ANÁLISES E SOLUÇÕES PERFORMÁTICAS PARA APLICAÇÕES EM REACTJS *

Huriel Ferreira Lopes† Jorge Luis Boeira Bavaresco‡

2023

Resumo

O ReactJS, assim como outras ferramentas, foi criado com o intuito de melhorar o processo de desenvolvimento web. Contudo, algumas aplicações desenvolvidas por meio desta ferramenta, tais como e-commerces e blogs, têm apresentado deficiências de desempenho que afetam a experiência do usuário, manifestando-se em tempos de carregamento maiores que o necessário, ausência de resposta durante a navegação, ordem inadequada de carregamento de conteúdo e diversos contratempos relacionados à usabilidade do site. Nesse contexto, o objetivo deste trabalho consiste em demonstrar os resultados de uma pesquisa em sites que apresentaram uma baixa performance e expor a frequência e natureza desses problemas em sites criados com React. Com relação a natureza desses empecilhos, foi realizada uma análise para incluir as causas, consequências e propostas de solução desses erros, por meio de códigos de exemplo em uma aplicação desenvolvida para estudo de caso, a qual comprovou que as técnicas estudadas e aplicadas contribuíram para a performance do projeto. Os esforços empregados nesse projeto e no trabalho em geral foram com o objetivo de contribuir para a área da Ciência da Computação no âmbito do Desenvolvimento Web.

Palavras-chaves: ReactJS. Desenvolvimento Web. Performance. Blocking Time. Front-end.

*Trabalho de Conclusão de Curso (TCC) apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense, Campus Passo Fundo, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação, na cidade de Passo Fundo, em 2023.

†huriel-lopes@outlook.com

‡jorgebavaresco@ifsul.edu.br

1 INTRODUÇÃO

O Desenvolvimento Web surgiu na década de 1990, com a World Wide Web (WWW) fundada por Tim Berners-Lee. Inicialmente, com a linguagem de marcação HyperText Markup Language (HTML), as primeiras páginas web estáticas surgiram. Ao longo do tempo, a evolução da Web permitiu páginas mais dinâmicas e interativas, com tecnologias como JavaScript e PHP, que tornaram possível o que se chamaria posteriormente de Desenvolvimento Fullstack. Esse modelo de desenvolvimento era responsável por realizar tudo que era necessário para o funcionamento do site, e não muito tempo depois, surgiu uma forma diferente de se trabalhar, a qual separava responsabilidades entre os desenvolvedores, surgindo o Desenvolvimento Back-end e Desenvolvimento Front-end, o qual foca no desenvolvimento de interfaces de usuário (UI).

De acordo com Fedosejev (2015), a necessidade de uma solução alternativa ao jQuery e outras bibliotecas no desenvolvimento de interfaces web, que fosse escalável, rápida e modular, resultou no surgimento do ReactJS, uma biblioteca JavaScript de código aberto para Desenvolvimento Front-end. Essa tecnologia revolucionou o modo como os desenvolvedores projetam e criam interfaces dinâmicas e responsivas, tornando-se altamente valorizada pelos profissionais da área. Em virtude da sua ampla adoção, o React.js foi eleito pelo StackOverflow (2022) como a ferramenta mais popular para o desenvolvimento Front-End.

Este trabalho tem como tema a pesquisa de desempenho em aplicações ReactJS, na qual foi realizada uma análise de sites e problemas relacionados à performance de aplicações web com potencial de otimização. Nesse sentido, o problema em questão é: pouca otimização de sites criados com o ReactJS. Esse desempenho abaixo do esperado, sobretudo em dispositivos com recursos limitados, pode impactar negativamente a experiência do usuário, tornando a navegação menos agradável.

Por conta de tais contratempos, a análise e correção de sites desenvolvidos utilizando a tecnologia ReactJS é importante, de tal forma que os problemas encontrados foram analisados e soluções foram propostas, bem como o desenvolvimento de um comércio eletrônico de exemplo focado em performance.

A situação problemática dos sites é preocupante em relação à usabilidade, uma vez que a web deve ser acessível em todas as plataformas. Como destacou Berners-Lee (1997), o poder da Web está em sua universalidade, pois é essencial permitir o acesso a todos, independentemente de deficiências. O acesso em dispositivos com recursos limitados que apresentam problemas de performance é um ponto importante a se considerar na construção de uma aplicação, uma vez que muitos usuários podem não ter condições de adquirir um dispositivo de última geração. Essa exclusão de usuários, além de não seguir os padrões da web, pode acarretar em problemas para quem está investindo no negócio, como um menor número de acessos ou até mesmo de vendas, no caso de comércios eletrônicos.

O React disponibiliza diversas ferramentas que podem ser utilizadas para melhorar a performance das aplicações web, como soluções de cache, além de outras bibliotecas que auxiliam a identificar e corrigir problemas a partir de análises e soluções que elas apresentam. Infelizmente, muitos projetos desenvolvidos com React não utilizam essas ferramentas, ocasionando em sites

lentos.

Diante desse contexto, os principais fatores que afetam a qualidade das aplicações web construídas com a biblioteca ReactJS foram analisados, a fim de melhorar o ecossistema web e garantir a acessibilidade a todos os usuários, independentemente dos dispositivos que utilizam.

As seções a seguir contém o referencial teórico de trabalhos correlatos que foram essenciais para o embasamento teórico deste trabalho, bem como as tecnologias principais envolvidas durante a análise e desenvolvimento. Ademais, a metodologia e o desenvolvimento do trabalho, sobre como foi feita a análise dos sites, a análise dos problemas encontrados e como se deu o desenvolvimento do trabalho. Por fim, há a conclusão final, onde é analisado se os objetivos propostos foram alcançados.

2 REFERENCIAL TEÓRICO

O referencial teórico busca elucidar a respeito das tecnologias abordadas ao longo da pesquisa. Também, tem como objetivo referenciar trabalhos similares que auxiliaram no embasamento das informações utilizadas ao longo do trabalho.

2.1 ReactJS

O ReactJS é uma biblioteca JavaScript criada pela Meta e usada para desenvolver interfaces de usuário altamente responsivas e escaláveis para aplicativos web. Ele foi lançado pela primeira vez em 2013 e desde então se tornou uma das bibliotecas JavaScript mais populares do mundo, conforme o StackOverflow (2022).

