UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

KAYNAN PEREIRA DE SOUSA

TYPESCRIPT E JAVASCRIPT NO DESENVOLVIMENTO WEB COM FRAMEWORKS MODERNOS: UM ESTUDO COMPARATIVO

QUIXADÁ

2026

KAYNAN PEREIRA DE SOUSA

TYPESCRIPT E JAVASCRIPT NO DESENVOLVIMENTO WEB COM FRAMEWORKS MODERNOS: UM ESTUDO COMPARATIVO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Jefferson de Carvalho.

QUIXADÁ

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S697t Sousa, Kaynan Pereira de.
TypeScript e JavaScript no Desenvolvimento Web com Frameworks Modernos: Um Estudo Comparativo / Kaynan Pereira de Sousa. – 2026.
46 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2026.
Orientação: Prof. Dr. Jefferson de Carvalho.

1. Comparativo de Tecnologias. 2. JavaScript. 3. TypeScript . 4. Desenvolvimento Web. 5. Vue.js e Next.js. I. Título.

CDD 004

KAYNAN PEREIRA DE SOUSA

TYPESCRIPT E JAVASCRIPT NO DESENVOLVIMENTO WEB COM FRAMEWORKS MODERNOS: UM ESTUDO COMPARATIVO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: 23/01/2026.

BANCA EXAMINADORA

Prof. Dr. Jefferson de Carvalho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Sidartha Azevedo Lobo de Carvalho
Universidade Federal do Ceará (UFC)

Prof[a]. M[a]. Lana Mesquita
Universidade Federal do Ceará (UFC)

Para todos aqueles que estavam comigo nessa trajetória, me incentivaram e não me deixaram desistir.

AGRADECIMENTOS

Em primeiro lugar, agradeço a Deus por ter me concedido força, perseverança e serenidade ao longo de toda esta trajetória, especialmente nos momentos em que estive distante da família e das pessoas que amo. Sua presença constante foi fundamental para que eu não desistisse e para que este caminho se tornasse menos desgastante.

Aos meus pais, Cicera Helânia Pereira Feitosa Sousa e Francisco Cláudio de Sousa, expresso minha mais profunda admiração e gratidão. Obrigado por todo o apoio, pelos conselhos e pelo incentivo contínuo, que foram essenciais para a construção da minha história acadêmica e pessoal, possibilitando que eu chegasse até aqui.

À minha família, agradeço pelo apoio incondicional ao longo dessa jornada, sempre presente, compartilhando cada conquista e oferecendo ajuda em todos os momentos possíveis.

Aos meus amigos, minha sincera gratidão. O apoio, os incentivos, as conversas, as risadas e os momentos compartilhados tornaram esta etapa significativamente mais leve. A amizade de vocês mostrou-se fundamental, especialmente nos momentos mais desafiadores.

Por fim, agradeço aos professores, pelos ensinamentos, orientações e oportunidades que me foram oferecidos ao longo da graduação. Cada sugestão, crítica e incentivo contribuiu de forma significativa para minha formação acadêmica e profissional, auxiliando no desenvolvimento das competências que me acompanham até hoje.

"Os rios não bebem sua própria água; as árvores não comem seus próprios frutos. O sol não brilha para si mesmo; e as flores não espalham sua fragrância para si. Viver para os outros é uma regra da natureza. (...)A vida é boa quando você está feliz; mas a vida é muito melhor quando os outros estão felizes por sua causa." (Papa Francisco, 2021)

RESUMO

O desenvolvimento *Web* tem ganhado destaque no mercado de tecnologia em função da crescente demanda por aplicações acessíveis, interativas e escaláveis. Nesse contexto, o *JavaScript* consolidou-se como a principal linguagem da *Web*, enquanto o *TypeScript*, como sua evolução tipada, vem sendo amplamente adotado por oferecer maior segurança e organização no desenvolvimento de *software*. Paralelamente, *frameworks* como *Vue.js* e *Next.js* têm se destacado por facilitar a construção de aplicações modernas e eficientes. Este trabalho apresenta uma análise comparativa prática entre o uso das linguagens *JavaScript* e *TypeScript* no desenvolvimento de aplicações *Web* utilizando os *frameworks Vue.js* e *Next.js*. A pesquisa foi conduzida por meio da implementação de quatro variações funcionais de uma mesma aplicação, combinando cada *framework* com ambas as linguagens, garantindo equivalência funcional entre as soluções. A avaliação das implementações foi realizada com o auxílio da ferramenta *Google Lighthouse*, permitindo a obtenção de métricas relacionadas à desempenho, acessibilidade, boas práticas e otimização para mecanismos de busca (SEO), além de uma análise qualitativa baseada na experiência prática do desenvolvedor durante o processo de implementação. Como resultado, o estudo oferece uma visão comparativa que evidencia as vantagens, limitações e cenários de uso mais adequados para cada combinação tecnológica, contribuindo como um guia de apoio à tomada de decisão na escolha de *stacks* de desenvolvimento, considerando fatores como escopo do projeto, escalabilidade, organização do código e esforço de desenvolvimento.

Palavras-chave: Comparativo de Tecnologias; *JavaScript*; *TypeScript*; Desenvolvimento *Web*; *Vue.js* e *Next.js*

ABSTRACT

Web development has gained significant relevance in the technology market due to the increasing demand for accessible, interactive, and scalable applications. In this context, JavaScript has become the core language of the Web, while TypeScript, as its typed superset, has been widely adopted for providing greater safety and code organization. At the same time, frameworks such as Vue.js and Next.js have emerged as prominent tools for building modern and efficient applications. This work presents a practical comparative analysis of the use of JavaScript and TypeScript in Web application development using the Vue.js and Next.js frameworks. The study was conducted through the implementation of four functionally equivalent versions of the same application, combining each framework with both languages to ensure a fair comparison. The evaluation was carried out using the Google Lighthouse tool, which provided metrics related to performance, accessibility, best practices, and search engine optimization (SEO), in addition to a qualitative analysis based on the developer's practical experience during the implementation process. The results offer a comparative perspective that highlights the strengths, limitations, and most suitable usage scenarios for each technological combination, serving as a practical guideline to support decision-making in the selection of development stacks according to project scope, scalability, code organization, and development effort.

Keywords: Technology Comparison; JavaScript; TypeScript; Web Development; Vue.js and Next.js

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
DOM	Document Object Model
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
REST	Representational State Transfer
SEO	Search Engine Optimization
SFC	Single File Component
SPA	Single Page Application
SSG	Static Site Generation
SSR	Server Side Rendering

SUMÁRIO

1 INTRODUÇÃO

A evolução da internet ao longo das últimas décadas transformou radicalmente a maneira como a sociedade se comunica, acessa informações e conduz negócios. O que inicialmente era uma rede restrita a fins acadêmicos e militares, hoje constitui uma infraestrutura global indispensável, integrando praticamente todos os setores da vida moderna. Nesse contexto, os sistemas *Web* e os aplicativos móveis emergiram como ferramentas essenciais, acompanhando o ritmo acelerado da digitalização e da mobilidade. Atualmente, essas tecnologias estão presentes nas mais diversas esferas do cotidiano, desde simples interações sociais em redes digitais até operações bancárias, serviços governamentais e plataformas de ensino a distância. O crescimento exponencial desses sistemas reflete não apenas os avanços tecnológicos, mas também a crescente demanda por soluções acessíveis, dinâmicas e centradas no usuário (Statista Research Department, 2025).

Diante da onipresença da internet e da ampla disseminação de dispositivos conectados, o desenvolvimento de *software* voltado para a plataforma *Web* assume uma importância estratégica e cada vez mais crítica. A *Web* consolidou-se como o principal canal pelo qual empresas, instituições e governos se comunicam com seu público, disponibilizam serviços, promovem produtos e fortalecem suas marcas. Programar para a *Web* não é apenas uma competência técnica desejável, mas uma necessidade diante da demanda por soluções acessíveis, escaláveis e multiplataformas. Além disso, a programação *Web* está no cerne da inovação tecnológica, sendo um dos pilares da transformação digital que redefine modelos de negócio, otimiza processos e amplia o alcance de serviços nos mais variados setores da economia e da sociedade (Miletto; Bertagnolli, 2014; Ferreira *et al.*, 2024).

No cenário do desenvolvimento *Web*, a linguagem *JavaScript* destaca-se como a linguagem de maior destaque para a criação de interfaces dinâmicas e interativas (Flanagan, 2012). Presente em praticamente todos os navegadores modernos, ele se consolidou como o principal recurso para dar vida às páginas web, permitindo desde animações simples até aplicações complexas de página única a Single Page Application (SPA) (Marcio Hanashiro, 2024). Com o surgimento do Node.js, o *JavaScript* também ultrapassou as fronteiras do front-end e passou a ser utilizado no back-end, possibilitando o desenvolvimento full stack com uma única linguagem. Nesse contexto, o *TypeScript* surge como uma poderosa evolução: um super conjunto do *JavaScript* que incorpora tipagem estática e outros recursos avançados. Ao introduzir uma camada de segurança e previsibilidade ao código, o *TypeScript* facilita a manutenção, a

escalabilidade e o desenvolvimento colaborativo, o que o torna especialmente atrativo para projetos de médio e grande porte no ecossistema web moderno (Awari, 2023; Typescriptutorial, 2025).

Com o amadurecimento do desenvolvimento *Web* e a crescente complexidade dos projetos, surgiram *frameworks* e bibliotecas que têm como objetivo acelerar a produção de *software*, padronizar processos e incentivar boas práticas de programação. No universo *JavaScript*/*TypeScript*, destacam-se ferramentas amplamente utilizadas tanto no *front-end* quanto no *back-end*. No desenvolvimento de interfaces, bibliotecas como o *React* e *frameworks* como *Angular* e *Vue.js* se consolidaram como escolhas populares por sua capacidade de modularizar o código, facilitar a criação de componentes reutilizáveis e promover interfaces dinâmicas e responsivas. No *back-end*, o *Node.js*[1] aliado ao *Express.js*[2] oferece uma estrutura leve e eficiente para criação de servidores e APIs, possibilitando uma arquitetura moderna e escalável com *JavaScript* em ambos os lados da aplicação. Essas tecnologias abstraem grande parte da complexidade inerente ao desenvolvimento *Web*, permitindo que os desenvolvedores foquem na lógica de negócio e entreguem soluções mais rápidas, robustas e sustentáveis (Aagam, 2024).

