UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Fabrício Fernandes Ziliotti

Otimização de aplicações web: Melhorando as métricas Core Web Vitals com NextJS

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Fabrício Fernandes Ziliotti

Otimização de aplicações web: Melhorando as métricas Core Web Vitals com NextJS

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2024

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade de Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4144 - http://www.portal.facom.ufu.br/ facom@ufu.br

ATA DE DEFESA - GRADUAÇÃO

Curso de Graduação em:	Bacharelado em Ciência da Computação				
Defesa de:	Projeto de Graduação 2 - GBC082				
Data:	30/04/2024	Hora de início:	10:00	Hora de encerramento:	10:50
Matrícula do Discente:	11711BCC002				
Nome do Discente:	Fabrício Fernandes Ziliotti				
Título do Trabalho:	Otimização de aplicações web: Melhorando as métricas Core Web Vitals com NextJS				
A carga horária curricular foi cumprida integralmente?	(X) Sim ( ) Não				

Reuniu-se virtualmente por meio da plataforma MS Teams, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Curso de Graduação em Ciência da Computação, assim composta: Professores: Dr. Daniel Duarte Abdala - FACOM/UFU; Dra. Maria Adriana Vidigal de Lima - FACOM/UFU; e Dr. Paulo Henrique Ribeiro Gabriel - FACOM/UFU, orientador(a) do(a) candidato(a).

Iniciando os trabalhos, o(a) presidente da mesa, Dr(a). Paulo Henrique Ribeiro Gabriel, apresentou a Comissão Examinadora e o candidato(a), agradeceu a presença do público, e concedeu ao discente a palavra, para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do curso.

A seguir o(a) senhor(a) presidente concedeu a palavra, pela ordem sucessivamente, aos(às) examinadores(as), que passaram a arguir o(a) candidato(a). Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o(a) candidato(a):

(X) Aprovado(a) Nota 95

OU

( ) Aprovado(a) sem nota.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Maria Adriana Vidigal de Lima**, **Professor(a) do Magistério Superior**, em 30/04/2024, às 10:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Daniel Duarte Abadala**, **Professor(a) do Magistério Superior**, em 30/04/2024, às 10:55, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Paulo Henrique Ribeiro Gabriel**, **Professor(a) do Magistério Superior**, em 30/04/2024, às 10:56, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5374452** e o código CRC **8246179B**.

Referência: Processo nº 23117.029660/2024-46 SEI nº 5374452

Agradecimentos

Em primeiro lugar, agradeço a todos que contribuíram para a realização deste trabalho e influenciaram o sucesso deste projeto.

Sou muito grato ao meu orientador, que me ajudou muito durante todo o processo. A sua orientação foi muito importante para a qualidade e desenvolvimento deste estudo.

Além disso, agradeço aos professores pelo compartilhamento de seus conhecimentos. Sem o aprendizado obtido com cada um deles, este trabalho não teria sido possível.

Eu expresso minha gratidão ao grupo de pesquisa **CompPet**, que desempenhou um papel crucial na minha formação acadêmica. As atividades de iniciação científica, ensino e extensão realizadas no grupo foram uma fonte constante de aprendizado e colaboração em equipe.

A empresa júnior **Techmob**, onde tive a oportunidade de conhecer o desenvolvimento web, é muito grata por me permitir desenvolver habilidades de liderança e trabalho em grupo, fundamentais para o mercado de trabalho.

Aos meus colegas de classe, agradeço a colaboração e o apoio nos momentos mais difíceis. Sem vocês, tudo teria sido mais difícil.

Agradeço em especial à Faculdade de Ciência da Computação e à Universidade Federal de Uberlândia por fornecerem os recursos e a infraestrutura necessários para a realização desta pesquisa.

Agradeço a todos os familiares, amigos e minha namorada pelo apoio e compreensão demonstrados durante esta jornada acadêmica, que foram fundamentais para superar os desafios enfrentados.

Por último, expresso minha sincera gratidão a todos que, de forma direta ou indireta, contribuíram para este empreendimento. Este projeto não seria possível sem a colaboração e o apoio de cada um de vocês.

Resumo

Este trabalho analisa a relevância do desempenho de aplicações web, demonstrando como otimizações específicas podem melhorar a experiência do usuário. A crescente demanda por websites rápidos e responsivos reforça a necessidade de métodos eficientes que garantam um desempenho satisfatório. Com base em literatura especializada e estudos de caso, como os divulgados pela Google, este estudo propõe uma abordagem prática de otimização utilizando as tecnologias React e Next.js. O método utilizado envolve a criação e a melhoria de um projeto de aplicação web, usando técnicas como renderização no lado do servidor (SSR), *lazy loading* e otimização de imagens. A metodologia é composta por análises quantitativas e qualitativas, utilizando a ferramenta Lighthouse para avaliar as métricas de desempenho pré e pós-otimização, particularmente as Core Web Vitals. Os resultados indicam que as otimizações implementadas contribuíram para melhorias significativas nas métricas de desempenho, destacando-se um aumento na velocidade de carregamento e na interatividade das páginas. Essas melhorias reforçam como alterações técnicas podem impactar positivamente na usabilidade e satisfação do usuário. Este estudo conclui que a aplicação de técnicas de otimização específicas em aplicações web pode levar a melhorias substanciais no desempenho, corroborando com a literatura existente e fornecendo um guia prático para desenvolvedores e designers aprimorarem seus projetos web. As percepções obtidas também contribuem para práticas de desenvolvimento baseadas em evidência, elevando o padrão de desempenho na indústria web.

Palavras-chave: Desempenho de aplicações web, otimização web, React, Next.js, Core Web Vitals.

Lista de ilustrações

Lista de tabelas

Lista de códigos fonte

Lista de abreviaturas e siglas

CDN	*Content Delivery Network*
CSS	*Cascading Style Sheets*
CSR	*Client-Side Rendering*
CLS	*Cumulative Layout Shift*
DOM	*Document Object Model*
FID	*First Input Delay*
HTTP	*HyperText Transfer Protocol*
HTML	*HyperText Markup Language*
INP	*Interaction to Next Paint*
LCP	*Largest Contentful Paint*
PNG	*Portable Network Graphics*
SEO	*Search Engine Optimization*
SSG	*Static Site Generation*
SSR	*Server-Side Rendering*
TTFB	*Time to First Byte*
WEBP	*Web Picture format*

Sumário

1 Introdução

Na era digital atual, o desempenho de uma aplicação web é crucial não apenas para a satisfação do usuário, mas também para a eficácia operacional das empresas. Os usuários esperam respostas rápidas e interações sem atrasos, aumentando a demanda por websites otimizados que carreguem rapidamente e que ofereçam uma experiência de navegação satisfatória.

De acordo com um estudo (DELOITTE, 2024a) realizado pela empresa Deloitte, uma melhoria de 1 milissegundo no carregamento de uma página já é suficiente para alterar o foco e a interação de um cliente, refletindo no ganho financeiro de empresas de diferentes setores do mercado. Apesar da importância conhecida do desempenho web e pela ampla divulgação de casos de sucesso pela plataforma digital (WEB.DEV, 2024a), muitas aplicações ainda enfrentam dificuldades para atingir níveis ótimos de desempenho, especialmente em dispositivos móveis. Neste cenário, como os desenvolvedores podem aprimorar o desempenho de aplicações web de forma efetiva e mensurável?

O objetivo deste trabalho é discutir a relevância do desempenho de um website e apresentar técnicas de otimização para melhorar o desempenho de uma aplicação web. Para alcançar este objetivo, propõe-se o desenvolvimento e a otimização de um projeto de aplicação web empregando as tecnologias React e Next.js. Identificaremos como diversas técnicas de otimização podem ter um impacto positivo no desempenho da página e, consequentemente, na experiência do usuário.

O trabalho está dividido em três partes. O capítulo de Fundamentação Teórica apresenta, inicialmente, conceitos fundamentais sobre a relevância do desempenho de um website para empresas, casos de estudos, além da base de conhecimento necessária para a implementação e análise de desempenho do projeto que será desenvolvido. Em seguida, o capítulo de Desenvolvimento apresentará as aplicações práticas das otimizações, abrangendo diferentes técnicas de otimização e utilizando a ferramenta Lighthouse para avaliar os resultados.

Espera-se que este trabalho demonstre que otimizações específicas levem a melhorias significativas nas métricas Core Web Vitals, fornecendo um caminho seguro para que desenvolvedores e designers melhorarem suas aplicações web. Além disso, os resultados visam contribuir para práticas de desenvolvimento mais informadas e fundamentadas em evidências, elevando o padrão de desenvolvimento web em termos de desempenho e experiência.

2 Fundamentação teórica

O desempenho de aplicações web é reconhecido como um componente crucial para a acessibilidade digital (MOZILLA DEVELOPER NETWORK, 2024). A acessibilidade e o desempenho são interdependentes, pois ambos consideram fatores como o tipo de dispositivo utilizado pelo usuário e a velocidade de sua conexão à internet. Uma página web que carrega lentamente, apresenta comportamento inadequado ou é acessada por uma conexão lenta pode frustrar os usuários, levando-os a abandonar o acesso.

A experiência de navegação otimizada é fundamental não apenas para a satisfação do usuário, mas também para o sucesso comercial de diversas plataformas online. No ambiente de e-commerce, por exemplo, uma navegação fluida e sem interrupções é diretamente ligada ao aumento nas taxas de conversão (PORTENT, 2024). Similarmente, portais de notícias, plataformas de entretenimento e redes sociais beneficiam-se de um desempenho web aprimorado para manter os usuários engajados e satisfeitos (CLOUDFLARE, 2024).

A velocidade de carregamento do conteúdo, a responsividade a interações como cliques e rolagens, são todos indicadores de desempenho que afetam diretamente a percepção do usuário sobre um site. Nesse contexto, o projeto *Core Web Vitals*, o qual será melhor detalhado nesse capítulo, desempenha um papel vital ao estabelecer padrões de desempenho, promover melhores práticas e fornecer métricas precisas para avaliação do desempenho na web (WEB.DEV, 2024g).

2.1 Contextualização

A navegação na internet, conforme discutido no blog Chromium (SAGOO; SULLIVAN; SEKHAR, 2024), é melhor descrita não como eventos isolados de acessos às páginas na internet, mas como uma jornada contínua de interações do usuário. Durante essa jornada, podem ocorrer interrupções, desvios e contratempos que comprometem a qualidade da experiência de navegação. Esses fatores são críticos, ao terem o potencial de diminuir a satisfação do usuário e aumentar o número de abandonos do site.

Neste cenário, a eliminação de barreiras e a minimização de interrupções emergem como elementos essenciais para assegurar uma experiência de usuário fluida e agradável. Diferentes tipos de aplicações, incluindo lojas virtuais, blogs, e páginas institucionais, reconhecem a importância dessa fluidez e frequentemente destinam consideráveis recursos financeiros e humanos para aprimorar a navegação. Além disso, muitas empresas de tecnologia compartilham publicamente suas descobertas e análises (WPOSTATS, 2024; WEB.DEV, 2024b).

2.1.1 Necessidade de Diminuir Carregamentos

As interrupções durante a navegação online são particularmente perceptíveis em operações assíncronas, embora sejam necessárias, não devem comprometer a experiência do usuário. É importante fornecer um feedback visual que mostre o progresso do carregamento, especialmente em páginas com abundância de conteúdo dinâmico, como listas de produtos em lojas virtuais ou plataformas de streaming de vídeo. Nestes contextos, a fluidez é crucial e deve ser uma prioridade (NIELSEN, 2010).

Pesquisas demonstram que atrasos no carregamento impactam negativamente a satisfação do usuário e diminuem a probabilidade de eles retornarem ao site (WO, 2024). Atrasos de apenas dois segundos podem já causar uma impressão negativa, especialmente em sites menos conhecidos. A capacidade do usuário de completar tarefas também diminui, com atrasos que ultrapassam quatro segundos (AKAMAI TECHNOLOGIES, 2009).

Um estudo sobre a navegação em menus aninhados revelou que, à medida que o tempo de carregamento de cada painel aumentava de 0 para 3 segundos e para 9 para 12 segundos, a satisfação do usuário diminui significativamente. A intenção de retornar ao site também diminuía significativamente com o atraso de 12 segundos, enquanto um intervalo de 6 segundos já era percebido como lento pelos usuários. (HOXMEIER; DICESARE, 2000)

Os especialistas sugerem que a velocidade de resposta de um sistema deve ser equivalente aos atrasos encontrados nas interações diárias entre humanos. Em geral, os tempos de resposta devem ser de 1 a 4 segundos para manter a satisfação do usuário (NIELSEN, 2010). Sendo assim, é crucial que os desenvolvedores web se dediquem à otimização dos tempos de carregamento para assegurar que a experiência do usuário se mantenha dentro desses requisitos.

2.1.2 Impacto conhecido no mercado

Como pode ser encontrado em plataformas de divulgação (WPOSTATS, 2024; WEB.DEV, 2024b), o desempenho de uma aplicação web está intrinsecamente ligado ao sucesso financeiro das empresas. Um indicador chave nesse contexto é a **taxa de conversão**, uma métrica essencial que quantifica a porcentagem de visitantes que completam uma ação desejada em um processo comercial, como preencher um formulário, inscrever-se para receber novidades por e-mail, ou realizar uma compra.

Um estudo (DELOITTE, 2024b) extensivo feito pela empresa Deloitte, demonstrou muito bem a relação entre os aprimoramentos no carregamento de sites para dispositivos móveis e seu impacto em empresas de diferentes setores. Perceberam uma correlação direta com o avanço dos usuários no processo de compra e também um impacto positivo no número de visualizações de páginas, taxas de conversão e também no valor médio dos

pedidos em todos os segmentos.

Ainda nessa pesquisa, foram encontradas algumas métricas bem interessantes com base em uma melhoria de 0,1 segundo na velocidade de sites em dispositivos móveis:

- **No varejo,** as vendas aumentaram em 8,48% e o preço médio dos pedidos aumentou em 9,25%.
- **No mercado de viagens**, as conversões aumentaram em 10,1% e o valor médio do pedido aumentou em 1,9%.
- **Em marcas de luxo**, as visualizações de página por sessão aumentaram em 8,6%.