Essa biblioteca é construída em torno do conceito de componentes, que são blocos reutilizáveis de código que podem ser compostos para criar interfaces de usuário complexas. Os componentes, segundo Pavić e Brkić (2021) são uma abstração poderosa que permite criar interfaces de usuário modulares e escaláveis, tornando o código mais fácil de manter e atualizar.

Cada componente no ReactJS tem sua própria lógica, estado e propriedades. O estado é uma representação de como o componente é exibido na interface e pode ser atualizado pelo usuário ou pelo próprio componente. As propriedades são valores passados para o componente. Juntos, o estado e as propriedades permitem criar interfaces dinâmicas e interativas.

O ciclo de vida dos componentes é bem definido. De acordo com a própria documentação publicada pela Meta (2023), os componentes passam por uma série de fases, desde a criação até a destruição, e cada fase é chamada em um determinado momento. O ciclo de vida de um componente inclui os seguintes métodos:

- constructor(): chamado quando o componente é inicializado;
- componentDidMount(): chamado depois que o componente é montado na árvore de elementos;
- shouldComponentUpdate(): chamado antes que o componente seja atualizado;

- componentWillUnmount(): chamado antes que o componente seja removido da árvore de elementos.

Esses métodos existem quando se programa utilizando o React no que é chamado de "Class Components" (Componentes de Classes), mas também é possível programar utilizando "Functional Components" (Componentes Funcionais), os quais também possuem o mesmo ciclo de vida, entretanto, o código é feito utilizando funções no lugar de classes, e o ciclo de vida é gerenciado através de "hooks", funções auxiliares que permitem manipular o ciclo de vida dos componentes.

Conforme Dwivedi, Kshamta e Joshi (2019), o ReactJS usa um Virtual DOM para gerenciar as alterações na interface do usuário de forma eficiente. Essa árvore virtual é uma representação em memória da árvore de elementos da interface, que é atualizada sempre que o estado do componente muda. O ReactJS compara o Virtual DOM atual com o anterior para identificar as alterações e atualiza apenas as partes da interface do usuário que foram alteradas, sem a necessidade de atualizar a página inteira.

Outro conceito importante é como os componentes em React são criados com o uso de JSX, que é uma extensão do JavaScript que permite escrever código HTML e CSS junto com a linguagem. Isso torna a criação de componentes mais fácil e intuitiva, permitindo que os desenvolvedores criem interfaces de usuário complexas com menos código. O JSX é convertido em JavaScript puro através de técnicas de compilação, como o Babel, antes de ser executado pelo navegador.

O React utiliza um fluxo de dados unidirecional para gerenciar o estado e as propriedades dos componentes. Isso significa que o estado e as propriedades são passados de pai para filho, e nunca o contrário. Isso ajuda a evitar problemas de sincronização de dados e torna o código mais fácil de entender e depurar.

O JSX, segundo Aggarwal (2018) é uma característica distintiva do ReactJS, mas também é controverso entre os desenvolvedores, já que alguns consideram a mistura de HTML e JavaScript uma má prática de codificação. No entanto, muitos desenvolvedores preferem o JSX por sua facilidade de uso e clareza.

Essa abordagem torna o ReactJS extremamente eficiente em termos de desempenho, pois minimiza a quantidade de processamento que precisa ser realizada toda vez que uma atualização é feita na interface do usuário. Além disso, o Virtual DOM permite que o ReactJS forneça recursos como a renderização do lado do servidor, facilitando a otimização do carregamento da página.

Em suma, o React é uma biblioteca JavaScript flexível para a web e interfaces nativas, segundo a própria Meta (2023). Ele oferece uma série de recursos avançados, como o Virtual DOM, componentes, estado, ciclo de vida, fluxo de dados unidirecional e JSX. Com o React, os desenvolvedores podem criar interfaces do usuário ricas e interativas com facilidade e eficiência.

2.2 Arquiteturas de Desenvolvimento

O desenvolvimento web é uma área em constante evolução, e sua arquitetura também está sofre mudanças. Segundo Gong et al. (2020), os grandes projetos desenvolvidos em Java, que consolidavam tanto o servidor quanto a interface juntos, causam uma pressão enorme no servidor, que caso caia, o serviço inteiro fica indisponível. Esse modelo de aplicação é chamado de Full-stack, pois o desenvolvedor cria tanto a interface quanto o servidor. Visando uma alternativa que não concentrasse tudo em um só lugar, até por questões de segurança, surgiu uma outra forma de se trabalhar com as aplicações, ao separar as responsabilidades de desenvolvimento. Dessa forma surgiram os termos Back-end e Front-end, sendo o Back-end responsável por cuidar do servidor, requisições, validações e interagir com o banco de dados, e o Front-end, responsável por lidar com a interface, usabilidade e interação do usuário.

O desenvolvimento full-stack é a prática de desenvolver tanto o frontend quanto o backend de uma aplicação web. Nas palavras de Gong et al. (2020), os desenvolvedores full-stack são capazes de trabalhar em todos os aspectos de uma aplicação web, desde o design e a funcionalidade do frontend até a lógica do servidor, banco de dados e segurança.

O desenvolvimento backend, segundo Madurapperuma, Shafana e Sabani (2022), se concentra no desenvolvimento do lado do servidor de uma aplicação web. Isso inclui a lógica de negócios, gerenciamento de banco de dados e segurança. Os desenvolvedores backend geralmente trabalham com linguagens de programação como Java, Ruby, Python, Node.js e PHP.

O desenvolvimento frontend, conforme Edkins et al. (2013), é a prática de desenvolver a interface do usuário e a interação do usuário em uma aplicação web. Os desenvolvedores frontend geralmente trabalham com HTML, CSS e JavaScript para criar o layout e o comportamento da interface do usuário. É nesse cenário onde o React se encontra, especializando ainda mais o desenvolvimento de interfaces.

