UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME DOS SANTOS CAVALCANTE

UMA ANÁLISE COMPARATIVA DE FRAMEWORKS DE DESENVOLVIMENTO WEB

QUIXADÁ

2025

GUILHERME DOS SANTOS CAVALCANTE

UMA ANÁLISE COMPARATIVA DE FRAMEWORKS DE DESENVOLVIMENTO WEB

Projeto de Pesquisa 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. Me. Carlos Roberto Rodrigues Filho.

QUIXADÁ

2025

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)

C364a Cavalcante, Guilherme dos Santos.
Uma análise comparativa de frameworks de desenvolvimento web / Guilherme dos Santos Cavalcante. – 2025.
40 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2025.
Orientação: Prof. Me. Carlos Roberto Rodrigues Filho.

1. Desenvolvimento Web. 2. Comparação de Frameworks. 3. React. 4. Angular. 5. Vue. I. Título.
CDD 004

GUILHERME DOS SANTOS CAVALCANTE

UMA ANÁLISE COMPARATIVA DE FRAMEWORKS DE DESENVOLVIMENTO WEB

Projeto de Pesquisa 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: 01/08/2025.

BANCA EXAMINADORA

Prof. Me. Carlos Roberto Rodrigues Filho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Bruno Góis Mateus
Universidade Federal do Ceará (UFC)

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

A Deus.

Aos meus pais, Marinalva e Claudemir.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Antonia Marinalva Lirada Cavalcante e Claudemir Vieira Cavalcante, pelo apoio, incentivo e amor incondicional ao longo de toda a minha trajetória acadêmica. Sem os valores, a força e os ensinamentos que recebi deles, este trabalho não teria sido possível.

Agradeço ao meu orientador, Prof. Me. Carlos Roberto Rodrigues Filho, pela orientação dedicada, pelas contribuições técnicas e pela paciência durante todo o desenvolvimento deste trabalho. Sua experiência e disponibilidade foram fundamentais para a realização deste projeto.

Registro também minha gratidão aos professores Prof. Dr. Bruno Góis Mateus e Prof. Dr. Jefferson de Carvalho Silva, membros da banca examinadora, pela leitura atenta, sugestões construtivas e contribuição valiosa para o aprimoramento deste trabalho.

A todos que, direta ou indiretamente, contribuíram para que eu chegasse até aqui, o meu sincero agradecimento.

“Sucesso é a soma de pequenos esforços repetidos dia após dia.”

Robert Collier

RESUMO

O desenvolvimento de aplicações *web* modernas exige a escolha criteriosa de *frameworks* que equilibrem desempenho, produtividade e acessibilidade. Diante da ampla adoção de *React*, *React* e *Vue*, este trabalho tem como objetivo comparar esses *frameworks* em duas dimensões fundamentais: curva de aprendizado e desempenho técnico. A curva de aprendizado foi avaliada por meio de um diário de bordo que registrou a experiência do autor durante a implementação, considerando critérios como clareza da documentação, complexidade da configuração inicial, verbosidade do código e tempo de desenvolvimento. Paralelamente foi desenvolvida uma aplicação de *To-do List* em cada tecnologia, utilizando um ambiente controlado para mensurar métricas de tempo de inicialização, uso de memória e CPU. Os resultados indicam que *React* apresentou o melhor desempenho geral, com baixo consumo de recursos e inicialização ágil, enquanto *Vue* destacou-se na curva de aprendizado, graças à documentação clara, suporte em português e configuração simples. *React*, por sua vez, mostrou alto consumo de recursos e curva de aprendizado mais acentuada, apesar de sua robustez. Conclui-se que a seleção do *framework* deve levar em conta o contexto do projeto e o perfil da equipe, sendo *Vue* mais indicado para iniciantes, *React* para projetos ágeis e *React* para sistemas corporativos de larga escala. Este estudo contribui com uma análise prática e comparativa que pode orientar decisões em ambientes acadêmicos e educacionais.

Palavras-chave: desenvolvimento *web*; *React*; *Angular*; *Vue*; comparação de *frameworks*; curva de aprendizado; desempenho.

