementares essenciais para o fluxo completo do e-commerce. Essas histórias ampliam a cobertura funcional e contribuem para validação estrutural da aplicação.

US-004 — Catálogo de Produtos
URL: /produtos

Objetivo
Permitir que usuários, autenticados ou não, visualizem a listagem de produtos disponíveis na loja, incluindo informações essenciais para decisão de compra.

Descrição
Como usuário visitante ou autenticado
Eu quero visualizar os produtos disponíveis
Para selecionar um item para compra

Critérios de Aceite

1. O sistema deve exibir a listagem de produtos ao acessar /produtos;
2. Cada produto deve apresentar:
    - Imagem;
    - Nome;
    - Preço;
3. O valor deve estar formatado em moeda brasileira (R$);
4. Deve existir paginação quando houver múltiplas páginas;
5. A ordenação padrão deve estar disponível ao usuário.

Regras de Negócio

- O catálogo deve estar acessível para usuários não autenticados;
- O carregamento dos produtos deve ocorrer automaticamente ao acessar a página;
- Caso não existam produtos cadastrados, o sistema deve exibir mensagem informativa apropriada.

US-005 — Painel Minha Conta
URL: /minha-conta (após login)

Objetivo
Permitir que o usuário autenticado visualize e gerencie sua área pessoal.

Descrição
Como usuário autenticado
Eu quero acessar minha área logada
Para gerenciar minhas informações

Critérios de Aceite
1. Apenas usuários autenticados devem acessar a página;
2. Deve ser exibida mensagem de boas-vindas contendo o nome do usuário;
3. Deve existir menu lateral contendo:
    - Painel;
    - Pedidos;
    - Downloads;
    - Endereços;
    - Detalhes da Conta;
    - Sair;
4. O botão “Sair” deve encerrar a sessão corretamente.

Regras de Negócio
- Caso o usuário não esteja autenticado, deve ser redirecionado para a página de login;
- O nome exibido deve corresponder ao usuário autenticado.

US-006 — Meus Pedidos
URL: /minha-conta/orders (após login)

Objetivo
Permitir que o usuário autenticado visualize o histórico de pedidos realizados.

Descrição
Como usuário autenticado
Eu quero consultar meus pedidos
Para acompanhar minhas compras

Critérios de Aceite
1. Apenas usuários autenticados devem acessar a página;
2. Caso não existam pedidos:
    - Exibir mensagem: “Nenhum pedido foi feito ainda.”;
3. Caso existam pedidos, devem ser exibidos:
    - Número do pedido;
    - Status;
    - Data;

    - Valor total;
4. Deve existir opção de visualização detalhada quando aplicável.

Regras de Negócio
- Apenas pedidos do usuário autenticado devem ser exibidos;
- Não deve haver exposição de pedidos de outros usuários.

US-007 — Endereços
URL: /minha-conta/edit-address/ (após login)

Objetivo
Permitir que o usuário visualize e edite seus endereços cadastrados.

Descrição
Como usuário autenticado
Eu quero gerenciar meus endereços
Para utilizá-los nas compras

Critérios de Aceite
1. A página deve exibir:
    - Billing Address;
    - Shipping Address;
2. Caso não exista endereço cadastrado:
    - Exibir mensagem informando ausência de configuração;
3. Deve existir botão “Edit” para cada tipo de endereço;
4. Após salvar, as alterações devem persistir.

Regras de Negócio
- O endereço deve estar associado exclusivamente ao usuário autenticado;
- As alterações devem ser persistidas no banco de dados.

US-008 — Detalhes da Conta
URL: /minha-conta/edit-account/ (após login)

Objetivo
Permitir que o usuário edite suas informações pessoais e credenciais.

Descrição
Como usuário autenticado
Eu quero atualizar minhas informações
Para manter meus dados corretos e atualizados

Critérios de Aceite
1. Devem ser exibidos os seguintes campos:
    - First Name;
    - Last Name;
    - Display Name;

    - Email;
2. Caso o usuário altere a senha:
    - Deve informar senha atual;
    - Nova senha;
    - Confirmação da nova senha;
3. O botão “Save Changes” deve persistir as alterações;
4. Deve ser exibida mensagem de sucesso após atualização.

Regras de Negócio

- O email deve possuir formato válido;
- A nova senha deve coincidir com a confirmação;
- A senha atual deve ser validada antes da alteração.

2.3 Casos de testes

A partir das histórias de usuário definidas, foram elaborados casos de teste estruturados com base em técnicas clássicas de Engenharia de Testes, priorizando cenários críticos e de maior risco.

Cada caso foi classificado quanto à técnica aplicada, tipo de teste e nível de automação.