Nos últimos anos, surgiram novas arquiteturas para lidar com esses problemas. A arquitetura de microserviços, por exemplo, mencionada por Gong et al. (2020) divide uma aplicação em componentes independentes e escaláveis, cada um com sua própria base de código e banco de dados. Outra arquitetura popular é a arquitetura serverless, onde a lógica de negócios é realizada em pequenos pedaços de código chamados de funções, que são executados em resposta a eventos.

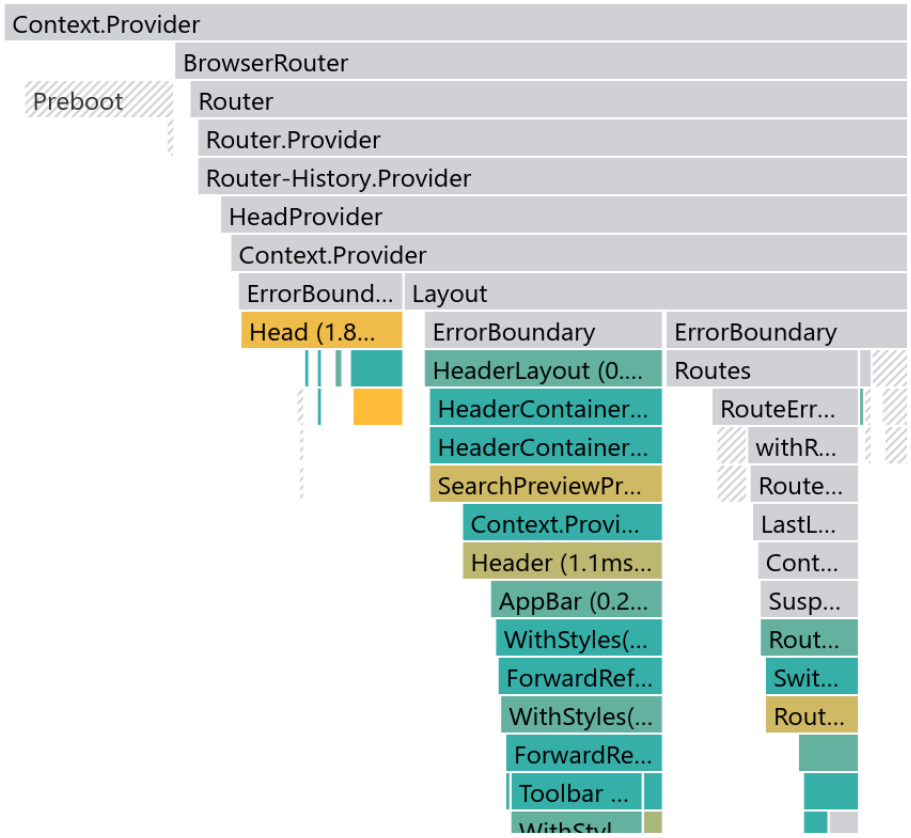
2.3 React Developer Tools

O React Developer Tools é uma extensão do navegador que ajuda a depurar aplicativos React. De acordo com Meta (2023), a ferramenta fornece uma interface gráfica do usuário (GUI) para examinar a hierarquia de componentes do React, bem como suas propriedades e estado em tempo real, permitindo inspecionar os componentes, hierarquia, desempenho e também manipular estados e propriedades.

Além dessas capacidades principais, o React Developer Tools também oferece várias outras ferramentas e recursos, como a capacidade de visualizar o histórico de mudanças de estado de um componente e a capacidade de filtrar e destacar componentes específicos na árvore de elementos, com uma interface bem intuitiva para auxiliar desenvolvedores, demonstrada na

Figura 1, na qual há uma cascata representativa da renderização dos componentes. Os itens preenchidos e traçejados em cinza foram componentes que atualizaram parcialmente na interface, pois possuem um bom controle da renderização de seu conteúdo. Já os itens coloridos realizaram uma renderização completa, sendo que em verde foi uma renderização rápida e em amarelo demorada.

Figura 1 – Análise do React Developer Tools e seus resultados



Fonte: do autor

Essa ferramenta é essencial para qualquer desenvolvedor que trabalha com React. Ele permite que você depure rapidamente problemas de estado, propriedades e renderização, além de ajudar a melhorar o desempenho do seu aplicativo.

2.4 Lighthouse

Lighthouse, na visão da Google (2016) é uma ferramenta de código aberto para auxiliar na análise de problemas das páginas da web, permitindo que os desenvolvedores identifiquem o que há de errado e corrijam erros em suas aplicações, para manter a qualidade do produto. Ele é usado para auditar as páginas da web em relação a uma série de métricas de desempenho, acessibilidade, melhores práticas e SEO (Search Engine Optimization).

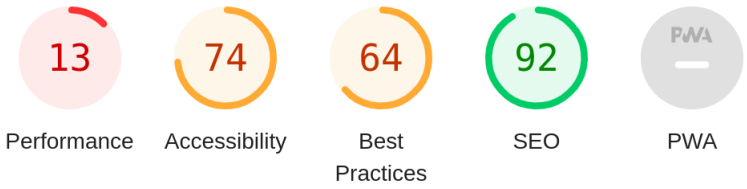
O Lighthouse funciona em qualquer site, independentemente da plataforma ou tecnologia utilizada para sua construção. A ferramenta é executada como uma extensão do Google Chrome

ou como uma linha de comando, oferecendo recursos para avaliar o desempenho da página em diferentes dispositivos móveis e de desktop.

Com relação às capacidades do Lighthouse, a ferramenta fornece métricas detalhadas sobre o tempo de carregamento da página, sugerindo maneiras de melhorar a velocidade de carregamento e a experiência do usuário. Ademais, avalia a acessibilidade da página, detectando problemas de contraste, estruturação de conteúdo, legibilidade, uso de imagens e outras questões. Além disso, também analisa a página para garantir que as práticas recomendadas de desenvolvimento web estejam sendo seguidas, como a compressão de arquivos e o uso de recursos de cache e avalia a compatibilidade da página com dispositivos móveis, garantindo que a página seja responsiva e compatível com diferentes tamanhos de tela. Por fim, avalia a otimização para mecanismos de busca (SEO) da página, conforme a Figura 2, fornecendo sugestões para melhorar a visibilidade e o ranking do site nos resultados de pesquisa.