ABSTRACT

Developing modern web applications requires the careful selection of frameworks that balance performance, productivity, and accessibility. Given the widespread adoption of React, React, and Vue, this work aims to compare these frameworks along two fundamental dimensions: learning curve and technical performance. The learning curve was assessed through a logbook that recorded the author's experience during implementation, considering criteria such as documentation clarity, initial setup complexity, code verbosity, and development time. In parallel, a To-do List application was developed using each technology, using a controlled environment to measure startup time, memory, and CPU usage. The results indicate that React presented the best overall performance, with low resource consumption and fast startup, while Vue excelled in the learning curve, thanks to its clear documentation, support in Portuguese, and simple setup. React, in turn, showed high resource consumption and a steeper learning curve, despite its robustness. We conclude that the selection of framework should take into account the project context and team profile, with Vue being more suitable for beginners, React for agile projects, and React for large-scale enterprise systems. This study contributes a practical and comparative analysis that can guide decisions in academic and educational settings.

Keywords: web development; React; Angular; Vue; framework comparison; learning curve; performance.

LISTA DE ILUSTRAÇÕES

Figura 1 – Imagem do fluxograma dos passos metodológicos 20
Figura 2 – Imagem da interface feita com *React* 22
Figura 3 – Imagem da interface feita com *Angular* 22
Figura 4 – Imagem da interface feita com *Vue* 23
Figura 5 – Imagem da aba de performance do *Chrome DevTools* 31
Figura 6 – Imagem da de memória do *Chrome DevTools* 32

LISTA DE TABELAS

Tabela 1 – Análise de arquivos: *React* 27
Tabela 2 – Análise de arquivos: *Angular* 29
Tabela 3 – Análise de arquivos: *Vue* 30
Tabela 4 – Valores de desempenho medidos para *React* (10 execuções) 32
Tabela 5 – Valores de desempenho medidos para *Angular* (10 execuções) 33
Tabela 6 – Valores de desempenho medidos para *Vue* (10 execuções) 33

LISTA DE QUADROS

Quadro 1 – Comparativo de trabalhos relacionados 19
Quadro 2 – Anotações sobre a curva de aprendizado em *React* 27
Quadro 3 – Anotações sobre a curva de aprendizado em *Angular* 28
Quadro 4 – Anotações sobre a curva de aprendizado em *Vue* 29

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Objetivos 13
2 FUNDAMENTAÇÃO TEÓRICA 14
2.1 *Frontend* 14
2.2 *Frameworks* de Desenvolvimento *Web* 14
2.2.1 *React* 15
2.2.2 *Angular* 16
2.2.3 *Vue* 16
3 TRABALHOS RELACIONADOS 17
3.1 Análise Comparativa Entre as Tecnologias de *Front-end React*, *Angular* e *Vue* 17
3.2 Análise Comparativa das Características de Performance dos *Frameworks JavaScript React* e *Vue* 17
3.3 Análise Comparativa entre *Frameworks Frontend* baseados em *JavaScritpt* para Aplicações *Web* 18
3.4 Comparação dos trabalhos relacionados com o trabalho proposto 18
4 METODOLOGIAS 20
4.1 Implementação da aplicação de *To-do List* 20
4.2 Análise da Curva de Aprendizado 23
4.3 Análise de Desempenho 24
5 RESULTADOS 26
5.1 Curva de aprendizado 26
5.1.1 *React* 26
5.1.2 *Angular* 28
5.1.3 *Vue* 29
5.1.4 Análise Comparativa 30
5.2 Desempenho 31
5.3 Discussão Geral 34
6 CONCLUSÕES E TRABALHOS FUTUROS 36
Referências 37

1 INTRODUÇÃO

O desenvolvimento de aplicações *web* tem crescido exponencialmente nos últimos anos, impulsionado pela demanda por soluções digitais rápidas, escaláveis e acessíveis em diversos setores da economia (Kolla; Ganta, 2025). Com o aumento da complexidade dos sistemas *web* modernos, tornou-se essencial o uso de ferramentas que otimizem o processo de desenvolvimento, melhorem a experiência do usuário e garantam alto desempenho (Kolla; Ganta, 2025). Nesse contexto, os *frameworks* de desenvolvimento *web* desempenham um papel central, fornecendo estruturas padronizadas que facilitam a criação de interfaces dinâmicas, reutilizáveis e eficientes (Kaur; Kaur, 2018).

