O IMPACTO DOS TESTES DE DESEMPENHO NA MELHORIA DE SISTEMAS WEB: UM ESTUDO DE CASO.

Vinicius Ferreira Camilo [1] Ana Cláudia Garcia Barbosa [2]

Resumo: Os testes de desempenho têm um papel fundamental na garantia da qualidade e eficiência de sistemas web, permitindo identificar possíveis gargalos e áreas de melhoria. Este estudo de caso teve como objetivo analisar os resultados de testes de desempenho realizados em um software web, utilizando a ferramenta JMeter e as ferramentas Elasticsearch, Logstash e Kibana para análise dos resultados. A pesquisa buscou compreender os testes de desempenho de softwares, suas metodologias e ferramentas, além de identificar possíveis melhorias para otimizar o desempenho do sistema. Após a implementação das melhorias sugeridas, foi realizado um segundo teste de desempenho para avaliar a eficácia das alterações realizadas. Os resultados obtidos demonstraram melhorias significativas no desempenho do sistema, com reduções nos tempos de resposta e nos tempos máximos de execução. A análise comparativa entre o teste inicial e o teste após as melhorias evidenciou os impactos positivos das alterações implementadas, destacando a importância dos testes de desempenho na melhoria contínua de sistemas web. Em suma, este estudo reforça a importância dos testes de desempenho na identificação de problemas, na otimização de sistemas e na garantia de uma experiência satisfatória para os usuários de plataformas online.

Palavras-chave: Testes de desempenho; software web; otimização de sistemas; melhoria contínua.

[1]Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), Criciúma - Santa Catarina - Brasil. fcvini@gmail.com

[2]Orientadora, Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense (Unesc), Criciúma - Santa Catarina - Brasil. agb@unesc.net

ABSTRACT: Performance tests play a crucial role in ensuring the quality and efficiency of web systems, allowing for the identification of potential bottlenecks and areas for improvement. This case study aimed to analyze the results of performance tests carried out on a web software, using the JMeter tool and the Elasticsearch, Logstash, and Kibana tools for result analysis. The research sought to understand software performance tests, their methodologies, and tools, in addition to identifying possible improvements to optimize system performance. After implementing the suggested improvements, a second performance test was conducted to assess the effectiveness of the changes made. The results obtained showed significant improvements in system performance, with reductions in response times and maximum execution times. The comparative analysis between the initial test and the test after the improvements highlighted the positive impacts of the implemented changes, emphasizing the importance of performance tests in the continuous improvement of web systems. In short, this study reinforces the importance of performance tests in identifying problems, optimizing systems, and ensuring a satisfactory experience for online platform users.

Keywords: Performance tests; web software; system optimization; continuous improvement.

1 INTRODUÇÃO

Os testes de software são uma parte fundamental do processo de desenvolvimento de um software e são realizados com o objetivo de garantir que o software seja funcional, seguro, eficiente e confiável. De acordo com Myers, Sandler, Badgett (2011), o processo de teste de software consiste em uma sequência de etapas planejadas para garantir que o código do computador execute precisamente o que foi concebido para fazer, sem realizar ações indesejadas. É fundamental que o software seja confiável e coerente, evitando assim quaisquer resultados inesperados para os seus usuários.

Os testes de software são importantes porque ajudam a reduzir os riscos e os custos associados a falhas em software. Como afirma Pressman (2011), o propósito do processo de teste de software é identificar falhas. Este objetivo é alcançado por meio de uma sequência de etapas de testes. Além disso, eles também ajudam a melhorar a qualidade do software, garantindo que ele atenda aos requisitos especificados.

Existem diversos tipos de testes de software, abrangendo áreas como funcionalidade, segurança, usabilidade e desempenho. Cada tipo de teste tem o objetivo de identificar diferentes tipos de problemas que podem afetar o funcionamento do software.

Os testes de desempenho em softwares são essenciais para garantir que um sistema seja capaz de lidar com uma carga de trabalho crescente e manter seu desempenho aceitável. Aprimorar a performance é um dos propósitos do teste de desempenho. Esses testes são capazes de identificar os módulos do software que são mais utilizados ou que demandam mais recursos do sistema. A partir dessa análise, é possível realizar ajustes e modificações no código desses módulos, com o objetivo de otimizá-los e garantir uma execução mais rápida e eficiente (Kaner; Falk; Nguyen, 1999).

Identificar e corrigir problemas de desempenho que possam afetar negativamente a experiência do usuário e prejudicar a reputação do software, é um dos principais objetivos dos testes de desempenho. Segundo Myers, Sandler, Badgett (2011), o desempenho do sistema tem um papel fundamental na formação da primeira impressão do cliente. Os usuários da internet buscam satisfação imediata e muitas vezes não estão dispostos a aguardar longos períodos de carregamento de páginas ou conclusão de transações. De fato, um atraso de apenas alguns segundos pode ser suficiente para que um cliente decida buscar outra opção de site. Além disso, um baixo desempenho pode causar dúvidas quanto à confiabilidade do site por parte dos clientes.

Os testes de desempenho podem incluir vários tipos de abordagens, como testes de carga, estresse e capacidade (Meier et al., 2007). Esses tipos de testes são fundamentais para avaliar diversos aspectos de desempenho de um sistema web e garantir que ele seja capaz de lidar com diferentes cenários de uso.

Tendo em vista o aumento da demanda por softwares web, é essencial garantir que esses sistemas sejam capazes de lidar com um grande número de usuários e ainda assim oferecer um alto desempenho. A falta de testes de desempenho adequados pode levar a problemas como lentidão, travamentos, queda do sistema e, consequentemente, uma má experiência do usuário. Isso pode resultar em prejuízos financeiros e de reputação para empresas e desenvolvedores, além de frustração e insatisfação por parte dos usuários.

Nesta pesquisa, teve-se como objetivo geral, analisar os resultados de teste de desempenho baseado em um estudo de caso realizado em

um software web. Tendo-se também como objetivos: compreender sobre testes de desempenho de softwares, ferramentas e metodologias de análise de resultados; executar testes de desempenho no sistema web por meio da ferramenta JMeter; analisar os resultados dos testes utilizando as ferramentas Elasticsearch, Logstash e Kibana, identificando possíveis gargalos de desempenho no sistema; identificar possíveis melhorias para otimizar o desempenho do sistema, com base nos resultados obtidos na análise; realizar um segundo teste de desempenho após a implementação das melhorias, comparando os resultados com o teste inicial e avaliando a eficácia das alterações realizadas.

As ações descritas tiveram como intuito evidenciar a importância da realização de testes de desempenho em softwares web, mostrando todo o processo prático, desde a etapa de desenvolvimento dos testes até a avaliação dos resultados obtidos, podendo assim no final, ser analisado o quão eficiente a análise prática foi para o software.