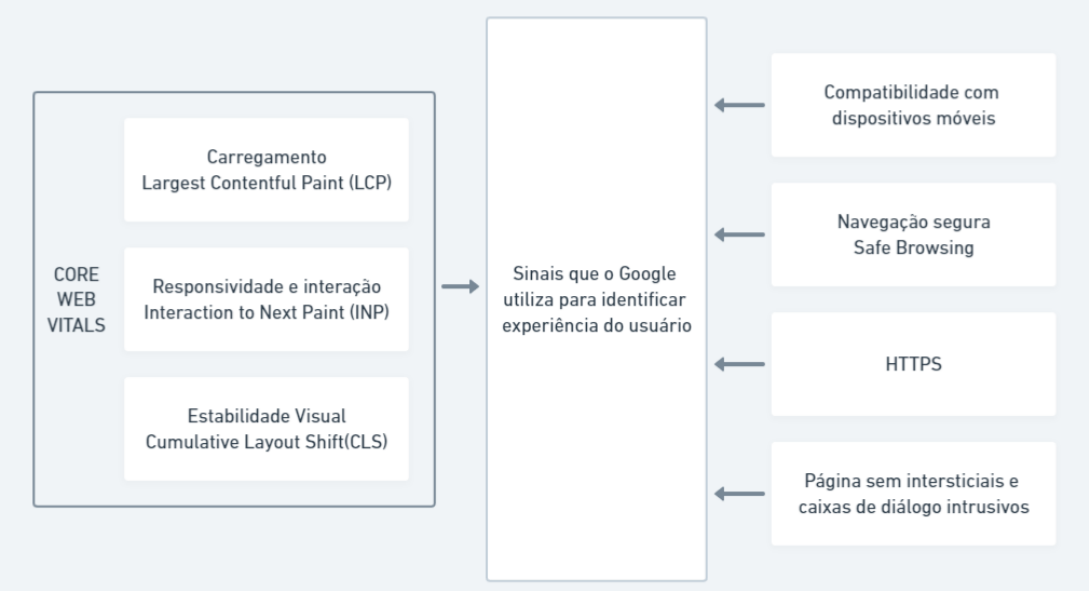
2.1.3 Aparecimento no Google

Historicamente, a velocidade da página tem sido um fator importante para a classificação nos resultados de pesquisa do Google. Inicialmente, em 2010, o Google começou a considerar a velocidade de carregamento das páginas como um critério de ranking para pesquisas em desktops (GOOGLE SEARCH CENTRAL, 2010). Essa política foi ampliada em 2018, quando a velocidade das páginas também começou a influenciar o ranking dos resultados em dispositivos móveis, demonstrando a crescente relevância da experiência móvel na busca online (GOOGLE, 2018)

Em 2021, o Google promoveu uma mudança significativa ao trocar o sinal de velocidade da página pelo sinal de experiência da página no critério para ranqueamento de sites. Este novo sinal engloba uma gama mais ampla de fatores, ilustrados na figura (1), que afetam a experiência do usuário, incluindo os Core Web Vitals (SOUTHERN, 2021). Essa evolução nos critérios de classificação demonstra a importância crescente que o Google tem dado à experiência geral do usuário, além simplesmente avaliar a velocidade de carregamento.

A relevância dessas métricas e o foco do Google na experiência do usuário têm implicações diretas para os desenvolvedores e gestores de website. Para se manterem competitivos e visíveis nos resultados de pesquisa, é crucial que eles implementem práticas de otimização que atendam não só aos aspectos técnicos de desempenho da página, mas também que proporcionem uma experiência de navegação agradável e eficaz para os usuários finais. Portanto, a velocidade da página e a experiência do usuário continuam sendo elementos centrais para qualquer estratégia de SEO (Search Engine Optimization) eficiente e para o sucesso online no cenário digital atual.

Figura 1 – Sinais que afetam a experiência do usuário sendo usadas pelo Google



Fonte: Autoria própria

2.2 Métricas Web Vitals

Como foi discutido na seção 2.1, o desempenho de um website pode ser um fator importante para o sucesso online, influenciando diretamente a experiência do usuário, o engajamento, a satisfação e o posicionamento nos resultados de busca. Um site que carrega rapidamente e opera de maneira fluida, promovendo uma interação positiva, essencial para reter usuários e melhorar conversões.

Para auxiliar desenvolvedores e administradores de sites a monitorar e otimizar essas experiências, o Google desenvolveu o conjunto de indicadores conhecidos como *Web Vitals* (WEB.DEV, 2024h). Estas métricas focam em aspectos cruciais do desempenho do site, incluindo usabilidade e estabilidade visual, e são vitais para entender como os usuários interagem com as páginas web.

2.2.1 Estabilidade das Métricas

As *Web Vitals* estão divididas em duas categorias: métricas candidatas e métricas estáveis. Conforme a documentação oficial do projeto (WEB.DEV, 2024h), uma métrica candidata, após uma avaliação e validação rigorosas, pode evoluir para uma métrica estável, sendo oficialmente reconhecida como parte das *Core Web Vitals*, este processo assegura que apenas métricas comprovadamente confiáveis e úteis sejam integradas como padrões na avaliação de desempenho web.

Métricas estáveis necessitam de suporte contínuo, incluindo correção de erros e

atualizações periódicas em suas definições. O projeto ainda destaca que alterações nas *Core Web Vitals* estáveis são limitadas a no máximo uma vez por ano (WEB.DEV, 2024h) para garantir consistência e previsibilidade.

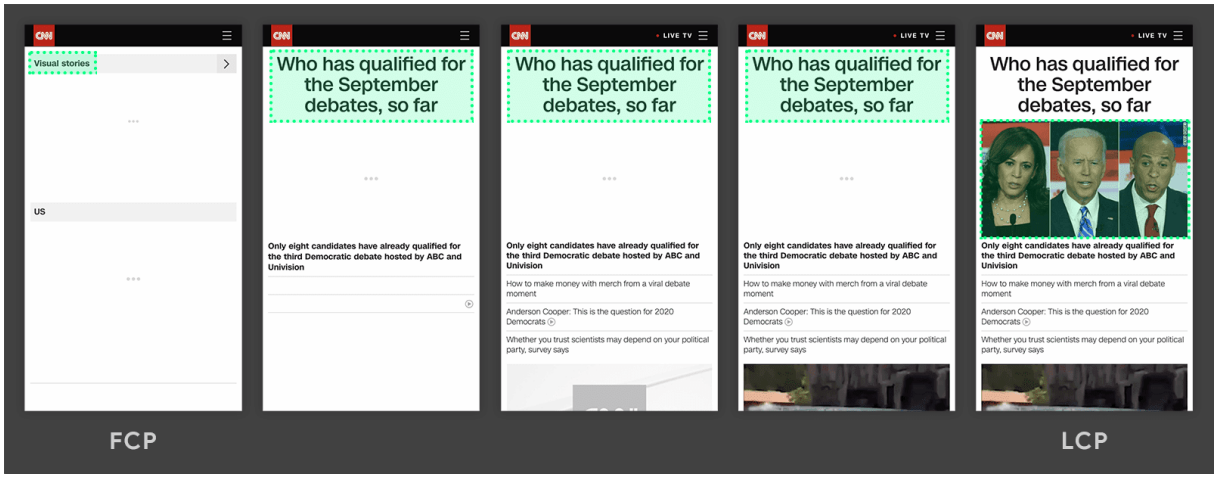
2.2.2 Largest Contentful Paint (LCP)

A métrica LCP é uma das Core Web Vitals (WEB.DEV, 2024e) responsável por medir o tempo necessário para que o maior bloco de conteúdo visível na área de visualização do usuário seja carregado completamente. Este conteúdo pode ser uma imagem de destaque, um bloco de texto volumoso ou outros elementos gráficos significativos.

Um LCP rápido indica que o site fornece um feedback visual útil ao usuário eficientemente, o que é crucial para manter o engajamento desde o início da visitação. A fim de assegurar uma experiência satisfatória ao usuário, o LCP deve ocorrer dentro de 2,5 segundos após o início do carregamento da página.

A figura (2) retrata de maneira visual o momento exato em que o LCP será identificado. No processo de carregamento da página, vários elementos estão sendo carregados, mas o maior elemento com conteúdo significativo, isto é, a imagem presente na parte superior da página representará o conteúdo de maior importância, e a métrica LCP representará o no momento em que o seu carregamento for concluído.

Figura 2 – Exemplificação da métrica LCP



Fonte: Autoria própria

2.2.3 Cumulative Layout Shift (CLS)

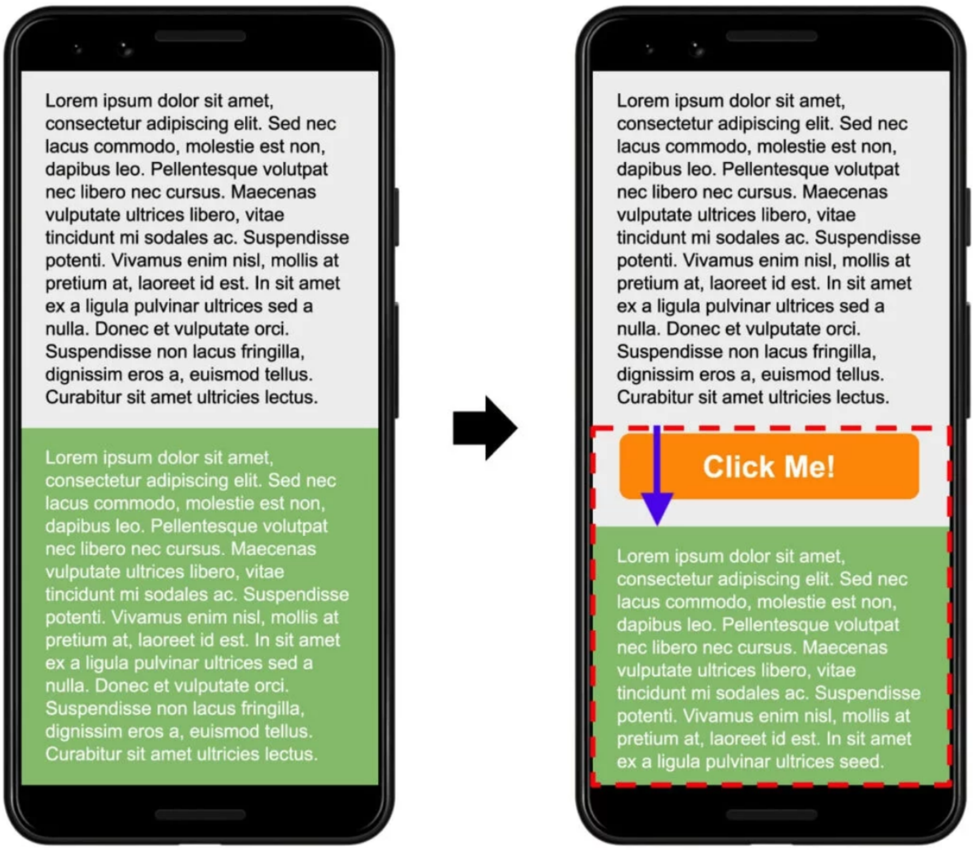
O Cumulative Layout Shift (CLS) é uma métrica importante para medir a estabilidade visual de páginas web (WEB.DEV, 2024c). Ele contabiliza todas as ocorrências de mudanças de layout inesperadas que ocorrem durante o ciclo de vida de uma página. As alterações podem ser causadas por imagens que carregam após o carregamento inicial,

anúncios que se ampliam ou redefinem, ou injeções dinâmicas de conteúdo que modificam a posição de elementos já exibidos na tela.

A métrica CLS é calculada em uma escala que varia de 0 a 1, sendo que 0 não indica nenhuma alteração inesperada de layout e 1 representa a máxima instabilidade visual possível. Otimizar o CLS para manter o valor abaixo de 0,1 é crucial para assegurar que o conteúdo da página não se mova inesperadamente. Isso evita cliques errôneos, melhora a visualização e a interação com o conteúdo, tornando a experiência do usuário mais agradável e segura.

O CLS é calculado a partir da soma de todos os deslocamentos de layout significativos que ocorrem entre duas renderizações consecutivas de quadros. Um deslocamento é considerado quando um elemento visível altera sua posição de um quadro para o outro. Na figura abaixo (3), o problema é evidenciado quando um botão surge inesperadamente na tela, deslocando outros elementos visualmente.

Figura 3 – Exemplificação de deslocamento de conteúdo



Fonte: Autoria própria

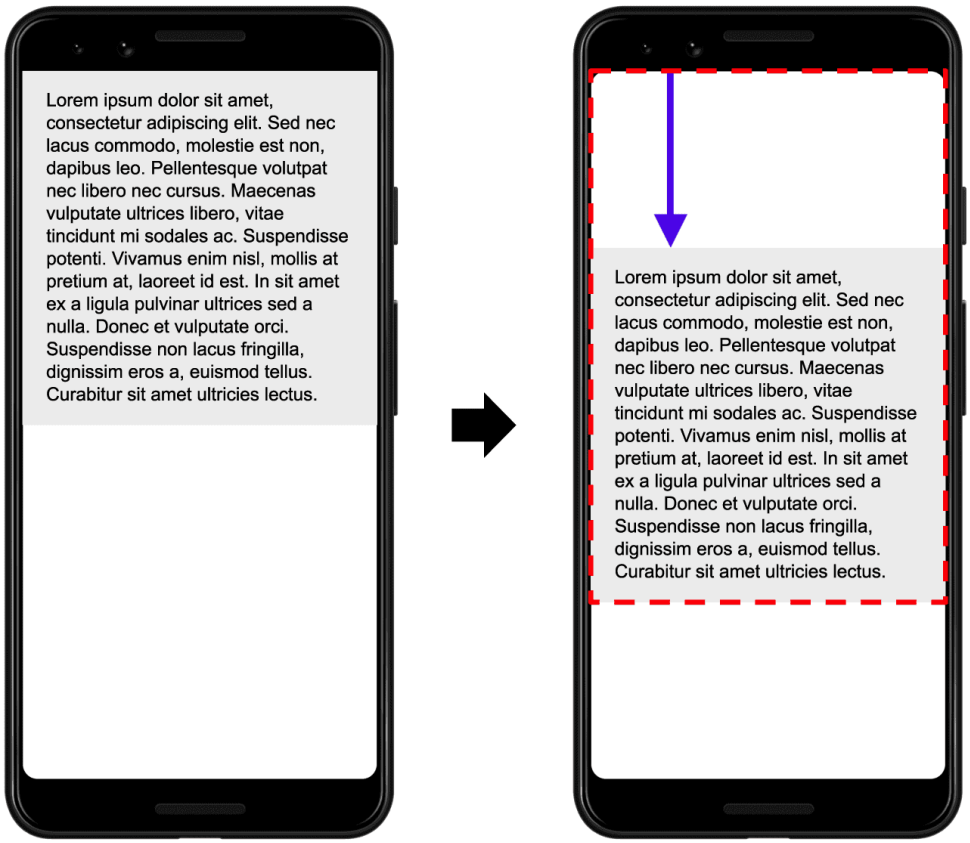
A explicação para o cálculo do CLS, conforme a documentação (WEB.DEV, 2024c), pode ser realizada utilizando as duas principais componentes: a fração de im-

pacto e a fração de distância.

Fração de Impacto: Esta medida avalia como os elementos instáveis afetam a área visível da tela entre dois quadros consecutivos. Para um quadro específico, a fração de impacto é calculada combinando as áreas visíveis de todos os elementos instáveis, tanto na posição anterior quanto na atual, em relação à área total da tela. Por exemplo (4), se um elemento ocupa a metade da tela em um quadro e, logo depois, se desloca para baixo, ocupando 25% a mais, sua área de impacto combinada é de 75% da tela, o que resulta numa fração de impacto de 0,75.

Fração de Distância: Esta componente calcula a distância que os elementos instáveis se movem em relação às dimensões da tela. É determinada pela maior distância horizontal ou vertical que um elemento instável se movimenta, dividida pela maior dimensão da tela (largura ou altura). No exemplo (4) apresentado, um elemento que se move a 25% da altura da tela resulta numa fração de distância de 0,25.