Dados do *Stack Overflow Developer Survey* indicam que *React*, *Angular* e *Vue* consistentemente figuram entre as tecnologias mais utilizadas por desenvolvedores *web*, reforçando seu impacto na indústria. Compreender as diferenças entre essas soluções pode auxiliar na tomada de decisões mais informadas, contribuindo para a otimização de tempo, recursos e produtividade no desenvolvimento de aplicações (Stack Overflow, 2025).

Essas tecnologias são utilizadas por empresas e desenvolvedores devido à sua maturidade, flexibilidade, desempenho e suporte ativo da comunidade. Cada um desses *frameworks* adota uma abordagem distinta quanto à arquitetura, organização de código e gerenciamento de estado, o que pode influenciar diretamente na curva de aprendizado, no desempenho da aplicação e na manutenibilidade do *software*. Apesar de sua popularidade, a escolha entre essas tecnologias não é trivial, pois envolve *trade-offs* entre complexidade, produtividade e requisitos específicos do projeto (Pinto; Hoffmann; Uriarte, 2023).

Diante disso, este trabalho busca realizar uma análise comparativa entre os *frameworks React*, *Angular* e *Vue*, com foco nas dimensões de curva de aprendizado e desempenho. Para isso, foi desenvolvida uma aplicação simples de lista de tarefas utilizando cada um dos três *frameworks*, permitindo uma avaliação prática e direta de suas características. A aplicação será descrita na Seção 1.1. A comparação foi estruturada de forma sistemática, com a aplicação de métricas como tempo de renderização, consumo de memória e uso de CPU.

Além da análise técnica, o estudo também considera a experiência subjetiva do autor durante o processo de implementação, com o intuito de avaliar a facilidade ou dificuldade de aprendizado de cada tecnologia. Essa abordagem híbrida (técnica e qualitativa) busca oferecer uma visão mais completa sobre os desafios enfrentados por desenvolvedores ao adotar novas ferramentas no ambiente profissional. Com isso, o público-alvo desta pesquisa inclui estudantes

e iniciantes no desenvolvimento *web* que necessitam de subsídios para a seleção de *frameworks* em novos projetos.

1.1 Objetivos

Esse trabalho tem como objetivo geral comparar os *frameworks* de desenvolvimento *web* *React*, *Angular* e *Vue* em termos de curva de aprendizado e desempenho por meio da implementação de uma mesma aplicação simples usando cada tecnologia. Os objetivos específicos estão listados a seguir:

- Desenvolver uma aplicação simples de *To-Do List* utilizando *React*, *Angular* e *Vue*.
- Avaliar a curva de aprendizado com base na experiência própria ao desenvolver a aplicação em cada *framework*.
- Fazer testes considerando métricas de tempo de renderização, uso de memória e uso de CPU para analisar o desempenho.
- Identificar vantagens e limitações de cada *framework* com base nos resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo são apresentados os conceitos que fundamentam o desenvolvimento deste trabalho. Primeiramente é tratado sobre o que é *frontend* e logo após são descritos os *frameworks* utilizados nesse trabalho.

2.1 *Frontend*

O *frontend* é a parte do desenvolvimento *web* responsável por criar a interface visual e interativa de uma aplicação. Ele envolve a implementação dos elementos gráficos e da experiência do usuário, garantindo que as páginas sejam acessíveis e responsivas em diferentes dispositivos (Marcotte, 2011).

Para isso, utiliza-se um conjunto de tecnologias essenciais, como *HTML*, responsável pela estruturação dos conteúdos, *CSS*, que define a aparência e o estilo dos elementos, e *JavaScript*, que possibilita interatividade e dinamismo, permitindo a manipulação da interface em tempo real (Flanagan, 2020). O avanço das aplicações *web* levou ao desenvolvimento de *frameworks* que facilitam a construção e manutenção de interfaces complexas (Flanagan, 2020). Tecnologias como *React*, *Angular* e *Vue* são amplamente utilizadas para criar componentes reutilizáveis, melhorar a desempenho da aplicação e oferecer uma experiência mais fluida ao usuário (Sutcliffe, 2016).