2.3.1 Casos de Teste – US-0001: Adicionar Item ao Carrinho

CT-001 — Adicionar produto com quantidade válida (Caminho Feliz)

- Técnica: Partição de Equivalência
- Tipo: Funcional
- Automação: Sim
- Pré-condição: Produto disponível em estoque

Passos:

1. Acessar página do produto;
2. Selecionar quantidade igual a 2;
3. Clicar em “Adicionar ao carrinho”.

Resultado Esperado:

- Produto adicionado ao carrinho;
- Quantidade exibida corretamente;
- Valor total atualizado conforme regra de cálculo.

CT-002 — Validar limite máximo de 10 itens (Valor Limite)

- Técnica: Análise de Valor Limite
- Automação: Não

Passos:

1. Selecionar quantidade igual a 10;
2. Adicionar ao carrinho.

Resultado Esperado:

- Produto adicionado com sucesso;
- Sistema aceita o valor máximo permitido.

CT-003 — Não permitir adicionar 11 itens (Valor Limite Negativo)

- Técnica: Análise de Valor Limite
- Automação: Sim

Passos:

1. Selecionar quantidade igual a 11;
2. Tentar adicionar ao carrinho.

Resultado Esperado:

- Sistema deve impedir a ação;
- Exibir mensagem de erro informando limite máximo.

Observação: Durante a execução automatizada foi identificado que essa regra não estava implementada na aplicação base.

CT-004 — Aplicar desconto de 10% (Tabela de Decisão)

- Técnica: Tabela de Decisão
- Automação: Não
- Pré-condição: Total entre R$200 e R$600

Passos:

1. Adicionar produtos até totalizar aproximadamente R$300.

Resultado Esperado:

- Aplicação automática de 10% de desconto;
- Valor final calculado corretamente.

CT-005 — Aplicar desconto de 15% (Tabela de Decisão)

- Técnica: Tabela de Decisão
- Automação: Não

Passos:

1. Adicionar produtos até totalizar aproximadamente R$700.

Resultado Esperado:

- Aplicação automática de 15% de desconto.

CT-006 — Não permitir ultrapassar R$990 (Regra de Negócio)

- Técnica: Partição de Equivalência + Regra de Negócio
- Automação: Sim

Passos:

1. Adicionar produtos até atingir R$990;
2. Tentar adicionar novo item.

Resultado Esperado:

- Sistema deve bloquear nova adição;
- Exibir mensagem informando valor máximo permitido.

Observação: Esta regra também não estava implementada no sistema base, configurando evidência de inconsistência funcional.

2.3.2 Casos de Teste – US-0002: Login na Plataforma

CT-007 — Login com credenciais válidas (Caminho Feliz)

- Técnica: Partição de Equivalência
- Automação: Sim

- Pré-condição: Usuário ativo cadastrado

Passos:

1. Acessar página de login;
2. Informar usuário válido;
3. Informar senha válida;
4. Clicar em “Entrar”.

Resultado Esperado:

- Usuário autenticado com sucesso;
- Redirecionamento para página “Minha Conta”.

CT-008 — Exibir erro para senha inválida

- Técnica: Partição de Equivalência
- Automação: Sim

Passos:

1. Informar usuário válido;
2. Informar senha inválida;
3. Clicar em “Entrar”.

Resultado Esperado:

- Exibição de mensagem de erro;
- Autenticação não realizada.

CT-009 — Não permitir login para usuário inativo

- Técnica: Partição de Equivalência
- Automação: Não

Passos:

1. Informar credenciais de usuário inativo;
2. Tentar autenticar.

Resultado Esperado:

- Sistema impede login;
- Exibição de mensagem de usuário inativo.

CT-010 — Bloquear usuário após 3 tentativas inválidas

- Técnica: Análise de Valor Limite
- Automação: Sim

Passos:

1. Informar senha incorreta três vezes consecutivas;
2. Realizar nova tentativa.

Resultado Esperado:

- Bloqueio temporário do usuário;
- Exibição de mensagem informando bloqueio por 15 minutos.

Observação: Regra não implementada na aplicação base.

CT-011 — Permitir login após período de bloqueio

- Técnica: Regra Temporal
- Automação: Não

Passos:

1. Aguardar período de bloqueio;

2. Informar credenciais válidas.

Resultado Esperado:

- Sistema permite autenticação normalmente.

2.3.3 Casos de Teste – US-0003: API de Cupons

CT-012 — Listar todos os cupons com autenticação válida

- Endpoint: GET /wc/v3/coupons
- Técnica: Partição de Equivalência
- Automação: Sim
- Pré-condição: Autenticação Basic válida

Resultado Esperado:

- Status Code 200;
- Retorno em formato array;
- Estrutura JSON válida.

CT-013 — Não permitir acesso com autenticação inválida

- Endpoint: GET /wc/v3/coupons
- Técnica: Partição de Equivalência
- Automação: Não

Resultado Esperado:

- Status Code 401;
- Mensagem de não autorizado.

CT-014 — Buscar cupom por ID válido

- Endpoint: GET /wc/v3/coupons/{id}
- Técnica: Partição de Equivalência
- Automação: Não

Resultado Esperado:

- Status Code 200;
- Retorno do objeto correspondente;
- ID igual ao solicitado.