Figura 4 – Exemplificação de deslocamento de conteúdo



Fonte: Retirada de (WEB.DEV, 2024c)

Combinando as duas medidas, o valor de deslocamento de layout para o exemplo seria calculado por 0,75 (fração de impacto) multiplicado por 0,25 (fração de distância),

resultando no valor de CLS igual a 0,1875.

2.2.4 Interaction to Next Paint (INP)

A métrica INP (WEB.DEV, 2024d) quantifica a interatividade de uma página, medindo o tempo entre uma interação do usuário (como um clique ou toque) e a atualização visual subsequente na página que reflete essa interação.

Esta métrica é importante por avaliar a responsividade de um site. Um INP de 200 milissegundos ou menos é considerado ideal, sugerindo que o site responde às ações do usuário de maneira rápida e fluida, proporcionando uma experiência de navegação ágil e satisfatória.

2.2.5 Valores recomendados para cada métrica

A tabela abaixo representa o intervalo de valores recomendados para as métricas do *Core Web Vitals*.

Tabela 1 – Intervalos de Valores Recomendados para Métricas do *Core Web Vitals*

Métrica	**Bom**	**Precisa de Melhoria**	**Ruim**
LCP	<= 2,5 segundos	Entre 2,5 e 4 segundos	> 4 segundos
INP	<= 200 milissegundos	Entre 200 e 500 milissegundos	> 500 milissegundos
CLS	<= 0,1	Entre 0.1 e 0,25	> 0,25

Fonte: Autoria própria

2.3 Ferramenta de Auditoria Lighthouse

Lighthouse (CHROME DEVELOPERS, 2024) é uma ferramenta de auditoria automatizada desenvolvida pelo Google, projetada para ajudar desenvolvedores a otimizar a qualidade das suas páginas web. Fundamentalmente, é utilizada para analisar aplicações web em diversas categorias de desempenho, acessibilidade, práticas recomendadas e SEO. A ferramenta realiza diversas avaliações em uma página web e gera um relatório que identifica oportunidades de aprimoramento para aprimorar a experiência do usuário. No cenário desse projeto, utilizaremos extensivamente a análise da categoria desempenho associadas tanto a dispositivos móveis ou desktops.

A relação dessa ferramenta com os Core Web Vitals é direta: ela é um dos principais meios pelos quais desenvolvedores podem testar e analisar como seus sites estão desempenhando em relação a essas métricas críticas. Ela está disponível nativamente no

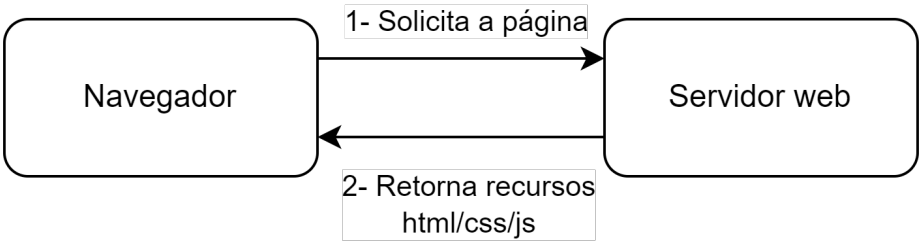
navegador Chrome, sendo utilizada para realizar testes in Lab (WEB.DEV, 2024f) (em laboratório), isto é, testes executados em ambientes controlados.

2.4 Renderização na Web

2.4.1 O que é um servidor web

Um servidor web, segundo a Mozilla (MOZILLA, 2024), pode ser entendido tanto como hardware e software. Software ao ser responsável por controlar o acesso de usuários a arquivos, como um servidor HTTP que entende endereços web e o protocolo HTTP que o navegador entende para visualizar as páginas. Já no lado de hardware, o servidor web é um computador que armazena os documentos HTML, CSS, *JavaScript* e se conecta na internet para trocar informação com outros dispositivos conectados.

Figura 5 – Exemplificação genérica de um modelo de renderização.



Fonte: Autoria própria

A renderização de conteúdo nos navegadores acaba sendo um elemento da arquitetura de uma aplicação, que influencia diretamente na complexidade de configuração deste servidor web, ela pode ser feita de diferentes formas e possui diferentes impactos no desempenho da aplicação e consequentemente, na experiência dos usuários em uma aplicação. A maioria das bibliotecas e *frameworks* web oferecem recursos que possibilita escolher diferentes padrões de renderização como as duas principais:

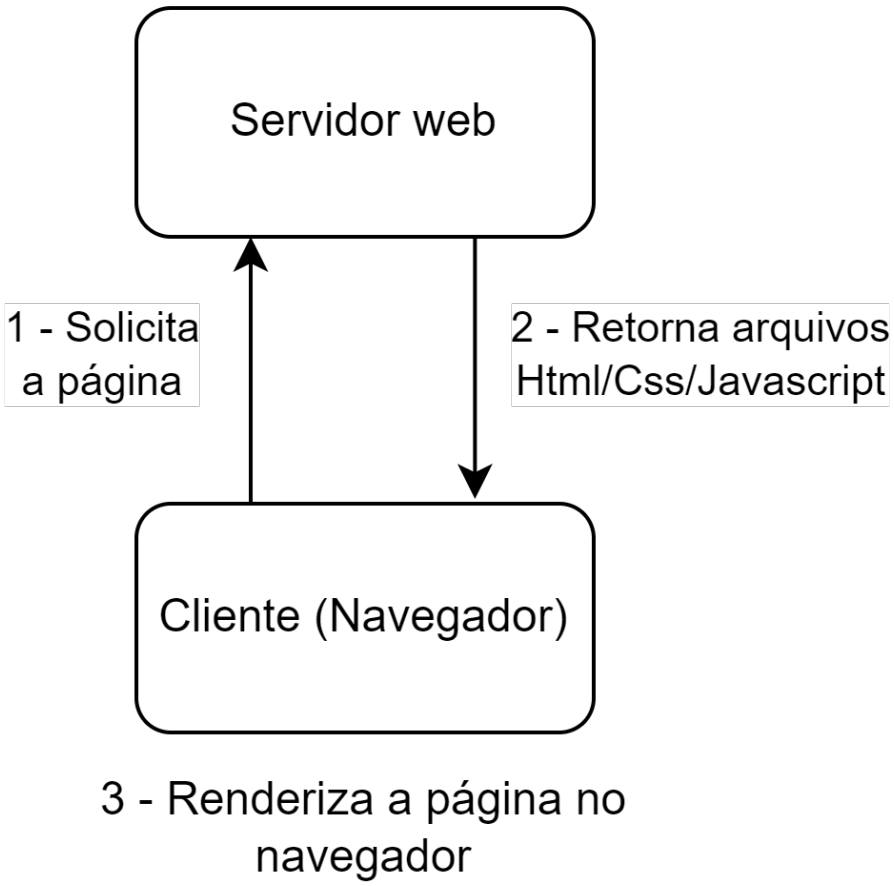
- Renderização no lado do servidor;
- Renderização no lado do cliente;

2.4.2 Client Side Rendering (CSR)

O CSR é um modelo de renderização de páginas web onde a lógica de renderização, incluindo a construção do conteúdo da página em HTML, é executada no navegador do usuário, utilizando *JavaScript*. Em aplicações CSR, o servidor geralmente envia apenas um arquivo HTML vazio com links para *scripts JavaScript* e folhas de estilo, sendo então

processados pelo navegador para renderizar a página. A imagem (6) abaixo ilustra melhor esse processo.

Figura 6 – Exemplo de renderização do lado do cliente.

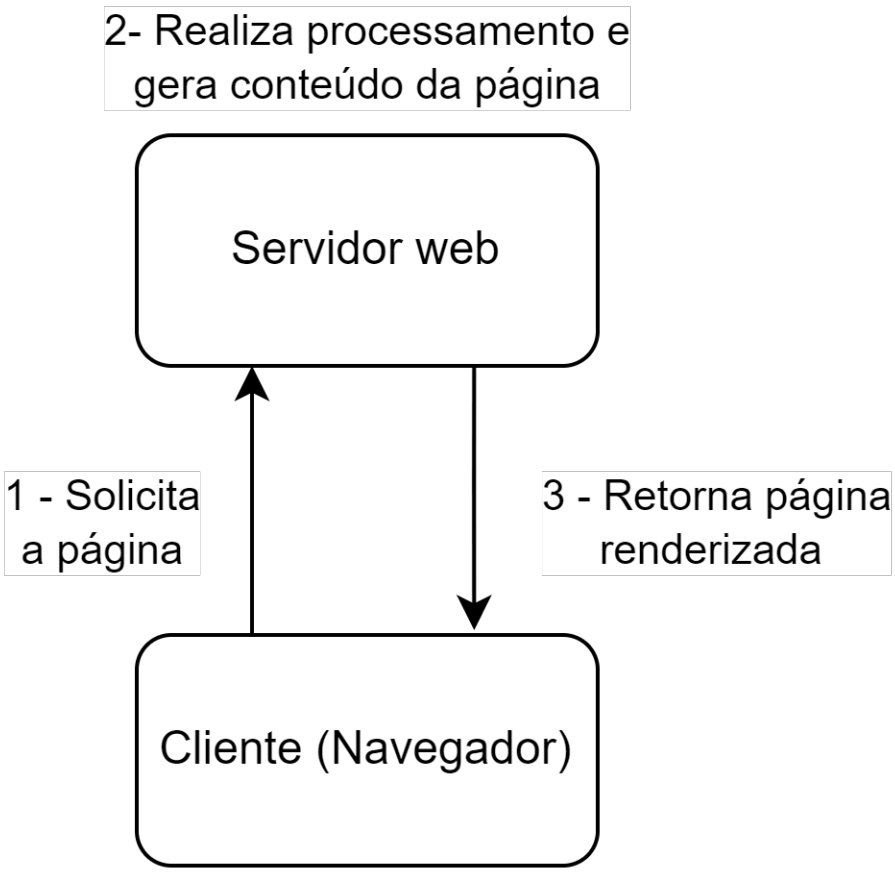


Fonte: Autoria própria

2.4.3 Server Side Rendering (SSR)

No modelo SSR, como exemplificado na figura (7), o HTML completo da página é gerado pelo servidor em resposta a cada solicitação, o que pode melhorar significativamente o tempo de carregamento inicial da tela no navegador e otimizar a página para os mecanismos de busca, que poderão capturar o conteúdo da página com maior facilidade do que no modelo CSR.

Figura 7 – Exemplo de renderização do lado do servidor



Fonte: Autoria própria

2.5 Biblioteca React e Framework Next.js

2.5.1 ReactJS

A biblioteca de JavaScript ReactJS foi criada e mantida pelo Facebook, e agora faz parte da Meta Platforms (FACEBOOK, 2024). Para criar interfaces de usuário (UI), React se destaca no desenvolvimento de aplicações para web por sua eficiência e grande comunidade de desenvolvimento. A popularidade de React pode ser atribuída a vários fatores-chave:

- **Componentes Reutilizáveis**: React encoraja a divisão da interface do usuário em componentes independentes Isso facilita o desenvolvimento, manutenção e teste, permitindo que equipes grandes trabalhem de forma mais eficiente e o código seja reutilizado em diferentes partes da aplicação ou projetos.
- **Abordagem Declarativa**: Ao utilizar uma abordagem declarativa, React torna o

código mais previsível e mais simples de depurar. Um UI declarativo é mais fácil de entender e manipular, o que é especialmente útil em projetos complexos.

- **Virtual DOM**: O React implementa um Virtual DOM (Document Object Model) que otimiza as atualizações no DOM real do navegador, aprimorando significativamente o desempenho das aplicações, especialmente em situações em que ocorrem constantes mudanças dinâmicas nos dados.

Esses recursos tornam o React uma solução altamente eficaz e flexível para o desenvolvimento de interfaces dinâmicas e responsivas, que se adaptam desde projetos menores até grandes aplicações empresariais. Por esse fator, o projeto web desenvolvido por este trabalho utilizará essa biblioteca.

2.5.2 Next.js

O *Next.js* é um *framework* de aplicação web criado pela Vercel, que torna o React mais simplificado e eficiente, adicionando funcionalidades e oferecendo uma estrutura mais opinativa, como a estruturação de pastas e componentes. Entre seus recursos mais notáveis, destacam-se:

- **Renderização do Lado do Servidor (SSR)**: Next.js disponibiliza suporte nativo para SSR, o que pode aprimorar significativamente o tempo de carregamento das páginas e ser benéfico para SEO, já que os motores de busca podem indexar o conteúdo pré-renderizado.
- **Geração de Sites Estáticos (SSG)**: Com Next.js, os desenvolvedores podem gerar páginas estáticas durante o **build time**. Esta abordagem é ideal para sites com conteúdo que não sofre alterações frequentes, oferecendo tempos de carregamento extremamente rápidos.
- **Rotas Baseadas em Arquivos**: Next.js simplifica o roteamento em aplicações React ao associar automaticamente os arquivos na pasta 'pages' a rotas correspondentes. Essa convenção de rotas reduz a complexidade e acelera o desenvolvimento.
- **Otimizações Automáticas**: Inclui otimizações de imagem, pré-fetching de links, e split de código automático. Essas características ajudam a melhorar o desempenho sem a necessidade de configurações complexas por parte do desenvolvedor.

Enquanto *React* fornece a base para a criação de interfaces de usuário com seu robusto sistema de componentes e um gerenciamento de estado eficiente, *Next.js* aprimora essas habilidades com estruturas e funcionalidades que visam a produtividade e o desempenho da aplicação. Juntas, eles formam uma base tecnológica poderosa para

o desenvolvimento moderno de aplicações web, que suporta desde projetos pessoais até grandes aplicações empresariais (NEXT.JS, 2024).

Essa integração entre React e Next.js é indicada especialmente para desenvolvedores que buscam uma solução completa que cubra tanto o desenvolvimento quanto a otimização de aplicações web, conforme recomendado pelos próprios criadores do React (REACT, 2024).

3 Desenvolvimento

Este capítulo detalha a evolução de um projeto de aplicação web construído com as tecnologias React e Next.js. O foco está nas intervenções de otimização realizadas para aprimorar o desempenho da página e experiência de usuário. As estratégias de otimização abordadas incluem o uso de Web Workers para processamento assíncrono, técnicas avançadas de otimização de imagens, implementação de *lazy loading* para um carregamento de conteúdo mais eficiente, e, finalmente, a aplicação de técnicas de renderização no lado do servidor.

3.1 Introdução

O objetivo deste projeto é desenvolver uma aplicação composta por cinco versões distintas de uma mesma página. A primeira versão apresentará problemas de desempenho e usabilidade, enquanto nas versões subsequentes, técnicas de otimização serão aplicadas e analisadas de maneira incremental. Esta abordagem permitirá uma análise clara e isolada do impacto de cada técnica de otimização, facilitando a auditoria de desempenho.

Para garantir uma avaliação objetiva e precisa do impacto dessas otimizações, ao final da implementação de cada versão, uma auditoria de desempenho será realizada utilizando a ferramenta Lighthouse, já mencionada em capítulos anteriores. A auditoria será executada via aba anônima do navegador para eliminar interferências externas e garantir a consistência dos resultados. O foco principal será avaliar as métricas do Core Web Vitals e o tempo de carregamento da página, proporcionando uma análise detalhada de como cada otimização afeta o desempenho percebido e medido da aplicação.