A pesquisa de performance do Lighthouse ocorre na página do link informado ao analisador. Por esse motivo, as demais rotas e páginas do site não são consideradas na análise.

Figura 2 – Resultado de Avaliação do Lighthouse



Fonte: do autor

Em suma, o Lighthouse é uma ferramenta valiosa para desenvolvedores web que desejam melhorar a qualidade de suas páginas, garantindo que elas sejam rápidas, acessíveis, compatíveis com dispositivos móveis e otimizadas para mecanismos de busca. Com o uso do Lighthouse, os desenvolvedores podem identificar problemas e corrigi-los antes que eles se tornem problemas para os usuários finais.

2.5 Next.js

Next.js é uma estrutura de aplicativos da web de código aberto para React, criada pela equipe da Vercel (2023). É uma das ferramentas mais populares para o desenvolvimento de aplicativos da web modernos e escaláveis. Next.js é usado por muitas empresas e organizações em todo o mundo para criar aplicativos da web de alto desempenho e escaláveis.

O objetivo principal do Next.js é fornecer uma experiência de desenvolvimento da web mais agradável e produtiva para desenvolvedores de todos os níveis de habilidade. Ele faz isso fornecendo uma ampla gama de recursos e ferramentas para ajudar os desenvolvedores a criar aplicativos da web rapidamente, incluindo suporte integrado para React, pré-renderização, rotas dinâmicas, otimização de imagem e muito mais. Essa ferramenta auxilia no desenvolvimento de

aplicativos da web modernos e escaláveis. Ele oferece uma ampla gama de recursos e ferramentas para ajudar os desenvolvedores a criar aplicativos da web de alta qualidade de forma mais rápida e eficiente.

2.6 Trabalhos relacionados

Foi realizada uma pesquisa a respeito de trabalhos já publicados que abordam teorias e técnicas relevantes para o desenvolvimento de aplicações web com ReactJS, mais especificamente com o tema de performance. Com esses trabalhos, foi possível obter um embasamento teórico mais eficiente para a busca de melhorias em aplicações que possuem problemas de performance, o que permitiu identificar mais facilmente o que pode resultar em um site lento.

Pavić e Brkić (2021) discutiram técnicas de otimização para aplicações React em seu trabalho. Tais técnicas incluíram empacotamento de código, divisão de código, eliminação de código morto e, principalmente, a otimização de imagens, um tema que foi abordado com maior foco, devido a importância que os autores deram ao tópico. O empacotamento de código reduz o número de arquivos "bundles" para diminuir a transmissão de arquivos, enquanto a divisão de código permite carregar partes do código sob demanda, o que reduz o carregamento inicial das páginas. No mesmo contexto, a eliminação de código morto reduz o tamanho da aplicação no geral, pois é um código que já deveria ter sido removido. Ademais, recomenda-se o uso de formatos de imagem que os autores denominam como "next-gen" para otimização de imagens, por serem mais eficientes para o uso na web. Ao longo do trabalho, também é realizado uma comparação de técnicas de otimização de desempenho em uma aplicação de aluguel de acomodações temporárias, testando diferentes ambientes e técnicas de carregamento dinâmico e otimização de imagens.

O trabalho de Javeed (2019) abordou técnicas para otimizar o desempenho de aplicações ReactJS, focando na redução de renderizações desnecessárias. Nesse processo, os métodos destacados incluiam a diminuição de mudanças de estado e propriedades, a divisão de componentes principais em componentes independentes e puros, a reutilização de instâncias de componentes e a otimização de algoritmos de pesquisa.

Os autores Sousa e Gonçalves (2020) apresentaram um estudo de caso do "humanportal", um aplicativo desenvolvido com ReactJS. Nesse estudo, foi destacado a utilidade e versatilidade do React no desenvolvimento, demonstrando como a migração para o React melhorou a satisfação dos clientes, por proporcionar rápido acesso a módulos de ação de treinamento e flexibilidade no uso dos módulos.

Em conjunto, esses trabalhos forneceram informações valiosas sobre técnicas de otimização e o impacto do React no desenvolvimento de aplicações web, destacando a importância do uso de ferramentas adequadas para melhorar o desempenho e a satisfação dos usuários.

3 METODOLOGIA E DESENVOLVIMENTO

Esta seção aborda em detalhes como a pesquisa foi desenvolvida, qual foi a amostra de sites analisados, como foram selecionados e classificados, quais foram os resultados obtidos da

análise, quais foram os erros encontrados com maior frequência nos sites, e principalmente, uma discussão das soluções propostas, com exemplos de código em detalhes de como implementar.

3.1 Análise de sites com deficiências de performance

A análise foi realizada em um total de 20 sites desenvolvidos com React, sendo dez de origem brasileira e dez de origem mundial. A seleção desses sites ocorreu com base nos resultados de pesquisa de ferramentas, mais precisamente o Google e o Bing. Para encontrar sites desenvolvidos com a ferramenta em questão, foi utilizado o React Developer Tools, que é capaz de identificar se o site acessado foi desenvolvido com React. Após a identificação, a análise foi realizada com a ferramenta Lighthouse, que fornece métricas de desempenho, acessibilidade, boas práticas, SEO e PWA (Progressive Web App). A análise foi focada principalmente na categoria de desempenho em dispositivos móveis. Nessa análise, foi considerado que sites que tivessem nota vermelha no Lighthouse, seriam classificados como "Ruins" ou não performáticos, e sites que obtivessem amarelo ou verde, como "Suficientes" ou performáticos. Com o resultado da análise inicial, a quantia de sites performáticos e não-performáticos foi descoberta, conforme a Tabela 1.

	Performáticos	Não performáticos
Sites mundiais	4	6
Sites brasileiros	2	8

Tabela 1 – Relação da quantidade de sites performáticos e não performáticos encontrados

Esses resultados têm implicações importantes para o desenvolvimento de sites no contexto principalmente brasileiro, mas também mundial. Para melhorar a performance dos sites, é necessário entender em detalhes o motivo da performance dos sites da amostra ter sido negativa, e em paralelo, a ferramenta Lighthouse ajudou a identificar os problemas.