CT-015 — Retornar erro para ID inexistente

- Endpoint: GET /wc/v3/coupons/{id}
- Técnica: Partição de Equivalência
- Automação: Não

Resultado Esperado:

- Status Code 404.

CT-016 — Cadastrar cupom com dados obrigatórios válidos

- Endpoint: POST /wc/v3/coupons
- Técnica: Partição de Equivalência
- Automação: Sim

Resultado Esperado:

- Status Code 201;
- Cupom criado com sucesso;
- Retorno contendo os campos enviados.

CT-017 — Não permitir cadastro com código duplicado

- Técnica: Tabela de Decisão
- Automação: Sim

Resultado Esperado:

- Retorno de erro;
- Não criação de cupom duplicado.

CT-018 — Não permitir cadastro sem campo obrigatório

- Técnica: Partição de Equivalência
- Automação: Não

Resultado Esperado:

- Status Code 400 (ou erro de validação);
- Mensagem indicando campo obrigatório ausente.

3. Testes automatizados

Os testes automatizados desenvolvidos neste projeto encontram-se disponíveis no repositório oficial:
Link: https://github.com/yassu974/TCC-EBAC-QE

A automação foi estruturada seguindo boas práticas de organização de código, separação de responsabilidades e padrão Page Object, garantindo manutenibilidade e escalabilidade.

3.1 Automação de UI

Contexto

A camada de interface web da aplicação EBAC-SHOP foi automatizada com o objetivo de validar:

- Fluxos críticos do usuário;
- Regras de negócio;
- Comportamentos esperados em cenários positivos e negativos;
- Regressão automatizada via pipeline.

A aplicação testada é baseada em **WordPress com WooCommerce**, executando em ambiente Docker local.

3.2 Justificativa da Escolha do Framework

Ferramenta escolhida: Cypress
Linguagem escolhida: JavaScript

Para definição da ferramenta de automação, foram analisadas três soluções amplamente utilizadas no mercado:

- Selenium WebDriver
- Playwright
- Cypress

A escolha foi realizada com base em critérios técnicos, facilidade de integração, curva de aprendizado e adequação ao escopo do projeto.

Selenium WebDriver
O Selenium é uma das ferramentas mais tradicionais do mercado, amplamente utilizada em ambientes corporativos.

Características:
- Arquitetura baseada em WebDriver;
- Suporte a múltiplas linguagens (Java, Python, C#, JavaScript);
- Execução multi-browser.

Vantagens:
- Alta maturidade e estabilidade;
- Ampla comunidade;
- Compatibilidade com diversos navegadores.

Desvantagens:
- Configuração mais complexa;
- Execução potencialmente mais lenta;
- Necessidade de gerenciamento manual de drivers;
- Debugging menos intuitivo.

Playwright
Ferramenta moderna desenvolvida pela Microsoft, com foco em performance e execução paralela.

Características:
- Suporte multi-browser real (Chromium, Firefox, WebKit);
- Execução paralela nativa;
- Suporte a múltiplas linguagens.

Vantagens:
- Alta performance;
- Excelente controle de contexto e abas;
- Arquitetura moderna.

Desvantagens:
- Curva de aprendizado maior;
- Comunidade menor comparada ao Selenium;
- Menor adoção em ambientes educacionais.

Cypress (Ferramenta Escolhida)
O Cypress é um framework moderno voltado especificamente para testes End-to-End em aplicações web.

Características:

- Executa diretamente no navegador;
- Arquitetura baseada em JavaScript;
- Controle automático de espera (auto-wait);
- Logs detalhados e visualização interativa.

Vantagens:

- Setup simples e rápido;
- Execução determinística;
- Excelente experiência de debugging;
- Integração simples com Docker;
- Sintaxe clara e legível;
- Forte aderência a aplicações web modernas.

Desvantagens:

- Restrito ao ecossistema JavaScript;
- Limitações históricas com múltiplas abas (mitigadas em versões recentes).

Justificativa da Escolha

O Cypress foi selecionado por apresentar o melhor equilíbrio entre:

- Facilidade de configuração;
- Experiência de desenvolvimento;
- Clareza na escrita de testes;
- Integração com pipelines de CI;
- Adequação ao escopo do projeto;
- Compatibilidade com aplicações WordPress/WooCommerce.

Como o foco principal era validar fluxos funcionais de uma aplicação web, o Cypress demonstrou-se plenamente adequado, permitindo rápida implementação, manutenção simples e excelente rastreabilidade de falhas.

3.3 Justificativa da Linguagem

Linguagem escolhida: JavaScript

A escolha da linguagem foi influenciada pelos seguintes fatores:

- Compatibilidade nativa com o Cypress;
- Linguagem base da aplicação web;
- Sintaxe moderna (ES6);
- Código mais conciso e legível;
- Facilidade de manutenção.

