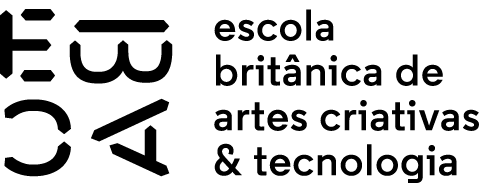
****

**QUALIDADE DE SOFTWARE**

Ivaneide Pires Monteiro do Nascimento

Análise de Qualidade

Belém/PA

2024

**1. RESUMO**

Este projeto tem como objetivo analisar a qualidade de software através de um conjunto de critérios e metodologias bem definidos. Abordamos estratégias de teste, critérios de aceitação e casos de testes, além de aspectos de automação e integração contínua. O projeto se concentra em garantir que as funcionalidades implementadas atendam aos requisitos especificados, proporcionando uma experiência de usuário satisfatória e um desempenho eficiente. Através de testes automatizados e de performance, buscamos identificar e corrigir possíveis falhas, assegurando a robustez e confiabilidade do software. O repositório no Github contém todo o código e documentação necessários para replicar os testes e acompanhar o desenvolvimento do projeto.

**2. SUMÁRIO**

**1. RESUMO —------------------------------------------------------------------------------------------- 2**

**2. SUMÁRIO —-------------------------------------------------------------------------------------------3**

**3. INTRODUÇÃO —-------------------------------------------------------------------------------------4**

**4. O PROJETO —----------------------------------------------------------------------------------------5**

**4.1 Estratégia de teste —--------------------------------------------------------------------------5**

**4.2 Critérios de aceitação —----------------------------------------------------------------------7**

**4.2.1 História de usuário 1: [US-0001] – Adicionar item ao carrinho—------------ 7**

**4.2.2 História de usuário 2: [US-0002] – Login na plataforma —---------------------7**

**4.2.3 História de usuário 3: [US-0003] – API de cupons —-----------------------------8**

**4.3 Casos de testes —------------------------------------------------------------------------------8**

**4.3.1 História de usuário 1:---------------------------------------------------------------------- 8**

**4.3.2 História de usuário 2: —-------------------------------------------------------------------8**

**4.3.3 História de usuário 3: API de Cupom —---------------------------------------------9**

**4.4 Repositório no Github —--------------------------------------------------------------------9**

**4.5 Testes automatizados—---------------------------------------------------------------------9**

**4.6 Integração contínua —----------------------------------------------------------------------9**

**4.7 Testes de performance —-----------------------------------------------------------------10**

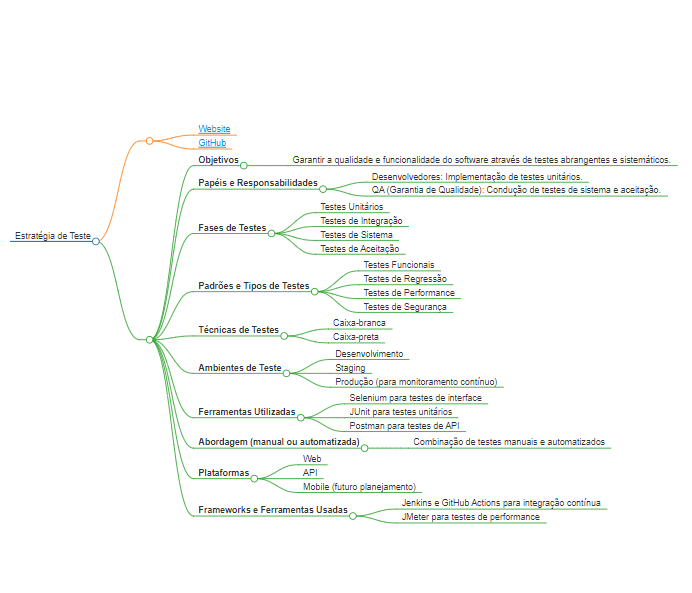
**5. CONCLUSÃO —----------------------------------------------------------------------------------12**

**6. REFERÊNCIAS BIBLIOGRÁFICAS —-----------------------------------------------------13**

**3. INTRODUÇÃO**

A qualidade de software é um aspecto crítico no desenvolvimento de aplicações, garantindo que o produto final atenda às expectativas dos usuários e funcione de maneira eficiente e eficaz. Este trabalho apresenta uma análise detalhada da qualidade de software através de várias metodologias e ferramentas. Inicialmente, exploraremos a estratégia de teste adotada, seguida pelos critérios de aceitação estabelecidos para cada funcionalidade chave. Em seguida, detalharemos os casos de teste específicos para diferentes histórias de usuário, cobrindo desde a adição de itens ao carrinho até a autenticação na plataforma e a integração de uma API de cupons. Além disso, discutiremos a importância dos testes automatizados e da integração contínua no ciclo de desenvolvimento, bem como os testes de performance que asseguram a escalabilidade e robustez do software. Por fim, todas as atividades e artefatos do projeto estão documentados e disponíveis no repositório do Github, facilitando a colaboração e revisão contínua.