3.2 Problemas encontrados e soluções

Foi analisada a frequência de ocorrência dos problemas encontrados através da análise do Lighthouse, para entender o que há de comum entre as amostras. Com a clareza dos problemas, diversas soluções surgiram tanto para projetos com Next.js, quanto projetos que utilizam apenas o React. Os problemas encontrados e suas soluções serão abordados nas subseções seguintes. Os problemas encontrados e o resultado da performance do Lighthouse é demonstrado pela Tabela 2. Nessa tabela, o nome dos sites não é divulgado pelo prezar da privacidade das amostras selecionadas.

Amostra	Resultado Lighthouse	Quantidade de erros
Site 1 Mundial	Vermelho: 31 (Não-performático)	6
Site 2 Mundial	Amarelo: 75 (Performático)	4
Site 3 Mundial	Amarelo: 82 (Performático)	4
Site 4 Mundial	Amarelo: 57 (Performático)	3
Site 5 Mundial	Vermelho: 31 (Não-performático)	6
Site 6 Mundial	Vermelho: 14 (Não-performático)	8
Site 7 Mundial	Vermelho: 29 (Não-performático)	8
Site 8 Mundial	Amarelo: 57 (Performático)	6
Site 9 Mundial	Vermelho: 26 (Não-performático)	6
Site 10 Mundial	Vermelho: 28 (Não-performático)	8
Site 1 Brasileiro	Vermelho: 41 (Não-performático)	9
Site 2 Brasileiro	Vermelho: 10 (Não-performático)	9
Site 3 Brasileiro	Vermelho: 45 (Não-performático)	6
Site 4 Brasileiro	Vermelho: 17 (Não-performático)	7
Site 5 Brasileiro	Amarelo: 65 (Performático)	5
Site 6 Brasileiro	Amarelo: 63 (Performático)	6
Site 7 Brasileiro	Vermelho: 21 (Não-performático)	8
Site 8 Brasileiro	Vermelho: 31 (Não-performático)	8
Site 9 Brasileiro	Vermelho: 30 (Não-performático)	7
Site 10 Brasileiro	Vermelho: 40 (Não-performático)	5

Tabela 2 – Resultados aprofundados da pesquisa

É importante ressaltar que cada erro encontrado possui seu nível de gravidade. Um site pode ser considerado performático com, por exemplo, quatro erros, e outro não performático mesmo que possua uma quantidade menor, porque a performance é relativa aos erros que estão ocorrendo, e não necessariamente à quantidade de erros, conforme ilustra a Tabela 2.

A seguir, serão listados os erros encontrados com suas respectivas soluções, além de exemplos de código quando é possível exemplificar.

3.2.1 Ausência de Compressão de Texto

Três sites analisados precisam realizar compressão de texto para possuir uma performance mais adequada. A compressão de texto é ativada por padrão em projetos Next.js, e nesses casos, deve-se desativar essa configuração somente quando o desenvolvedor entender o processo e houver essa intenção. No caso de demais projetos React.js, o `compression-webpack-plugin` é uma biblioteca que pode ser instalada e essa configuração deverá ser adicionada no arquivo de configuração do Webpack, conforme o Código Fonte 1.

Código Fonte 1 – Compressão de texto

```
1 plugins: [
2     new webpack.DefinePlugin({
3     'process.env': {
4         'NODE_ENV': JSON.stringify('production')
5     }
6     }),
7     new webpack.optimize.DedupePlugin(),
```

```
8       new webpack.optimize.UglifyJsPlugin(),
9       new webpack.optimize.AggressiveMergingPlugin(),
10      new CompressionPlugin({
11          asset: "[path].gz[query]",
12          algorithm: "gzip",
13          test: /\.js$|\.css$|\.html$/,
14          threshold: 10240,
15          minRatio: 0.8
16      })
17  ]
```

Posteriormente, é possível adicionar um Middleware nas requisições do servidor, o qual irá retornar os arquivos comprimidos, ao invés do código fonte. Caso esteja-se utilizando Express, o Código Fonte 2 resolve o problema:

Código Fonte 2 – Middleware de envio de compressão

```
1  app.get('*.js', function (req, res, next) {
2      req.url = req.url + '.gz';
3      res.set('Content-Encoding', 'gzip');
4      next();
5  });
```

3.2.2 Thread Principal Excessivamente Grande

Em dezesseis sites foi encontrado esse problema. Para resolvê-lo, pode se utilizar a divisão do código em diferentes bundles com o lazy load, carregar scripts e CSS posteriormente, e otimizar imagens. Diversas das técnicas de correção abordadas ao longo do trabalho causam a correção desse problema também.

Uma abordagem eficaz para minimizar a thread principal é implementar o lazy load e dividir o código em bundles separados. Sem essas técnicas, todo o código é carregado em um único bundle principal, resultando em carregamento desnecessário de partes não utilizadas. No React, você pode usar a função `lazy` para carregar componentes de forma assíncrona e separada, como mostrado no Código Fonte 3:

Código Fonte 3 – Uso de Lazy Component no React

```
1  import { lazy } from 'react';
2  const LazyComponent = lazy(() => import('./LazyComponent'));
```

Isso carrega o código do componente separadamente, reduzindo a carga na thread principal e eliminando código JavaScript não utilizado. No caso do Next.js, o conceito é semelhante. Você pode usar `dynamic` para carregamento assíncrono, otimizando ainda mais o `React.lazy`. O uso desse componente é simples, conforme o Código Fonte 4:

Código Fonte 4 – Uso de Lazy Component no Next.js

```
1  import dynamic from 'next/dynamic';
2  const LazyComponent = dynamic(
3      () => import('../components/LazyComponent')
4  );
```

Essa abordagem no Next.js também carrega o código do componente separadamente, e através de um segundo parâmetro, é possível renderizar um outro componente enquanto o principal está sendo carregado.

3.2.3 Excesso de JavaScript Não Utilizado

Dezoito sites, quase todas as amostras possuíam JavaScript que não estava sendo utilizado. Para eliminar esse problema, a primeira etapa é identificar código morto. A medida que os projetos crescem, partes do código podem deixar de ser usadas e não serem removidas. É importante verificar periodicamente a presença de código morto. Isso pode ser feito com ferramentas como o Webpack Bundle Analyzer ou a aba Compiler do Developer Tools do navegador. Nessa solução, é necessário realizar a remoção desse código manualmente após a análise.