2 PESQUISAS ASSOCIADAS A TESTES DE DESEMPENHO

O artigo "Comparative Analysis Of Web Application Performance Testing Tools"de Kołtun e Pańczyk publicado em 2020, compara três ferramentas de teste de desempenho de aplicativos web: Apache JMeter, LoadNinja e Gatling. O objetivo é determinar a ferramenta mais adequada com base na interface do usuário, parametrização de solicitações e flexibilidade na criação de scripts de teste.

O artigo "Software Performance Testing"de Melkozyorova e Rassomakhin, publicado em 2020, discute a importância do teste de desempenho no desenvolvimento de software, com especial ênfase na ferramenta Apache JMeter. Destaca-se a importância de avaliar sistemas e aplicativos sob diferentes cargas, com o JMeter emergindo como uma solução eficaz para esse propósito.

O artigo "Testing Self-adaptive Software With Probabilistic Guarantees On Performance Metrics: Extended And Comparative Results"de 2022 feito por Claudio Mandrioli e Martina Maggio, é apresentada uma estratégia para testar o desempenho de sistemas autoadaptativos com garantias probabilísticas. A abordagem envolve a realização de testes aleatórios repetidos, cada um representando uma possível execução do sistema, e a avaliação dos resultados desses testes usando métodos estatísticos tradicionais e a Teoria do Cenário.

O presente trabalho se diferencia dos demais citados acima, ao

aplicar ferramentas de teste de desempenho em um estudo de caso real, envolvendo um software web, com duas fases de testes para identificar gargalos e avaliar melhorias. Enquanto os artigos de Kołtun e Pańczyk (2020) e de Melkozyorova e Rassomakhin (2020) focam na comparação de ferramentas e na eficácia do JMeter, e o de Mandrioli e Maggio (2022) em estratégias probabilísticas para sistemas autoadaptativos, esse estudo destaca a integração prática do JMeter com o ELK Stack para análise de resultados.

3 MATERIAIS E MÉTODOS

A análise dos testes de desempenho realizados em um software web neste trabalho tiveram como objetivo identificar possíveis gargalos e fornecer sugestões de melhorias. Esta pesquisa foi conduzida com uma abordagem tecnológica, descritiva e baseada em estudos bibliográficos. Para isso, foram empregadas diversas ferramentas ao longo do processo de desenvolvimento, execução e análise de resultados dos testes.

3.1 FERRAMENTAS UTILIZADAS PARA O TESTE

O Apache JMeter foi escolhido para o desenvolvimento e execução dos testes, oferecendo uma solução robusta e flexível para simular o comportamento dos usuários e gerar carga no sistema. Este é um software livre e de código aberto desenvolvido em Java, utilizado para testar o desempenho e o comportamento funcional de aplicativos web. Ele também permite realizar análises gráficas do desempenho de servidores, scripts e objetos sob carga simultânea pesada (Memon; Hafiz; Bhatti, 2018).

Por sua vez, o pacote ELK Stack foi utilizado para analisar os resultados obtidos durante a execução dos testes. Esse conjunto de ferramentas, composto por Elasticsearch, Logstash e Kibana, permite uma análise detalhada e eficiente dos dados coletados.

O Elasticsearch é o componente central do ELK Stack, ele centraliza o armazenamento de dados, possibilitando buscas ágeis, refinadas configurações de relevância e análises robustas, com a flexibilidade para ser escalado com facilidade (Elastic, 2024a). Trata-se de um busca distribuído e escalável, que possibilita pesquisas em texto completo. Escrito em Java, o Elasticsearch é independente de plataforma e é considerado bastante estável. Por meio de sua combinação de recursos, é possível obter flexibilidade em relação a requisitos específicos e opções simples para expansão. Todas essas características são extremamente úteis para análise

de big data em tempo real (Shah; Willick; Mago, 2018).

O Elasticsearch também pode ser usado em conjunto com outras ferramentas para se ter resultados melhores. Segundo Shah, Willick, Mago (2018) o Elasticsearch é amplamente utilizado para análises em tempo real e é aprimorado por plugins estendidos, como o Kibana e o Logstash, que oferecem representações funcionais de big data em tempo real. De acordo com Elastic (2024b) o Logstash é uma ferramenta gratuita e de código aberto, situada no servidor, que atua como um sistema de canalização de dados. Sua principal função é coletar informações de diversas fontes, realizar transformações necessárias e direcionar esses dados para o destino escolhido, funcionando como um intermediário de armazenamento.

A ferramenta Kibana vem com várias visualizações padrão, facilitando o desenvolvimento de visualizações para usuários finais por meio de um recurso de arrastar e soltar. Além disso, como o Kibana é suportado pela arquitetura do Elasticsearch, é extremamente rápido e eficiente para análises em tempo real. Por fim, ele permite a interação gráfica durante a construção e manipulação de consultas, fornecendo uma visão acessível da integridade e das propriedades do cluster no banco de dados (Shah; Willick; Mago, 2018).

A escolha dessas ferramentas se deu pela complementaridade entre elas, permitindo a extração dos resultados gerados pelo JMeter e sua importação para o Elasticsearch, possibilitando uma análise detalhada.

Essas ferramentas desempenharam papéis fundamentais em cada etapa do ciclo de teste de desempenho, garantindo uma abordagem abrangente e eficaz. O Apache JMeter facilitou a simulação de cenários de uso realista e a coleta de dados relevantes sobre o desempenho do sistema sob diferentes condições de carga. Por sua vez, o ELK Stack ofereceu recursos avançados de análise e visualização de dados, permitindo a identificação de padrões, tendências e possíveis áreas de melhoria no sistema.

A integração dessas ferramentas proporcionou uma análise completa e detalhada do desempenho do software web, resultando em informações valiosas para a otimização e aprimoramento contínuo do sistema.

3.2 APLICAÇÃO WEB USADA PARA ANÁLISE PRÁTICA

O software utilizado para a condução da pesquisa é uma plataforma de gestão de contratos, cujo nome e detalhes específicos foram omitidos devido a restrições impostas pelo proprietário. Esta aplicação desempenha um papel fundamental na administração eficiente e segura de

contratos, oferecendo uma ampla gama de funcionalidades destinadas a facilitar o processo de gerenciamento contratual.

Entre as principais características do software, destacam-se sua capacidade de criar, gerenciar, assinar e visualizar contratos de forma intuitiva e eficaz. Além disso, a plataforma oferece recursos avançados, como armazenamento seguro de documentos contratuais, rastreamento de obrigações, geração de alertas e notificações por e-mail, fornecendo assim um ambiente completo para o ciclo de vida dos contratos.

Uma das vantagens distintivas deste software é sua capacidade de personalização e adaptação às necessidades específicas de cada organização ou setor. Com uma interface flexível e ferramentas de relatórios avançadas, os usuários podem moldar a aplicação de acordo com seus requisitos exclusivos, garantindo uma experiência sob medida para suas operações contratuais.