Apesar do avanço nas ferramentas de desenvolvimento e do amplo ecossistema de tecnologias disponíveis, a escolha inicial de uma linguagem, *framework* ou biblioteca para iniciar um projeto *Web* ou móvel continua sendo uma das etapas mais críticas e desafiadoras. A diversidade de opções pode confundir até mesmo desenvolvedores experientes, especialmente quando se leva em conta a necessidade de criar aplicações responsivas, que funcionem bem em diferentes dispositivos e tamanhos de tela. Uma escolha inadequada pode acarretar sérias consequências, como comprometimento do desempenho, dificuldades futuras na manutenção do código, aumento dos custos de desenvolvimento e, em casos mais graves, o fracasso do projeto. Assim, a decisão tecnológica precisa considerar não apenas a curva de aprendizado e a popularidade da ferramenta, mas também sua capacidade de atender às exigências de responsividade, escalabilidade e sustentabilidade do sistema ao longo do tempo (Gizas *et al.*, 2012).

Diante da complexidade envolvida na escolha de tecnologias para o desenvolvimento *Web* e móvel, diversos pesquisadores e profissionais têm se dedicado a analisar e comparar diferentes linguagens, *frameworks* e bibliotecas. Esses estudos visam fornecer subsídios para decisões mais embasadas, considerando fatores como desempenho, curva de aprendizado, po-

[1] **Node.js:** https://nodejs.org/pt/learn/getting-started/introduction-to-nodejs
[2] **Express.js:** https://expressjs.com/pt-br/

pularidade, ecossistema e adequação a tipos específicos de projeto. Por exemplo, um estudo comparativo realizou uma análise entre JavaScript e TypeScript, abordando aspectos tanto quantitativos quanto qualitativos dessas linguagens, analisando dez características qualitativas distintivas e realizando testes quantitativos de consumo de memória e tempo de execução em algoritmos análogos (Mororó, 2024). Além disso, análises detalhadas, como a comparação entre *Angular* e *React* publicada pela Kinsta, exploram aspectos como tempo de carregamento, scripting e renderização, auxiliando desenvolvedores na escolha da tecnologia mais adequada às necessidades do projeto. Essas comparações são fundamentais para orientar decisões tecnológicas que impactam diretamente na qualidade e sustentabilidade das aplicações desenvolvidas (Powell, 2023).

Este trabalho se propõe a oferecer uma análise que auxilie desenvolvedores, gestores de projetos e demais interessados na escolha mais adequada de tecnologias para o desenvolvimento de aplicações Web e móveis. Por meio de um panorama comparativo entre diferentes linguagens, *frameworks* e bibliotecas, com foco na responsividade e na adaptabilidade aos diversos contextos de uso, busca-se contribuir para decisões mais estratégicas e bem fundamentadas.

Para isso, é apresentada uma análise comparativa prática entre o uso das linguagens *JavaScript* e *TypeScript* no desenvolvimento de aplicações *Web* utilizando os *frameworks Vue.js* e *Next.js*. A pesquisa foi conduzida a partir da implementação de quatro variações funcionais de uma mesma aplicação, avaliadas por meio da ferramenta *Google Lighthouse*, que forneceu métricas relacionadas à desempenho, acessibilidade, boas práticas e otimização para mecanismos de busca o Search Engine Optimization (SEO). Além dos dados técnicos, a análise considera aspectos práticos observados durante o desenvolvimento, como curva de aprendizado, facilidade de implementação e adequação a diferentes contextos de projeto.

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica necessária para a compreensão dos principais conceitos abordados. O Capítulo 3 discute os trabalhos relacionados, com destaque para estudos comparativos relevantes na área. O Capítulo 4 descreve a metodologia adotada para o desenvolvimento da pesquisa. No Capítulo 5 são apresentados e analisados os resultados obtidos a partir das implementações realizadas. Por fim, o Capítulo 6 apresenta as considerações finais e as perspectivas para trabalhos futuros.

1.1 Objetivo Geral

Analisar e comparar, de forma prática, o uso das linguagens *JavaScript* e TypeScript em conjunto com os *frameworks Vue.js* e *Next.js* no desenvolvimento de aplicações *Web*, considerando métricas técnicas de qualidade e a experiência de desenvolvimento, com o objetivo de fornecer subsídios que auxiliem na escolha da combinação tecnológica mais adequada a diferentes cenários de projeto.

1.1.1 Objetivos Específicos

- Implementar uma mesma aplicação *Web* utilizando quatro combinações tecnológicas distintas: *Vue.js* com *JavaScript*, *Vue.js* com *TypeScript*, *Next.js* com *JavaScript* e *Next.j*s com *TypeScript*;
- Avaliar as implementações por meio da ferramenta *Google Lighthouse*, analisando métricas de desempenho, acessibilidade, boas práticas e SEO;
- Analisar aspectos qualitativos relacionados ao processo de desenvolvimento, como curva de aprendizado, organização do código e facilidade de manutenção;
- Comparar os resultados obtidos entre as diferentes combinações de linguagem e *framework*;
- Propor um guia de apoio à decisão tecnológica, indicando cenários nos quais cada combinação analisada se mostra mais adequada.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir neste Capítulo, serão expostos os conceitos fundamentais que embasam a criação e implementação do projeto apresentado. Inicialmente, a Seção 2.1 contextualizará o Desenvolvimento *Web*. Em seguida, a Seção 2.2 abordará a linguagem *JavaScript* , e a Seção 2.3 se aprofundará no *TypeScript*. Por fim, a Seção 2.4 explorará os *Frameworks* e Bibliotecas relevantes como *React*, *Next.js* e *Vue.js* , o papel das *APIs* nesse contexto, e as considerações sobre as tecnologias que norteiam este estudo.

2.1 Desenvolvimento Web

O desenvolvimento *web* refere-se ao processo de criação e manutenção de sites e aplicações acessíveis por meio de navegadores, sendo uma das áreas mais dinâmicas e em constante evolução dentro da tecnologia da informação. Ele abrange tanto o *front-end* (interface com o usuário) quanto o *back-end* (lógica de negócios e comunicação com bancos de dados), exigindo conhecimentos em diversas linguagens, *frameworks*, bibliotecas e práticas de design responsivo (Miletto; Bertagnolli, 2014).

Nos últimos anos, o desenvolvimento *web* passou de páginas estáticas simples para aplicações complexas e interativas, capazes de oferecer experiências ricas ao usuário (Turemuratova, 2025). Esse avanço foi impulsionado pela crescente demanda por soluções acessíveis, multiplataforma e escaláveis. Nesse cenário, a *Web* consolidou-se como um canal essencial para negócios, instituições públicas e interações sociais, assumindo papel estratégico na transformação digital.

A evolução do desenvolvimento *web* também trouxe consigo o aumento da complexidade dos projetos, exigindo abordagens mais organizadas e eficientes, como a divisão em componentes reutilizáveis, integração contínua, e o uso de ferramentas modernas de versionamento e automação. Essa transformação contribuiu para o surgimento de uma vasta gama de tecnologias específicas, voltadas para diferentes etapas do desenvolvimento, e tornou o domínio dessas ferramentas uma competência indispensável para profissionais da área (Ferreira *et al.*, 2024).

2.2 JavaScript

JavaScript é uma linguagem de programação de alto nível, interpretada e orientada a objetos, criada em 1995 por *Brendan Eich* enquanto trabalhava na *Netscape Communications*. Inicialmente concebida para adicionar interatividade e dinamismo às páginas HyperText Markup Language (HTML), sua criação veio como resposta à necessidade de uma linguagem de *script* que fosse executada diretamente no navegador do cliente. A primeira versão foi lançada como parte do navegador *Netscape Navigator*, com o nome inicial de *LiveScript*, sendo posteriormente rebatizada para *JavaScript* por motivos de *marketing*, aproveitando a popularidade da linguagem *Java* na época. Com o passar dos anos, a linguagem evoluiu significativamente, tornando-se um padrão oficial com a criação da especificação *ECMAScript*[1], mantida atualmente pela *ECMA International*. Essa padronização permitiu que navegadores distintos implementassem o *JavaScript* de forma mais consistente, incentivando seu crescimento e consolidando-o como uma linguagem essencial para o desenvolvimento *Web* moderno (Deno, 2025).

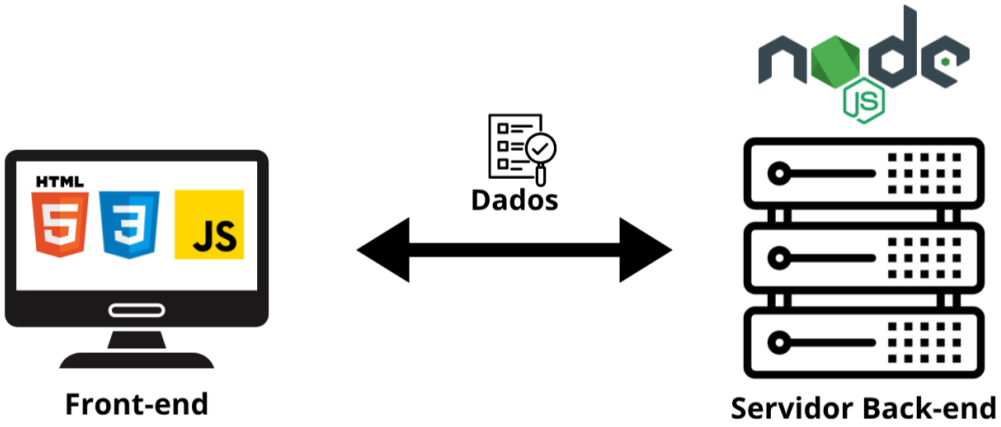
JavaScript é conhecido por sua flexibilidade e por permitir o desenvolvimento de aplicações interativas e dinâmicas diretamente no navegador. Suas principais características incluem a tipagem dinâmica, que permite alterar o tipo de uma variável durante a execução, e a interpretação imediata, sem necessidade de compilação prévia. Adicionalmente, funções são tratadas como cidadãos de primeira classe, o que significa que podem ser atribuídas a variáveis, passadas como argumentos e retornadas de outras funções. A linguagem também se destaca por seu modelo de evento assíncrono, especialmente com o uso de callbacks, Promises e async/await , e pela manipulação direta do Document Object Model (DOM), que possibilita atualizações em tempo real na interface do usuário. Essas características permitem uma programação altamente expressiva e rápida, embora possam introduzir desafios em aplicações de grande porte, como dificuldade de manutenção e erros em tempo de execução. Tradicionalmente associado ao ambiente de navegador, o *Web* ultrapassou essa limitação com o surgimento do *Node.js*, em 2009.

O *Node.js* é uma plataforma que permite a execução de código *JavaScript* no lado do servidor, utilizando o motor V8 do *Google Chrome*. Com isso, tornou-se possível utilizar *JavaScript* para o desenvolvimento de aplicações *full-stack*, empregando a mesma linguagem tanto no *front-end* quanto no *back-end*. Essa mudança transformou o ecossistema da linguagem, dando origem a uma série de *frameworks*, bibliotecas e ferramentas que permitem desenvolver *APIs*, servidores *Web*, aplicações em tempo real (como *chats*), e até mesmo aplicativos *desktop* e

[1] **ECMAScript:** https://ecma-international.org/technical-committees/tc39/

mobile. A Figura 1 ilustra a atuação do *JavaScript* no cliente e no servidor com o uso de *Node.js*

Figura 1 – atuação do JavaScript no *client* e *server* (com uso de *Node.js*).