3.2.4 Excesso de CSS Não Utilizado

Quatro sites possuiam um número excessivo de CSS que não estava sendo utilizado na página. Para eliminar estilos CSS não utilizados, pode-se utilizar a biblioteca PurgeCSS, cuja finalidade consiste em suprimir estilos CSS sem uso. Em contextos que envolvem aplicações Next.js e React, essa operação de depuração se desencadeia durante a etapa de construção (build) da aplicação. Como resultado, os arquivos CSS gerados não incluirão classes e estilos que não tenham sido empregados no código.

Para proceder com a configuração da biblioteca, é necessário instalar o pacote npm `purgecss` e `@fullhuman/postcss-purgecss`, e configurá-lo no seu projeto conforme o Código Fonte 5:

Código Fonte 5 – Configuração da biblioteca Purge CSS

```
1 module.exports = {
2     plugins: [
3         require('@fullhuman/postcss-purgecss')({
4             content: ['./src/**/*.js', './pages/**/*.js'],
5         }),
6     ],
7 };
```

3.2.5 Imagens não Otimizadas

Dezessete sites não estão utilizando otimizações adequadas nas imagens. No contexto das aplicações Next.js, a recomendação mais sólida reside no uso do componente `next/image`, que automaticamente implementa diversas otimizações em relação às imagens. Além disso, é crucial, mesmo quando tratando de aspectos responsivos, estabelecer as dimensões (largura e altura) das imagens. Isso não apenas incide positivamente na performance, mas também mantém a proporção de aspecto (aspect-ratio) das imagens. Outro ponto relevante é aplicar a técnica de `lazy load` às imagens que não são primárias e que inicialmente se encontram fora do campo de visualização, dado que as imagens podem representar conteúdo pesado que afeta o tempo de carregamento da página em sua totalidade.

O Next.js gera distintas versões de uma mesma imagem em diferentes formatos, a fim de carregar apenas a versão mais eficiente no momento em que o usuário acessa o site. Isso otimiza o consumo de recursos e promove elevados níveis de desempenho. Também é viável configurar um elemento de preenchimento (placeholder) que será exibido ao usuário enquanto a imagem é carregada, proporcionando uma navegação melhor ao usuário. Os exemplos mencionados podem ser visualizados no exemplo de componente do Código Fonte 6.

Código Fonte 6 – Uso do Componente Image do Next.js

```
1  import Image from 'next/image';
2  export const DefaultImage = () => {
3      return (
4          <div>
5              <Image
6                  src="https://avatars.githubusercontent.com/u/61247833"
7                  alt="Foto de Huriel, 21 anos, caucasiano"
8                  width={400} height={200} placeholder="blur"
9              />
10         </div>
11     );
12 };
```

Para enriquecer ainda mais um componente de imagem e torná-lo mais eficiente, dependendo do contexto, é possível utilizar a propriedade `quality` para indicar um valor numérico de 1 a 100, referente à qualidade da imagem. Dado o significativo impacto da qualidade da imagem no desempenho, fornecer um valor preciso ao Next.js é uma consideração relevante nesse contexto. Por fim, a propriedade `priority` admite um valor booleano que determina se a imagem deve ser carregada com prioridade em relação a outras imagens da página.

3.2.6 Scripts Concorrendo Com Conteúdo Principal

Na amostra de vinte sites, sete possuem scripts de terceiro carregando junto com o conteúdo principal. Em aplicações Next.js, é altamente recomendável empregar o componente `Script` fornecido pelo Next.js. Este componente, por si só, já incorpora várias melhorias de desempenho relacionadas ao carregamento de scripts. Além disso, simplifica as configurações relacionadas ao momento de carregamento de cada script por meio da propriedade `strategy` do componente. Essa propriedade permite carregar o script antes de qualquer outro conteúdo na página, durante o carregamento da página, após o carregamento ou até mesmo em um web worker (experimental), o que viabiliza o carregamento do script sem impactar o restante da página, conforme está sendo utilizado na Linha 8 do Código Fonte 7.

Código Fonte 7 – Uso do Componente Script do Next.js

```
1  import Script from 'next/script'
2  export default function MyApp({ Component, pageProps }) {
3      return (
4          <>
5              <Component {...pageProps} />
6              <Script
7                  src="https://example.com/script.js"
```

```
8                   strategy="afterInteractive"
9               />
10          </>
11      );
12  }
```

Em aplicações React.js em geral, seja com ou sem Next.js, é possível recorrer à biblioteca `Partytown`, que se especializa em carregar scripts em um web worker para aprimorar o desempenho. Para isso, basta instalar o pacote `@builder.io/partytown` e configurá-lo na seção `Head` das páginas da aplicação, assim como o Código Fonte 8 demonstra.

Código Fonte 8 – Configuração do Partytown no React

```
1  import { Partytown } from '@builder.io/partytown/react';
2  export function Head() {
3      return (
4          <Partytown debug={true} forward={['dataLayer.push']} />
5      );
6  }
```

Após essa configuração, é possível inserir o script, definindo o tipo como `text/partytown` no Código Fonte 9.

Código Fonte 9 – Uso do Partytown no React

```
1  <script
2      type="text/partytown"
3      dangerouslySetInnerHTML={{
4          __html: '/* Script de Terceiro */',
5      }}
6  />
```

Essas abordagens, tanto no contexto Next.js quanto em aplicações React.js em geral, contribuem significativamente para otimizar o carregamento de scripts e, consequentemente, melhorar a performance das aplicações.

3.2.7 Ausência de Uso de Cache

Quinze sites analisados deveriam implementar melhorias de cache. No Next.js, essas soluções podem ser implementadas de várias maneiras. O framework já incorpora diversas otimizações e soluções por padrão, embora todas elas sejam ajustáveis conforme as necessidades do projeto. No entanto, é fundamental ter cautela ao realizar ajustes nas configurações, pois configurações incorretas podem prejudicar o desempenho da aplicação. Em ambiente de produção, o Next.js sobrescreve as configurações definidas pelo desenvolvedor no arquivo `next.config.js` para assegurar que as páginas geradas estaticamente tenham a melhor configuração de cache possível.