O desenvolvimento *frontend* também está diretamente ligado à experiência do usuário (UX) e ao *design* de interface (UI), garantindo que os elementos visuais sejam intuitivos e funcionais. Estratégias como *Mobile-First* e *Design* Responsivo são essenciais para adaptar as aplicações a diferentes tamanhos de tela, proporcionando acessibilidade e melhor usabilidade (Mullins, 2015).

2.2 *Frameworks* de Desenvolvimento *Web*

Frameworks de desenvolvimento *web* são conjuntos de ferramentas e bibliotecas que fornecem uma estrutura padronizada para a criação de aplicações *web* (Vyas, 2022). Eles estabelecem convenções e padrões de desenvolvimento, facilitando a escrita, organização e manutenção do código (Vyas, 2022). Ao invés de construir aplicações do zero, os desenvolvedores podem utilizar essas estruturas para implementar funcionalidades comuns de forma mais efici-

ente, reduzindo o tempo de desenvolvimento e garantindo maior consistência no projeto (Vyas, 2022).

A maioria dos *frameworks* de desenvolvimento *web* adota uma arquitetura bem definida para facilitar a separação de responsabilidades dentro da aplicação. Um dos modelos mais comuns é a arquitetura Modelo-Visão-Controlador (MVC), onde o modelo gerencia os dados e regras de negócio, a visão cuida da interface gráfica e o controlador faz a intermediação entre os dois (Nguyen *et al.*, 2022). No entanto, *frameworks* modernos como *React*, *Angular* e *Vue*, seguem um modelo baseado em componentes, onde a interface é construída a partir de pequenas partes reutilizáveis que encapsulam lógica e apresentação (Chen; Shimomura, 2009).

Além de definir uma estrutura para o código, os *frameworks* de desenvolvimento *web* também incluem recursos adicionais que aumentam a produtividade e segurança do desenvolvimento. Entre essas funcionalidades estão: roteamento, manipulação de estado, integração com APIs, *templates* dinâmicos, suporte a testes automatizados e ferramentas de segurança. Esses recursos garantem que os desenvolvedores possam criar aplicações robustas sem precisar programar tudo do zero e, em simultâneo, adotam boas práticas recomendadas pelo próprio *framework* (Krishna, 2020).

2.2.1 *React*

O *React* é um *framework* de desenvolvimento *web* baseado em JavaScript, criado pelo *Facebook* em 2013, voltado para a construção de interfaces de usuário dinâmicas e reutilizáveis. Seu principal foco está no desenvolvimento de *Single Page Applications*(SPAs), onde a navegação ocorre sem a necessidade de recarregar completamente a página, garantindo uma experiência mais fluida para o usuário (Meta, 2025).

O *React* utiliza uma abordagem baseada em componentes, onde cada parte da interface é representada por um bloco independente que pode ser reutilizado em diferentes contextos dentro da aplicação. Esse modelo modular facilita a manutenção e a escalabilidade do projeto. Outro conceito central do *React* é o *Virtual Document Object Model* (*DOM*), um mecanismo que otimiza a renderização da interface ao criar uma representação virtual do *DOM* real, aplicando apenas as modificações necessárias, o que resulta em um melhor desempenho e menor consumo de recursos (Pinto; Hoffmann; Uriarte, 2023).

O gerenciamento de estado no *React* ocorre de forma unidirecional, garantindo previsibilidade no fluxo de dados. Para aplicações mais complexas, é comum o uso de bibliotecas

auxiliares, como *Redux*, *Recoil* e *Context API*, que oferecem soluções para um controle eficiente do estado global da aplicação (Ferreira; Zuchi, 2018).

2.2.2 *Angular*

O *Angular* é um *framework* de desenvolvimento *web* de código aberto, criado pelo *Google*, focado na construção de aplicações *web* dinâmicas e escaláveis. Lançado inicialmente como *AngularJS* em 2010, o *framework* passou por uma reformulação completa em 2016 com a introdução do *Angular* 2, adotando *TypeScript* como linguagem principal, o que trouxe benefícios como tipagem estática e maior organização do código (Google, 2025).