Tendo compreendido as funcionalidades e vantagens do *Next.js*, agora nos voltamos para a aplicação prática dessas tecnologias na construção de um projeto web inovador, detalhado no próximo capítulo. Este capítulo abordará a evolução de uma aplicação web desenvolvida com React e Next.js, focando especialmente nas intervenções de otimização que visam melhorar o desempenho e a experiência do usuário. Serão exploradas estratégias avançadas como o uso de Web Workers para processamento assíncrono, otimização de imagens, implementação de *lazy loading*, e técnicas de renderização do lado do servidor. Este estudo de caso não apenas aplicará os conceitos discutidos anteriormente, mas também proporcionará uma análise detalhada e incremental do impacto de cada técnica de otimização, utilizando a ferramenta Lighthouse para auditorias de desempenho. Este projeto exemplificará a aplicação prática de Next.js e React em um cenário real, destacando como essas tecnologias podem ser efetivamente implementadas para atender às exigências modernas de desenvolvimento web.

3.2 Descrição dos Componentes do Projeto

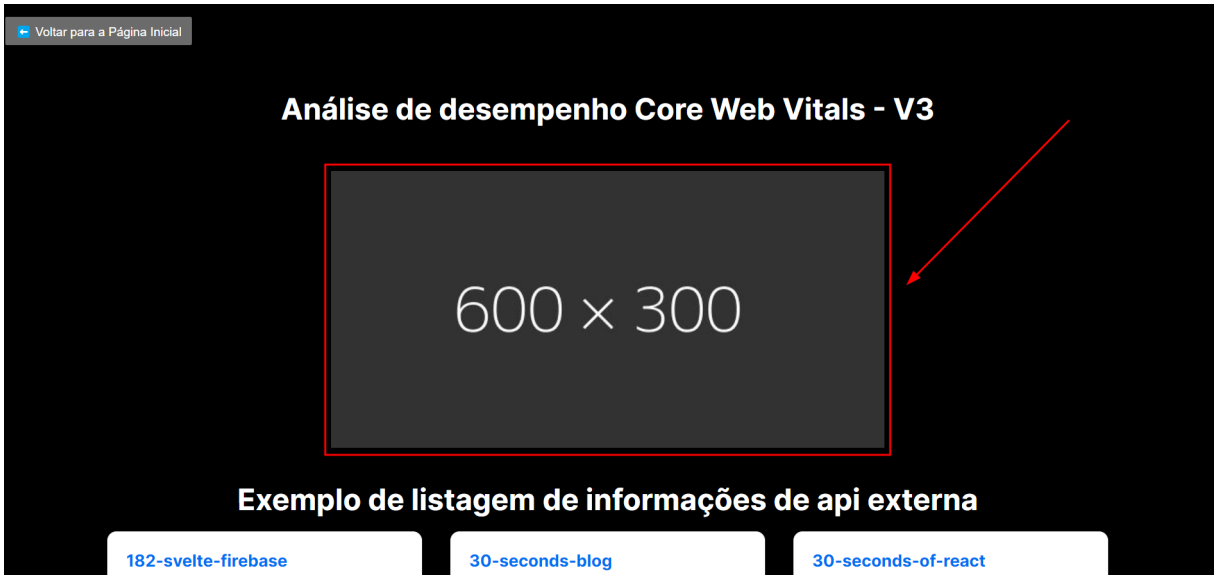
Esta seção apresentará uma descrição detalhada de cada componente e seção da página web, bem como seus códigos-fonte iniciais. Cada componente é crucial para a interface do usuário, e vão fornecer desafios únicos para a otimização de desempenho de cada uma das versões do site apresentadas posteriormente.

3.2.1 Componente ImagemComAtraso

O componente **ImagemComAtraso** simula uma situação comum em que imagens grandes demoram para ser carregadas, impactando a experiência do usuário. Este componente utiliza um timer para atrasar a exibição de uma imagem, representando os desafios de carregar conteúdo visual pesado.

A figura (8) apresentada abaixo demonstra como este componente é renderizado no navegador.

Figura 8 – Componente ImagemComAtraso destacado em vermelho



Fonte: Autoria própria

O código React referente ao componente também está disponível abaixo:

Código 3.1 – Código-fonte do Componente Imagem com Atraso

```
1  function ImagemComAtraso() {
2    const [mostrarImagem, setMostrarImagem] = useState(false);
3
4    useEffect(() => {
5      const temporizador = setTimeout(() => {
6        setMostrarImagem(true);
7      }, 2000);
8
```

```
9       return () => clearTimeout(timer);
10    }, []);
11
12    return (
13      <div>
14        {mostrarImagem ? (
15          <img src="https://dummyimage.com/600x300/323232/fff.jpg" height={300}
      width={600} style={{margin: "1rem 0 2rem 0"}} />
16        ) : (
17          <div>Carregando imagem...</div>
18        )}
19      </div>
20    );
21  }
22
23  export default ImagemComAtraso;
```

Explicação do Código: Este componente define um temporizador que atrasa a exibição de uma imagem por dois segundos, simulando um carregamento tardio. Ele também é renderizado no lado do cliente, por conta da diretiva "use client".

Implicações para o Desempenho: Ao adiar propositadamente o carregamento de uma imagem grande sem reservar previamente um espaço para a imagem na tela, o componente **ImagemComAtraso** deve impactar negativamente o valor da métrica CLS, dado que por conta de seu comportamento, haverá deslocamento de conteúdo. Além disso, como esse componente será renderizado na porção superior da página, a métrica LCP também será afetada.

3.2.2 Componente RepositoriosGithub

O componente **RepositoriosGithub** é responsável por listar informações que serão buscadas de repositórios existentes na plataforma Github. O objetivo desse componente é ilustrar uma situação comum em aplicações web, que é a busca por informações externas para apresentação na página web. Esse processamento assíncrono pode ser realizado tanto pelo próprio navegador quanto pelo servidor web.

A figura (9) abaixo demonstra como este componente será renderizado no navegador.

Figura 9 – Componente RepositoriosGithub destacado em vermelho

Fonte: Autoria própria

O código React referente ao componente também está disponível abaixo:

Código 3.2 – Código-fonte do Componente RepositoriosGithub

```
1  const RepositoriosGithub = () => {
2    const [repositorios, setRepositorios] = useState([]);
3    const [carregando, setCarregando] = useState(true);
4
5    useEffect(() => {
6      fetch('https://api.github.com/users/fziliotti/repos')
7        .then(response => response.json())
8        .then(data => {
9          setRepositorios(data);
10         setCarregando(false);
11       })
12       .catch(erro => {
13         console.error("Erro ao buscar repositorios", erro);
14         setCarregando(false);
15       });
16   }, []);
17
18   if (carregando) {
19     return <div className={styles.loader}>Carregando repositorios...</div>;
20   }
21
22   return (
23     <div className={styles.grid}>
24       {repositorios.map(repo => (
25         <div key={repo.id} className={styles.card}>
26           <a href={repo.html_url} target="_blank" rel="noopener noreferrer"
     className={styles.link}>
27             <div className={styles.title}>{repo.name}</div>
28           </a>
```

```
29            <p className={styles.description}>{repo.description || 'Nenhuma
    descricao disponivel.'}</p>
30          </div>
31        ))}
32      </div>
33    );
34  };
35
36  export default RepositoriosGithub;
```

Explicação do Código: O código 3.2 em questão é responsável por listar os repositórios retornados pela chamada a API. Como ele é renderizado no lado do cliente, o estado 'carregando' é usado para controlar a exibição de um indicador de carregamento enquanto os dados não são recebidos.

Implicações para o Desempenho: As chamadas de API podem causar uma longa espera, especialmente se a API responder de forma lenta ou o volume de dados for grande. Gerenciar bem o carregamento e usar técnicas como cache de dados ou SSR podem melhorar a resposta do componente e a experiência do usuário.

3.2.3 Componente Galeria de Imagens

O componente **GaleriaDeImagens** por sua vez, será responsável por exibir uma coleção de imagens em alta resolução, mas que de maneira proposital, estará prejudicando o desempenho da aplicação, uma vez que quanto maior a quantidade de bytes das imagens, mais tempo o site demorará para carregar e renderizar no navegador do usuário.

A figura (10) apresentada abaixo demonstra como este componente será apresentado ao usuário.

Figura 10 – Componente GaleriaDeImagens destacado em vermelho

Fonte: Autoria própria

O código React referente ao componente também está disponível abaixo:

Código 3.3 – Código-fonte do Componente Galeria de Imagens

```
const GaleriaDeImagens = () => {
  const images = [
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
    'https://dummyimage.com/9000x3600/F2F230/000.jpg',
  ];
  return (
    <div className={styles.gallery}>
      {images.map((image, index) => (
        <div key={index} className={styles.imageContainer}>
          <img src={image} alt={`Imagem ${index + 1}`} className={styles.image}
  />
        </div>
      ))}
    </div>
  );
};

export default GaleriaDeImagens;
```

Explicação do Código: Este componente usa um array para armazenar URLs de imagens e o método `map` do JavaScript para iterar este array, criando um novo elemento `div` com uma `img` para cada imagem.

Implicações para o desempenho: Carregar muitas imagens simultaneamente pode afetar negativamente o desempenho da página, aumentando o tempo de carregamento e consumindo significativa largura de banda. Implementar técnicas como lazy loading ou o uso de formatos de imagem mais eficientes, como WebP, podem ajudar a mitigar esses impactos, melhorando a experiência do usuário e a eficiência do carregamento. Além disso, escolher uma resolução apropriada e minificar a imagem, podem ser processos importantes.

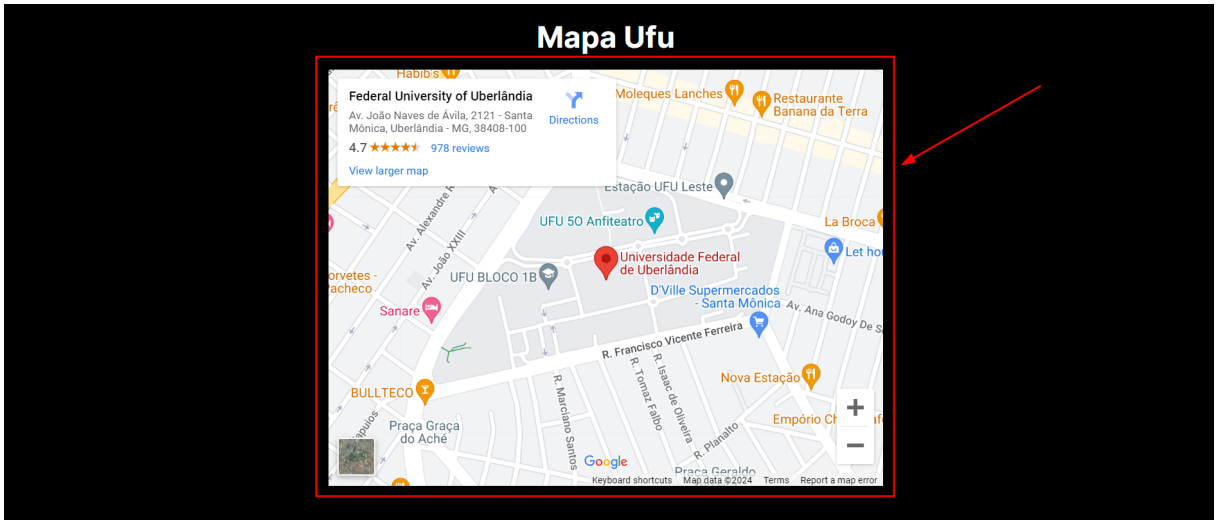
3.2.4 Componente MapaGoogle

O **MapaGoogle** integra um mapa interativo do Google Maps na aplicação por meio de um `iframe`. Este elemento é frequente em diversas aplicações web que requerem

recursos de mapeamento para aprimorar a experiência do usuário, fornecendo direções, localizações de serviços ou imagens geográficas.

A figura (10) apresentada abaixo demonstra como este componente é renderizado no navegador.

Figura 11 – Componente MapaGoogle destacado em vermelho.



Fonte: Autoria própria

O código React referente ao componente também está disponível abaixo:

Código 3.4 – Código-fonte do Componente MapaGoogle

```
export const MapaGoogle = () => {
    return (
        <iframe
         src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3774
    .2918218912764!2d-48.260751224798604!3d-18.918472982253935!2m3!1f0!2f0!3f0!3
    m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x94a44574c062bad3%3A0x656d242f316ee167!2
    sFederal%20University%20of%20Uberl%C3%A2ndia!5e0!3m2!1sen!2sbr!4
    v1714012397399!5m2!1sen!2sbr"
         width="600"
         height="450"
         style={{ border: 0, margin: "1rem 0"}}
         referrerPolicy="no-referrer-when-downgrade">
        </iframe>
    );
}
```

Explicação do código: Este código utiliza React para renderizar um `iframe` que carrega um mapa do Google Maps. O `src` do `iframe` contém uma URL complexa que configura o mapa para exibir uma localização específica, neste caso, a Universidade Federal de Uberlândia, com parâmetros de visualização e interatividade definidos.

Implicações para o desempenho: Mapas interativos são geralmente pesados e carregam muitos recursos. Se um mapa é carregado de imediato como parte do carrega-

mento inicial da página, isso aumentará o tempo para que a página se torne interativa. O **lazy loading** permite que o mapa seja carregado somente quando necessário, isto é, quando o usuário desliza até a parte da página no qual o mapa está exposto. Isso reduz o tempo de carregamento inicial percebido, melhorando a experiência do usuário.

3.2.5 Componente ScriptPesado

O **ScriptPesado** retrata um cenário comum em aplicações web onde um script de execução longa pode impactar significativamente o desempenho do navegador. Este tipo de script é frequentemente encontrado em processamentos de dados intensivos ou operações complexas realizadas no lado do cliente.

Código 3.5 – Código-fonte do script que simula um processamento demorado

```
1  console.log("Iniciando processamento de 5 segundos...");
2
3  function processamentoDemorado() {
4    const inicio = Date.now();
5    let agora = inicio;
6
7    while (agora - inicio < 5000) {
8      agora = Date.now();
9    }
10
11   console.log("Processamento concluido apos 5 segundos.");
12 }
13
14 processamentoDemorado();
```

Explicação do Código: A função **processamentoDemorado** é definida para realizar um loop que simula uma operação de longa duração. O loop utiliza a função ‘Date.now()‘ para obter o tempo atual repetidamente até que cinco segundos se passem desde o início da execução. Após terminar o processo, uma mensagem é exibida para mostrar que o processo foi concluído.

Implicações para o Desempenho:

A utilização de scripts como este torna a experiência do usuário ruim, uma vez que a interface permanece irresponsiva. A inclusão deste componente tem o objetivo de demonstrar as consequências de não utilizar práticas como Web Workers ou técnicas assíncronas para gerenciar tarefas pesadas.