Já as páginas renderizadas pelo servidor, podem ser melhoradas através do controle das configurações de cache no cabeçalho de cada página utilizando o método `setHeader`, definido na Linha 2 do Código Fonte 10.

Código Fonte 10 – setHeader de cache no Next.js

```
export async function getServerSideProps({ req, res }) {
    res.setHeader(
        'Cache-Control',
        'public, s-maxage=10, stale-while-revalidate=59'
    )

    return {
        props: {
            // Propriedades a serem retornadas
        },
    }
}
```

Em aplicações React é viável implementar soluções de cache eficientes utilizando os hooks `useMemo` e `useCallback`. Esses hooks se baseiam no conceito de memoização, permitindo preservar o resultado de uma função ou um dado por um período mais longo, o que evita o processamento desnecessário de JavaScript.

O `useCallback` é útil para preservar uma função e somente atualizá-la quando necessário. Normalmente, a função seria recriada a cada renderização do componente, o que poderia resultar em perda de desempenho. Por exemplo, em um menu que gerencia cliques em itens, o `useCallback` garante que a função só seja atualizada quando a dependência real for alterada, ou seja, a função `onItemClick`, definida do vetor da Linha 5 do Código Fonte 11, muda.

Código Fonte 11 – Uso do useCallback no React

```
import React, { useCallback } from 'react';
function Menu({ items, onItemClick }) {
    const handleItemClick = useCallback((item) => {
        onItemClick(item);
    }, [onItemClick]);

    // return ...
}
```

O `useMemo` segue um processo semelhante, mas em vez de memoizar uma função, ele memoiza um dado. O valor desse dado só é processado novamente quando as dependências definidas no vetor de dependências são alteradas. Isso é particularmente útil para componentes que formatam dados, como cartões (comumente chamados de “cards”), onde o processamento deve ser evitado sempre que possível, conforme o Código Fonte 12.

Código Fonte 12 – Uso do useMemo no React

```
import React, { useMemo } from 'react';
function Card({ data }) {
    const formattedData = useMemo(() => {
        const formattedData = {
            // Processamento de dados
        };

        return formattedData;
```

```
9       }, [data]);
10
11      // return ...
12  }
```

3.2.8 Tamanho Elevado da DOM

Onze sites apresentam esse problema. No React, é possível evitar isso de várias maneiras. Uma dessas maneiras é através do uso apropriado do hook `useEffect`. Este hook é responsável por executar uma função quando um estado especificado pelo desenvolvedor tem seu valor alterado. Portanto, é essencial implementar o `useEffect` de forma adequada para evitar que o hook ative em momentos desnecessários e acabe renderizando diversas vezes.

Outra abordagem eficaz é utilizar o `React.memo` para criar componentes puros, que são componentes sem estados, e por esse motivo, a lógica para renderizá-los deveria ser executada somente uma ou pouquíssimas vezes. No exemplo Código Fonte 13, há um componente simples, e devido à sua simplicidade, não é necessário consumir recursos do dispositivo do usuário para renderizar essa página muitas vezes, o que conta muito para uma boa performance no carregamento de páginas do React. Para evitar isso, é recomendável utilizar o `React.memo`, declarado na linha 11 do Código Fonte 13.

Código Fonte 13 – Uso do memo no React

```
1  function LandingPage({ title, description, image }) {
2      return (
3          <div>
4              <h1>{title}</h1>
5              <p>{description}</p>
6              <img src={image} alt="Landing" />
7          </div>
8      );
9  }
10
11 export default React.memo(LandingPage);
```

Ao empregar essas técnicas, é possível manter o tamanho do DOM sob controle e, assim, melhorar significativamente o desempenho da aplicação.

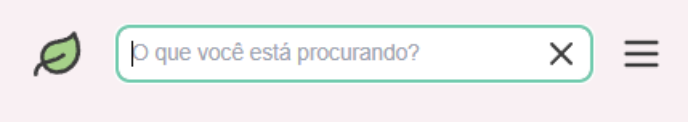
3.2.9 Largest Contentful Paint (LCP) Demorado

Três sites analisados precisam empregar uma técnica dos navegadores que realiza um pré-carregamento (Preload) das páginas da web para entender o conteúdo. Durante esse processo, se os maiores elementos que irão aparecer na interface forem imagens ou conteúdos pesados em termos de recursos de memória, é crucial empregar algumas técnicas para instruir o navegador sobre como lidar com esses elementos, a fim de acelerar seu carregamento. Uma dessas técnicas envolve o uso do atributo `fetchpriority` no HTML, que pode ser aplicado a elementos de imagem para indicar uma prioridade mais alta no carregamento. Para fazer isso, é necessário fornecer o valor `high` para o atributo `fetchpriority`.

3.3 Desenvolvimento do Estudo de Caso

No desenvolvimento do estudo de caso, aplicaram-se as técnicas performáticas para garantir a performance da aplicação. Ao longo do desenvolvimento, surgiram dificuldades em manter performance principalmente na criação do cabeçalho e do menu. O cabeçalho do Leafy possui um recurso interessante, que é incluir o sistema de busca do site no cabeçalho ilustrado na Figura 3, e por esse motivo, se tornou complexo. Por conta de utilizar a interação do usuário para realizar a busca com o texto, o componente precisou ser desenvolvido como um "Client Component" padrão, e as animações do surgimento do campo de busca foi feita com CSS puro. Nesse contexto de animações, também há o menu do site, o qual também utiliza animação para o seu surgimento, e por conta da dificuldade, bibliotecas responsáveis por animação como o Framer Motion foram testadas, porém o foco do projeto era performance, e por esse motivo, foi explorado como realizar essas animações com CSS também, o que resultou em uma ótima performance para o site.[1]

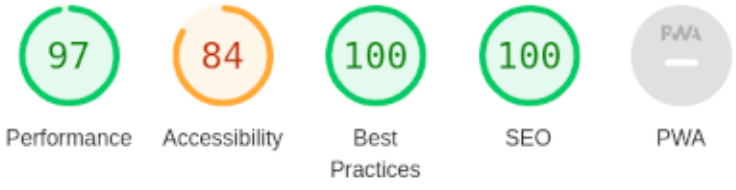
Figura 3 – Cabeçalho do Leafy com a busca expandida



Fonte: do autor

Após o desenvolvimento do projeto, foi realizado um teste com o Lighthouse para avaliar a performance, e o resultado obtido está na Figura 4. A performance se encontra em 97 por conta de um aviso que o Lighthouse retornou relacionado a ela, que seria um alerta do uso de "Client Components", os quais estão sendo utilizados no cabeçalho.