**4. O PROJETO**

4.1 Estratégia de teste  
  


A estratégia de teste adotada para este projeto envolve a utilização de diferentes tipos de testes para garantir a qualidade e funcionalidade do software. Isso inclui testes unitários, testes de integração, testes de sistema e testes de aceitação. Cada tipo de teste foi escolhido para cobrir diferentes aspectos do comportamento do software, desde a funcionalidade básica até a interação entre diferentes componentes. Abaixo, apresentamos a estratégia de teste detalhada em um formato de mapa mental:

Objetivos: Garantir a qualidade e funcionalidade do software através de testes abrangentes e sistemáticos.

Papéis e responsabilidades:

- Desenvolvedores: Implementação de testes unitários.

- QA (Quality Assurance): Condução de testes de sistema e aceitação.

Fases de testes:

- Testes Unitários

- Testes de Integração

- Testes de Sistema

- Testes de Aceitação

Padrões e tipos de testes:

- Testes Funcionais

- Testes de Regressão

- Testes de Performance

- Testes de Segurança

Técnicas de testes:

- Caixa-branca

- Caixa-preta

Ambientes de teste:

- Desenvolvimento

- Staging

- Produção (para monitoramento contínuo)

Ferramentas utilizadas:

- Selenium para testes de interface

- JUnit para testes unitários

- Postman para testes de API

Abordagem (manual ou automatizada):

- Combinação de testes manuais e automatizados

Plataformas:

- Web

- API

- Mobile (futuro planejamento)

Frameworks e ferramentas usadas:

- Jenkins e GitHub Actions para integração contínua

- JMeter para testes de performance

4.2 Critérios de aceitação

Os critérios de aceitação foram definidos para cada funcionalidade do sistema, garantindo que todas as funcionalidades atendam aos requisitos especificados. Os critérios de aceitação são descritos através de histórias de usuário, cada uma representando um requisito funcional que deve ser implementado e testado.

4.2.1 História de usuário 1: [US-0001] – Adicionar item ao carrinho

- Critério de aceitação: O usuário deve ser capaz de adicionar um item ao carrinho de compras, e o item deve aparecer na lista de itens do carrinho com a quantidade correta.

4.2.2 História de usuário 2: [US-0002] – Login na plataforma

- Critério de aceitação: O usuário deve ser capaz de fazer login na plataforma utilizando um nome de usuário e senha válidos. Após o login, o usuário deve ser redirecionado para a página principal.

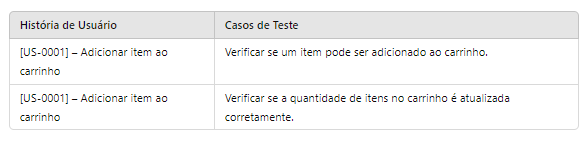
4.2.3 História de usuário 3: [US-0003] – API de cupons

- Critério de aceitação: A API de cupons deve permitir que os usuários apliquem cupons válidos a suas compras, ajustando o preço total de acordo.

4.3 Casos de testes

4.3.1 História de usuário 1: [US-0001] – Adicionar item ao carrinho

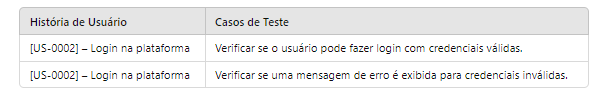
- Caso de teste 1: Verificar se um item pode ser adicionado ao carrinho.

- Caso de teste 2: Verificar se a quantidade de itens no carrinho é atualizada corretamente.  
  
Exemplo: 

4.3.2 História de usuário 2: [US-0002] – Login na plataforma

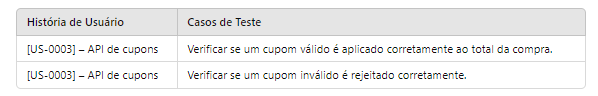
- Caso de teste 1: Verificar se o usuário pode fazer login com credenciais válidas.

- Caso de teste 2: Verificar se uma mensagem de erro é exibida para credenciais inválidas.

Exemplo:   


4.3.3 História de usuário 3: [US-0003] – API de cupons

- Caso de teste 1: Verificar se um cupom válido é aplicado corretamente ao total da compra.