Comparativo entre Linguagens

Linguagem	Ferramenta Principal	Vantagens	Desvantagens
Java	Selenium	Forte no mercado corporativo	Sintaxe mais verbosa
Python	Selenium / Playwright	Sintaxe simples	Menor uso em frontend
JavaScript	Cypress / Playwright	Nativa para aplicações web	Restrita ao ecossistema JS

A utilização do JavaScript permitiu manter coerência tecnológica entre aplicação e automação, reduzindo complexidade e aumentando a eficiência do desenvolvimento.

3.4 Conclusão da Automação de UI

A adoção do Cypress com JavaScript mostrou-se adequada ao contexto do projeto, proporcionando:

- Escrita clara e organizada de testes;
- Estrutura baseada em Page Object Pattern;
- Uso de Custom Commands;
- Geração de relatórios automatizados;
- Integração com pipeline de CI;
- Execução determinística e reprodutível.

Essa abordagem garantiu robustez, manutenibilidade e alinhamento às boas práticas modernas de Engenharia de Qualidade.

3.5 Análise das Automações UI

A automação da camada de interface teve como objetivo validar não apenas o funcionamento esperado da aplicação, mas também verificar a aderência às regras de negócio especificadas nos critérios de aceitação.

Além da validação funcional, esta etapa permitiu identificar inconsistências e lacunas na implementação das regras definidas.

3.5.1 US-0001 – Adicionar item ao carrinho

CT-001 — Caminho Feliz

Objetivo

Validar que um produto variável pode ser adicionado corretamente ao carrinho com seleção adequada de atributos e quantidade válida.

Fluxo Automatizado

1. Acessar página do produto;
2. Selecionar variações (tamanho e cor);
3. Validar que o botão de compra está habilitado;
4. Adicionar produto ao carrinho;
5. Validar mensagem de sucesso;
6. Limpar carrinho após execução para garantir isolamento.

Resultado

- Produto adicionado corretamente;
- Mensagem de confirmação exibida;
- Quantidade e valor atualizados corretamente.

Conclusão

O fluxo positivo apresentou comportamento conforme esperado, validando a funcionalidade básica de adição ao carrinho.

CT-003 — Não permitir adicionar mais de 10 itens

Regra Esperada

O sistema não deve permitir a inserção de mais de 10 unidades do mesmo produto.

Fluxo Automatizado

1. Selecionar variações do produto;
2. Inserir quantidade igual a 11;
3. Adicionar ao carrinho;
4. Validar quantidade exibida no carrinho.

Resultado

- Sistema permitiu adicionar 11 unidades;
- Nenhuma mensagem de erro foi exibida.

Conclusão

Foi evidenciada ausência de validação da regra de limite máximo por item.
A regra de negócio especificada não está implementada na aplicação base.

CT-006 — Não permitir ultrapassar R$ 990,00

Regra Esperada

O valor total do carrinho não pode ultrapassar R$ 990,00.

Fluxo Automatizado

1. Selecionar variações do produto;
2. Inserir quantidade suficiente para ultrapassar o limite;
3. Validar valor total exibido no carrinho.

Resultado

- Sistema permitiu total acima de R$ 990,00;
- Não houve bloqueio ou mensagem de erro.

Conclusão

Foi identificada falha no controle de limite financeiro do carrinho.
A regra de negócio definida não está implementada na aplicação.

Análise Técnica Geral – US-0001
A automação da US-0001 demonstrou:

- Cobertura de fluxo positivo;
- Cobertura de cenários de valor limite;
- Identificação de falhas em regras de negócio;
- Evidência objetiva de ausência de validações;
- Isolamento adequado entre testes (limpeza de estado);
- Aplicação de boas práticas de automação.

Além de validar funcionalidades, os testes automatizados agregaram valor ao processo ao evidenciar inconsistências funcionais reais.

3.5.2 US-0002 – Login na Plataforma

CT-007 — Login com credenciais válidas
Objetivo
Validar que um usuário ativo consegue autenticar corretamente na aplicação.

Fluxo Automatizado

1. Acessar página de login;
2. Informar e-mail válido;
3. Informar senha válida;
4. Submeter formulário;
5. Validar mensagem de boas-vindas;
6. Realizar logout para manter isolamento.

Resultado

- Login realizado com sucesso;
- Sessão criada corretamente;
- Logout executado corretamente.

Conclusão
O fluxo positivo de autenticação está funcionando conforme esperado.

CT-008 — Exibir erro para senha inválida
Objetivo
Validar tratamento adequado para credenciais incorretas.

Fluxo Automatizado

1. Informar e-mail válido;
2. Informar senha inválida;
3. Submeter formulário;
4. Validar mensagem de erro;
5. Confirmar ausência de autenticação.

Resultado

- Mensagem de erro exibida corretamente;
- Usuário permaneceu na tela de login;

- Sessão não foi criada.

Conclusão
O sistema trata adequadamente tentativas de login com credenciais inválidas.