3.3 DESENVOLVIMENTO DO TESTE

Os testes de desempenho buscam replicar a experiência de múltiplos usuários interagindo com o sistema simultaneamente, mas com uma abordagem distinta dos testes de interface do usuário (UI). Enquanto os testes de UI reproduzem a interação de um usuário real, navegando por páginas e aguardando o carregamento dos elementos, os testes de desempenho se concentram exclusivamente nas requisições feitas durante esse fluxo.

Em outras palavras, o teste de desempenho é centrado nas ações realizadas pelo usuário no sistema, mas sem a necessidade de simular visualmente a navegação por páginas web. Em vez disso, o foco está nas requisições enviadas ao servidor enquanto o usuário percorre o fluxo de interações. Essa abordagem oferece uma visão mais direta e detalhada do desempenho do sistema sob carga, permitindo identificar e resolver potenciais gargalos com maior precisão.

Antes de iniciar os testes, é necessário estabelecer diversas configurações fundamentais, tais como o ambiente de execução, o fluxo de testes a ser seguido, a estratégia de aumento gradual de usuários durante a execução (rampa de usuários) e outras definições relevantes para garantir a precisão e eficácia dos testes.

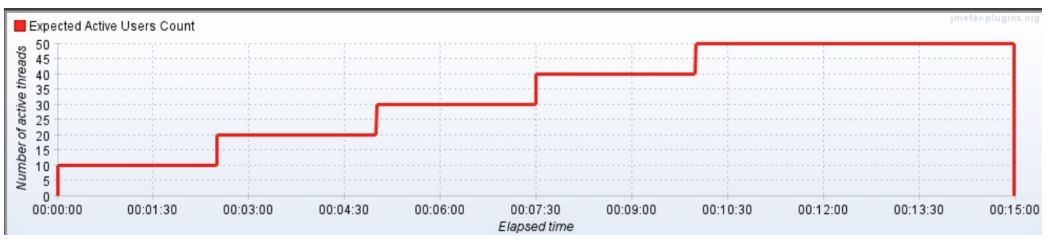
3.3.1 Definições Iniciais

Devido à natureza contínua da operação da aplicação, que requer funcionamento ininterrupto em produção, optou-se por desenvolver e executar os testes em um ambiente separado. Esse ambiente, embora simulasse as condições de produção, dispunha de recursos substancialmente inferiores, resultando em uma redução significativa na carga de usuários durante os testes.

Para determinar a carga máxima de usuários, foram realizados alguns testes preliminares (smoke testes) para estimar a capacidade aproximada do sistema em lidar com usuários simultâneos. Quanto ao tempo de execução, optou-se por um intervalo relativamente curto, considerando que o teste precisava ser finalizado no dia, por conta de que o ambiente era usado para outras atividades que foram paralisadas para esse teste. Portanto, era fundamental manter o tempo de teste o mais eficiente possível, permitindo ajustes caso ocorressem problemas durante os testes preliminares ou durante a execução oficial.

Dessa forma, definiu-se uma rampa de usuários que começava com 10 usuários simultâneos, adicionando-se mais 10 usuários a cada 2 minutos e 30 segundos, até atingir um máximo de 50 usuários simultâneos. Essa carga máxima foi mantida durante 5 minutos, totalizando aproximadamente 15 minutos de execução para o teste, conforme ilustrado na Figura 1.

Figura 1 - Rampa de usuários.



Fonte: O autor, 2024.

3.3.1.1 Fluxo de Teste

Considerando a relevância das interações relacionadas à gestão de contratos, o fluxo escolhido para ser testado foi o de visualização de contrato. Nesse processo, o usuário realiza uma série de etapas, iniciando-se com o login no sistema, o carregamento da página principal da aplicação, o acesso à tela de contratos, a aplicação de um filtro pelo nome de um con-

trato específico (sorteado para o teste) e, por fim, a visualização detalhada deste contrato na tela correspondente.

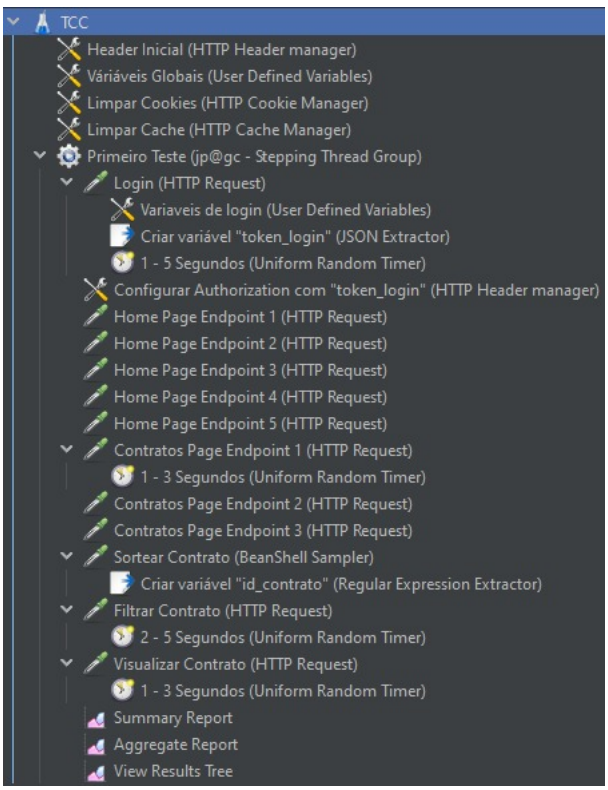
3.3.2 Apache JMeter

O teste de desempenho foi desenvolvido com auxílio da ferramenta Apache JMeter, a qual opera em um ambiente Java e tem o propósito de simular o comportamento de vários usuários acessando uma aplicação ou serviço ao mesmo tempo. Isso é alcançado por meio da criação de planos de teste, nos quais os usuários virtuais, são configurados para realizar uma variedade de ações. Durante a execução dos testes, o JMeter monitora métricas fundamentais, como tempo de resposta, taxa de transferência, latência e ocorrência de erros, proporcionando uma avaliação minuciosa do desempenho do sistema sob diferentes níveis de carga.

3.3.2.1 Componentes do JMeter Usados no Teste

Durante o desenvolvimento dos testes, empregou-se uma variedade de componentes do JMeter para garantir a eficácia e precisão das simulações, como mostra a Figura 2.

Figura 2 - Componentes do JMeter usados + Fluxo de teste.



Fonte: O autor, 2024.

Utilizou-se o HTTP Cookie Manager e o HTTP Cache Manager para limpar cookies e cache em cada execução de usuário, promovendo assim uma interação mais autêntica com o sistema. Além disso, recorreu-se ao User Defined Variables para estabelecer variáveis globais e ao HTTP Header Manager para controlar e personalizar os cabeçalhos HTTP, aprimorando a precisão e flexibilidade dos testes.