Fonte: Elaborado pelo autor.

2.3 TypeScript

TypeScript surgiu com o propósito de ampliar e aperfeiçoar as capacidades do JavaScript, oferecendo um conjunto de funcionalidades que proporcionam mais previsibilidade, clareza e segurança durante o desenvolvimento. Criado pela *Microsoft* e lançado oficialmente em 2012, o *TypeScript* foi pensado como uma ferramenta para enfrentar os desafios que emergem à medida que aplicações *JavaScript* crescem em tamanho e complexidade (Ritika, 2024).

A linguagem mantém compatibilidade com o *JavaScript*, pois todo código *TypeScript* é convertido, ou "transpilado", para *JavaScript* tradicional antes de ser executado. Dessa forma, é possível escrever código mais organizado e com menos propensão a erros, sem perder a ampla compatibilidade que o *JavaScript* oferece com navegadores e plataformas (bluebirdinternational, 2024).

A principal característica que distingue o *TypeScript* do *JavaScript* é a possibilidade de usar tipagem estática. Essa tipagem não é obrigatória, mas altamente recomendada, permitindo que o desenvolvedor defina previamente o tipo de cada variável, argumento de função ou valor de retorno. Isso torna o código mais descritivo e facilita tanto o entendimento quanto a manutenção do sistema. Além de evitar falhas comuns em tempo de execução, o uso de tipos permite que ferramentas de desenvolvimento identifiquem problemas logo durante a escrita do código. Isso se traduz em *feedback* imediato por parte do editor, melhor suporte a autocompletar, e uma experiência de codificação mais segura e fluida. Outros recursos avançados também são

oferecidos, como interfaces, tipos genéricos, *decorators* e *namespaces*, que ajudam a estruturar melhor o projeto e a aplicar princípios da programação orientada a objetos e funcional.

TypeScript tem se consolidado como uma escolha preferencial para projetos de médio a grande porte, especialmente onde a escalabilidade e a legibilidade do código são cruciais. Empresas que buscam mais controle sobre suas bases de código e equipes de desenvolvimento que valorizam organização e produtividade encontram na linguagem um aliado importante. *Frameworks* modernos como *Angular* utilizam *TypeScript* como base, justamente por sua robustez e pela integração facilitada com padrões arquiteturais complexos. Outros ambientes e bibliotecas, como *React* e *Vue*, também oferecem suporte pleno ao *TypeScript*, o que tem contribuído significativamente para sua disseminação. As Figuras 2 e 3 apresentam exemplos de código de uma função em *JavaScript* e *TypeScript*, respectivamente, para ilustrar essa diferença.

Figura 2 – Código de uma função em JavaScript.

```javascript
function somaDasRaizes(a, b) {
  return Math.sqrt(a) + Math.sqrt(b);
}

// Exemplo de uso:
console.log(somaDasRaizes(9, 16)); // 3 + 4 = 7
```

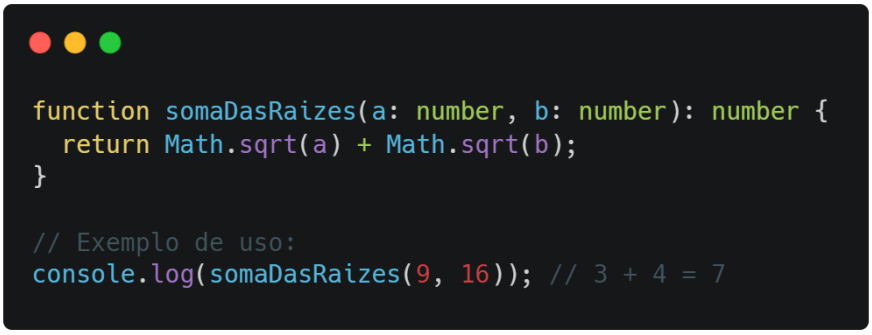
Fonte: Elaborado pelo autor.

2.4 Frameworks e Bibliotecas

Frameworks e bibliotecas são essenciais para acelerar o desenvolvimento e facilitar a manutenção de aplicações modernas. Dentre os mais utilizados estão o *React*[2], *Vue.js*[3] e *Next.js*[4], cada um com características específicas (Aagam, 2024).

[2] **React:** https://react.dev
[3] **Vue.js:** https://vuejs.org
[4] **Next.js:** https://nextjs.org/docs

Figura 3 – Código de uma função em TypeScript.



```typescript
function somaDasRaizes(a: number, b: number): number {
  return Math.sqrt(a) + Math.sqrt(b);
}

// Exemplo de uso:
console.log(somaDasRaizes(9, 16)); // 3 + 4 = 7
```

Fonte: Elaborado pelo autor.

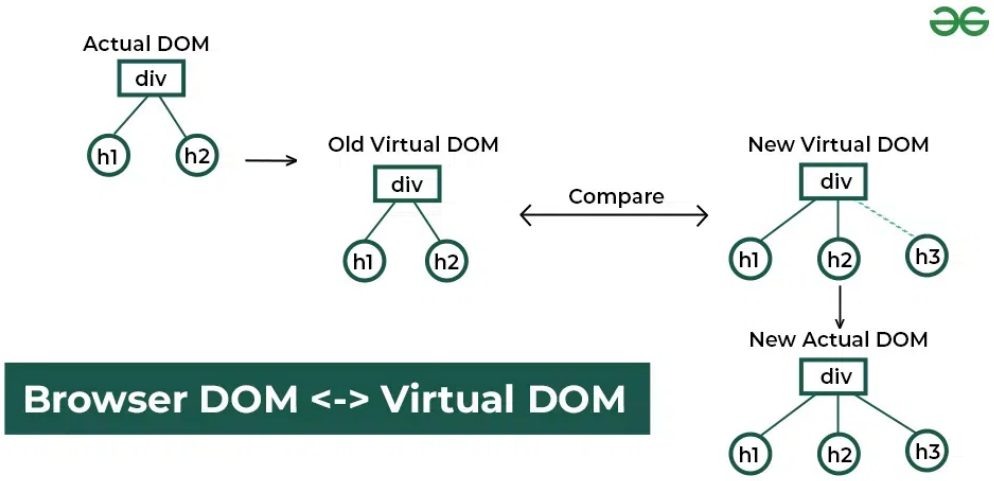
2.4.1 React

O *React* é uma biblioteca *JavaScript* de código aberto desenvolvida pelo *Facebook* (atualmente Meta), lançada em 2013. Foi criado inicialmente para lidar com a complexidade crescente da interface do usuário de aplicações *Web* modernas, permitindo que desenvolvedores criem interfaces reativas e performáticas de forma mais simples e declarativa.

Um dos marcos principais do *React* foi a introdução do conceito de *Virtual* DOM, que revolucionou a maneira como as atualizações na interface do usuário são gerenciadas. Ao invés de manipular diretamente o DOM real, o *React* cria uma representação virtual da interface e realiza atualizações de forma eficiente, minimizando a quantidade de interações com o DOM real, conhecido por ser um processo custoso em termos de desempenho (Banks; Porcello, 2020). Como mostrado na Figura 4 abaixo.

A arquitetura do *React* é baseada em componentes reutilizáveis, onde cada parte da interface é um componente isolado, podendo ser funcional ou baseado em classes, embora os componentes funcionais com *hooks* sejam mais utilizados atualmente. Seus principais conceitos incluem o JSX (*JavaScript* XML), uma sintaxe que permite escrever *HTML* dentro do *JavaScript*, tornando o código mais legível e intuitivo. Também adota o Unidirectional *Data Flow* (fluxo de dados unidirecional), onde os dados fluem do componente pai para os filhos através de props, o que facilita o controle e depuração do estado da aplicação. Os Hooks, introduzidos a partir da versão 16.8, como *useState*, *useEffect* e *useContext*, permitem manipular o estado e o ciclo de vida em componentes funcionais, sem necessidade de classes. *React* possui um ecossistema robusto e em constante expansão, com ferramentas como *React Router* para gerenciamento de

Figura 4 – React virtual DOM.



Fonte: geeksforgeeks (2025)

rotas , *Redux* e *Context API* para gerenciamento de estado , e *React Query* para gerenciamento de dados assíncronos. Integradores como *Vite*, *Webpack* e *Babel* também fazem parte desse ecossistema, tornando o *React* altamente personalizável e adaptável a projetos de diversas escalas. Além disso, com a introdução do *React Native*, é possível utilizar os mesmos conceitos da biblioteca para desenvolvimento de aplicativos móveis nativos (Powell, 2023).

2.4.2 Next.js

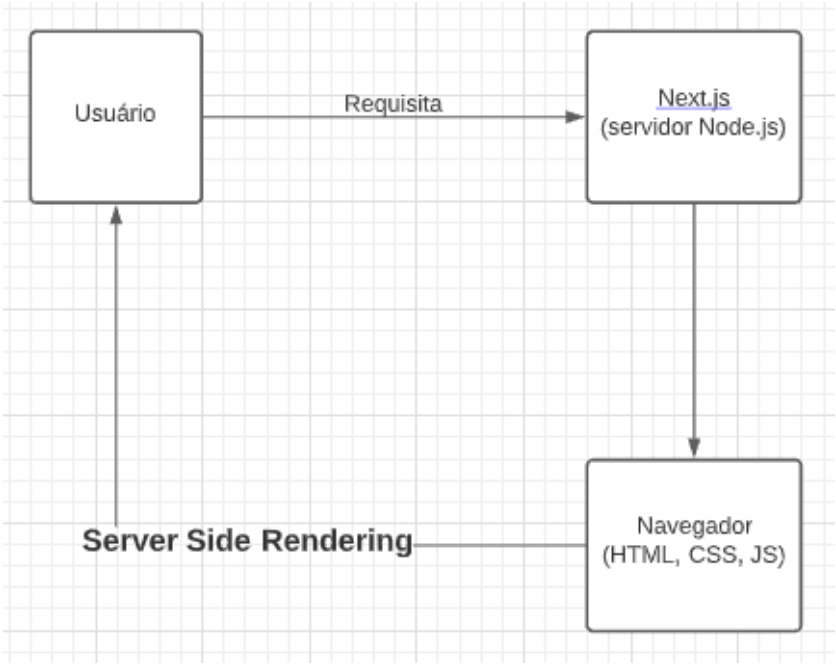
O *Next.js* é um framework de *React* criado pela empresa *Vercel*, com o objetivo de fornecer uma solução completa para aplicações web modernas, combinando o poder do *React* com funcionalidades como renderização no servidor Server Side Rendering (SSR) e geração estática de páginas Static Site Generation (SSG).