CT-010 — Bloqueio após três tentativas inválidas
Objetivo
Validar regra de segurança referente a bloqueio após múltiplas tentativas incorretas.

Fluxo Automatizado
1. Realizar três tentativas com senha inválida;
2. Realizar uma quarta tentativa;
3. Validar comportamento do sistema.

Resultado
- Sistema não bloqueou o usuário;
- Permitiu tentativas ilimitadas;
- Nenhuma mensagem de bloqueio foi exibida.

Conclusão
Foi identificada ausência de controle de bloqueio por múltiplas tentativas.
A regra de segurança especificada não está implementada na aplicação base.

Análise Técnica Geral – US-0002
A automação da US-0002 demonstrou:
- Cobertura de fluxo positivo;
- Cobertura de fluxo negativo;
- Validação de regra de segurança;
- Uso de massa de dados dinâmica;
- Isolamento adequado entre execuções;
- Estrutura organizada via Page Object e Custom Commands.

Além disso, a automação permitiu identificar falha relevante de segurança relacionada à ausência de bloqueio após múltiplas tentativas inválidas, evidenciando contribuição efetiva do processo de qualidade.

3.5.3 Considerações Gerais sobre as Automações UI

As automações implementadas não se limitaram à validação do caminho feliz. Foram construídas para:
- Testar regras de negócio;
- Validar comportamentos negativos;
- Evidenciar lacunas funcionais;
- Garantir reprodutibilidade;
- Manter independência entre testes;
- Simular cenário real de uso.

A abordagem adotada demonstra aplicação prática dos princípios de Engenharia de Qualidade, indo além da simples execução de testes e agregando valor analítico ao processo.

3.6 Automação de API

A camada de API foi automatizada com o objetivo de validar regras de negócio, autenticação e contrato de dados da funcionalidade de gerenciamento de cupons.

Os testes foram implementados utilizando Supertest em conjunto com Jest, executando requisições HTTP diretamente contra a API REST do WooCommerce.
A automação contemplou validação de cenários positivos e negativos, além de verificação de estrutura mínima do contrato retornado.

3.6.1 US-0003 – API de Cupons

CT-012 — Listar todos os cupons (GET)

Objetivo

Validar que a API retorna corretamente a lista de cupons cadastrados quando a autenticação é válida.

Validações Realizadas

- Status Code 200;
- Retorno no formato Array;
- Validação de contrato mínimo contendo:
  - id
  - code
  - amount
  - discount_type
- Verificação de autenticação via Basic Auth.

Resultado

- API retornou status 200 conforme esperado;
- Estrutura mínima do contrato validada com sucesso;
- Autenticação Basic Auth funcionando corretamente.

Conclusão

A API apresentou comportamento adequado para requisição de listagem autenticada, validando contrato mínimo e controle de acesso.

CT-016 — Criar cupom válido (POST)

Objetivo

Validar a criação de um novo cupom contendo os campos obrigatórios exigidos pela API.

Estratégia

- Geração dinâmica do campo code para evitar conflitos entre execuções;
- Envio de payload válido conforme documentação do WooCommerce;
- Validação de resposta com status 201 (Created).

Validações Realizadas

- Status Code 201;
- Presença do campo id no retorno;
- Validação dos campos:
    - code
    - amount
    - discount_type
- Comparação entre payload enviado e resposta recebida.

Observação Técnica

Foi identificado que a API realiza normalização automática do campo code, convertendo-o para letras minúsculas.

Essa característica exigiu ajuste na validação automatizada, evidenciando comportamento implícito da API não explicitado inicialmente.

Resultado

- Cupom criado com sucesso;
- Estrutura do contrato validada;
- Comportamento real da API devidamente documentado.

Conclusão

A criação de cupons com dados válidos está implementada corretamente e segue o contrato esperado, com normalização automática aplicada ao campo code.

CT-017 — Não permitir criação de cupom duplicado

Objetivo

Validar a regra de negócio que impede a criação de cupons com código repetido.

Estratégia

1. Criar um cupom válido com código específico;
2. Tentar criar novo cupom com o mesmo código;
3. Validar resposta de erro.

Validações Realizadas

- Status Code 400;
- Presença dos campos:
    - code
    - message
    - data.status
- Confirmação de que o cupom não foi criado novamente**.**

Resultado

- API bloqueou corretamente a criação duplicada;
- Contrato de erro validado;
- Regra de negócio implementada corretamente.

Conclusão
A API possui controle adequado de unicidade de código de cupom, garantindo integridade dos dados e prevenção de duplicidade.

Análise Técnica Geral
A automação da API demonstrou:

- Uso adequado do framework Supertest para requisições HTTP;
- Validação de autenticação via Basic Auth;
- Verificação de contratos mínimos de sucesso e erro;
- Cobertura de fluxo positivo e negativo;
- Geração dinâmica de dados para evitar dependência de estado prévio;
- Identificação de comportamento implícito da API (normalização de código).