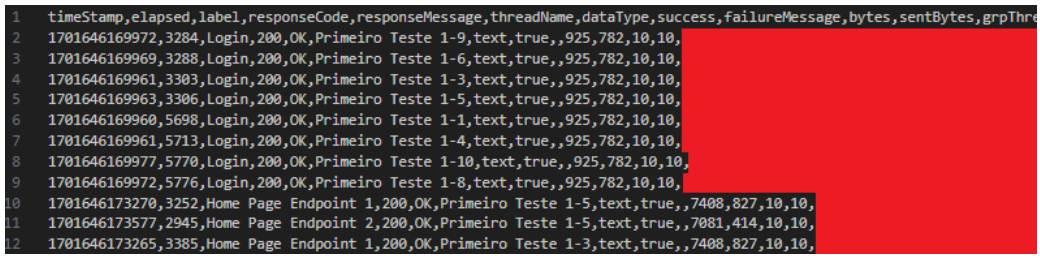
O componente jp@gc - Stepping Thread Group foi usado para configurar a rampa de usuários e simular a execução de cada Thread (usuário), permitindo uma distribuição gradual e controlada das cargas de trabalho. Para as chamadas HTTP, utilizou-se o componente HTTP Request, enquanto o BeanShell Sampler foi empregado para sortear IDs de contratos existentes, introduzindo uma variabilidade realista nas interações do teste. Adicionalmente, implementou-se o Uniform Random Timer para introduzir intervalos aleatórios antes de determinadas requisições, refletindo de forma mais precisa o comportamento dos usuários reais. Também foi utilizado o componente JSON Extractor, a fim de extrair informações da resposta de uma requisição HTTP, e o componente Regular Expression Extractor para extrair uma resposta do componente BeanShell Sampler.

Por fim, para monitorar e analisar os resultados dos testes, foram integrados três componentes do tipo Listener: View Results Tree, Summary Report e Aggregate Report. Esses componentes fornecem uma visão detalhada das métricas de execução em tempo real, além de permitirem exportar arquivos de resultados de execução.

3.3.3 Execução do Teste

Após o desenvolvimento, o teste foi executado utilizando o Apache JMeter e os resultados foram armazenados em um arquivo .jtl, como mostra a Figura 3. Previamente configurado a partir dos componentes de tipo Listener usados no teste conforme apresentado na Figura 2.

Figura 3 - Arquivo de resultados .jtl.



```
timeStamp,elapsed,label,responseCode,responseMessage,threadName,dataType,success,failureMessage,bytes,sentBytes,grpThre
1701646169972,3284,Login,200,OK,Primeiro Teste 1-9,text,true,,925,782,10,10,
1701646169969,3288,Login,200,OK,Primeiro Teste 1-6,text,true,,925,782,10,10,
1701646169961,3303,Login,200,OK,Primeiro Teste 1-3,text,true,,925,782,10,10,
1701646169963,3306,Login,200,OK,Primeiro Teste 1-5,text,true,,925,782,10,10,
1701646169960,5698,Login,200,OK,Primeiro Teste 1-1,text,true,,925,782,10,10,
1701646169961,5713,Login,200,OK,Primeiro Teste 1-4,text,true,,925,782,10,10,
1701646169977,5770,Login,200,OK,Primeiro Teste 1-10,text,true,,925,782,10,10,
1701646169972,5776,Login,200,OK,Primeiro Teste 1-8,text,true,,925,782,10,10,
1701646173270,3252,Home Page Endpoint 1,200,OK,Primeiro Teste 1-5,text,true,,7408,827,10,10,
1701646173577,2945,Home Page Endpoint 2,200,OK,Primeiro Teste 1-5,text,true,,7081,414,10,10,
1701646173265,3385,Home Page Endpoint 1,200,OK,Primeiro Teste 1-3,text,true,,7408,827,10,10,
```

Fonte: O autor, 2024.

3.4 ANÁLISE DE RESULTADOS DO PRIMEIRO TESTE

Após a conclusão das etapas de desenvolvimento e execução, iniciou-se a fase de análise de resultados. Ao conduzir testes de desempenho, é fundamental não apenas coletar dados, mas também analisá-los de maneira eficaz para identificar gargalos, prever problemas futuros e tomar decisões informadas sobre otimizações necessárias. Ferramentas especializadas são essenciais nesse processo, pois permitem uma visualização clara e detalhada dos resultados, facilitando a interpretação dos dados e a implementação de melhorias.

3.4.1 Pacote de Ferramentas ELK Stack

Para analisar os resultados do teste foi utilizado o pacote de ferramentas ELK Stack, o qual é composto por Elasticsearch, Logstash e Kibana. Sendo o Elasticsearch o responsável por armazenar, pesquisar e analisar grandes volumes de dados rapidamente, o Logstash o responsável por coletar, processar e transformar dados de diversas fontes antes de enviá-los ao Elasticsearch, e o Kibana responsável por fornecer visualizações e dashboards interativos para analisar os dados armazenados no Elasticsearch.

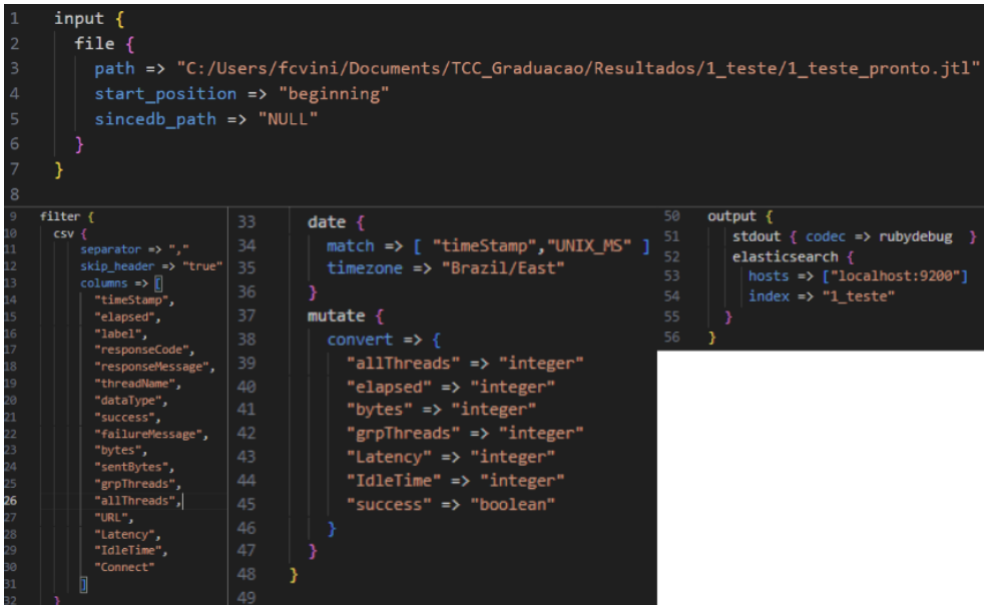
3.4.1.1 Preparações para a Análise

A primeira etapa envolveu a criação e configuração de uma instância no Elasticsearch para receber os dados. Em seguida, a ferramenta Kibana foi configurada para apontar para essa instância do Elasticsearch, permitindo a visualização dos dados armazenados.