Lançado em 2016, o *Next.js* resolveu várias limitações enfrentadas por aplicações *React* tradicionais no que diz respeito a desempenho, SEO e estrutura de roteamento. O diferencial do *Next.js* está em sua abordagem híbrida de renderização e em sua organização orientada a convenções, o que simplifica o desenvolvimento de aplicações escaláveis.

Algumas de suas características importantes incluem o roteamento baseado em arquivos, onde cada arquivo dentro da pasta *pages/* representa uma rota. Permite também a renderização híbrida, possibilitando misturar SSR, SSG e do Client-Side Rendering (CSR) em uma mesma aplicação. Outros recursos são a *Incremental Static Regeneration* (ISR), que é a capacidade de regenerar páginas estáticas dinamicamente , e as *API Routes*, que facilitam a criação de rotas de API diretamente no projeto *Next.j*s, permitindo a criação de um *back-end*

leve integrado. Como mostrado na figura 5 vejamos como funciona o fluxo do SSR

Figura 5 – Fluxo SSR(*Server Side Rendering*).



Fonte: geeksforgeeks (2025).

Next.js é amplamente adotado por grandes empresas como *Nike*, *OpenIA*, *Netflix Jobs* e muitas outras, sendo reconhecido como a principal escolha para projetos *React* que demandam desempenho e SEO (Vercel, 2026). Seu ecossistema inclui ferramentas como *Image Optimization* para carregamento eficiente de imagens e *Middleware* para interceptação de requisições. A nova estrutura do App Router oferece rotas aninhadas e *layouts* persistentes com maior flexibilidade.

2.4.3 Vue.js

O *Vue.js* é um *framework* progressivo de *JavaScript* criado por Evan You, lançado em 2014. Sua proposta é oferecer uma estrutura leve, reativa e de fácil integração com outras bibliotecas e projetos existentes, sendo ideal tanto para aplicações pequenas quanto grandes sistemas corporativos.

Evan You, ex-funcionário da *Google*, desenvolveu o *Vue.js* após trabalhar com *AngularJS*, com o objetivo de criar algo mais leve, simples e que mantivesse a flexibilidade sem abrir mão do poder expressivo das interfaces dinâmicas. *Vue.js* é projetado para ser adotado de forma incremental, o que significa que se pode começar com funcionalidades básicas e, conforme o projeto cresce, utilizar recursos avançados do framework.

Entre os conceitos fundamentais do *Vue.js* estão os componentes reativos, simi-

larmente ao React, onde o *Vue.js* também é baseado em componentes reutilizáveis. Utiliza *templates* declarativos, com uma sintaxe baseada em *HTML* para definir o *layout* da interface , e possui reatividade automática, na qual o sistema de observação do *Vue.js* detecta automaticamente alterações de estado e atualiza a interface de forma eficiente. O *Vue.js* também oferece *Two-Way Data Binding*, um sistema de ligação bidirecional semelhante ao *Angular*, que facilita formulários e entrada de dados (Cuomo; Lee, 2024). O *Vue.js* possui ferramentas oficiais para rotas (*Vue Router*), gerenciamento de estado (Pinia, anteriormente Vuex), e empacotamento de projetos (*Vite*, *Vue CLI*). É amplamente utilizado por empresas e comunidades na China, Europa e América Latina. Por ser mais opinativo e com documentação extremamente clara, é muitas vezes preferido por desenvolvedores iniciantes e por equipes que buscam produtividade rápida sem abrir mão da escalabilidade.

2.4.4 API

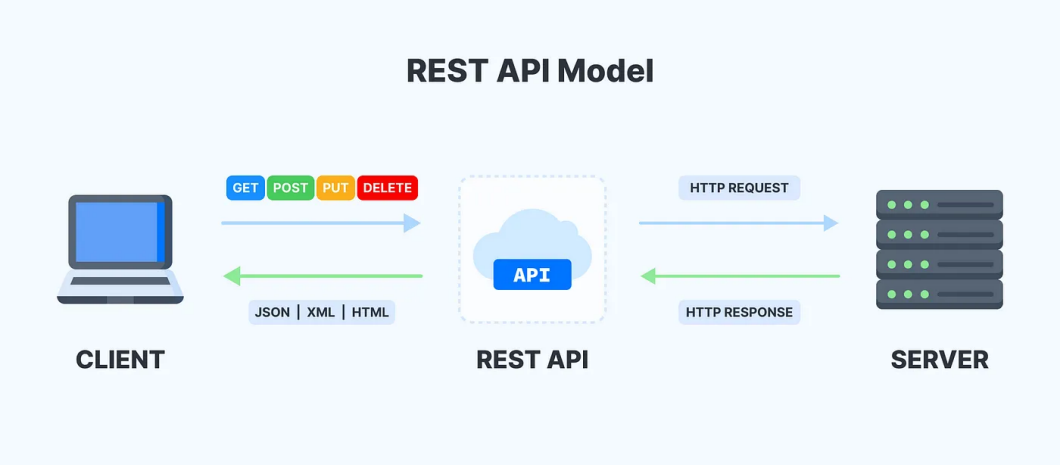
No desenvolvimento de *software*, a Application Programming Interface (API) são essenciais, funcionando como contratos que permitem a comunicação e troca de dados entre diferentes sistemas de *software*. Elas definem regras e protocolos para interação, abstraindo a complexidade interna e simplificando o uso de funcionalidades. No contexto *Web*, as APIs são cruciais para que aplicações, como navegadores ou serviços de terceiros, possam trocar informações de forma padronizada e eficiente.

A relevância das APIs no desenvolvimento *Web* moderno é inegável. Elas sustentam arquiteturas como microsserviços, onde componentes independentes se comunicam via APIs, promovendo escalabilidade (Fowler, 2014). As APIs também são vitais para melhorar a experiência do usuário, possibilitando interações dinâmicas (como com AJAX) e a integração de serviços externos, o que acelera o ciclo de desenvolvimento ao permitir o reuso de funcionalidades já existentes.

Dentre os diversos tipos e estilos arquiteturais de APIs, o Representational State Transfer (REST) é um dos mais proeminentes no desenvolvimento *Web*. As *APIs RESTful* são projetadas com base em princípios como a separação cliente-servidor, a ausência de estado (*statelessness*), ou seja,onde cada requisição do cliente para o servidor deve conter toda a informação necessária para ser compreendida, sem que o servidor precise lembrar de estados anteriores, e a capacidade de cache, que otimiza as interações e melhora a desempenho (Fielding, 2000). Elas utilizam os métodos padrão do protocolo *HTTP* (como *GET*, *POST*, *PUT*, *DELETE*) para

realizar operações sobre os recursos e geralmente trocam dados em formatos como JavaScript Object Notation (JSON) [5] ou XML (Extensible Markup Language)[6]. A flexibilidade e a independência de tecnologia das *APIs RESTful* permitem que cliente e servidor evoluam de forma independente, tornando-as uma escolha popular para construir serviços web escaláveis e de fácil integração. Vejamos abaixo por meio da figura 6 como segue o funcionamento de uma comunicação Cliente/Servidor por meio de APIs REST.

Figura 6 – Funcionamento de uma API REST



Fonte: geeksforgeeks (2025)

No ecossistema *JavaScript/TypeScript*, *Node.js*, com *frameworks* como *Express.js*[7], é frequentemente usado para construir o lado servidor de APIs, definindo endpoints e lógica de negócios. *Frameworks front-end* como *React*, *Vue.js* e *Angular* consomem essas APIs para buscar e exibir dados. O *Next.js*, por exemplo, como mostrado anteriormente, integra a criação de APIs com as "*API Routes*", permitindo adicionar funcionalidades de *back-end* diretamente ao projeto. O formato de dados predominante nessas trocas é o JSON, devido à sua simplicidade e facilidade de parseamento em *JavaScript*, sendo mais leve que o XML, ainda usado em alguns contextos.

Finalmente, a segurança é um aspecto crítico em APIs. A autenticação (verificar quem faz a requisição) e a autorização (determinar se há permissão para acessar um recurso) são fundamentais. Métodos como *tokens* JWT[8] e OAuth 2.0[9] são comuns, e a transmissão de dados deve sempre ocorrer sobre HTTPS para proteger informações sensíveis.

[5] **JSON:** https://www.json.org/json-en.html
[6] **XML:** https://en.wikipedia.org/wiki/XML
[7] **Express.js:** https://expressjs.com/pt-br/
[8] **JWT:** https://jwt.io
[9] **OAuth 2.0:** https://oauth.net/2/

2.4.5 Google Lighthouse

O *Google Lighthouse* é uma ferramenta de auditoria automatizada desenvolvida pelo *Google* com o objetivo de avaliar a qualidade de aplicações *Web* sob diferentes aspectos técnicos e de experiência do usuário. Integrado ao navegador *Google Chrome*, o *Lighthouse* permite a execução de análises padronizadas que resultam em relatórios detalhados, fornecendo indicadores quantitativos e qualitativos sobre o estado da aplicação avaliada.

Entre os principais critérios analisados pelo *Lighthouse* destacam-se métricas relacionadas à desempenho, acessibilidade, conformidade com boas práticas de desenvolvimento e otimização para mecanismos de busca o SEO. A avaliação de desempenho considera fatores como tempo de carregamento, velocidade de renderização e eficiência no uso de recursos, enquanto a análise de acessibilidade verifica o atendimento a diretrizes que visam garantir o acesso adequado a usuários com diferentes necessidades. As métricas de boas práticas abordam aspectos de segurança, uso adequado de tecnologias *Web* modernas e padrões recomendados, e a análise de SEO avalia a estrutura da aplicação quanto à sua visibilidade em mecanismos de busca.

Por sua padronização e ampla adoção na indústria, o *Google Lighthouse* tem sido utilizado tanto no contexto profissional quanto acadêmico como referência para avaliação técnica de aplicações *Web*. Sua capacidade de gerar relatórios comparáveis, aliada à facilidade de uso e integração com ambientes de desenvolvimento e produção, torna a ferramenta adequada para estudos comparativos que buscam analisar diferentes tecnologias, arquiteturas ou abordagens de implementação, contribuindo para decisões mais fundamentadas no processo de desenvolvimento de software.

2.4.6 Considerações sobre as tecnologias

As linguagens *JavaScript* e *TypeScript*, os *frameworks* e bibliotecas como *React*, *Next.js* e *Vue.js*, juntamente com o uso de APIs para a comunicação de dados, constituem o núcleo das tecnologias abordadas nesta seção e são amplamente empregadas no ecossistema de desenvolvimento web moderno. A compreensão aprofundada de suas arquiteturas, funcionalidades distintas, vantagens e limitações inerentes é, portanto, crucial para a tomada de decisões estratégicas e bem fundamentadas ao selecionar as ferramentas mais adequadas para as especificidades de cada projeto.