O *Angular* oferece uma solução completa para o desenvolvimento *web*, incluindo funcionalidades nativas como injeção de dependências, roteamento, manipulação de formulários, comunicação com APIs e testes automatizados. Uma das características centrais do *Angular* é o *Two-Way Data Binding*, que permite uma sincronização bidirecional entre o modelo de dados e a interface do usuário. Isso significa que qualquer alteração nos dados reflete automaticamente na interface, e vice-versa, reduzindo a necessidade de manipulação direta do *DOM*. Além disso, o *Angular* adota uma arquitetura baseada em componentes, facilitando a organização do código e promovendo a reutilização de funcionalidades (Pinto; Hoffmann; Uriarte, 2023).

2.2.3 *Vue*

O *Vue* é um *framework* de desenvolvimento *web* progressivo e de código aberto, criado por Evan You em 2014, voltado para a construção de interfaces de usuário dinâmicas e SPAs (Vue.js, 2025).

Uma das principais características do *Vue* é sua arquitetura progressiva, permitindo que ele seja utilizado de forma incremental em projetos existentes ou como um *framework* completo para desenvolvimento de aplicações *web*. O *Vue* utiliza um sistema de reatividade onde a interface é atualizada automaticamente sempre que há mudanças nos dados (Ferreira; Zuchi, 2018). O *Vue* adota uma sintaxe declarativa e intuitiva, sem abrir mão da escalabilidade necessária para projetos mais complexos. Sua estrutura modular, baseada em componentes reutilizáveis, facilita a organização e manutenção do código. Para gerenciamento de estado, o *Vue* oferece o Pinia, garantindo um controle eficiente dos dados da aplicação (Ferreira; Zuchi, 2018).

3 TRABALHOS RELACIONADOS

Neste Capítulo, serão apresentados alguns trabalhos relacionados destacando as semelhanças e diferenças com o desenvolvido neste trabalho.

3.1 Análise Comparativa Entre as Tecnologias de *Front-end React*, *Angular* e *Vue*

Um dos trabalhos mais relevantes para a presente pesquisa é o artigo "Análise Comparativa Entre as Tecnologias de *Front-end React*, *Angular* e *Vue*" de Pinto, Hoffmann e Uriarte (2023), que apresenta uma avaliação detalhada desses três *frameworks* de desenvolvimento *web* amplamente utilizadas. O objetivo do artigo é comparar as tecnologias em termos de popularidade e desempenho, fornecendo uma visão sobre suas vantagens e limitações para orientar desenvolvedores na escolha da melhor ferramenta.

No artigo, a popularidade foi avaliada com base em métricas como *downloads* no *Node Package Manager* (NPM), número de repositórios no *GitHub* e estrelas atribuídas a cada tecnologia. Além disso, os autores conduziram pesquisas próprias para medir o reconhecimento e a utilização de cada *framework* em diferentes ambientes de trabalho. Os resultados indicaram que o *React* é a tecnologia mais popular, seguido pelo *Angular* e pelo *Vue*.

Já o desempenho foi analisado através de testes de renderização realizados em uma aplicação simples de gerenciamento de contatos. Foram mensuradas métricas como o tempo necessário para adicionar elementos. Os resultados desses testes apontaram que o *Vue* apresentou o melhor desempenho em todos os cenários, seguido pelo *Angular* e, por último, pelo *React*.

3.2 Análise Comparativa das Características de Performance dos *Frameworks JavaScript React* e *Vue*

Outro trabalho relevante para a presente pesquisa é o artigo "Análise Comparativa das Características de Performance dos *JavaScript Frameworks React* e *Vue*" de Almeida *et al.* (2022), que realiza uma avaliação quantitativa das características de desempenho desses dois *frameworks* amplamente utilizados no desenvolvimento de aplicações *web*. O artigo tem como objetivo principal comparar o *React* e o *Vue* em três aspectos específicos: *speed index*, consumo de memória gráfica e tempo de controle de estado, fornecendo insights sobre o desempenho de cada tecnologia.

O estudo utiliza técnicas de mineração de repositórios no *GitHub* para selecionar 30 aplicações de *To-do List* (15 desenvolvidas em *React* e 15 em *Vue*). Em seguida, realiza testes utilizando ferramentas como *Lighthouse* e *DevTools* do *Google Chrome* para coletar métricas de desempenho. Os resultados indicaram que o *React* teve melhor desempenho no *speed index*, enquanto o *Vue* apresentou vantagens em consumo de memória gráfica e controle de estado.