Figura 4 – Análise do Lighthouse do Leafy



Fonte: do autor

4 CONCLUSÃO

O trabalho apresenta uma análise detalhada sobre a performance de aplicações em ReactJS, destacando os principais problemas que afetam a experiência do usuário e apresentando

[1] O código fonte desenvolvido para o estudo de caso Leafy está disponível publicamente no GitHub através do link: https://github.com/huri3l/leafy

soluções para resolvê-los. O objetivo do trabalho era analisar sites desenvolvidos com React, entender os problemas de performance que possuíam e analisar correções, bem como desenvolver um estudo de caso baseado nisso. Durante o desenvolvimento, foi possível identificar que muitos sites criados com essa ferramenta apresentam problemas de desempenho, principalmente em dispositivos móveis, o que pode impactar negativamente a navegação e a usabilidade.

Por esse motivo, o trabalho destaca a importância de se investir em técnicas de otimização de desempenho desde o início do processo de desenvolvimento, a fim de evitar problemas futuros. Além disso, é ressaltada a importância de utilizar todo o conhecimento do React para criar sites performáticos e de alta qualidade.

O objetivo principal foi atingido, pois foi possível encontrar soluções para os erros analisados, criar um projeto de exemplo que não possui tais erros através das soluções propostas, além de ser um material útil e disponível para desenvolvedores que trabalham com ReactJS e desejam melhorar a performance de suas aplicações, através das sugestões de correção dos problemas, a fim de tornar a web um local mais acessível e agradável a todos.

PERFORMANCE ANALISYS AND SOLUTIONS FOR REACTJS APPLICATIONS

Huriel Ferreira Lopes[†] Jorge Luis Boeira Bavaresco[‡]

2023

Abstract

ReactJS, like other web development tools, was created with the intention of improving the web development process. However, certain applications built with this tool, such as e-commerce websites and blogs, suffer from performance issues that greatly impact user experience, such as slow load times, blocking time during navigation, incorrect content loading order, among other usability problems. Therefore, the goal of this paper is to demonstrate the results of a study on websites that exhibited low performance and statistically analyze the frequency and nature of these issues in sites created with React. Regarding the nature of these hindrances, an analysis was conducted to include the causes, consequences, and proposed solutions for these errors through example code in a study case application. The study case was developed with the techniques learned along the research, and proved that the correct usage of them has improved the performance of the project. The ultimate goal of this research is to contribute to the field of Computer Science and Web Development.

Key-words: ReactJS. Web Development. Performance. Blocking Time. Front-end.

Referências

AGGARWAL, S. *Modern Web-Development using ReactJS*. 2018. Disponível em: <http://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf>. Acesso em 26 Outubro 2023. Citado na página 4.

[†]huriel-lopes@outlook.com
[‡]jorgebavaresco@ifsul.edu.br

BERNERS-LEE, T. *World Wide Web Consortium Launches International Program Office for Web Accessibility Initiative*. 1997. Disponível em: <https://www.w3.org/Press/IPO-announce>. Acesso em 09 Agosto 2023. Citado na página 2.

DWIVEDI, P.; KSHAMTA; JOSHI, A. *ReactJS For Trading Applications*. 2019. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9995932>. Acesso em 11 Setembro 2023. Citado na página 4.

EDKINS, A. et al. *Exploring the front-end of project management*. 2013. Disponível em: <https://www.tandfonline.com/doi/full/10.1080/21573727.2013.775942>. Acesso em 27 Outubro 2023. Citado na página 5.

FEDOSEJEV, A. 1. ed. Birmingham: Packt Publishing Ltd, 2015. Acesso em: 08 set 2023. Citado na página 2.

GONG, Y. et al. *The Architecture of Micro-services and the Separation of Frond-end and Back-end Applied in a Campus Information System*. 2020. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9213662>. Acesso em 12 Setembro 2023. Citado na página 5.

GOOGLE. *Lighthouse*. 2016. Disponível em: <https://developer.chrome.com/docs/lighthouse/overview/>. Acesso em 09 Setembro 2023. Citado na página 6.

JAVEED, A. *Performance Optimization Techniques for ReactJS*. 2019. Disponível em: <https://ieeexplore.ieee.org/abstract/document/8869134/>. Acesso em 13 Setembro 2023. Citado na página 8.

MADURAPPERUMA; SHAFANA; SABANI. *State-of-Art Frameworks for Front-end and Back-end Web Development*. 2022. Disponível em: <http://ir.lib.seu.ac.lk/handle/123456789/6339>. Acesso em 28 Outubro 2023. Citado na página 5.

META. *React*. 2023. Disponível em: <https://react.dev/>. Acesso em 07 Setembro 2023. Citado 3 vezes nas páginas 3, 4 e 5.

PAVIć, F.; BRKIć, L. *Methods of Improving and Optimizing React Web-applications*. 2021. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9596762/>. Acesso em 11 Setembro 2023. Citado 2 vezes nas páginas 3 e 8.

SOUSA, M. de; GONçALVES, A. *humanportal – Um caso de estudo usando React.js*. 2020. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9141070/>. Acesso em 16 Setembro 2023. Citado na página 8.

STACKOVERFLOW. *Stack Overflow Developer Survey*. 2022. Disponível em: <https://survey.stackoverflow.co/2022/#section-most-loved-dreaded-and-wanted-web-frameworks-and-technologies>. Acesso em 07 Agosto 2023. Citado 2 vezes nas páginas 2 e 3.

VERCEL. *Next.js*. 2023. Disponível em: <https://nextjs.org/>. Acesso em 07 Setembro 2023. Citado na página 7.