Nesse sentido, ao longo deste trabalho foi conduzida uma análise comparativa prática, baseada no desenvolvimento e na avaliação de uma aplicação *Web* implementada de forma independente utilizando os *frameworks Vue.js* e *Next.js*. Um dos elementos centrais dessa análise consistiu no consumo de dados provenientes de uma API pública, comum a todas as implementações, o que possibilitou avaliar e contrastar as diferentes abordagens de desenvolvimento adotadas, bem como aspectos relacionados à produtividade e ao desempenho das aplicações resultantes.

3 TRABALHOS RELACIONADOS

Neste Capítulo, serão apresentados alguns trabalhos relacionados com o projeto proposto, a fim de analisar diferenças e semelhanças de estudo. A análise visa identificar abordagens similares, metodologias aplicadas e os focos investigativos adotados por outros autores.

3.1 *Análise Comparativa de Tecnologias JavaScript Focadas no Front-End para Desenvolvimento Web*

O estudo de Martins Filho (2023), apresenta uma análise comparativa entre React, Vue e Next.js no desenvolvimento front-end. A pesquisa envolveu a construção de uma aplicação (uma Pokédex) com cada uma dessas tecnologias, analisando aspectos como documentação, configuração do ambiente, suporte, atividade das comunidades, curva de aprendizagem e inserção no mercado de trabalho. Também foram avaliados dados de desempenho, como tempo de renderização, tamanho e tempo de build e tempo de acesso à API. O autor conduziu ainda uma pesquisa de campo com desenvolvedores experientes, oferecendo uma visão abrangente sobre as particularidades de cada *framework*.

Esse trabalho se assemelha ao presente estudo por realizar uma análise comparativa de frameworks como *Vue.js* e *Next.js*, com base na implementação de uma aplicação e na coleta da percepção de desenvolvedores. Em ambos os casos, o desempenho é avaliado por meio de métricas como tempo de renderização e acesso a dados. A principal diferença, no entanto, está no foco do presente trabalho, que aprofunda a análise comparando versões *JavaScript* e *TypeScript* de cada *framework*, algo não abordado por Martins Filho (2023), cuja investigação se concentra nos *frameworks* como unidades isoladas, incluindo React.

3.2 *Análise comparativa entre frameworks front-end baseados em Javascript para aplicações web*

A pesquisa de Ferreira e Zuchi (2018), também oferece uma comparação entre tecnologias populares como *AngularJS*, *React*, *Angular* e *Vue*. Os autores desenvolveram aplicações similares com funcionalidades de formulário, listagem, edição e exclusão de dados, e analisaram elementos como arquitetura, documentação, suporte da comunidade, compatibilidade entre navegadores, tamanho dos pacotes e tempo de renderização. Os resultados indicaram que

cada *framework* se destacou em diferentes aspectos — o React, por exemplo, gerou o menor pacote de arquivos, enquanto o *Vue* apresentou melhor desempenho em renderização.

Esse estudo se aproxima do presente trabalho ao empregar a implementação prática de aplicações para fins comparativos, analisando fatores relacionados ao desenvolvimento e desempenho, com o objetivo de fornecer subsídios para a escolha de tecnologias. A distinção principal está no escopo: Ferreira e Zuchi compararam três frameworks utilizando exclusivamente *JavaScript*, enquanto este trabalho se dedica a examinar as versões em JavaScript e TypeScript dos *frameworks Vue.js* e *Next.js*, com foco nos impactos da tipagem estática no processo de desenvolvimento.

3.3 *Um estudo comparativo entre JavaScript e TypeScript*

Mororó (2024), em seu trabalho, foca diretamente na comparação entre as duas linguagens. A metodologia envolve revisão bibliográfica, implementação de algoritmos semelhantes em ambas as linguagens e a coleta de dados quantitativos (como consumo de memória e tempo de execução), além de uma pesquisa qualitativa com desenvolvedores. Os resultados mostram que, embora o desempenho em tempo de execução e memória seja semelhante, o *TypeScript* apresenta vantagens significativas em aspectos como legibilidade, manutenibilidade e curva de aprendizado. Para ilustrar as diferenças, Mororó desenvolveu uma aplicação (jogo Flappy Bird) em ambas as linguagens.

Esse estudo dialoga com o presente trabalho ao explorar a comparação entre *JavaScript* e *TypeScript* e ao utilizar tanto métodos qualitativos quanto implementações práticas. No entanto, difere por focar nas linguagens de forma isolada, enquanto o presente estudo analisa a interação dessas linguagens com frameworks específicos, observando como a escolha entre JS e TS afeta o desenvolvimento de aplicações reais com *Vue.js* e *Next.js*. Além disso, enquanto Mororó utiliza algoritmos e jogos simples como base prática, o presente estudo foca na construção de uma aplicação simples com *frameworks* modernos.

3.4 *Migrating from JavaScript to TypeScript and its advantages*

O trabalho de Holmberg (2023), traz uma abordagem prática baseada na experiência de migração de um sistema real de *JavaScript* para *TypeScript*. O autor analisa as principais vantagens da tipagem estática oferecida pelo TS, como maior segurança e qualidade do código, e

relata dificuldades comuns enfrentadas no processo, como o alto volume de erros e incompatibilidades com bibliotecas de terceiros. O estudo é enriquecido com entrevistas realizadas com desenvolvedores que vivenciaram o processo de migração.

Apesar de compartilhar com o presente estudo o interesse na comparação entre *JavaScript* e *TypeScript* e os benefícios do uso de TypeScript, o trabalho de Holmberg se concentra no processo de migração em si, sem explorar a comparação entre *frameworks*. O foco está na transição de projetos existentes, enquanto o presente estudo investiga o impacto da linguagem na construção de novas aplicações com diferentes tecnologias, a partir de uma abordagem mais controlada e sistemática.

3.4.1 Análise comparativa

Entre os estudos destacados, o trabalho de Martins Filho (2023) realiza uma comparação entre *frameworks* como *React*, *Vue.js* e *Next.js* no contexto do *front-end*, com foco em aspectos técnicos e na percepção de desenvolvedores. Já Ferreira e Zuchi (2018) exploram a comparação entre *frameworks JavaScript* com base em aplicações práticas e análise de desempenho. O estudo de Mororó (2024) oferece uma visão direta sobre as diferenças entre *JavaScript* e *TypeScript*, abordando tanto aspectos qualitativos quanto quantitativos das linguagens. Por fim, o trabalho de Holmberg (2023) trata da experiência de migração de projetos *JavaScript* para *TypeScript*, ressaltando os ganhos em qualidade de código e os desafios do processo.

Esses estudos contribuem para o embasamento do presente trabalho, ao fornecerem perspectivas complementares sobre o uso de *frameworks* modernos e linguagens no desenvolvimento *Web*, embora com escopos e abordagens distintas. A presente pesquisa se diferencia ao unir a análise entre linguagens (*JavaScript* e *TypeScript*) e *frameworks* (*Vue.js e Next.js*) em um contexto prático e sistemático.

O Quadro 1 resume os trabalhos relacionados, comparando as suas principais características com as da presente pesquisa.

Quadro 1 – Comparação entre estudos relacionados e o trabalho proposto.

Trabalho	**Frameworks Usados**	**Tecnologia Usada**	**Intuito Principal**	**Método de Análise**
Martins Filho (2023)	Vue.js e Next.js	JavaScript	Fornecem uma visão abrangente das características distintas de React, Vue e Next	Análise por meio de implementações com os frameworks e questionários com a comunidade
Ferreira & Zuchi (2018)	Angular e Vue.js	JavaScript	Apontar os principais frameworks JavaScript do mercado e mostrar suas características mais marcantes	Análise por meio de implementações utilizando os frameworks previstos
Mororó (2024)	Nenhuma	JavaScript e TypeScript	Comparação quantitativa e qualitativa entre JavaScript e TypeScript	Análise por meio do tempo de execução de funções em ambas as linguagens e Questionário
Holmberg (2023)	Nenhuma	JavaScript e TypeScript	Mostrar se a tarefa de migrar um sistema de JavaScript para Typescript é recomendada, as dificuldades, mas também os benefícios associados e o resultado	Análise por meio do processo migração de um software de uma linguagem para outra
Trabalho Proposto	**Vue.js e Next.js**	**JavaScript e TypeScript**	**Propõe a oferecer uma análise que auxilie desenvolvedores, gestores de projetos e demais interessados na escolha mais adequada de tecnologias para o desenvolvimento de aplicações Web e móveis.**	**Análise por meio de 4 implementações e métricas obtidas pelo Google Lighthouse e análise por baseado na experiência prática dessas implementações**

Fonte: elaborado pelo autor.

4 METODOLOGIA

Este capítulo detalha o percurso metodológico adotado para o desenvolvimento deste trabalho, compreendendo desde a seleção das tecnologias até a análise técnica das aplicações construídas. A pesquisa fundamenta-se em uma abordagem experimental e comparativa, focada no impacto do uso de *JavaScript* e *TypeScript* em frameworks distintos para o desenvolvimento Web moderno.

4.1 Definição das Tecnologias

As tecnologias utilizadas neste estudo foram selecionadas com base em sua relevância no cenário atual do desenvolvimento *Web* e em sua ampla adoção no mercado. Foram empregadas as linguagens *JavaScript* e *TypeScript*, em conjunto com os *frameworks Vue.js* e *Next.js*, possibilitando a construção de quatro variações distintas de uma mesma aplicação.

Além das tecnologias de desenvolvimento, foi utilizada a ferramenta *Google Lighthouse* como instrumento central de avaliação técnica das implementações. O *Lighthouse* é uma ferramenta de auditoria automatizada, integrada ao ecossistema do *Google Chrome*, que permite analisar aplicações *Web* a partir de métricas padronizadas relacionadas à desempenho, acessibilidade, boas práticas e otimização para mecanismos de busca (SEO). A adoção dessa ferramenta possibilitou a obtenção de relatórios comparáveis entre as diferentes implementações, fornecendo dados objetivos para a análise dos resultados.

4.2 Definição do Sistema Implementado

O sistema desenvolvido neste trabalho consiste em uma aplicação *Web* voltada ao consumo e à visualização de dados provenientes da API pública COVID-19. A aplicação permite a filtragem e a exibição de informações relacionadas à evolução da pandemia, simulando um cenário comum no desenvolvimento de aplicações *Web* modernas que dependem de dados externos.

Com o objetivo de viabilizar a análise comparativa, a mesma aplicação foi implementada em quatro variações funcionais equivalentes: *Vue.js* com *JavaScript*, *Vue.js* com *TypeScript*, *Next.js* com *JavaScript* e *Next.js* com *TypeScript*. Todas as versões compartilham os mesmos requisitos funcionais, estrutura de dados e lógica de negócio, garantindo que as diferenças observadas estejam diretamente relacionadas às tecnologias adotadas, e não a variações no escopo da

aplicação.