3.3 Análise Comparativa entre *Frameworks Frontend* baseados em *JavaScritpt* para Aplicações *Web*

O artigo "Análise Comparativa entre *Frameworks Frontend* baseados em *JavaScript* para Aplicações *Web*" autorado por Ferreira e Zuchi (2018) procura auxiliar desenvolvedores na escolha do *framework* mais adequado para seus projetos, com foco nos *frameworks React*, *Angular* e *Vue*. O estudo analisa aspectos fundamentais das tecnologias, como arquitetura, documentação e suporte da comunidade, compatibilidade com navegadores, tamanho dos pacotes gerados e tempo de renderização.

A metodologia utilizada no artigo consistiu no desenvolvimento de aplicações simples, do tipo SPAs, com formulários para manipulação de dados e estilização baseada no *Bootstrap*. As aplicações foram executadas em um ambiente de comparação controlado, onde ferramentas como o painel *network* do *Chrome DevTools* foram utilizadas para medir tempos de carregamento e desempenho. Além disso, os autores exploraram métricas como tamanho do arquivo final gerado por cada *framework* e a compatibilidade com diferentes navegadores.

Os resultados do estudo indicaram que o *Vue* se destacou em tempo de renderização, enquanto o *React* apresentou menor tamanho de arquivo gerado. Já o *Angular*, apesar de ser robusto, teve resultados menos favoráveis nesses quesitos. O artigo destaca ainda a importância da documentação e da comunidade, sendo que o *Vue* foi o único a oferecer suporte oficial em português do Brasil.

3.4 Comparação dos trabalhos relacionados com o trabalho proposto

A análise dos trabalhos relacionados permitiu identificar diferentes abordagens utilizadas para avaliar *frameworks* de desenvolvimento *web*. Cada estudo analisado apresenta contribuições relevantes, mas também limitações que o diferenciam do trabalho proposto. A seguir, é feita uma comparação geral entre os trabalhos relacionados e este estudo, destacando seus escopos, metodologias e critérios de análise.

Enquanto alguns estudos focam estritamente no desempenho técnico, outros consideram aspectos mais subjetivos, como a curva de aprendizado. No entanto, a maioria dos estudos não combina análises de desempenho e aprendizado prático com a implementação real de um mesmo projeto em diferentes tecnologias, o que constitui um dos principais diferenciais deste trabalho. Outro ponto de distinção do estudo proposto está na abrangência dos critérios de comparação.

Enquanto a maioria dos trabalhos analisados se concentram em métricas isoladas, este estudo adota uma visão mais ampla, incorporando curva de aprendizado e desempenho.

Além disso, a metodologia adotada neste trabalho se diferencia por empregar um mesmo projeto simples desenvolvido em *React*, *Angular* e *Vue*, permitindo uma avaliação prática e direta das três tecnologias. Essa abordagem garante maior controle sobre os testes e assegura uma comparação mais objetiva entre os *frameworks*. O Quadro 1 sintetiza as principais diferenças entre os trabalhos relacionados e o presente estudo.

Quadro 1 – Comparativo de trabalhos relacionados

	Curva de aprendizado	Desempenho
Pinto, Hoffmann e Uriarte (2023)		x
Almeida *et al.* (2022)		x
Ferreira e Zuchi (2018)	x	x
Esse trabalho	x	x