3.3 Versão 1: Estado Inicial dos Componentes

Nessa primeira versão, os componentes foram integrados sem otimizações específicas, utilizando renderização no lado do cliente. Esta abordagem inicial ajuda a estabelecer uma linha de base para o desempenho do site, destacando áreas críticas que necessitam

de melhorias nas versões subsequentes. A lista de componentes, em resumo do que foi explicado na seção anterior, incluirão:

- **ScriptPesado**: O script externo, o qual simula um processamento que demora 5segundos para ser concluído, será carregado na página, afetando potencialmente o tempo de interatividade.
- **ImagemcomAtraso**: Demonstração do impacto nas métricas LCP e CLS, pelo fato do componente simular uma imagem com atraso intencional de 2 segundos.
- **RepositóriosGithub**: Carrega dados de uma API externa, exemplificando o comportamento de chamadas de rede.
- **GaleriadeImagens**: Exibe várias imagens grandes e desproporcionais, ideal para testar técnicas de otimização de imagens e lazy loading.
- **MapaGoogle**: Componente responsável por renderizar o mapa do Google dentro de um iframe, mostrando como conteúdo externo e pesado pode afetar o desempenho.

3.3.1 Código da página V1

Os componentes da seção anterior serão incluídos na estrutura da página V1 3.6 juntamente com um botão de retorno e uma área de rodapé, proporcionando uma visão completa da interface do usuário na sua forma mais elementar.

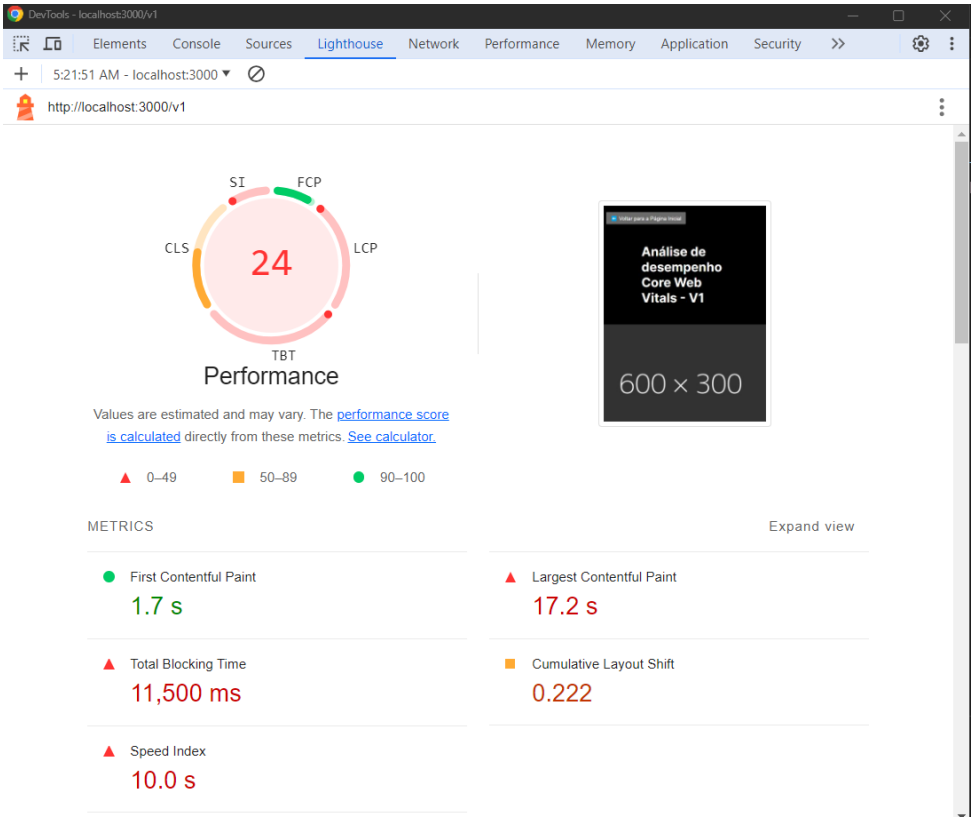
Código 3.6 – Código-fonte da Versão 1 da aplicação

```
export default function V1() {
  return (
      <main className={styles.main}>
        <Script src="/scripts/script-pesado1.js"/>
        <BotaoVoltar />

        <h1 className={styles.title}>Analise de desempenho Core Web Vitals - V1
   </h1>
        <ImagemComAtraso />

        <h1>Exemplo de listagem de informacoes de api externa</h1>
        <RepositoriosGithub/>

        <h1>Exemplo de galeria de imagens</h1>
        <GaleriaDeImagens />

        <h1>Mapa Ufu</h1>
        <MapaGoogle/>

        <footer className={styles.footer}>Desenvolvido por Fabricio Ziliotti -
   FACOM UFU</footer>
      </main>
  );
}
```

3.3.2 Resultados das Auditorias

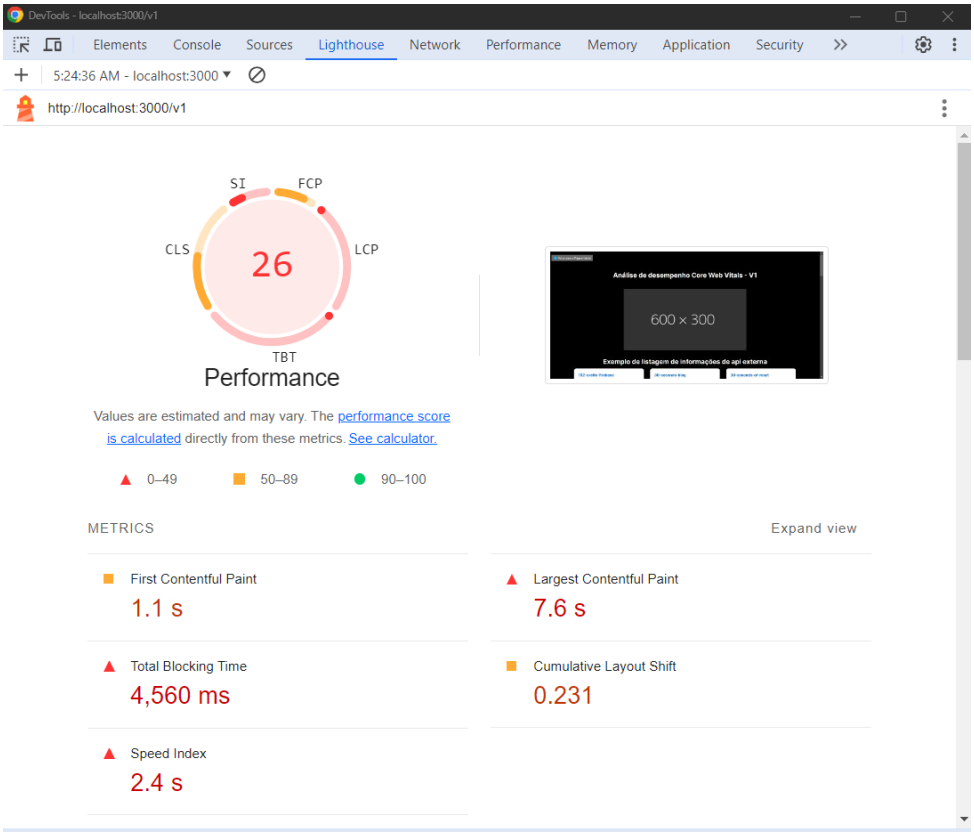
A seguir, estão os resultados das auditorias realizadas com a ferramenta **lighthouse**.Tanto para dispositivos móveis(figura 12) quanto para dispositivo *desktop*(figura 13).

Figura 12 – Resultado auditoria para versão 1 em dispositivo móvel



Fonte: Autoria própria

Figura 13 – Resultado auditoria para versão 1 em dispositivo móvel



Fonte: Autoria própria

3.4 Versão 2: Otimização com Web Worker

Na segunda versão do projeto, será demonstrado uma forma de resolver o problema de bloqueio de renderização pela questão de scripts externos contendo longos processamentos.

Para alcançar o objetivo, o componente ScriptPesado presente no código figura 3.5 não será mais utilizado dentro da página. A ideia agora será criar um novo Web Worker, que é uma funcionalidade do JavaScript que permite a execução de scripts em segundo plano, em uma thread separada da thread principal do navegador. Isso possibilita realizar operações pesadas ou de longa duração sem bloquear ou afetar o desempenho da página.

A implementação do novo Worker (3.7) é demonstrada abaixo:

Código 3.7 – Código-fonte worker-pesado.js

```
1 self.onmessage = function (e) {
2   console.log('Mensagem recebida no worker:', e.data);
3   let result = realizaLongoProcessamento(e.data);
4   self.postMessage(result);
5 };
6
7 function realizaLongoProcessamento(data) {
```

```
8   console.log(data)
9   const inicio = Date.now();
10  let agora = inicio;
11
12  // Loop que dura aproximadamente 5 segundos
13  while (agora - inicio < 5000) {
14    agora = Date.now();
15  }
16
17  return "Processamento concluido ap s 5 segundos.";
18 }
```

O script é destinado a ser executado como um Web Worker. A comunicação entre o Worker e o *thread* principal (geralmente, a interface do usuário) é feita por mensagens. Dessa maneira, o processamento não irá mais bloquear a renderização no navegador, desbloqueando a thread principal do navegador e, consequentemente, melhorando a experiência do usuário da aplicação.

3.4.1 Código da página V2

Com o novo script, no componente que corresponde à Versão 2 do projeto (3.8), o antigo script-pesado foi removido e uma nova lógica será adicionada para criar o novo Worker e realizar a comunicação entre a página e o worker, no momento em que o processamento for finalizado, a página vai receber a informação e fazer o log no console do navegador.

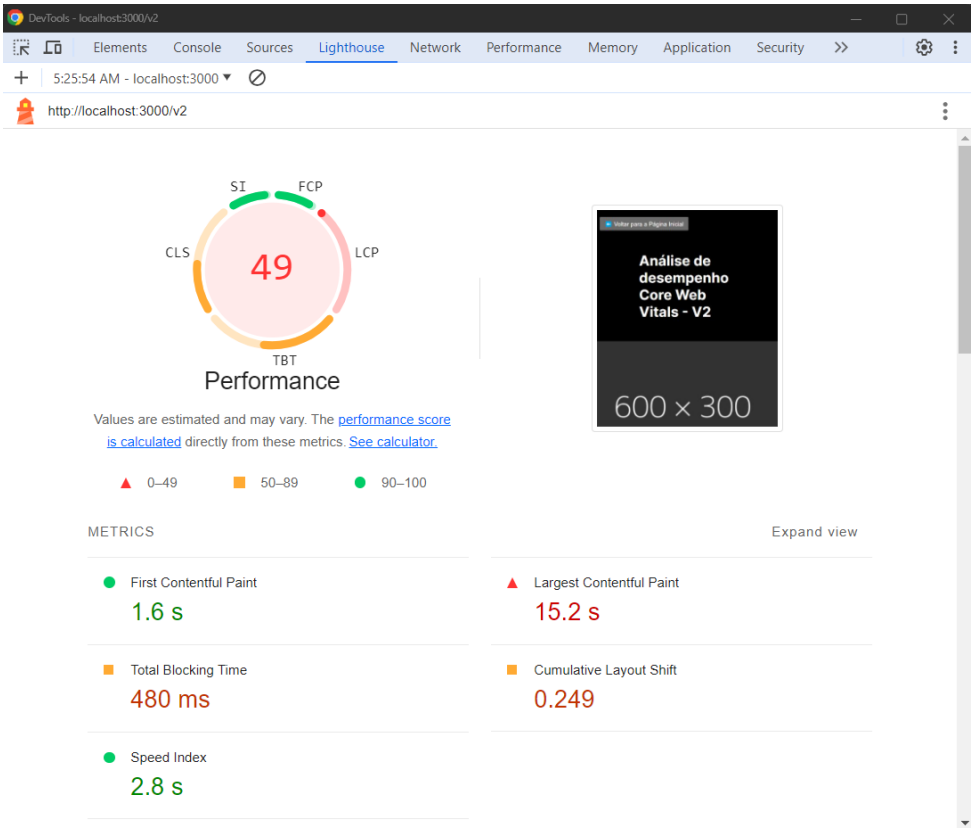
Código 3.8 – Código-fonte da Versão 2 da aplicação

```
1  export default function V2() {
2    % Logica responsavel por iniciar o Web Worker assim que a p gina for
       renderizada no cliente pela primeira vez.
3    useEffect(() => {
4      if (typeof window === 'object' && window.Worker) {
5        const worker = new Worker('/scripts/worker-pesado1.js');
6        worker.onmessage = function (e) {
7          console.log('Mensagem recebida do worker:', e.data);
8        };
9        worker.postMessage('Inicie a tarefa pesada');
10       return () => worker.terminate(); // Limpeza ao desmontar
11     }
12   }, []);
13
14   return (
15     <>
16       {/* <Script src="/scripts/script-pesado1.js"/> O processamento agora sera
       realizado no Worker*/}
17
18       % Omitindo o codigo da versao 1, que permanecera o mesmo.
19     </>
20   );
21 }
```

3.4.2 Resultados das Auditorias

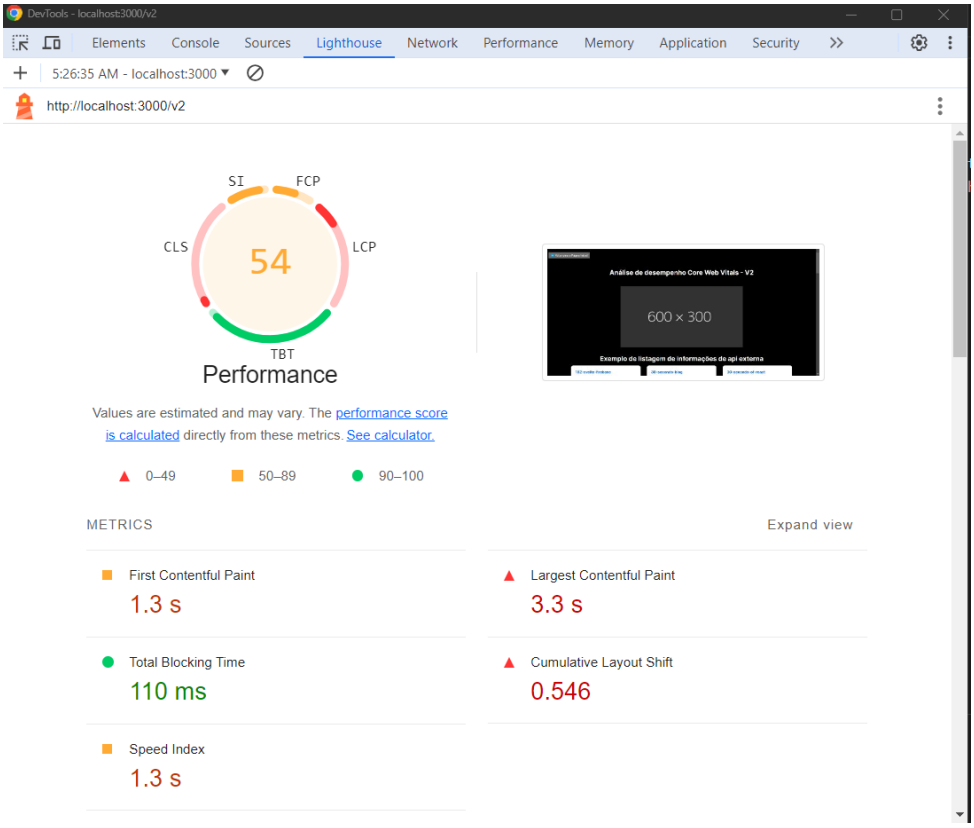
A seguir, estão os resultados das auditorias realizadas com a ferramenta **lighthouse**.Tanto para dispositivos móveis(figura 14) quanto para dispositivo *desktop*(figura 15).