4.3 Métricas utilizadas

A avaliação das implementações foi conduzida a partir da combinação de métricas objetivas e análise subjetiva. As métricas técnicas foram obtidas por meio dos relatórios gerados pelo *Google Lighthouse*, contemplando indicadores de desempenho, acessibilidade, conformidade com boas práticas e SEO. Esses indicadores permitem avaliar aspectos como tempo de carregamento, eficiência na renderização, uso adequado de recursos e aderência a recomendações consolidadas para aplicações *Web*.

Complementarmente, foi realizada uma análise qualitativa baseada na experiência prática do autor durante o desenvolvimento das quatro implementações. Essa análise considerou fatores como curva de aprendizado, facilidade de configuração do ambiente, clareza e organização do código, legibilidade, esforço de manutenção e fluidez no processo de desenvolvimento. A combinação dessas perspectivas permitiu uma avaliação mais abrangente, unindo dados técnicos mensuráveis a percepções obtidas durante a implementação real dos sistemas.

4.4 Análise e Consolidação dos Resultados

A etapa final consistiu na interpretação cruzada dos relatórios técnicos com as observações práticas documentadas ao longo do desenvolvimento. Sem a necessidade de validação externa por terceiros, a análise focou na síntese dos resultados obtidos para fundamentar recomendações estratégicas sobre qual tecnologia utilizar em diferentes contextos. O objetivo dessa consolidação foi determinar as condições ideais para cada combinação, relacionando aspectos como a rapidez de entrega para projetos simples e a necessidade de escalabilidade para sistemas complexos.

5 RESULTADOS

Este capítulo apresenta os resultados obtidos a partir da implementação prática das quatro variações funcionais do sistema proposto, bem como a análise técnica derivada dessas construções. Inicialmente, descrevem-se a arquitetura da solução e os padrões de organização de pastas adotados em cada combinação de framework e linguagem, de modo a contextualizar as decisões estruturais tomadas durante o desenvolvimento.

Na sequência, são apresentados os dados extraídos a partir das auditorias realizadas com a ferramenta *Google Lighthouse*, contemplando métricas relacionadas à desempenho, acessibilidade, conformidade com boas práticas e otimização para mecanismos de busca. Esses dados servem de base para uma análise comparativa entre as implementações, permitindo confrontar os resultados técnicos obtidos com a experiência prática de desenvolvimento, a fim de fundamentar recomendações de uso para cada combinação tecnológica analisada.

Os experimentos foram conduzidos em ambiente de produção, com todas as implementações devidamente implantadas em servidores acessíveis publicamente. As aplicações foram hospedadas em um servidor do tipo *Virtual Private Server* (VPS), executando sistema operacional *Linux*, com configurações de *hardware* padronizadas para todas as versões, composto por 4 núcleos de CPU, 16 GB de memória RAM e 200 GB de espaço em disco. Além disso, foi utilizada a plataforma *Coolify*[1] para orquestração de containers *Docker* na instância VPS. Essa padronização teve como objetivo reduzir a influência de fatores externos nos resultados obtidos, garantindo maior consistência e comparabilidade entre as auditorias realizadas.

As quatro versões da aplicação encontram-se disponíveis publicamente nos seguintes endereços: Vue.js com TypeScript[2], Vue.js com JavaScript[3], Next.js com TypeScript[4] e Next.js com JavaScript[5]. As auditorias do Google Lighthouse foram executadas diretamente sobre essas aplicações em ambiente de produção, garantindo que os resultados refletissem condições reais de uso, incluindo tempo de resposta do servidor, carregamento de recursos e renderização no navegador.

Os relatórios completos gerados a partir das auditorias são apresentados nos Apêndices deste trabalho, por meio de capturas de tela representativas dos resultados obtidos para

[1] https://coolify.io
[2] https://vue-com-ts.veaser.com.br/
[3] https://vue-com-js.veaser.com.br/
[4] https://next-com-ts.veaser.com.br/
[5] https://next-com-js.veaser.com.br/

cada implementação. A partir desses relatórios, foi possível extrair métricas consolidadas que permitem uma comparação objetiva entre as diferentes combinações de linguagem e framework analisadas.

Tabela 1 – Comparação dos resultados das auditorias do Google Lighthouse.

Stack	Desempenho	Acessibilidade	Boas Práticas	SEO
Vue.js + JavaScript	94	75	100	92
Vue.js + TypeScript	94	75	100	92
Next.js + JavaScript	94	84	100	100
Next.js + TypeScript	93	83	100	100

Fonte: elaborado pelo autor.

A análise dos resultados apresentados na Tabela 1 evidencia que todas as implementações analisadas alcançaram pontuações elevadas nas métricas avaliadas pelo *Google Lighthouse*, indicando um bom nível geral de qualidade técnica independentemente da combinação de linguagem e *framework* utilizada. Destaca-se, especialmente, o fato de todas as versões terem obtido pontuação máxima no critério de boas práticas, o que demonstra aderência às recomendações técnicas estabelecidas pela ferramenta.

No que se refere ao desempenho, observa-se um comportamento bastante semelhante entre as quatro implementações, com pontuações variando entre 93 e 94. Esse resultado indica que, para o escopo da aplicação desenvolvida, tanto o *Vue.js* quanto o *Next.js* apresentaram desempenho equivalente, não sendo possível identificar diferenças significativas relacionadas ao uso de *JavaScript* ou *TypeScript* nesse aspecto.

Em relação à acessibilidade, as implementações baseadas em *Next.js* apresentaram resultados superiores quando comparadas às versões desenvolvidas com *Vue.js*. Enquanto as aplicações em *Vue.js* obtiveram pontuação de 75, as versões em *Next.js* alcançaram valores acima de 80, o que pode estar associado às configurações padrão do *framework*, bem como ao suporte mais explícito a práticas voltadas à acessibilidade em sua estrutura inicial.

No critério de SEO, as aplicações desenvolvidas com *Next.js* obtiveram pontuação máxima, evidenciando a eficácia do *framework* em atender aos requisitos de indexação e otimização para mecanismos de busca, especialmente em razão de seus recursos nativos de renderização híbrida. Já as versões desenvolvidas com *Vue.js* apresentaram resultados satisfatórios, porém inferiores, refletindo a necessidade de ajustes adicionais para alcançar níveis semelhantes de otimização.

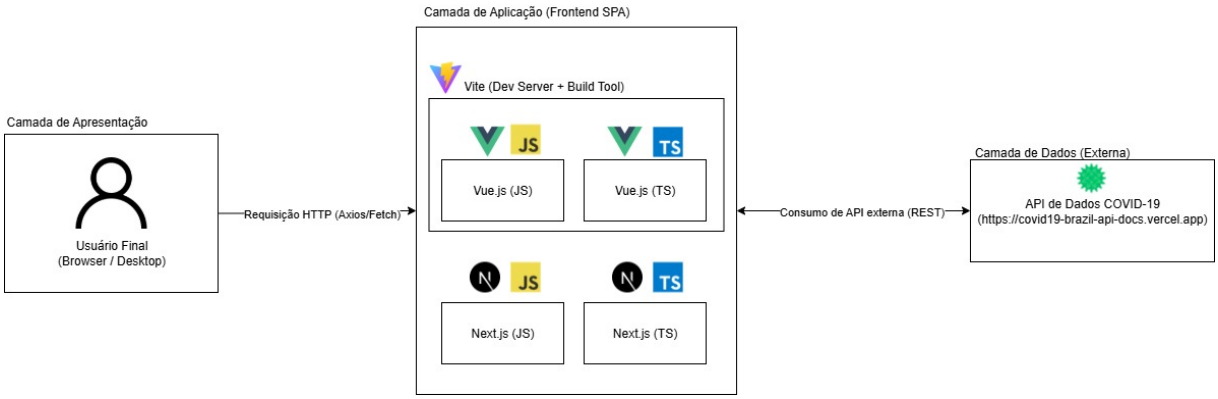
Quanto à comparação entre *JavaScript* e *TypeScript*, os resultados indicam que a adoção do *TypeScript* não impactou negativamente as métricas avaliadas, uma vez que as pontuações permaneceram equivalentes ou muito próximas entre as versões. Esse comportamento reforça que a escolha pelo *TypeScript* está mais relacionada a benefícios estruturais e de manutenção do código do que a ganhos ou perdas diretas de desempenho mensurados pelas auditorias automatizadas.

Além das pontuações numéricas, o *Google Lighthouse* também indicou oportunidades de melhoria comuns às implementações, como o aprimoramento de atributos relacionados à acessibilidade, ajustes finos em elementos semânticos e a otimização complementar de recursos. Essas recomendações evidenciam que, embora os resultados sejam positivos, ainda há margem para refinamentos que podem elevar ainda mais a qualidade técnica das aplicações, independentemente da tecnologia adotada.

5.1 Arquitetura da Solução

Para viabilizar o estudo comparativo, foi definida uma arquitetura de *software* de três camadas, conforme ilustrado na Figura 7, que visa isolar a lógica de *front-end* e padronizar a forma de consumo de dados.

Figura 7 – Arquitetura das implementações



Fonte: Elaborado pelo autor.

A arquitetura é composta pelas seguintes camadas:

Camada de Apresentação: Representa o cliente, onde o Usuário Final interage com a aplicação por meio de um navegador web (*Browser/Desktop*). As interações do usuário, como o preenchimento de filtros, disparam requisições *HTTP* (utilizando Axios ou a API *Fetch* nativa) para a camada de aplicação.

Camada de Aplicação (*Frontend SPA*): É o núcleo do estudo, onde as quatro variações da aplicação são implementadas. Esta camada utiliza ferramentas de construção e desenvolvimento, como o

- Vite (*Dev Server + Build Tool*), para processar o código-fonte escrito em *JavaScript* ou *TypeScript* com o *framework Vue.js*, já que o *Next.js* já possui seu próprio ambiente de *build/server* embutido. A responsabilidade desta camada é gerenciar o estado da aplicação, processar os dados recebidos e renderizar a interface para o usuário.
- Camada de Dados (Externa): A aplicação consome dados de uma fonte externa, a API de Dados COVID-19. A comunicação é realizada via o padrão REST, onde a camada de aplicação solicita os dados e recebe uma resposta no formato JSON.
- Este design arquitetônico garante que todas as versões da aplicação operem sob as mesmas condições de acesso a dados, permitindo que a análise comparativa se concentre efetivamente nas diferenças de implementação, desempenho e manutenibilidade de cada stack tecnológica.

Este design arquitetônico garante que todas as versões da aplicação operem sob as mesmas condições de acesso a dados, permitindo que a análise comparativa se concentre efetivamente nas diferenças de implementação, desempenho e manutenibilidade de cada tecnologia.