Logo após, foi necessário editar alguns dados no arquivo .jtl. Apenas os resultados dos componentes HTTP Request utilizados para fazer requisições aos endpoints da aplicação eram relevantes para a análise. No entanto, os componentes Listener do Apache JMeter exportam dados de todos os componentes do tipo Sampler usados no teste. Isso incluiu os resultados do componente BeanShell Sampler usado para sortear os contratos a serem visualizados, que também foram exportados para o arquivo .jtl. Foi necessário remover manualmente do arquivo esses dados considerados irrelevantes.

Posteriormente, foi criado um arquivo de configuração (.conf) ilustrado na Figura 4, com o intuito de ler os arquivos de resultados de execução (.jtl) obtidos durante os testes. Esse arquivo de configuração foi dividido em três partes: input, filter e output.

Figura 4 - Arquivo de configuração .conf.



Fonte: O autor, 2024.

Na seção de input, foi definido qual arquivo com dados seria lido. A seção de filter foi responsável por processar e transformar para um formato que o Elasticsearch pudesse interpretar. Por fim, na seção de output, foi especificado o destino final dos dados processados, direcionando-os para o Elasticsearch.

Usando o Logstash, o arquivo de configuração (.conf) foi executado, enviando todos os dados para a instância previamente configurada do Elasticsearch. Em seguida, foi criado um novo dashboard no Kibana. Dentro desse dashboard, foram criadas e configuradas várias visualizações diferentes, com o objetivo de facilitar a análise dos resultados dos testes.

3.4.2 Sugestões de melhorias

Após a importação dos dados no Elasticsearch e a devida configuração do Kibana, tornou-se possível visualizar diversas métricas, com o objetivo de identificar possíveis gargalos no fluxo da aplicação, essas métricas serão detalhadas na seção de resultados e discussões. Em seguida, foi elaborado um relatório detalhando os possíveis gargalos e apresentando sugestões de melhorias, que foi enviado ao arquiteto de desenvolvimento da aplicação.

3.5 ANÁLISE DE RESULTADOS DO SEGUNDO TESTE

Após a implementação das alterações sugeridas, um segundo teste foi realizado para permitir a comparação dos resultados com os do teste inicial. As etapas desse segundo teste incluíram o desenvolvimento de um novo arquivo de teste para verificar se algum endpoint havia mudado, sido acrescentado ou removido no fluxo do teste.

Para a parte de análise de resultados, o processo foi mais simples, já que o arquivo .conf e as ferramentas do ELK Stack já estavam devidamente configurados. Foi necessário apenas retirar os dados irrelevantes no arquivo de resultados .jtl, assim como na primeira execução, carregar os resultados do arquivo no Elasticsearch e realizar uma nova análise dos resultados por meio do Kibana, comparando o desempenho da primeira execução com a segunda.

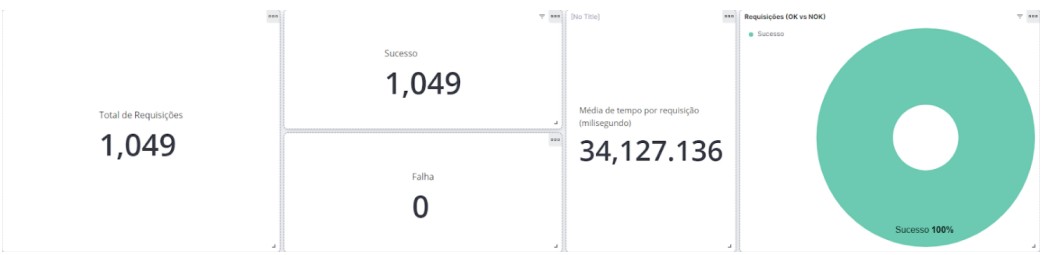
4 RESULTADOS E DISCUSSÃO

Nesta seção, serão apresentados os resultados dos dois testes de desempenho realizados, por meio dos gráficos e tabelas mais relevantes, analisando as métricas obtidas e discutindo as principais observações. Inicialmente, serão destacados os resultados do primeiro teste, identificando possíveis gargalos e áreas de melhoria. Em seguida, será abordado os resultados do segundo teste, realizado após as implementações das alterações sugeridas.

4.1 RESULTADOS PRIMEIRO TESTE

O primeiro teste executou um total de 1.049 requisições, todas bem-sucedidas, com um tempo médio de 34,127 segundos por requisição, conforme mostra a Figura 5.

Figura 5 - Sumário de execução 1° teste.

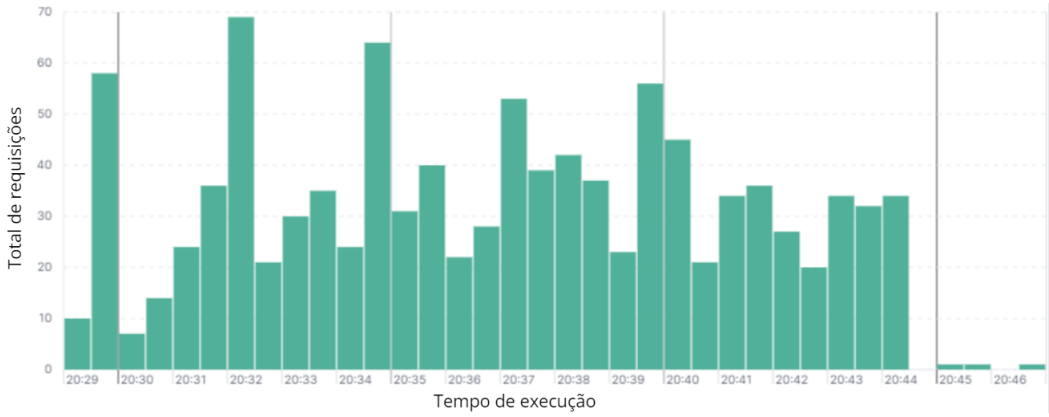


Fonte: O autor, 2024.

A Figura 6 mostra a quantidade de requisições executadas a

cada 30 segundos.