Além disso, a estrutura do projeto foi organizada separando testes de UI e testes de API em camadas distintas, reforçando boas práticas de arquitetura e permitindo escalabilidade futura.

A abordagem adotada evidencia maturidade na validação de contratos e regras de negócio, indo além da simples verificação de status code e garantindo confiabilidade da camada de integração.

3.7 Automação Mobile

A camada mobile foi automatizada com o objetivo de validar a navegação e o comportamento funcional da aplicação Android da EBAC-SHOP.

A automação foi estruturada seguindo boas práticas modernas de testes mobile, incluindo uso de Page Object Model, sincronização adequada e geração de relatórios detalhados.

Ferramentas Utilizadas

- Node.js
- WebdriverIO
- Appium
- Android Emulator
- Mocha
- Allure Report
- Visual Studio Code
- Git

3.7.1 Objetivo

Implementar automação mobile utilizando:

- WebdriverIO como framework de execução;
- Appium como servidor de automação;
- Page Object Model para organização estrutural;
- Mocha como framework de testes;
- Allure para geração de relatórios detalhados.

O foco principal foi validar o fluxo de navegação e carregamento do catálogo de produtos no aplicativo Android.

3.7.2 Análise das Automações

CT-001 — Validar navegação para o catálogo e exibição de preço

Objetivo

Validar que, ao acessar a aba *Browse*, os produtos são carregados corretamente e exibem preço no formato monetário brasileiro (R$).

Fluxo Automatizado

1. Abrir o aplicativo;
2. Garantir estado inicial na aba *Home*;
3. Clicar na aba *Browse*;
4. Executar navegação adicional Home → Browse (estratégia de estabilização);
5. Aguardar carregamento dos produtos;
6. Capturar o primeiro preço exibido;
7. Validar que o texto contém o prefixo “R$”.

Implementação Técnica

A automação foi estruturada com base nas seguintes boas práticas:

- Padrão Page Object Model;
- Uso de waits explícitos para sincronização;
- Tratamento de timeouts;
- Separação entre camada de teste e camada de elementos;
- Geração de relatórios via Allure;
- Captura automática de screenshot em caso de falha.

Essa abordagem garantiu organização, legibilidade e facilidade de manutenção.

Resultado Obtido

Durante as execuções automatizadas foram observados dois comportamentos distintos:

Execução Manual

- Produtos carregaram imediatamente;
- Preços exibidos corretamente;
- Nenhum comportamento inesperado identificado.

Execução Automatizada

- Em execuções intermitentes, os produtos não foram carregados;
- O teste, por vezes, falha por timeout na validação de exibição;
- O erro ocorre na etapa de espera pelo elemento de preço.

Evidências

Relatório gerado via Allure demonstrou:

- 1 caso de teste executado;
- Registro de falha;

- Stacktrace detalhado;
- Evidência de timeout;
- Screenshot anexado no momento da falha.

Essas evidências foram fundamentais para análise técnica da ocorrência.

Análise Técnica
A estrutura do teste foi avaliada e encontra-se corretamente implementada:
- Seletores válidos;
- Estratégia de sincronização aplicada;
- Uso de waits explícitos;
- Implementação conforme padrão arquitetural;
- Validação objetiva do comportamento esperado.

A falha observada indica possível instabilidade no carregamento da tela de catálogo quando acionada via automação.

O comportamento sugere:
- Problema no lifecycle da aplicação;
- Condição de sincronização interna do aplicativo;
- Carregamento assíncrono não concluído adequadamente.

Não foram identificados indícios de erro estrutural na automação.

Conclusão
O teste automatizado está corretamente estruturado e implementado seguindo boas práticas de automação mobile.

A falha registrada indica possível instabilidade intermitente no carregamento da aba *Browse* quando executada via automação, caracterizando um cenário real de investigação de qualidade.

Essa etapa demonstrou capacidade de:
- Implementar automação mobile completa;
- Trabalhar com sincronização e estados de aplicação;
- Analisar falhas com base em evidências;
- Utilizar relatórios profissionais;
- Diferenciar problema de aplicação versus problema de teste.

Considerações Finais sobre Mobile
A automação mobile permitiu ampliar o escopo do projeto para além da camada web, demonstrando aplicação prática de Quality Engineering em ambiente Android.

Mesmo com instabilidade identificada, a estrutura de automação mostrou-se robusta, organizada e aderente às boas práticas modernas de testes mobile.

Essa etapa reforça a maturidade técnica do projeto ao simular cenários reais de investigação de falhas e análise de comportamento sob automação.

4. Integração contínua

A etapa de Integração Contínua (CI) representa um dos pilares fundamentais da Engenharia de Qualidade moderna, permitindo validação automatizada e contínua do sistema a cada alteração realizada no código-fonte.

A implementação dessa etapa elevou o projeto para um nível mais próximo de práticas corporativas reais.