Figura 14 – Resultado auditoria para versão 2 em dispositivo móvel



Fonte: Autoria própria

Figura 15 – Resultado auditoria para versão 2 em dispositivo móvel



Fonte: Autoria própria

Como esperado, o **score** obtido pela ferramenta apresentou uma melhoria significativa em relação aos resultados da primeira versão. Isso foi particularmente notável na métrica de Tempo Total de Bloqueio (TBT), que, em análises realizadas em laboratório, serve como um excelente substituto para a métrica de Interação até a Próxima Pintura (INP). A métrica TBT quantifica o período em que a thread principal fica bloqueada durante o carregamento da página, impactando diretamente na responsividade do site.

3.5 Versão 3: Otimização de Imagens e Melhoria do CLS

Nesta terceira versão do projeto, será demonstrado a técnica de "skeleton screens"(ou "telas esqueleto") é um método utilizado para aprimorar a experiência do usuário durante o carregamento de conteúdo na web, especificamente para melhorar a métrica chamada Cumulative Layout Shift (CLS).

No novo código do componente ImagemComAtraso (3.9), pode-se perceber que existe um novo atributo de style no elemento que contem o texto "Carregando imagem...", a ideia é preencher o espaço onde as imagens irão carregar com um esqueleto, mantendo o layout estável e melhorando a percepção de desempenho pelo usuário.

Código 3.9 – Código-fonte da componente ImagemComAtraso depois da otimização

```
function ImagemComAtrasoOtimizada() {
  const [mostrarImagem, setMostrarImagem] = useState(false);
  useEffect(() => {
    const timer = setTimeout(() => {
      setMostrarImagem(true);
    }, 2000); // Atraso de 2000 milissegundos (2 segundos)
    return () => clearTimeout(timer);
  }, []);

  return (
    <div>
      {mostrarImagem ? (
        <img src="https://dummyimage.com/600x300/323232/fff.jpg" height={300}
    width={600} style={{margin: "1rem 0 2rem 0"}} />
      ) : (
        <div
          style={{
            // Os estilos abaixo sao responsaveis por eliminar o deslocamento de
     conteudo da pagina e melhorar a metrica CLS.
            margin: "1rem 0 2rem 0", height:"300px", width: "600px",
           ...EstilosTextoCarregando
           }}>
          Carregando imagem...
        </div>
      )}
    </div>
  );
}

export default ImagemComAtrasoOtimizada;
```

Com a nova implementação, será criado o componente **ImagemComAtrasoOtimizada**, que será responsável por manter a estabilidade visual da página, evitando deslocamentos de conteúdo. O código da terceira versão do projeto (3.10) deverá importar esse novo componente.

3.5.1 Código da página V3

Após as otimizações nas imagens e no componente ImagemComAtrasom a página V4 ficará com o seguinte código:

Código 3.10 – Código-fonte da Versão 3 da aplicação

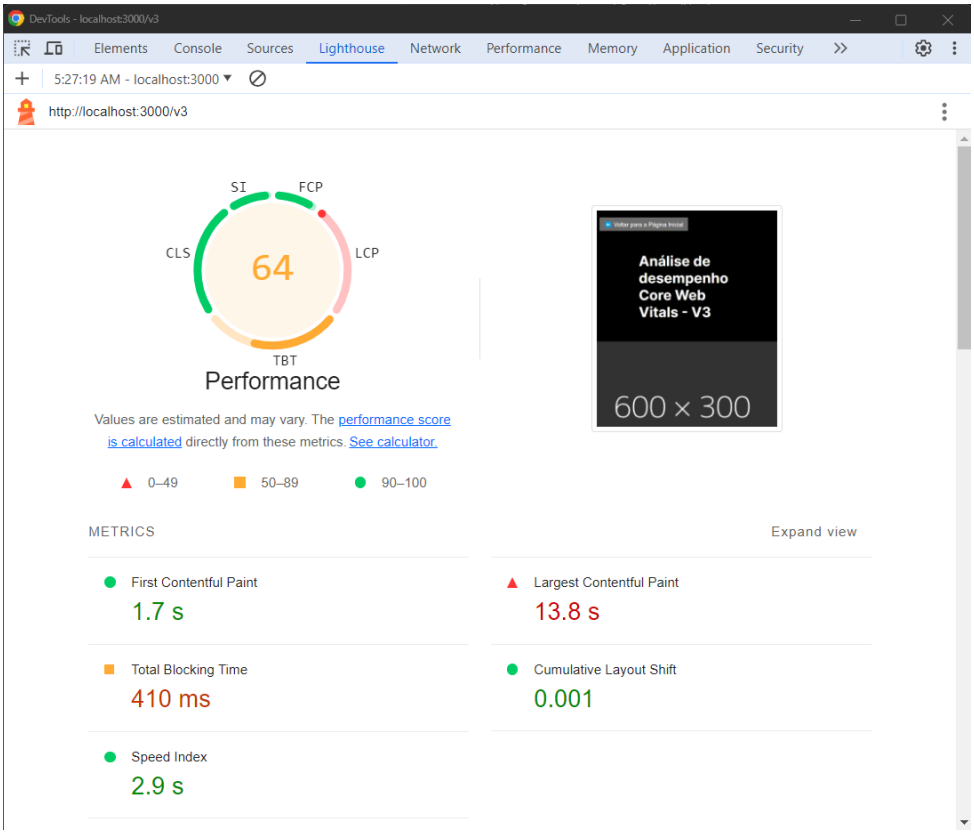
```
export default function V3() {
  useEffect(() => {
    if (typeof window === 'object' && window.Worker) {
      const worker = new Worker('/scripts/worker-pesado1.js');
      worker.onmessage = function (e) {
        console.log('Mensagem recebida do worker:', e.data);
      };
      worker.postMessage('Inicie a tarefa pesada');
      return () => worker.terminate();
```

```
10      }
11    }, []);
12
13    return (
14      <>
15        <main className={styles.main}>
16          <BotaoVoltar />
17
18          <h1 className={styles.title}>
19            An lise de desempenho Core Web Vitals - V3
20          </h1>
21
22          <ImagemComAtrasoOtimizada /> // Alterac o nessa terceira versao,
      otimizando CLS
23
24          <h1>Exemplo de listagem de informa  es de api externa</h1>
25          <RepositoriosGithub/>
26
27          <h1>Exemplo de galeria de imagens</h1>
28          <GaleriaDeImagens />
29
30          <h1>Mapa Ufu</h1>
31          <MapaGoogle/>
32
33          <footer className={styles.footer}>
34            Desenvolvido por Fabricio Ziliotti - FACOM UFU
35          </footer>
36        </main>
37      </>
38    );
39  }
```

3.5.2 Resultados das Auditorias

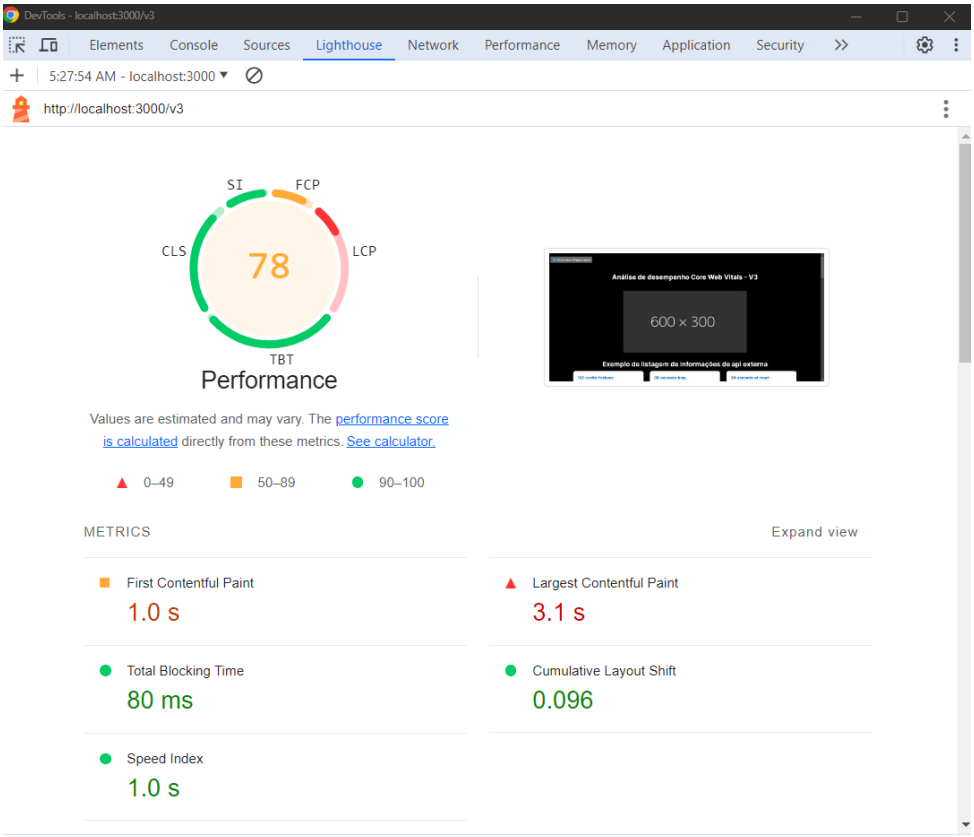
A seguir, estão os resultados das auditorias realizadas com a ferramenta **lighthouse**.Tanto para dispositivos móveis(figura 16) quanto para dispositivo *desktop*(figura 17).

Figura 16 – Resultado auditoria para versão 3 em dispositivo móvel



Fonte: Autoria própria

Figura 17 – Resultado auditoria para versão 3 em dispositivo móvel



Fonte: Autoria própria

3.6 Versão 4: Lazy Loading e Otimização de Imagens

Na quarta versão do projeto, implementamos técnicas avançadas para otimizar o carregamento de componentes pesados, como mapas e galerias de imagens, visando melhorar significativamente o desempenho e a experiência do usuário.

3.6.1 Implementação de Lazy Loading no Iframe do Mapa do Google

Introduzimos o atributo `lazy` no `iframe` que contém o Mapa do Google (**??**). Este ajuste assegura que o mapa só será carregado quando o usuário rolar a página até a galeria de imagens.

Código 3.11 – Código-fonte do componente MapaGoogle após otimização

```
1 export const MapaGoogleLazyLoading = () => {
2   return (
3     <iframe
4       src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3774
    .2918218912764!2d-48.260751224798604!3d-      18.918472982253935!2m3!1f0!2f0
    !3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x94a44574c062bad3%3
    A0x656d242f316ee167!2sFederal%20University%20of%20Uberl%C3%A2ndia!5e0!3m2!1
    sen!2sbr!4v1714012397399!5m2!1sen!2sbr"
```

```
5         width="600"
6         height="450"
7         style={{ border: 0, margin: "1rem 0"}}
8         loading="lazy" //Novo atributo para aplicar tecnica de lazy loading de
      maneira nativa na web
9         referrerPolicy = "no-referrer-when-downgrade" >
10              </iframe>
11    );
12  }
```

3.6.2 Otimizações na Galeria de Imagens

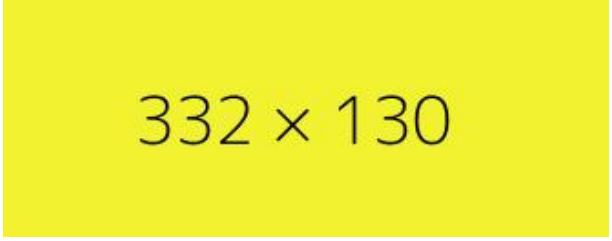
Realizamos uma série de otimizações nas imagens utilizadas na galeria de imagens do site para reduzir o volume de dados transferidos e acelerar o carregamento das páginas:

1. **Download das Imagens para o Projeto**: Anteriormente, as imagens eram carregadas diretamente de URLs externas através de requisições GET, o que podia resultar em latências adicionais. Agora, as imagens (18) são hospedadas localmente, eliminando a dependência de respostas de servidores externos e reduzindo o tempo de carregamento.

2. **Redimensionamento das Imagens**: As imagens foram redimensionadas para dimensões mais apropriadas (332 pixeis de lagura e 130px de altura ao invés de 9000 pixeis de largura e 3600 pixeis de altura). Esse redimensionamento assegura que não será enviado mais pixels do que necessário, otimizando ainda mais o desempenho ao diminuir o volume de bytes transferidos. O tamanho da imagem reduziu de 613kb para 3,38KB.

3. **Conversão para Formato WebP**: As imagens serão convertidas do formato JPG para WebP, o qual oferece uma compressão superior sem perda de qualidade visual. Esta mudança reduz o tamanho das imagem de 3,38KB para 1,73KB comparado ao JPEG, o que é crucial para a velocidade de carregamento em conexões de internet mais lentas.

Figura 18 – Imagem otimizada que será renderizada pelo componente GaleriaDeImagens



Fonte: Autoria própria

Estas otimizações pretendem não apenas acelerar o tempo de carregamento da página, mas também melhorar a experiência do usuário ao interagir com a aplicação web, garantindo que os recursos visuais sejam entregues eficientemente e sem comprometer a qualidade visual.

No código, o componente GaleriaDeImagensOtimizada (3.12) agora deverá inserir o caminho relativo dentro do projeto onde está localizada a imagem otimizada (18).

Código 3.12 – Código-fonte da GaleriaDeImagens após otimização

```
const GaleriaDeImagensOtimizada = () => {
  return (
    <div className={styles.gallery}>
      {new Array(15).fill(0).map((_, index) => (
        <div key={index} className={styles.imageContainer}>
          <img src="/assets/otimizados/imagem-tamanho-correto.webp" alt={`Imagem
    ${index + 1}`} className={styles.image} />
        </div>
      ))}
    </div>
  );
};

export default GaleriaDeImagensOtimizada;
```

3.6.3 Código da página V4

Por fim, o código final (3.13) do componente correspondente à página da Versão 4 atualizado com os dois novos componentes otimizados.