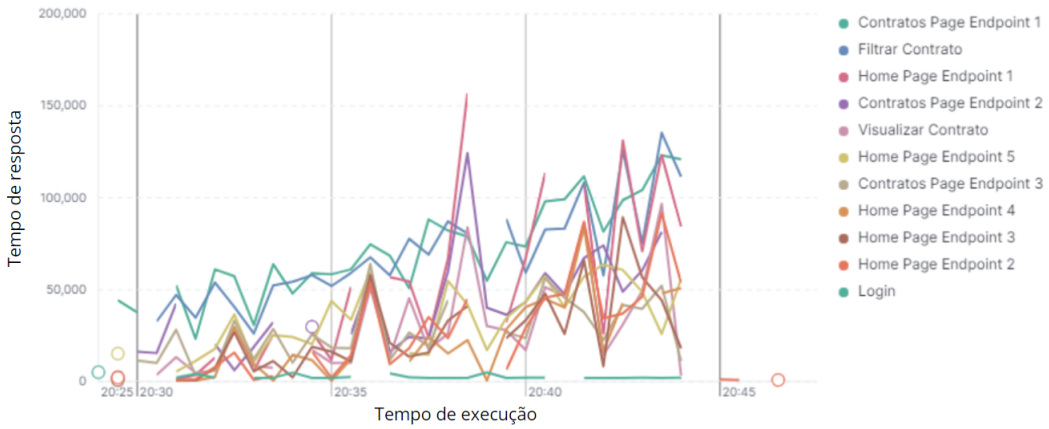
Figura 6 - Total de requisições por 30 segundos 1° teste.



Fonte: O autor, 2024.

A Figura 7 apresenta o tempo médio de resposta que cada endpoint teve durante o teste.

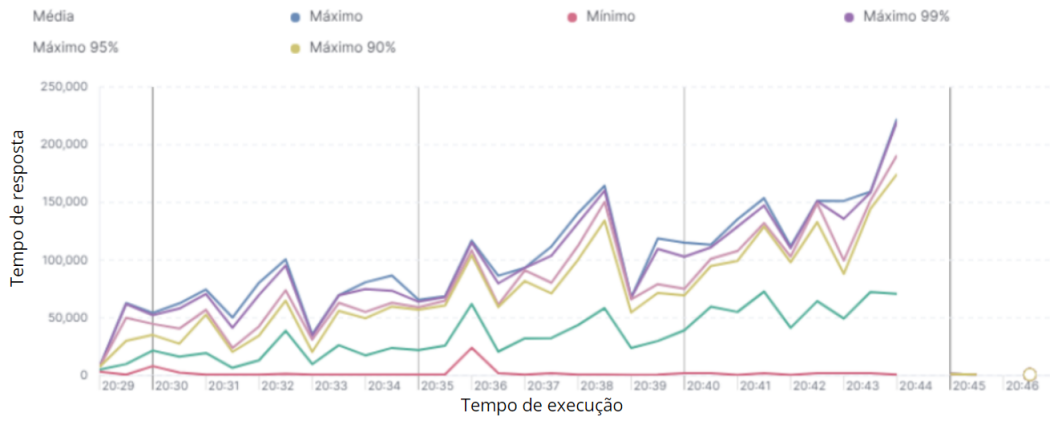
Figura 7 - Tempo médio de resposta por endpoint 1° teste.



Fonte: O autor, 2024.

Já a Figura 8 ilustra o tempo de todas as requisições feitas durante o teste com diferentes métricas.

Figura 8 - Tempo de resposta de todas as requisições 1° teste.



Fonte: O autor, 2024.

A Tabela 1 apresenta dados como total de requisições feitas para cada endpoint, média, mínima e máxima de tempo das requisições, e a média de bytes trafegados.

Tabela 1 – Tempo de resposta por requisição 1° teste

Requisição	Total de requisições	Média (seg)	Min (seg)	Max (seg)	Média de Bytes
Login	113	2,233	1,956	7,936	925
Home Page Endpoint 1	113	26,260	0,882	168,529	7.410
Home Page Endpoint 2	113	9,426	0,659	144,363	7.083
Home Page Endpoint 3	107	10,817	0,373	127,590	1.882
Home Page Endpoint 4	103	6,509	0,266	107,048	4.979
Home Page Endpoint 5	100	24,231	4,772	123,203	497.148
Contratos Page Endpoint 1	99	65,612	20,874	215,423	742.623
Contratos Page Endpoint 2	82	30,107	6,244	143,324	282.575
Contratos Page Endpoint 3	78	17,762	2,665	100,804	195.506
Filtrar Contrato	75	60,618	15,838	222,344	906.576
Visualizar Contrato	66	24,788	1,240	112,187	25.204

Fonte: O autor, 2024.

Analisando as métricas fornecidas pelo Kibana, a primeira coisa que se nota é que o sistema não apresentou erros ou indisponibilidade em momento algum, mesmo sob condições severas de carga. Embora a aplicação tenha ficado praticamente inutilizável devido à lentidão, com requisições levando até 3 minutos e 42 segundos para serem executadas, o sistema demonstrou resiliência e não foi derrubado.

Nota-se também que o sistema foi submetido a uma carga de estresse bastante alta. Isso é evidente ao analisar as colunas de tempo mínimo e máximo de cada requisição na Tabela 1. Observa-se uma grande variação nos tempos de resposta: os tempos menores ocorrem no início do teste, quando havia menos usuários simultâneos, e os tempos maiores aparecem no final, quando o número de usuários atingiu seu pico. As figuras 7 e 8 ilustram esse comportamento, mostrando gráficos onde o tempo de resposta aumenta conforme o teste avança e o número de usuários simultâneos cresce.

Ao examinar os endpoints que mais impactaram o desempenho da aplicação, observa-se que as requisições que levaram mais tempo para serem processadas foram as de "Contratos Page Endpoint 1"e "Filtrar Contrato". A Tabela 1 mostra que essas requisições também foram as que trafegaram mais bytes durante o teste. Analisando os dados de resposta dessas requisições, percebeu-se que muitas informações trafegadas eram aparentemente desnecessárias.

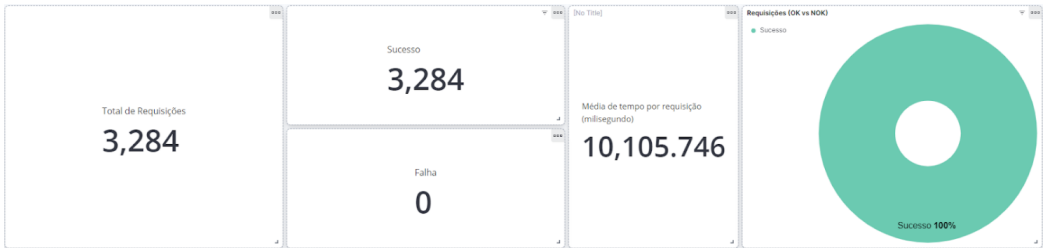
Essa análise revelou que o problema era recorrente em várias requisições do fluxo de teste. Muitos endpoints, que deveriam retornar apenas informações básicas sobre determinado registro, traziam diversas configurações e dados de outros vínculos que não eram relevantes para o sistema, inclusive com muitos campos nulos.

Portanto, a principal recomendação de melhoria foi revisar a quantidade de informações trafegadas em cada endpoint, a fim de diminuir o volume de bytes transferidos e retornar apenas as informações essenciais.