5.2 Detalhes da Implementação e Padrões Arquiteturais

Nesta seção, detalham-se as escolhas técnicas e a estrutura organizacional aplicada em cada uma das quatro implementações. Embora o objetivo funcional de consumir a API da COVID-19 seja idêntico em todas as frentes, a organização interna variou para respeitar as convenções de cada tecnologia e o nível de complexidade introduzido pela tipagem.

Com o intuito de garantir transparência e possibilitar a reprodução dos experimentos apresentados, todos os códigos-fonte desenvolvidos neste trabalho foram disponibilizados em repositórios públicos. Os respectivos repositórios podem ser acessados por meio dos links disponibilizados nas notas de rodapé ao longo desta seção, permitindo a consulta detalhada das implementações, da organização de pastas e das decisões arquiteturais adotadas.

- Vue.js com JavaScript[6]

[6] https://github.com/kaynan951/TCC-Vue_JS

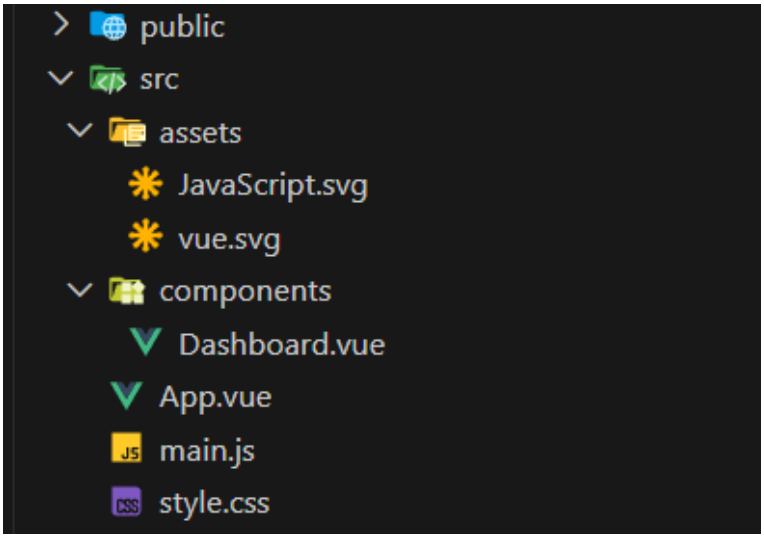
- Vue.js com TypeScript[7]
- Next.js com JavaScript[8]
- Next.js com TypeScript[9]

5.2.1 Vue.js com JavaScript

A implementação utilizando *Vue.js* com *JavaScript* adotou o padrão Pure Monolithic Single File Component (SFC), no qual a maior parte da lógica, estrutura e estilização da aplicação encontra-se concentrada em componentes únicos. Essa abordagem favorece a simplicidade e a rapidez no desenvolvimento inicial, sendo especialmente adequada para projetos de pequeno porte ou com escopo bem definido.

Do ponto de vista arquitetural, foi utilizada uma *Flat Architecture*, caracterizada por uma estrutura de diretórios pouco profunda e com baixo nível de separação. A organização do código segue o modelo *Feature-in-File*, no qual cada funcionalidade é encapsulada em um único arquivo, reunindo template, lógica e estilo. Essa estratégia reduz a complexidade inicial, porém pode dificultar a manutenção à medida que a aplicação cresce.

Figura 8 – organização de pastas da implementação Vue.js + JavaScript.



Fonte: Elaborado pelo autor.

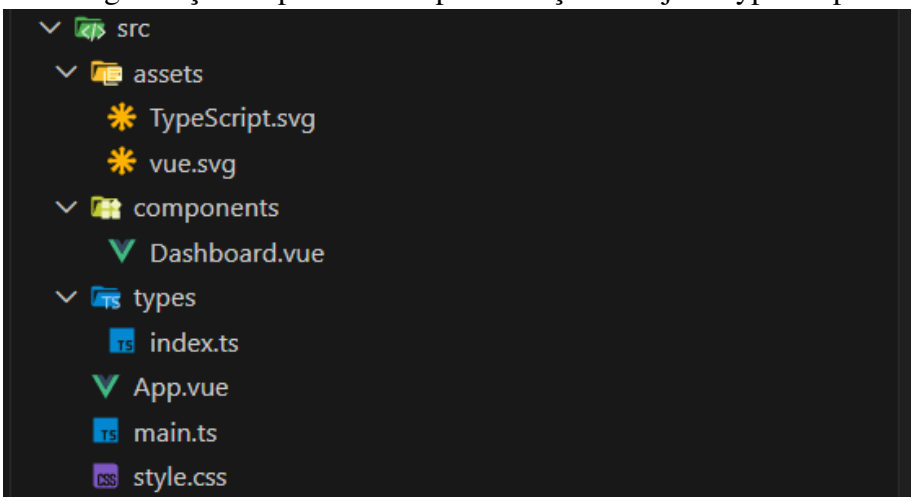
[7] https://github.com/kaynan951/TCC-Vue_TS
[8] https://github.com/kaynan951/TCC-Next_JS
[9] https://github.com/kaynan951/TCC-Next_TS

5.2.2 *Vue.js com TypeScript*

Na versão desenvolvida com *Vue.js* e *TypeScript*, foi adotado o padrão *Type-Layered* SFC *Architecture*, no qual há uma separação mais explícita entre a definição de tipos e a implementação dos componentes. Essa abordagem contribui para maior clareza na modelagem dos dados e maior segurança durante o desenvolvimento.

A arquitetura pode ser caracterizada como uma *Two-Layer Architecture*, composta por uma camada de definição de tipos e uma camada de componentes. A organização segue uma Arquitetura Monolítica Modular, na qual as funcionalidades são agrupadas de forma estruturada, favorecendo a escalabilidade e a manutenibilidade da aplicação em comparação à versão desenvolvida em *JavaScript*.

Figura 9 – organização de pastas da implementação Vue.js + TypeScript.



Fonte: Elaborado pelo autor.

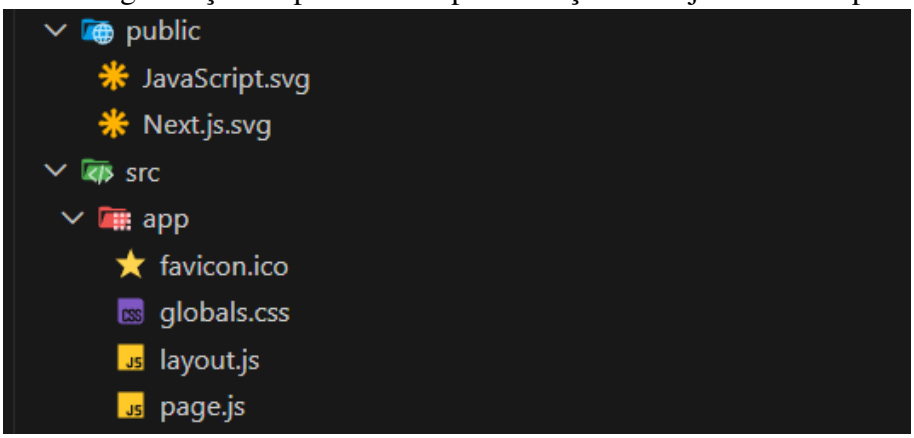
5.2.3 *Next.js com JavaScript*

A implementação em *Next.js* utilizando *JavaScript* seguiu o padrão *Monolithic Page Component*, no qual cada página concentra tanto a lógica de negócio quanto a renderização da interface. Essa abordagem está alinhada ao modelo tradicional do *framework*, especialmente ao utilizar o *App Router*.

A arquitetura baseia-se no *Next.js App Router with Flat Pages*, fazendo uso do roteamento baseado em arquivos. A organização do código segue o princípio de *File-based Routing with Monolithic Logic*, em que cada rota corresponde a um arquivo que centraliza a lógica da página. Embora essa estratégia simplifique o entendimento inicial, pode resultar em

arquivos extensos e menos organizados à medida que o projeto cresce.

Figura 10 – organização de pastas da implementação Next.js + JavaScript.



Fonte: Elaborado pelo autor.

5.2.4 Next.js com TypeScript

Na versão desenvolvida com *Next.js* e *TypeScript*, foi adotado o padrão *Type-Layered Page Architecture*, também conhecido como *Next.js Type-First Pattern*. Nesse modelo, há uma separação clara entre a camada de definição de tipos e a implementação das páginas e componentes como mostrado na Figura 11.

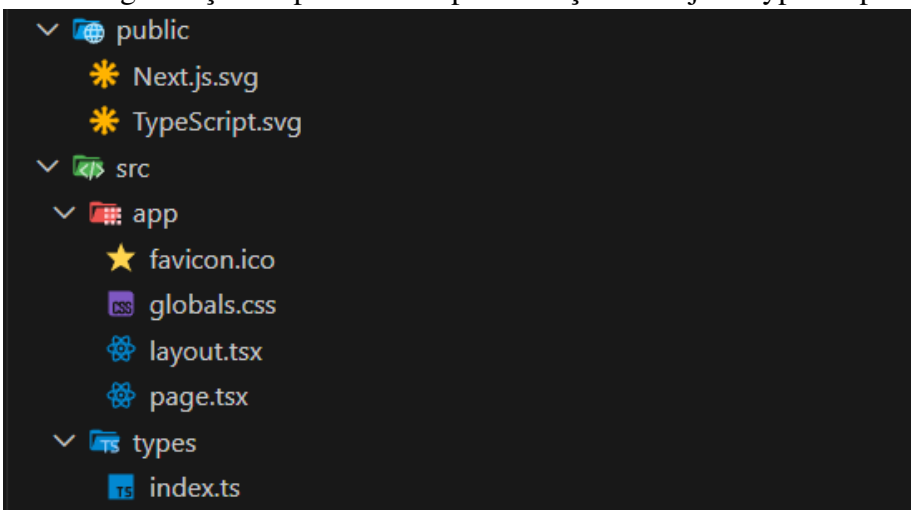
A arquitetura utilizada baseia-se no *Next.js App Router with Type Definition Layer*, incorporando uma camada adicional dedicada à tipagem estática. A organização segue uma Arquitetura Monolítica Modular, proporcionando maior controle estrutural, melhor legibilidade do código e maior facilidade de manutenção, sendo especialmente adequada para aplicações mais robustas e com potencial de crescimento, assim como no caso do *Vue.js* com *TypeScript*.

5.2.5 Guia Comparativo das Recomendações

Nesse sentido, uma das principais contribuições deste trabalho é a proposição de um guia prático de apoio à tomada de decisão, construído a partir dos resultados obtidos e da experiência prática adquirida durante as implementações. O objetivo desse guia é servir como referência para desenvolvedores que necessitam escolher uma combinação tecnológica de forma consciente, considerando características do projeto, prazos, escalabilidade e nível de maturidade técnica da equipe. O Quadro 2 sintetiza essas recomendações, destacando cenários indicados, pontos positivos e limitações observadas.