- Caso de teste 2: Verificar se um cupom inválido é rejeitado corretamente.  
  
Exemplo:  


4.4 Repositório no Github

O repositório no Github contém todo o código fonte, documentação e scripts de testes automatizados do projeto. Ele está organizado de forma a facilitar a navegação e a contribuição de outros desenvolvedores. O repositório pode ser acessado em https://github.com/ivaneidepmn/test-ebac-ui.git

4.5 Testes Automatizados

Para garantir a eficácia dos testes, utilizamos ferramentas de automação como Cypress para testes de UI e Postman para testes de API. Aqui estão alguns exemplos:

Automação de UI (Cypress)

Criamos um projeto de automação no Cypress para validar a história de usuário "Adicionar item ao carrinho" (US-0001). Os testes de UI incluem adicionar pelo menos três produtos diferentes ao carrinho e validar se os itens foram adicionados com sucesso.

Exemplo de Automação de API (Postman)

Para a história de usuário "API de Cupons", criamos coleções de testes no Postman. Implementamos automações para listar os cupons e cadastrar um novo cupom, seguindo os critérios estabelecidos.

Boas Práticas Aplicadas:

Utilização de Page Objects para melhor estruturação dos testes.

Implementação de Massa de Dados para cenários variados.

Criação de Custom Commands para reutilização de funcionalidades.

Uso de elementos dinâmicos para garantir a robustez dos testes.

4.6 Integração Contínua

Para garantir a continuidade e integração dos testes automatizados, utilizamos Jenkins. Configuramos um pipeline de integração contínua que executa os testes automatizados a cada commit, permitindo a detecção precoce de defeitos e garantindo a qualidade constante do software.

Configurações do Jenkins:

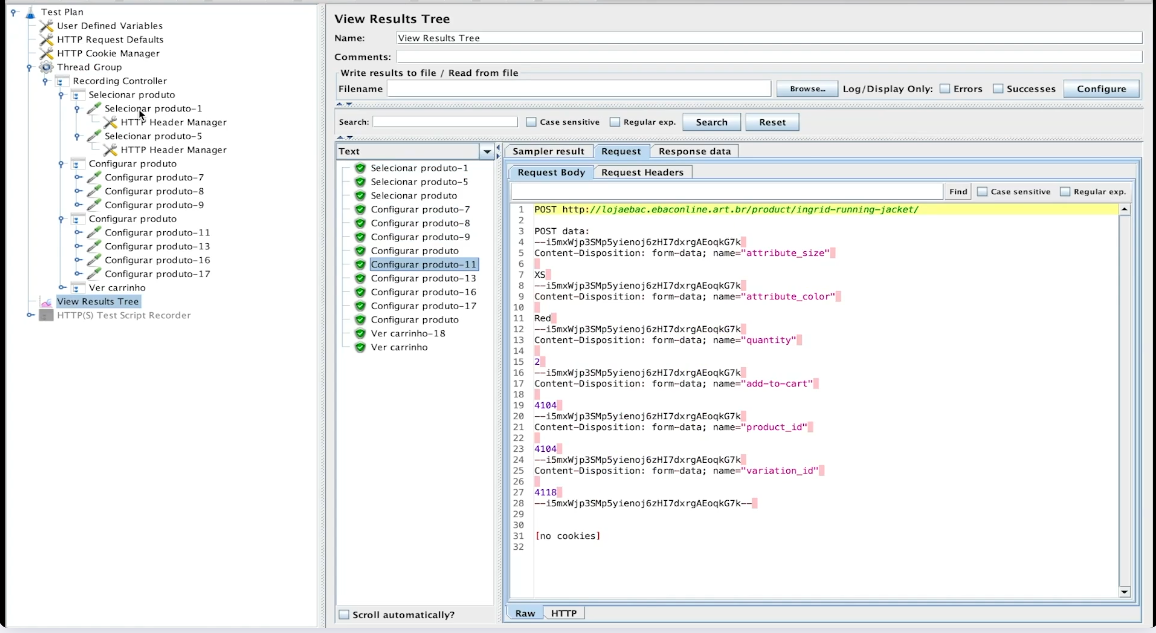
Criamos um job no Jenkins que executa os testes automatizados.