4.1 Objetivo

A etapa de Integração Contínua teve como principal objetivo automatizar a execução dos testes desenvolvidos ao longo do projeto, garantindo que, a cada alteração no código (push ou pull request), o conjunto de testes fosse executado automaticamente.

Essa abordagem permite:

- Validar a estabilidade da aplicação de forma contínua;
- Detectar regressões precocemente;
- Reduzir risco de falhas em produção;
- Garantir rastreabilidade das execuções.

A CI assegura que nenhuma modificação seja incorporada ao projeto sem passar por validação automatizada.

4.2 Estratégia Adotada

Foi utilizada a ferramenta GitHub Actions como plataforma de automação da pipeline, por sua integração nativa ao repositório e ampla adoção no mercado.

A pipeline foi configurada para executar automaticamente nos seguintes eventos:

- Push na branch main;
- Pull Request direcionado à branch main.

A execução ocorre em ambiente Linux (ubuntu-latest), garantindo:

- Ambiente limpo e isolado;
- Reprodutibilidade;
- Independência de configurações locais.

4.3 Estrutura da Pipeline

O workflow implementado contempla as seguintes etapas:

1. Checkout do repositório;
2. Configuração do Node.js (versão 18);
3. Criação de rede Docker dedicada;
4. Subida do banco de dados via container;
5. Subida da aplicação WordPress + WooCommerce via container;
6. Aguardar disponibilidade da aplicação;
7. Instalação das dependências do projeto de API;

8. Execução dos testes de API (Jest + Supertest);
9. Instalação das dependências do projeto de UI;
10. Execução dos testes E2E (Cypress);
11. Upload dos relatórios como artifacts.

A decisão de subir a aplicação via Docker dentro da própria pipeline garante que os testes sejam executados contra um ambiente controlado, isolado e reprodutível, eliminando dependências externas e reduzindo variabilidade ambiental.

4.4 Execução dos Testes na CI

Durante a execução da pipeline, são realizados:

Testes de API

- Validação de autenticação;
- Listagem de cupons;
- Criação de cupons;
- Regra de duplicidade.

Testes E2E (UI)

- Login com credenciais válidas;
- Tratamento de erro para senha inválida;
- Validação de regra de bloqueio (não implementada);
- Adição de produto ao carrinho;
- Validação de regras de negócio não implementadas.

Destaca-se que os testes de login foram refatorados para criar usuários dinamicamente antes da execução, garantindo que a pipeline não dependa de estado prévio do banco de dados.

Essa abordagem assegura maior confiabilidade e independência entre execuções.

4.5 Resultados Obtidos

Após ajustes relacionados a caminhos relativos do Cypress na pipeline, a execução passou a ocorrer com sucesso.

Resultados alcançados:

- Execução automatizada e estável dos testes de API;
- Execução automatizada e estável dos testes de UI;
- Subida automática do ambiente via Docker;
- Geração de relatórios como artifacts;
- Execução consistente em ambiente limpo e isolado.

A pipeline demonstrou comportamento estável e reprodutível, validando a robustez da estrutura implementada.

4.6 Análise Técnica

A implementação da Integração Contínua proporcionou benefícios relevantes:

- Redução significativa de risco de regressão;
- Validação automática de novas implementações;
- Reprodutibilidade completa do ambiente;
- Eliminação de dependência de execução manual;
- Simulação de cenário corporativo real;
- Estrutura preparada para escalabilidade futura.

Além disso, a identificação e correção de inconsistências relacionadas a caminhos relativos do Cypress durante a configuração da pipeline evidenciam a importância de considerar diferenças entre ambiente local e ambiente de CI.

Essa etapa consolidou o projeto como uma solução estruturada, automatizada e alinhada às práticas modernas de Engenharia de Qualidade.

5. Testes de performance

A etapa de testes de performance foi implementada com o objetivo de avaliar o comportamento da aplicação sob carga controlada, analisando sua estabilidade, capacidade de resposta e tolerância a concorrência.

Os testes foram conduzidos utilizando a ferramenta k6, amplamente adotada para simulação de carga e validação de métricas de desempenho.

5.1 Objetivo

O objetivo desta etapa foi avaliar o comportamento da aplicação sob carga simultânea, identificando:

- Possíveis gargalos de desempenho;
- Degradações progressivas;
- Falhas sob concorrência;
- Impacto no tempo de resposta.

Foram definidos dois cenários críticos:

- Autenticação de usuário (Login);
- Requisição HTTP representando acesso à funcionalidade principal da aplicação.

A meta estabelecida foi validar a estabilidade do sistema com:

- 20 usuários virtuais simultâneos;
- 2 minutos de execução;
- Ramp-up gradual de 20 segundos.

Esses parâmetros seguem a especificação proposta para o projeto.

5.2 Estratégia

A estratégia adotada baseou-se em testes de carga controlada com definição de thresholds, isto é, limites aceitáveis de desempenho.