A partir dessa análise, o trabalho cumpre sua motivação central ao demonstrar que a

Figura 11 – organização de pastas da implementação Next.js + TypeScript.



Fonte: Elaborado pelo autor.

Quadro 2 – Guia comparativo de recomendação das tecnologias analisadas

Stack	**Situação recomendada**	**Ponto positivo**	**Ponto negativo**
Vue.js + JS	Projetos pequenos ou acadêmicos, com escopo reduzido e prazos curtos	Rapidez na implementação e baixa curva de aprendizado	Menor escalabilidade e maior dificuldade de manutenção futura
Vue.js + TS	Projetos de médio porte que exigem maior organização e segurança no código	Melhor tipagem, organização e manutenibilidade	Curva de aprendizado mais elevada em relação ao *JavaScript* puro
Next.js + JS	Aplicações *Web* que demandam roteamento avançado e renderização híbrida, com foco em produtividade inicial	Estrutura robusta e facilidade de criação de rotas	Maior propensão a erros e menor controle de tipos
Next.js + TS	Projetos de médio a grande porte, com foco em escalabilidade, manutenção e equipes maiores	Alta organização, segurança de tipos e melhor suporte à evolução do projeto	Maior complexidade inicial e tempo de configuração

escolha de tecnologias deve ser orientada por critérios técnicos e contextuais, e não apenas por tendências de mercado. As evidências levantadas oferecem subsídios para que desenvolvedores avaliem, de forma mais crítica, qual combinação tecnológica se mostra mais eficiente para determinado cenário de aplicação.

6 CONSIDERAÇÕES FINAIS E PERPECTIVAS FUTURAS

Este trabalho teve como principal objetivo analisar, de forma prática e comparativa, o uso das linguagens *JavaScript* e *TypeScript* em conjunto com os frameworks *Vue.js* e *Next.js*, a partir da implementação real de quatro variações funcionais de uma mesma aplicação *Web*. Diferentemente de uma abordagem exclusivamente teórica, a pesquisa foi fundamentada na experiência direta de desenvolvimento, na coleta de métricas automatizadas por meio do *Google Lighthouse* e na análise crítica conduzida sob a perspectiva do desenvolvedor responsável pelas implementações.

Ao longo do desenvolvimento, foi possível compreender como cada combinação tecnológica impacta não apenas aspectos técnicos mensuráveis, como desempenho, acessibilidade e aderência a boas práticas, mas também fatores práticos do cotidiano do desenvolvimento, como curva de aprendizado, organização do código, facilidade de manutenção e produtividade. Essa vivência reforçou a constatação de que não existe uma tecnologia universalmente superior, mas sim escolhas mais adequadas conforme o contexto do projeto, o perfil da equipe e os objetivos estabelecidos.

No que se refere às perspectivas futuras, diversas extensões podem ser exploradas a partir desta pesquisa. Uma possibilidade consiste na incorporação de ferramentas baseadas em Inteligência Artificial para auxiliar na análise automática do código, identificação de padrões arquiteturais e sugestão de melhorias relacionadas a desempenho, legibilidade e manutenibilidade.

Outra vertente seria a ampliação ou substituição das tecnologias analisadas, incluindo outros *frameworks* amplamente utilizados no mercado, ou ainda a realização de comparativos mais específicos, isolando variáveis como linguagem ou *framework* para análises aprofundadas de desempenho e escalabilidade. Adicionalmente, trabalhos futuros podem incorporar a aplicação de questionários ou *surveys* com um número significativo de desenvolvedores, possibilitando a coleta de percepções do mercado e a triangulação desses dados com novas implementações práticas, enriquecendo e validando ainda mais as conclusões obtidas.

REFERÊNCIAS

AAGAM. **The Most Popular JavaScript Frameworks in 2025**. 2024. Disponível em: https://hygraph.com/blog/javascript-frameworks. Acesso em: 13 mai. 2025.

Awari. **TypeScript: A evolução do JavaScript para desenvolvedores modernos**. 2023. Disponível em: https://awari.com.br/typescript/. Acesso em: 13 mai. 2025.

BANKS, A.; PORCELLO, E. **Learning React**: Modern patterns for developing react apps. O'Reilly Media, 2020. ISBN 9781492051671. Disponível em: https://books.google.com.br/books?id=tjjrDwAAQBAJ. Acesso em: 14 ago. 2025.

BLUEBIRDINTERNATIONAL. **A Brief History of JavaScript**. 2024. Disponível em: https://bluebirdinternational.com/javascript-vs-typescript-performance/. Acesso em: 03 jun. 2025.

CUOMO, S.; LEE, T. **Vue.js 3 for Beginners**: Learn the essentials of vue.js 3 and its ecosystem to build modern web applications. Packt Publishing, 2024. ISBN 9781805123293. Disponível em: https://books.google.com.br/books?id=N-gWEQAAQBAJ. Acesso em: 14 ago. 2025.

DENO. **A Brief History of JavaScript**. 2025. Disponível em: https://deno.com/blog/history-of-javascript. Acesso em: 03 jun. 2025.

FERREIRA, E. d. S.; SOUZA, C. A.; OLIVEIRA, A. P. DESENVOLVIMENTO DE SISTEMAS WEB MODERNOS: UMA ANÁLISE DAS TECNOLOGIAS E TENDÊNCIAS ATUAIS. **Revista Científica Eletrônica UNIFIA**, v. 1, n. 2, p. 215–229, jan./jun. 2024. ISSN 2966-0648. Disponível em: https://portal.unisepe.com.br/unifia/wp-content/uploads/sites/10001/2024/06/Desenvolvimento-de-Sistemas-Web-Modernos-p%C3%A1g-215-a-229-2.pdf. Acesso em: 13 mai. 2025.

FERREIRA, H. K.; ZUCHI, J. D. AnÁlise comparativa entre frameworks frontend baseados em javascript para aplicaÇÕes web. **Revista Interface Tecnológica**, v. 15, n. 2, p. 111–123, dez. 2018. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/502. Acesso em: 10 jun. 2025.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) – University of California, Irvine, 2000.

FLANAGAN, D. **JavaScript**: O guia definitivo. Bookman Editora, 2012. ISBN 9788565837484. Disponível em: https://books.google.com.br/books?id=zWNyDgAAQBAJ. Acesso em: 14 ago. 2025.

FOWLER, M. **Microservices**. 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 03 jun. 2025.

GEEKSFORGEEKS. **ReactJS Virtual DOM**. 2025. Disponível em: https://www.geeksforgeeks.org/reactjs-virtual-dom/. Acesso em: 03 jun. 2025.

GIZAS, A.; CHRISTODOULOU, S.; PAPATHEODOROU, T. Comparative evaluation of javascript frameworks. In: INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, 21. **Proceedings [...]**. New York, NY, USA: Association for Computing Machinery, 2012. (WWW '12 Companion), p. 513–514. ISBN 9781450312301. Disponível em: https://doi.org/10.1145/2187980.2188103. Acesso em: 13 mai. 2025.

HOLMBERG, O. **Migrating from JavaScript to TypeScript and its advantages.** 2023, 40 p. Bachelor's Thesis – Metropolia University of Applied Sciences, Helsinki, 2023. Disponível em: https://www.theseus.fi/bitstream/handle/10024/793478/Holmberg_Oliver.pdf?sequence=2. Acesso em: 10 jun. 2025.

Marcio Hanashiro. **JavaScript: conheça a evolução da linguagem de programação**. 2024. Disponível em: https://www.locaweb.com.br/blog/temas/codigo-aberto/evolucao-do-javascript. Acesso em: 13 mai. 2025.

MARTINS FILHO, F. R. F. **ANÁLISE COMPARATIVA DE TECNOLOGIAS JAVASCRIPT FOCADAS NO FRONT-END PARA DESENVOLVIMENTO WEB**. 2023, 40 p. Trabalho de Conclusão de Curso(Graduação em Engenharia de Computação) – Universidade Federal do Ceará, Campus Quixadá, Quixadá, 2023. Disponível em: https://repositorio.ufc.br/bitstream/riufc/75925/1/2023_tcc_frfmartinsfilho.pdf. Acesso em: 10 jun. 2025.

MILETTO, E.; BERTAGNOLLI, S. de C. **Desenvolvimento de Software II**: Introdução ao desenvolvimento web com html, css, javascript e php - eixo: Informação e comunicação - série tekne. Bookman Editora, 2014. (Tekne). ISBN 9788582601969. Disponível em: https://books.google.com.br/books?id=lcLFAwAAQBAJ. Acesso em: 14 ago. 2025.

MORORÓ, J. F. **UM ESTUDO COMPARATIVO ENTRE JAVASCRIPT E TYPESCRIPT**. 2024, 153 p. Trabalho de Conclusão de Curso – Universidade Federal do Ceará, Campus Sobral, Sobral, 2024. Disponível em: https://repositorio.ufc.br/bitstream/riufc/78817/1/2024_tcc_jfmororo.pdf. Acesso em: 10 jun. 2025.

POWELL, Z. **Angular vs React: Uma Comparação Detalhada**. 2023. Disponível em: https://kinsta.com/pt/blog/angular-vs-react/. Acesso em: 13 mai. 2025.

RITIKA. **What is TypeScript? Definition, History, Features and Uses**. 2024. Disponível em: https://invedus.com/blog/what-is-typescript-definition-history-features-and-uses-of-typescript/. Acesso em: 03 jun. 2025.

Statista Research Department. **Internet Usage Worldwide - Statistics & Facts**. 2025. Disponível em: https://www.statista.com/topics/1145/internet-usage-worldwide/#topicOverview. Acesso em: 13 mai. 2025.

TUREMURATOVA, G. The evolution of web development: From static pages to dynamic web applications. **International Journal of Artificial Intelligence**, v. 1, n. 1, p. 818–820, 2025.

Typescriptutorial. **Guia e Tutorial de TypeScript**. 2025. Disponível em: https://typescriptutorial.com/pt/guia-e-tutorial/. Acesso em: 13 mai. 2025.

Vercel. **Next.js Showcase**. 2026. Disponível em: https://nextjs.org/showcase. Acesso em: 14 jan. 2026.

APÊNDICE A – RELATÓRIO GOOGLE LIGHTHOUSE - VUE.JS COM JAVASCRIPT

Fonte: Elaborado pelo autor.

APÊNDICE B – RELATÓRIO GOOGLE LIGHTHOUSE - VUE.JS COM TYPESCRIPT

Fonte: Elaborado pelo autor.

APÊNDICE C – RELATÓRIO GOOGLE LIGHTHOUSE - NEXT.JS COM JAVASCRIPT

Fonte: Elaborado pelo autor.

APÊNDICE D – RELATÓRIO GOOGLE LIGHTHOUSE - NEXT.JS COM TYPESCRIPT

Fonte: Elaborado pelo autor.