4.2 RESULTADOS SEGUNDO TESTE

Ao comparar o segundo teste com o primeiro, percebe-se uma grande diferença na quantidade de requisições que o sistema conseguiu realizar durante os testes. Isso pode ser visto ao comparar as Figuras 5 e 9, nas quais o primeiro teste executou um total de 1.049 requisições, com uma média de aproximadamente 34 segundos por requisição, enquanto o segundo teste executou um total de 3.284 requisições, com uma média de aproximadamente 10 segundos por requisição. Assim como no primeiro teste, o sistema não apresentou indisponibilidade em nenhum momento durante o segundo teste.

Figura 9 - Sumário de execução 2° teste.

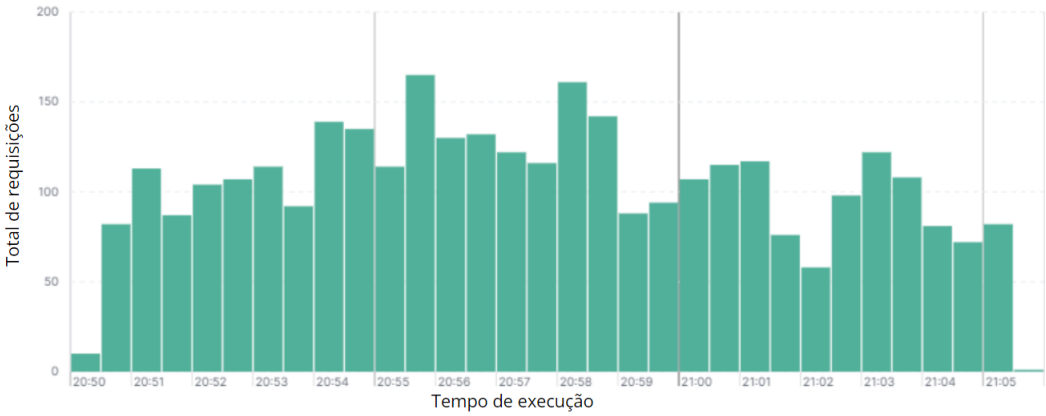


Fonte: O autor, 2024.

Na Figura 6 observa-se um aumento significativo a cada 2 minutos e meio, seguido por uma queda, devido à execução de novas threads conforme a rampa de usuários. Embora o login, em um domínio diferente, fosse bem-sucedido mesmo com o sistema sobrecarregado, as demais páginas enfrentaram mais dificuldades para carregar. Enquanto na Figura 10 nota-se um gráfico de barras com menos picos, o que demonstra que o sistema conseguiu lidar com a carga de usuários de maneira muito mais eficiente.

Além disso, no primeiro teste, o tempo observado foi maior que o estipulado na Figura 1, pois ao final do teste, as threads demoravam para finalizar as requisições antes de serem encerradas devido à sobrecarga do sistema. Em contraste, no segundo teste, esse problema não ocorreu e o teste foi concluído em 15 minutos, conforme planejado.

Figura 10 - Total de requisições por 30 segundos 2° teste.

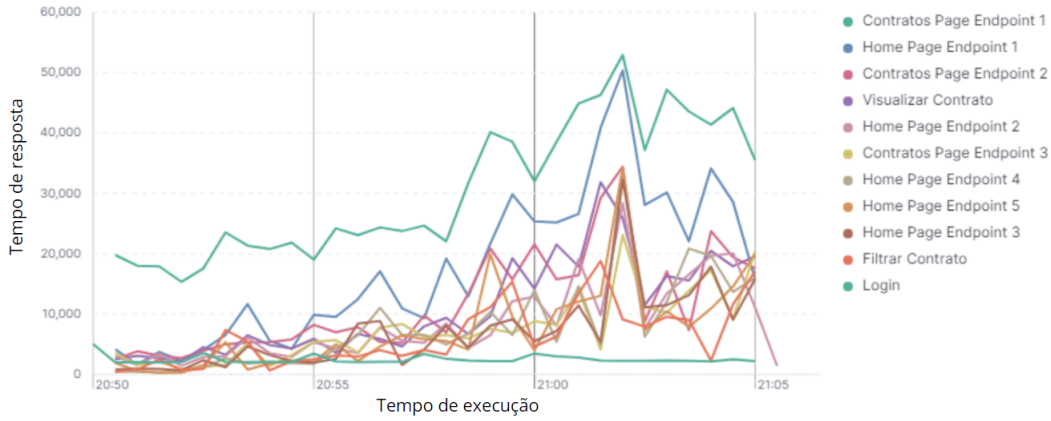


Fonte: O autor, 2024.

Na Figura 11, nota-se uma queda significativa no tempo de resposta de cada endpoint em relação a Figura 7, especialmente nos que fo-

ram os principais agressores no primeiro teste.

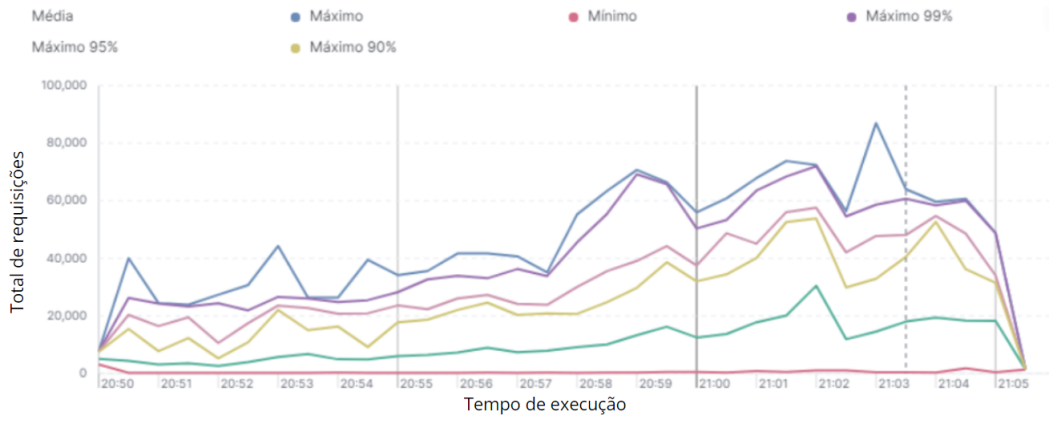
Figura 11 - Tempo médio de resposta por endpoint 2° teste.



Fonte: O autor, 2024.

Comparando os gráficos das Figuras 8 e 12, observa-se uma grande diferença nos tempos de resposta gerais dos testes. A média de todas as requisições a cada 30 segundos no segundo teste atingiu um pico máximo de aproximadamente 30 segundos, enquanto no primeiro teste o pico foi de aproximadamente 73 segundos.

Figura 12 - Tempo de resposta de todas as requisições 2° teste.



Fonte: O autor, 2024.

Também percebe-se melhorias evidentes ao comparar as Tabelas 1 e 2.