Os critérios definidos foram:

- Percentil 95 (p95) inferior a 1500ms para o cenário de Login;
- Percentil 95 (p95) inferior a 1200ms para requisição HTTP geral;
- Taxa de falhas HTTP inferior a 5%.

A utilização do percentil 95 permite avaliar o comportamento do sistema considerando a maior parte das requisições, ignorando picos isolados que não representam padrão sistêmico.

Foram utilizados dados previamente cadastrados, simulando múltiplos usuários reais executando ações simultaneamente.

5.3 Configuração do Teste

Parâmetros de Execução

- Usuários Virtuais (VUs): 20
- Tempo total de execução: 2 minutos
- Ramp-up: 20 segundos
- Massa de dados: 5 usuários distintos
- Thresholds Configurados

Login

- p(95) < 1500ms
- http_req_failed < 5%

Requisição HTTP

- p(95) < 1200ms
- http_req_failed < 5%

5.4 Resultados – Teste de Login

Métricas Obtidas

- Média de resposta: 544ms
- p90: 850ms
- p95: 1.16s
- Taxa de falhas: 0%
- Total de requisições: 1827
- Throughput: 11.32 requisições por segundo

Análise

O sistema apresentou estabilidade durante todo o período de execução.
O percentil 95 ficou em 1.16 segundos, mantendo-se abaixo do limite definido de 1500ms.

Não houve falhas HTTP registradas, indicando:

- Controle adequado de autenticação;
- Ausência de erros de processamento;
- Estabilidade sob concorrência moderada.

Mesmo com 20 usuários simultâneos, o sistema manteve tempos consistentes e comportamento previsível.

5.5 Resultados – Teste de Requisição HTTP

Métricas Obtidas

- Média de resposta: 500ms
- p90: 796ms
- p95: 948ms
- Taxa de falhas: 0%
- Total de requisições: 2836
- Throughput: 17.56 requisições por segundo

Análise

Neste cenário, o sistema apresentou desempenho superior ao teste de login, com p95 inferior a 1 segundo.

O comportamento indica:

- Capacidade adequada de processamento;
- Baixo impacto de concorrência;
- Ausência de degradação progressiva;
- Estabilidade durante toda a execução.

Não foram observados erros HTTP ou interrupções.

5.6 Análise Comparativa

Métrica	Login	Requisição HTTP
Média	544ms	500ms
p95	1.16s	948ms
Falhas	0%	0%
Throughput	11.32 req/s	17.56 req/s

Ambos os cenários atenderam integralmente aos thresholds estabelecidos, apresentando margem confortável de segurança.

A diferença entre os cenários é justificada pelo processamento adicional envolvido na autenticação (validação de credenciais e criação de sessão), enquanto a requisição HTTP geral exige menor processamento interno.

5.7 Conclusão

Com base nos testes realizados, conclui-se que a aplicação:

- Suporta 20 usuários simultâneos sem falhas;
- Mantém tempo de resposta inferior a 1,5 segundos no percentil 95;

- Não apresenta erros HTTP sob carga moderada;
- Demonstra estabilidade durante execução contínua de 2 minutos;
- Atende integralmente aos requisitos definidos para esta etapa.

Os resultados indicam que o sistema está apto a suportar cenários de uso simultâneo moderado, mantendo desempenho dentro de padrões aceitáveis.

A implementação de testes de performance complementa a estratégia de qualidade do projeto, garantindo não apenas correção funcional, mas também confiabilidade operacional.

6 Conclusão Final

O desenvolvimento deste projeto permitiu aplicar, de maneira prática e integrada, os conceitos fundamentais da Engenharia de Qualidade de Software em um ambiente controlado e próximo à realidade do mercado.

A estratégia de testes elaborada contemplou diferentes camadas da aplicação — UI, API e Mobile — além de incluir testes não funcionais de performance e implementação de Integração Contínua. Essa abordagem multicamadas garantiu maior cobertura, redução de riscos e validação abrangente do sistema.

Durante a execução das automações, foram identificadas inconsistências em regras de negócio especificadas, como ausência de validação de limite de quantidade no carrinho, controle de valor máximo e bloqueio de usuário após múltiplas tentativas de login. Essas evidências demonstram que o processo de qualidade foi capaz de agregar valor real ao produto, indo além da validação superficial de funcionalidades.

Os testes de performance confirmaram estabilidade sob carga moderada, mantendo tempos de resposta dentro dos thresholds definidos e sem ocorrência de falhas HTTP. A implementação de pipeline de Integração Contínua consolidou o projeto em um modelo automatizado e reprodutível, alinhado às práticas modernas de desenvolvimento.

Conclui-se que o projeto atingiu seus objetivos, demonstrando aplicação consistente de técnicas de teste, uso adequado de ferramentas modernas, organização arquitetural estruturada e visão estratégica de qualidade.

O trabalho evidencia que a atuação do Quality Engineer é fundamental para garantir não apenas o funcionamento correto do software, mas também sua confiabilidade, estabilidade e evolução sustentável ao longo do tempo.