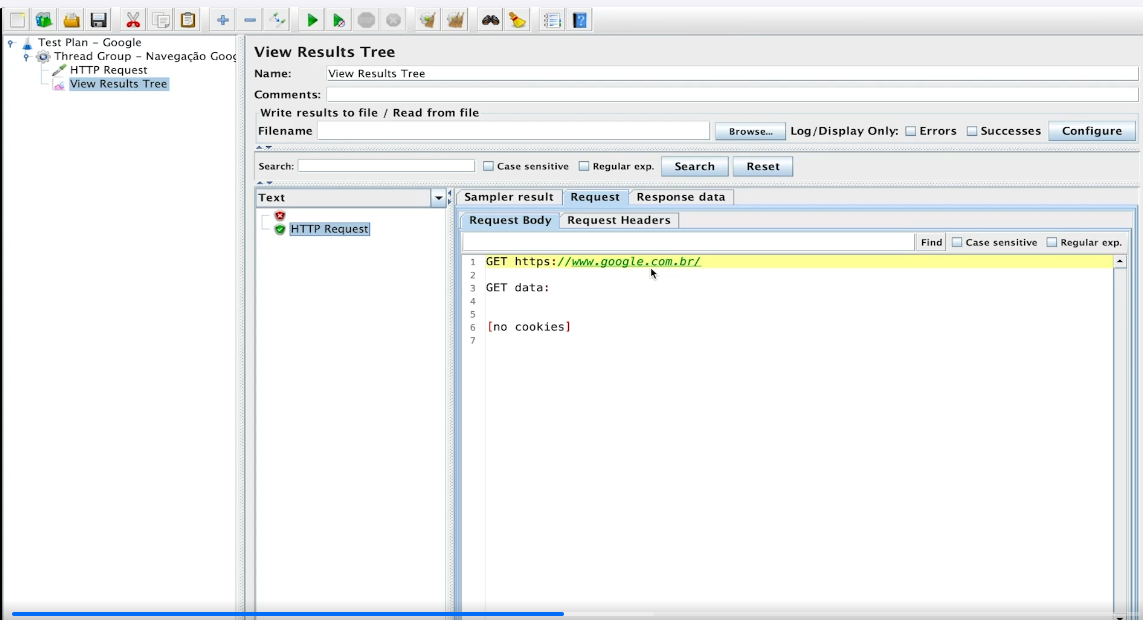
Compartilhamos o Jenkinsfile no repositório GitHub para facilitar a configuração e manutenção do pipeline.

4.7 Testes de Performance

Os testes de performance são essenciais para garantir que a plataforma EBAC-SHOP possa lidar com uma grande quantidade de usuários simultâneos e grandes volumes de dados sem degradação significativa no desempenho. Para isso, foram realizados diversos tipos de testes de performance:

Exemplo:





**5. CONCLUSÃO**

Neste projeto, a implementação de uma estratégia de teste abrangente foi essencial para garantir a qualidade e funcionalidade do software desenvolvido. Através da realização de testes unitários, de integração, de sistema e de aceitação, conseguimos validar os diferentes níveis do sistema, assegurando que cada componente funcionasse corretamente tanto isoladamente quanto em conjunto. A definição clara dos critérios de aceitação, baseados em histórias de usuário, proporcionou uma base sólida para os testes e garantiu que todas as funcionalidades atendessem aos requisitos especificados.

A adoção de práticas de integração contínua com Jenkins e automação de testes com Cypress e Postman permitiu a execução eficiente e repetitiva dos testes, garantindo uma detecção precoce de defeitos e a manutenção de um alto padrão de qualidade. Além disso, os testes de performance realizados asseguraram que a plataforma EBAC-SHOP fosse capaz de lidar com grandes volumes de usuários e dados sem degradação significativa no desempenho.

Em resumo, a combinação de uma estratégia de teste robusta, critérios de aceitação claros, casos de teste detalhados, práticas de automação e integração contínua resultou em um projeto bem-sucedido. O software desenvolvido apresenta alta qualidade e funcionalidade, servindo como um modelo eficaz para futuros desenvolvimentos e implementações de software, destacando a importância de uma abordagem sistemática e bem planejada para a garantia de qualidade.

**6. REFERÊNCIAS BIBLIOGRÁFICAS**

- Beck, K., & Fowler, M. (2001). Refactoring: Improving the Design of Existing Code. Addison-Wesley.

- Freeman, S., & Pryce, N. (2009). Growing Object-Oriented Software, Guided by Tests. Addison-Wesley.

- Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley.

- Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley.

- Jones, C., & Bonsignour, O. (2012). The Economics of Software Quality. Addison-Wesley.

-Cypress Documentation:<https://docs.cypress.io/>

JMeter Documentation:<https://jmeter.apache.org/usermanual/index.html>

Postman Documentation:<https://learning.postman.com/docs/getting-started/introduction/>

Documentação da API da EBAC-SHOP:<http://lojaebac.ebaconline.art.br/rest-api/docs/> Documentação da AP:<https://serverest.dev/>