Código 3.13 – Código-fonte da Versão 4 da aplicação

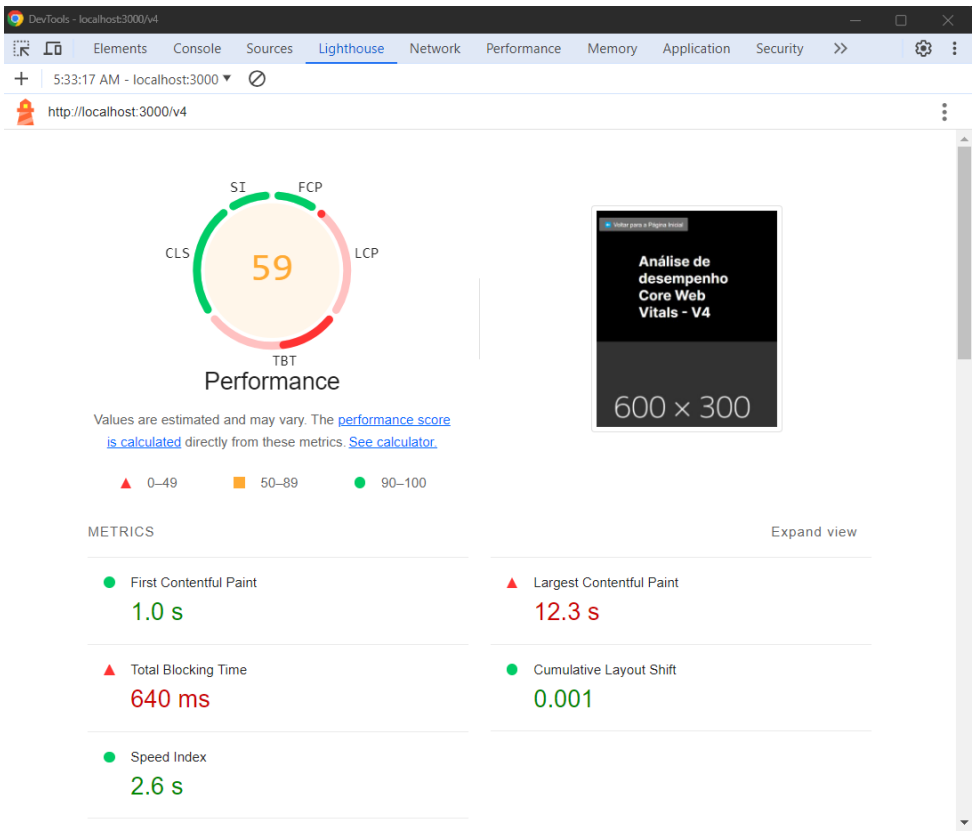
```
export default function V4() {
  useEffect(() => {
    if (typeof window === 'object' && window.Worker) {
      const worker = new Worker('/scripts/worker-pesado1.js');
      worker.onmessage = function (e) {
        console.log('Mensagem recebida do worker:', e.data);
      };
      worker.postMessage('Inicie a tarefa pesada');
      return () => worker.terminate();
    }
  }, []);

  return (
    <>
      <main className={styles.main}>
        <BotaoVoltar />

        <h1 className={styles.title}>
          An lise de desempenho Core Web Vitals - V4
        </h1>

        <ImagemComAtrasoOtimizada />
```

```
23
24          <h1>Exemplo de listagem de informacoes de api externa</h1>
25          <RepositoriosGithub/>
26
27          <h1>Exemplo de galeria de imagens</h1>
28          <GaleriaDeImagensOtimizada />  {/* Otimizacao nas imagens*/}
29
30          <h1>Mapa Ufu</h1>
31
32          <MapaGoogleLazyLoading/>  {/* Otimizacao no MapaGoogle*/}
33
34          <footer className={styles.footer}>
35            Desenvolvido por Fabricio Ziliotti - FACOM UFU
36          </footer>
37        </main>
38      </>
39    );
40  }
```

3.6.4 Resultados das Auditorias

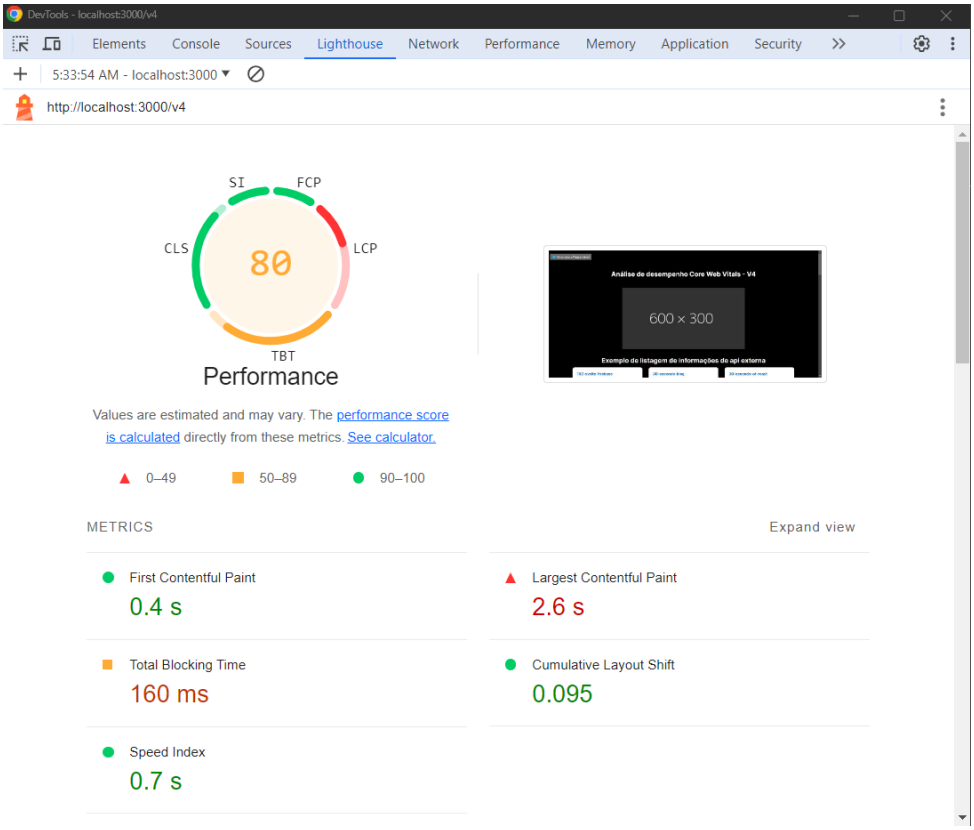
A seguir, estão os resultados das auditorias realizadas com a ferramenta **lighthouse**.Tanto para dispositivos móveis(figura 19) quanto para dispositivo *desktop*(figura 20).

Figura 19 – Resultado auditoria para versão 4 em dispositivo móvel



Fonte: Autoria própria

Figura 20 – Resultado auditoria para versão 4 em dispositivo desktop



Fonte: Autoria própria

3.7 Versão 5: Otimizando componentes para serem renderizados no servidor

Na versão final do projeto, a Versão 5, adotamos intensivamente o Server-Side Rendering (SSR) para renderizar componentes críticos no servidor antes de enviá-los ao navegador. Esta abordagem reduz significativamente o tempo de carregamento percebido pelo usuário e melhora as métricas de Core Web Vitals, como Largest Contentful Paint (LCP) e First Input Delay (FID).

3.7.1 Otimização dos Componentes

3.7.1.1 Componente RepositóriosGithub

O componente `RepositóriosGithub`, que anteriormente realizava chamadas API no lado do cliente, agora é renderizado no servidor. Isso significa que todas as informações são carregadas e prontas quando a página chega ao navegador do usuário, eliminando a necessidade de processamento adicional e reduzindo o tempo de interatividade.

Abaixo, encontra-se a nova implementação do componente 3.14, que agora deve utilizar o recurso de SSR do NextJs para fazer a busca dos repositórios diretamente no servidor, através da função buscarRepositoriosSSR.

Código 3.14 – Código-fonte da RepositóriosGithub renderizado no servidor

```
async function buscarRepositoriosSSR() {
  const res = await fetch('https://api.github.com/users/fziliotti/repos');
  if (!res.ok) {
    throw new Error('Falha ao buscar repositorios')
  }
  return res.json()
}

const RepositoriosGithubSSR = async () => {
  const repositorios = await buscarRepositoriosSSR()
  return (
    <div className={styles.grid}>
      {repositorios?.map(repo => (
        <div key={repo.id} className={styles.card}>
          <a href={repo.html_url} target="_blank" rel="noopener noreferrer"
    className={styles.link}>
            <div className={styles.title}>{repo.name}</div>
          </a>
          <p className={styles.description}>{repo.description || 'Nenhuma
    descricao disponivel.'}</p>
        </div>
      ))}
    </div>
  );
};

export default RepositoriosGithubSSR;
```

3.7.1.2 Componente ImagemSemAtraso

A imagem que inicialmente era carregada com um atraso para simular carregamento lento agora é entregue diretamente no carregamento inicial da página, graças ao SSR. Isso não apenas melhora a LCP, mas também elimina qualquer impacto negativo no Cumulative Layout Shift (CLS), assegurando que a página seja visualmente estável desde o momento em que é exibida.

No código abaixo 3.15, encontra-se a implementação simplificada do componente, removendo a simulação de atraso.

Código 3.15 – Código-fonte do componente ImagemComAtraso renderizada no servidor

```
import React from 'react';

function ImagemSSR() {
  return (
    <div>
        <img src="https://dummyimage.com/600x300/323232/fff.jpg" height={300}
    width={600} style={{margin: "1rem 0 2rem 0"}} />
```

```
7       </div>
8     );
9   }
10
11  export default ImagemSSR;
```

3.7.2 Código da página V5

Após a implementação de todas as otimizações de desempenho e pela alteração de todos os componente para a renderização no lado do servidor, o código da página V5 3.16 ficou na versão final.

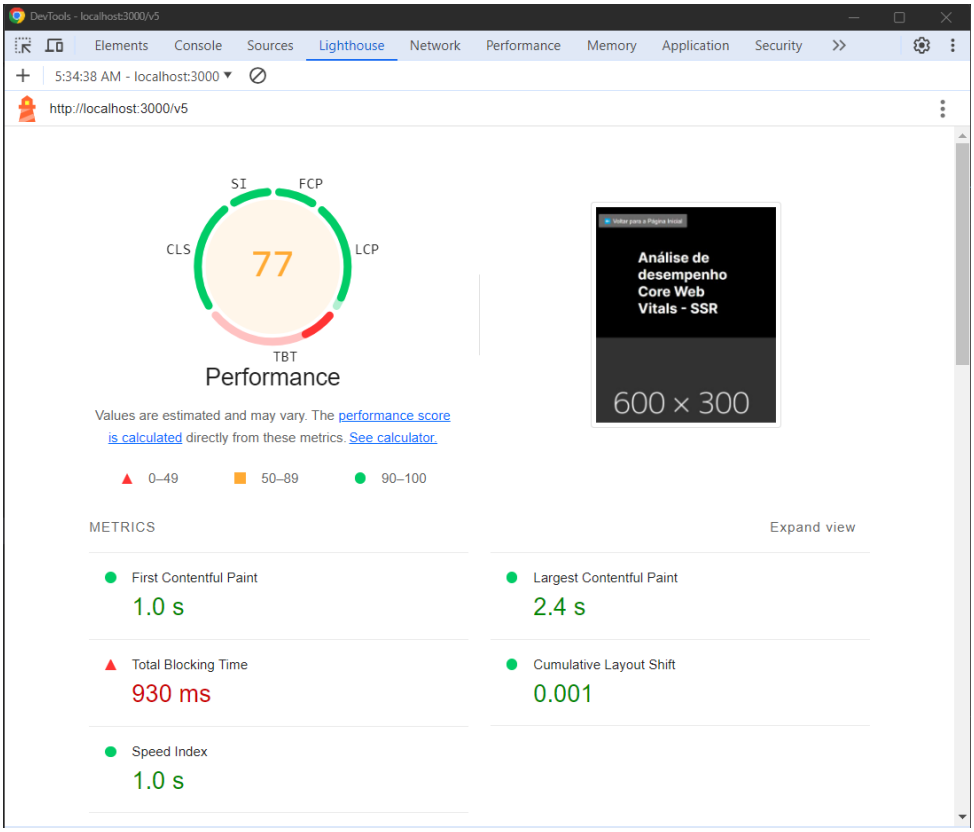
Código 3.16 – Código-fonte da versão 5 da aplicação

```
1   export default function V5() {
2     return (
3       <>
4         <main className={styles.main}>
5           <BotaoVoltar />
6
7           <h1 className={styles.title}>
8             Analise de desempenho Core Web Vitals - SSR
9           </h1>
10
11          <ImagemSSR />
12
13          <h1>Exemplo de listagem de informacoes de api externa</h1>
14          <RepositoriosGithubSSR/>
15
16          <h1>Exemplo de galeria de imagens</h1>
17          <GaleriaDeImagensOtimizada />
18
19          <h1>Mapa Ufu</h1>
20          <MapaGoogleLazyLoading/>
21
22          <footer className={styles.footer}>
23            Desenvolvido por Fabricio Ziliotti - FACOM UFU
24          </footer>
25        </main>
26      </>
27    );
28  }
```

3.7.3 Resultados das Auditorias

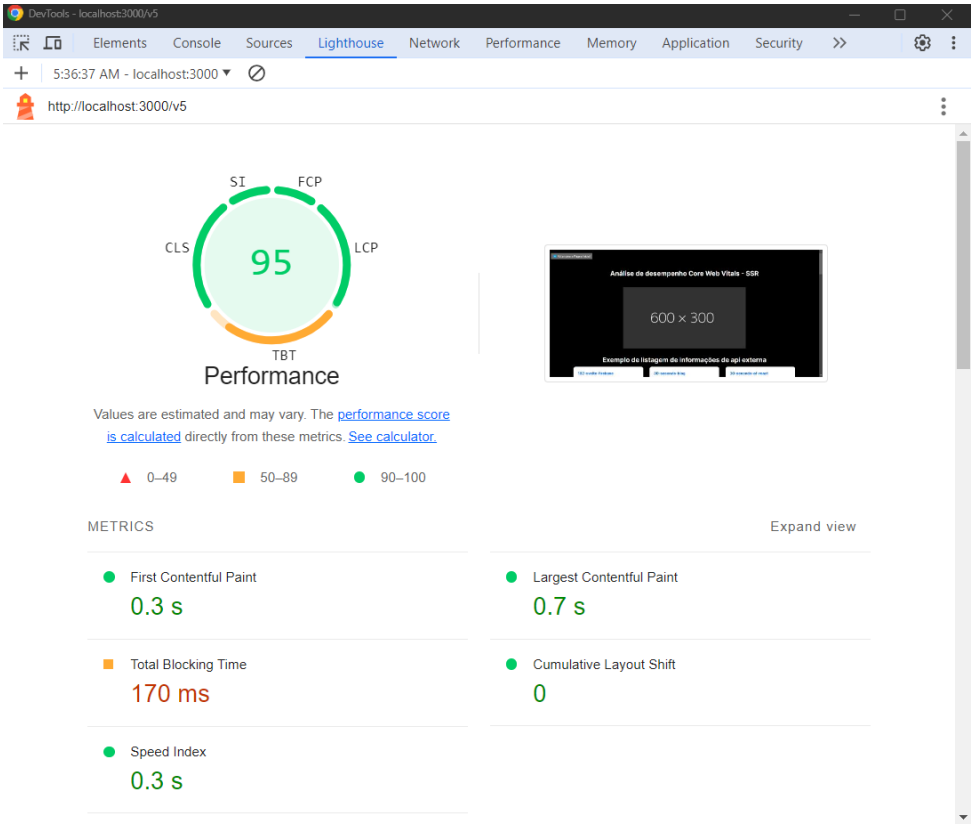
A seguir, estão os resultados das auditorias realizadas com a ferramenta **lighthouse**.Tanto para dispositivos móveis(figura 21) quanto para dispositivo *desktop*(figura 22).