Tabela 2 – Tempo de resposta por requisição 2° teste

Requisição	Total de requisições	Média (seg)	Min (seg)	Max (seg)	Média de Bytes
Login	315	2,146	1,934	8,022	925
Home Page Endpoint 1	315	7,940	0,859	86,967	7.407
Home Page Endpoint 2	315	3,944	0,640	51,342	7.080
Home Page Endpoint 3	313	2,047	0,361	48,551	1.882
Home Page Endpoint 4	308	1.934	0,259	52,786	4.979
Home Page Endpoint 5	303	1,372	0,222	46,502	22.534
Contratos Page Endpoint 1	298	26,153	7,247	72,391	71.774
Contratos Page Endpoint 2	288	7,138	1,274	54,018	9.451
Contratos Page Endpoint 3	284	3,393	0,490	47,061	19.501
Filtrar Contrato	275	2,231	0,299	39,247	4.466
Visualizar Contrato	270	4,932	1,174	52,797	25.218

Fonte: O autor, 2024.

No primeiro teste, os dois endpoints mais prejudiciais ao desempenho da aplicação foram o "Contratos Page Endpoint 1", que apresentou um tempo médio de resposta de aproximadamente 66 segundos e um tempo máximo de 215 segundos. No segundo teste, esses valores caíram para aproximadamente 26 segundos e 72 segundos, respectivamente. O segundo principal agressor foi o "Filtrar Contrato", que no primeiro teste teve uma média de aproximadamente 61 segundos e um tempo máximo de 222 segundos. No segundo teste, esses valores reduziram-se para uma média de aproximadamente 2 segundos e um tempo máximo de 39 segundos, tornando esse endpoint apenas o sétimo na lista de principais agressores ao desempenho da aplicação no segundo teste.

Outro ponto importante é a diminuição da média de bytes trafegados por cada endpoint, sendo essa a principal sugestão de melhoria após a análise do primeiro teste. O endpoint que teve a maior redução na quantidade de bytes trafegados foi o "Filtrar Contrato", que no primeiro teste apresentou uma média de 906.576 bytes, enquanto no segundo teste apresentou apenas 4.466 bytes. Sendo esse o endpoint que demonstrou a maior melhora de desempenho em comparação com o primeiro teste.

Um dado adicional significativo foi a quantidade de vezes que o endpoint "Visualizar Contrato"foi executado. Este endpoint, sendo o último do fluxo, indica que cada vez que foi executado o fluxo chegou ao final com sucesso. No primeiro teste, o fluxo foi completado 66 vezes, enquanto no segundo teste foi completado 270 vezes, demonstrando mais uma vez a

eficiência das melhorias implementadas.

5 CONCLUSÃO

Os testes de desempenho são essenciais na melhoria contínua de sistemas web, garantindo que aplicativos e plataformas online atendam às expectativas de desempenho dos usuários. Por meio do estudo de caso apresentado neste trabalho, foi possível observar como a realização de testes sistemáticos e a análise criteriosa dos resultados podem identificar gargalos, áreas de melhoria e oportunidades de otimização.

Ao longo dos testes realizados, foi evidenciado que a identificação e correção de pontos críticos, como os endpoints mais prejudiciais ao desempenho da aplicação, resultaram em melhorias significativas. Reduções substanciais nos tempos de resposta, nos tempos máximos de execução e na quantidade de bytes trafegados foram alcançadas após a implementação das alterações sugeridas.

Além disso, a comparação entre o primeiro e o segundo teste demonstrou claramente os impactos positivos das melhorias implementadas, refletindo em um aumento significativo na eficiência e na confiabilidade do sistema. A frequência de execução bem-sucedida do fluxo de operação também evidenciou a eficácia das alterações realizadas.

Olhando para o futuro, uma área promissora de pesquisa seria explorar novas técnicas de teste de desempenho e utilizar outras ferramentas, fazendo um comparativo entre as mesmas. Isso poderia fornecer insights adicionais sobre a eficácia de diferentes abordagens e ferramentas, contribuindo ainda mais para a otimização e aprimoramento de sistemas web.

Diante disso, fica evidente que os testes de desempenho fazem parte de um papel fundamental na garantia da qualidade e no aprimoramento contínuo de sistemas web. A análise criteriosa dos resultados, a identificação de gargalos e a implementação de melhorias são etapas essenciais para assegurar que os sistemas atendam às demandas dos usuários e ofereçam uma experiência satisfatória.

Portanto, conclui-se que investir em testes de desempenho e na otimização de sistemas web não apenas contribui para a satisfação dos usuários, mas também para a eficiência operacional e a competitividade no mercado digital. A constante busca por aprimoramento e a atenção aos detalhes técnicos são fundamentais para o sucesso e a excelência dos sistemas online.

REFERÊNCIAS

ELASTIC. **Elasticsearch**: o coração do elastic stack gratuito e aberto. 2024a. <https://www.elastic.co/pt/elasticsearch>. Acessado em: 13 de fevereiro de 2024.

ELASTIC. **Logstash**: centralize, transforme e oculte seus dados. 2024b. <https://www.elastic.co/pt/logstash>. Acessado em: 13 de fevereiro de 2024.

KANER, C.; FALK, J.; NGUYEN, H. Q. **Testing Computer Software**. 2. ed. [S.l.]: John Wiley & Sons, 1999.

KOłTUN, A.; PAńCZYK, B. Comparative analysis of web application performance testing tools. **Journal of Computer Sciences Institute**, Politechnika Lubelska, v. 17, p. 351–357. ISSN 2544-0764. Disponível em: <http://dx.doi.org/10.35784/jcsi.2209>.

MANDRIOLI, C.; MAGGIO, M. Testing self-adaptive software with probabilistic guarantees on performance metrics: Extended and comparative results. **IEEE Transactions on Software Engineering**, Institute of Electrical and Electronics Engineers (IEEE), v. 48, n. 9, p. 3554–3572. ISSN 2326-3881. Disponível em: <http://dx.doi.org/10.1109/tse.2021.3101130>.

MEIER, J. D. et al. **Performance Testing Guidance for Web Applications**. 1st. ed. Redmond, Washington, USA: Microsoft Press US, 2007. 288 p. ISBN 9780735625709.

MELKOZYOROVA, O. M.; RASSOMAKHIN, S. G. Software performance testing. **Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems»**, V. N. Karazin Kharkiv National University, n. 45.

MEMON, P. et al. Comparative study of testing tools blazemeter and apache jmeter. **Sukkur Iba Journal Of Computing And Mathematical Sciences**, Sindh, p. 70–76.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing. 3**. New Jersey, USA: John Wiley & Sons, 2011.

PRESSMAN, R. S. **Engenharia de Software: uma abordagem profissional**. 7. ed. New York: Amgh, 2011. Tradução de: Ariovaldo Griesi, Mario Moro Fecchio.

SHAH, N.; WILLICK, D.; MAGO, V. A framework for social media data analytics using elasticsearch and kibana. **Wireless Networks**, Springer Science and Business Media LLC, v. 28, n. 3, p. 1179–1187. ISSN 1572-8196. Disponível em: <http://dx.doi.org/10.1007/s11276-018-01896-2>.