Figura 21 – Resultado auditoria para versão 5 em dispositivo móvel



Fonte: Autoria própria

Figura 22 – Resultado auditoria para versão 5 em dispositivo móvel



Fonte: Autoria própria

3.8 Análise dos Resultados do Lighthouse

As avaliações de desempenho conduzidas com a ferramenta Google Lighthouse oferecem uma visão importante sobre o progresso das melhorias implementadas em cada versão do projeto. As tabelas 2 e 3 apresentam os dados acumulados para dispositivos móveis e desktops, respectivamente.

Tabela 2 – Resultados da auditoria In Lab da ferramenta Lighthouse para cada versão da aplicação para dispositivos móveis.

Versão do projeto	LCP	TBT (INP)	CLS	Page Load	Score
Versão 1	17,2 segundos	11500 milissegundos	0,222	10 segundos	24
Versão 2	15,2 segundos	480 milissegundos	0,249	2,8 segundo	49
Versão 3	13,8 segundos	410 milissegundos	0,001	2,9 segundos	64
Versão 4	2,6 segundos	640 milissegundos	0,001	2,6 segundos	59
Versão 5	2,4 segundos	930 milissegundos	0,001	1 segundo	77

Fonte: Autoria própria

Tabela 3 – Resultados da auditoria In Lab da ferramenta Lighthouse para cada versão da aplicação para dispositivos desktop.

Versão do projeto	LCP	TBT (INP)	CLS	Page Load	Score
Versão 1	7,6 segundos	4560 milissegundos	0,231	2,4 segundos	26
Versão 2	3,3 segundos	110 milissegundos	0,546	1,3 segundo	54
Versão 3	3,1 segundos	80 milissegundos	0,096	1 segundo	78
Versão 4	2,6 segundos	160 milissegundos	0,095	0,7 segundo	80
Versão 5	0,7 segundo	170 milissegundos	0	0,3 segundo	95

Fonte: Autoria própria

3.8.1 Dispositivos Móveis

A tabela 2 demonstra melhorias contínuas em várias métricas chave:

- **Largest Contentful Paint (LCP):** Reduzido de 17,2 segundos na Versão 1 para 2,4 segundos na Versão 5, indicando um carregamento visual mais rápido dos elementos principais da página.
- **Total Blocking Time (TBT) / Interaction to Next Paint (INP):** Diminuído de 11500 milissegundos para 930 milissegundos, melhorando significativamente a responsividade da página.
- **Cumulative Layout Shift (CLS):** Estabilizado em um valor muito baixo de 0,001, refletindo uma interface visualmente mais estável ao longo das otimizações.
- **Page Load Time:** Reduzido de 7,6 para 1 segundo na Versão 5, demonstrando um carregamento da página extremamente eficiente na versão final.
- **Score:** Aumentado para 77 na Versão 5, evidenciando uma experiência de usuário aprimorada em termos gerais.

3.8.2 Dispositivos Desktop

Analogamente, a tabela 3 reflete melhorias similares para a versão desktop:

- **Largest Contentful Paint (LCP):** Redução de 7,6 segundos para 0,7 segundo, destacando um carregamento acelerado do conteúdo mais crítico.
- **Total Blocking Time (TBT) / Interaction to Next Paint (INP):** Reduzido de 4560 milissegundos para 170 milissegundos, indicando uma melhora substancial na interatividade.

- **Cumulative Layout Shift (CLS):** Atingiu o valor ideal de zero na Versão 5, representando a ausência total de movimentos inesperados de conteúdo.
- **Page Load Time:** Diminuição para 0,3 segundo na Versão 5, proporcionando um tempo de resposta quase instantâneo.
- **Score:** Elevado para 95 na Versão 5, refletindo uma melhoria notável na qualidade e na experiência geral do usuário.

3.8.3 Considerações

As auditorias realizadas com a ferramenta Google Lighthouse revelaram impactos bastante positivos nas métricas do Web Vitals, corroborando com a literatura. Estas melhorias, que foram consistentemente observadas ao longo das versões incrementais do projeto, confirmam a eficácia das estratégias de otimização planejadas para dispositivos móveis e desktop.

Todavia, é importante considerar que alguns resultados inesperados surgiram durante a análise. Por exemplo, apesar de melhorias gerais, observou-se que em certas versões, o aumento no *Total Blocking Time* (TBT) para dispositivo desktop não foi tão significativo quanto esperado, sugerindo que certas otimizações podem ter tido um impacto diferenciado dependendo da configuração do dispositivo ou do conteúdo específico da página. Estas variações evidenciam a complexidade dos sistemas web e a necessidade de uma análise contínua e aprofundada para compreender completamente os efeitos de cada alteração.

Além disso, é crucial destacar que os dados obtidos nas auditorias são dados de laboratório, conforme descrito em (WEB.DEV, 2024f). Estes dados, embora úteis para testar alterações em um ambiente controlado e para isolar variáveis, podem diferir de dados de campo, que capturam as experiências reais dos usuários em condições variadas. Sendo assim, apesar de os resultados de laboratório indicarem melhorias claras, fundamentais para o diagnóstico e a otimização inicial, eles não refletem todos os aspectos do desempenho em situações reais.

Essas considerações enfatizam o êxito das otimizações, ao mesmo tempo em que reforçam a necessidade de uma avaliação constante e uma abordagem multidisciplinar para otimização de desempenho, que considere tanto dados de laboratório quanto de campo para uma melhor compreensão do impacto das otimizações na experiência do usuário final.

4 Conclusão

Este trabalho teve como objetivo estudar e implementar técnicas de otimização em uma aplicação web, utilizando as tecnologias React e Next.js, e avaliar especialmente as métricas Web Vitals, que se concentram na experiência do usuário. Ao longo do desenvolvimento do projeto, diversas técnicas foram estudadas e aplicadas, incluindo otimização de imagens, lazy loading, uso de Web Workers e renderização no lado do servidor. Essas intervenções resultaram em melhorias significativas e mensuráveis no desempenho da aplicação.

As análises realizadas, conforme apresentadas no Capítulo 3.7.3, e apoiadas por auditorias de desempenho com a ferramenta Lighthouse, demonstraram que as intervenções propostas contribuíram significativamente para a redução dos tempos de carregamento, melhor interatividade e maior estabilidade das páginas.

Esses resultados validam a eficácia das técnicas de otimização discutidas e demonstram como tais intervenções podem ser aplicadas para resolver problemas específicos de desempenho em aplicações reais, impactando positivamente diversos setores da sociedade, como observado nos casos de estudo abordados no Capítulo 2.1.2.

A relevância deste trabalho estende-se além das melhorias técnicas, contribuindo para o campo de desenvolvimento web ao fornecer um modelo prático e replicável que outros desenvolvedores podem seguir. As estratégias testadas e demonstradas podem servir como um guia para a otimização de aplicações web modernas, oferecendo percepções valiosas sobre a abordagem sistemática e efetiva do aprimoramento do desempenho.

Por fim, espera-se que as descobertas deste estudo inspirem mais pesquisas na área de otimização de desempenho web e impulsionem uma mudança positiva nas práticas de desenvolvimento, elevando a qualidade e a eficiência das aplicações web em um mercado cada vez mais competitivo e digitalizado. Assim, este trabalho não só contribui para a melhoria técnica das aplicações, mas também tem o potencial de influenciar positivamente a experiência do usuário final e, consequentemente, o sucesso comercial das plataformas online.

4.1 Limitações do Trabalho

Este estudo abordou uma variedade de técnicas de otimização para aplicações web, utilizando as tecnologias React e Next.js e a ferramenta Lighthouse para avaliar o desempenho das páginas web. Uma limitação relevante deste estudo é a eficácia do Lighthouse predominante em ambientes de desenvolvimento controlado, o que pode resultar na impos-

sibilidade de capturar as variáveis dos ambientes de produção reais nos quais as aplicações serão aplicadas. No entanto, apesar das limitações, o uso do Lighthouse se mostrou crucial, fornecendo percepções substanciais sobre áreas de melhoria de desempenho e identificação de problemas. Essa ferramenta confirmou sua relevância no ciclo de desenvolvimento e otimização, apontando para a necessidade de avaliações complementares que considerem condições de uso mais diversificadas.

4.2 Trabalhos Futuros

Estudos futuros podem expandir a integração do desenvolvimento web com modelos de linguagem de grande escala (LLMs), focando na otimização de prompts para criar páginas web com alto desempenho, alinhados às Web Vitals. Essa abordagem poderia incluir a adaptação de prompts para incorporar técnicas de otimização em componentes web existentes, aumentando tanto a funcionalidade quanto a eficiência dos sites. A investigação sobre a integração de LLMs no desenvolvimento web teria como objetivo não somente avançar no campo da engenharia de software, como também facilitar novas metodologias para a automação e melhorar continuamente o processo de desenvolvimento de aplicações web. Considerar a adoção de métodos avançados de otimização, específicos para diferentes contextos tecnológicos, também é uma direção valiosa para o trabalho futuro.

Referências

AKAMAI TECHNOLOGIES. Loading time impacts on user behavior. **Akamai**, 2009. Disponível em: <https://www.akamai.com/newsroom/press-release/akamai-releases-spring-2017-state-of-online-retail-performance-report>. Acesso em: 21 abr. 2024. Citado na página 14.

CHROME DEVELOPERS. **Overview of Lighthouse**. 2024. Disponível em: <https://developer.chrome.com/docs/lighthouse/overview>. Acesso em: 21 abr. 2024. Citado na página 20.

CLOUDFLARE. **Why Site Speed Matters**. 2024. Disponível em: <https://www.cloudflare.com/learning/performance/why-site-speed-matters/>. Acesso em: 25 abr. 2024. Citado na página 13.

DELOITTE. **Milliseconds Make Millions**. 2024. Disponível em: <https://www.deloitte.com/ie/en/services/consulting/research/milliseconds-make-millions.html>. Acesso em: 23 mar. 2024. Citado na página 12.

_____. **Milliseconds Make Millions**. 2024. Disponível em: <https://www.deloitte.com/content/dam/Deloitte/ie/Documents/Consulting/Milliseconds_Make_Millions_report.pdf>. Acesso em: 21 abr. 2024. Citado na página 14.

FACEBOOK. **React - A JavaScript library for building user interfaces**. 2024. Disponível em: <https://opensource.fb.com/projects/react/>. Acesso em: 25 abr. 2024. Citado na página 23.

GOOGLE SEARCH CENTRAL. **Using site speed in web search ranking**. 2010. Disponível em: <https://developers.google.com/search/blog/2010/04/using-site-speed-in-web-search-ranking>. Acesso em: 20 abr. 2024. Citado na página 15.

_____. **Using page speed in mobile search ranking**. 2018. Disponível em: <https://developers.google.com/search/blog/2018/01/using-page-speed-in-mobile-search>. Acesso em: 20 abr. 2024. Citado na página 15.

HOXMEIER, J. A.; DICESARE, C. System response time and user satisfaction: An experimental study of browser-based applications. In: ASSOCIATION FOR INFORMATION SYSTEMS. **AMCIS 2000 Proceedings**. 2000. p. 347. Disponível em: <http://aisel.aisnet.org/amcis2000/347>. Citado na página 14.

MOZILLA, D. **O que é um servidor web (web server)?** 2024. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server#sum%C3%A1rio>. Acesso em: 23 abr. 2024. Citado na página 21.

MOZILLA DEVELOPER NETWORK. **Why Web Performance Matters**. 2024. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Performance/why_web_performance>. Acesso em: 23 mar. 2024. Citado na página 13.

NEXT.JS. **What is Next.js?** 2024. Disponível em: <https://nextjs.org/docs#what-is-nextjs>. Acesso em: 24 abr. 2024. Citado na página 25.

NIELSEN, J. Website response times. **Nielsen Norman Group**, 2010. Disponível em: <https://www.nngroup.com/articles/website-response-times/>. Acesso em: 20 mar. 2024. Citado na página 14.

PORTENT. **Research: Site Speed Is Hurting Everyone's Revenue**. 2024. Disponível em: <https://www.portent.com/blog/analytics/research-site-speed-hurting-everyones-revenue.htm>. Acesso em: 22 abr. 2024. Citado na página 13.

REACT. **Start a New React Project**. 2024. Disponível em: <https://react.dev/learn/start-a-new-react-project>. Acesso em: 21 mar. 2024. Citado na página 25.

SAGOO, A.; SULLIVAN, A.; SEKHAR, V. **The Science Behind Web Vitals**. 2024. Disponível em: <https://blog.chromium.org/2020/05/the-science-behind-web-vitals.html>. Acesso em: 20 mar. 2024. Citado na página 13.

SOUTHERN, M. G. Google page experience update begins rolling out. **Search Engine Journal**, 2021. Disponível em: <https://www.searchenginejournal.com/google-page-experience-update-begins-rolling-out/410660/>. Acesso em: 14 abr. 2024. Citado na página 15.

WEB.DEV. **Case Studies**. 2024. Disponível em: <https://web.dev/case-studies>. Acesso em: 23 mar. 2024. Citado na página 12.

_____. **Case Studies**. 2024. Disponível em: <https://web.dev/case-studies?hl=pt-br>. Acesso em: 21 abr. 2024. Citado 2 vezes nas páginas 13 e 14.

_____. **Cumulative Layout Shift**. 2024. Disponível em: <https://web.dev/articles/cls?hl=pt-br>. Acesso em: 20 mar. 2024. Citado 3 vezes nas páginas 17, 18 e 19.

_____. **Interaction to Next Paint**. 2024. Disponível em: <https://web.dev/articles/inp?hl=pt-br>. Acesso em: 21 mar. 2024. Citado na página 20.

_____. **Largest Contentful Paint**. 2024. Disponível em: <https://web.dev/articles/lcp?hl=pt-br>. Acesso em: 20 mar. 2024. Citado na página 17.

_____. **Understanding Differences Between Lab and Field Data in Web Performance**. 2024. Disponível em: <https://web.dev/articles/lab-and-field-data-differences>. Acesso em: 23 abr. 2024. Citado 2 vezes nas páginas 21 e 53.

_____. **User-Centric Performance Metrics**. 2024. Disponível em: <https://web.dev/articles/user-centric-performance-metrics>. Acesso em: 23 mar. 2024. Citado na página 13.

_____. **Vitals**. 2024. <https://web.dev/articles/vitals>. Acesso em: 20 de abril de 2024. Citado 2 vezes nas páginas 16 e 17.

WEBSITE OPTIMIZATION. **The Psychology of Web Performance: How slow response times affect user psychology**. 2024. Disponível em: <https://www.websiteoptimization.com/speed/tweak/psychology-web-performance/>. Acesso em: 25 abr. 2024. Citado na página 14.

WPOSTATS. **WPOStats**. 2024. Disponível em: <https://wpostats.com/>. Acesso em: 21 abr. 2024. Citado 2 vezes nas páginas 13 e 14.