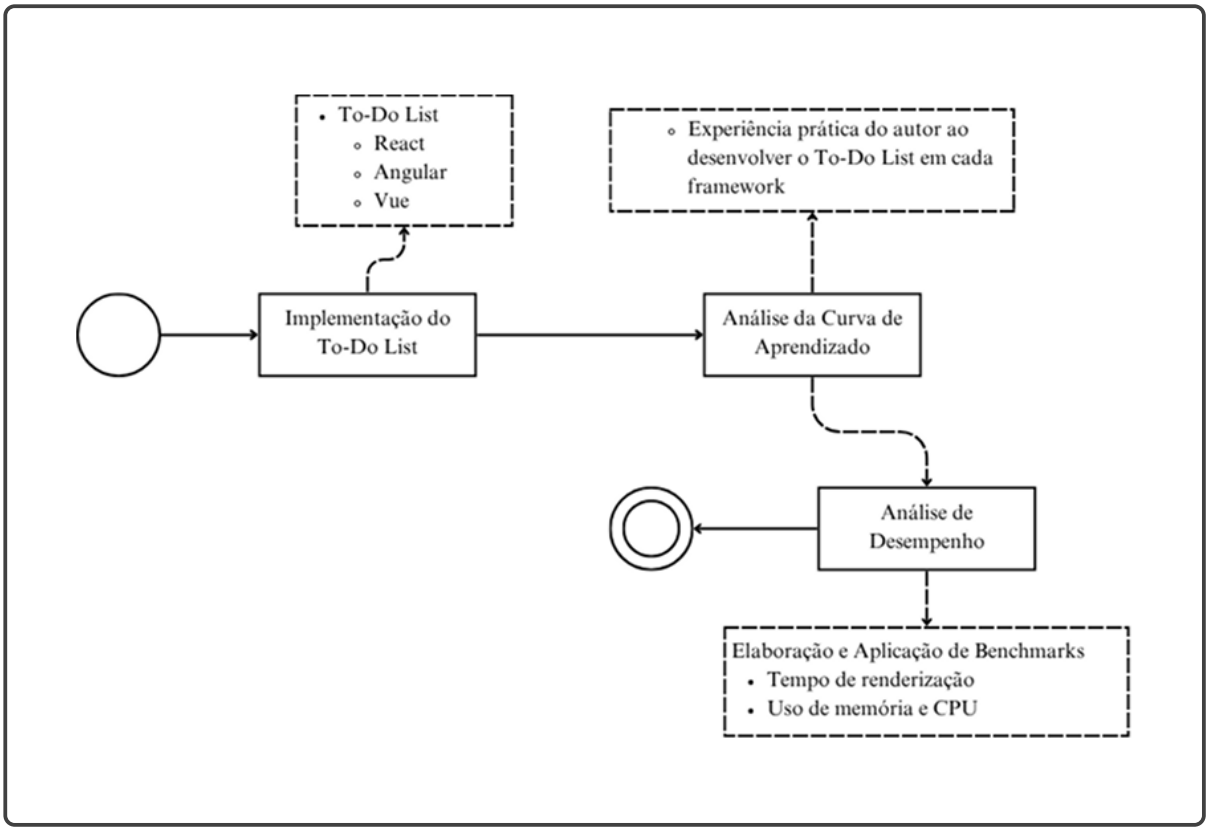
Fonte: elaborado pelo auto (2025)

Com base nessas diferenças, fica evidente que o presente trabalho busca preencher lacunas deixadas pelos estudos existentes ao oferecer uma análise comparativa mais prática e integrada.

4 METODOLOGIAS

Nesta Seção serão apresentadas as metodologias usadas para o desenvolvimento desse trabalho. As etapas a serem seguidas estão representadas no fluxograma da Figura 1.

Figura 1 – Imagem do fluxograma dos passos metodológicos



Fonte: elaborado pelo autor (2025)

4.1 Implementação da aplicação de *To-do List*

Para garantir uma base consistente para a análise, foi desenvolvida uma mesma aplicação utilizando cada um dos três *frameworks React*, *Angular* e *Vue*, permitindo uma avaliação direta sob as mesmas condições. Todo o código-fonte desenvolvido neste trabalho está disponível publicamente no repositório: `https://github.com/Guilhermedsc/to_to_list`.

A aplicação escolhida para ser desenvolvida foi uma lista de tarefas (*To-do List*), por ser um exemplo clássico utilizado em vários estudos comparativos de *frameworks web* (Almeida *et al.*, 2022; Carniato, 2021; Tong; Jikson; Gunawan, 2023). Essa escolha se justifica por

apresentar funcionalidades suficientemente representativas do desenvolvimento moderno de interfaces, sem introduzir complexidade excessiva. As funcionalidades implementadas incluem:

- Adição de tarefas
- Remoção de tarefas
- Edição de tarefas
- Marcação de tarefas como concluídas
- Atualização reativa da interface

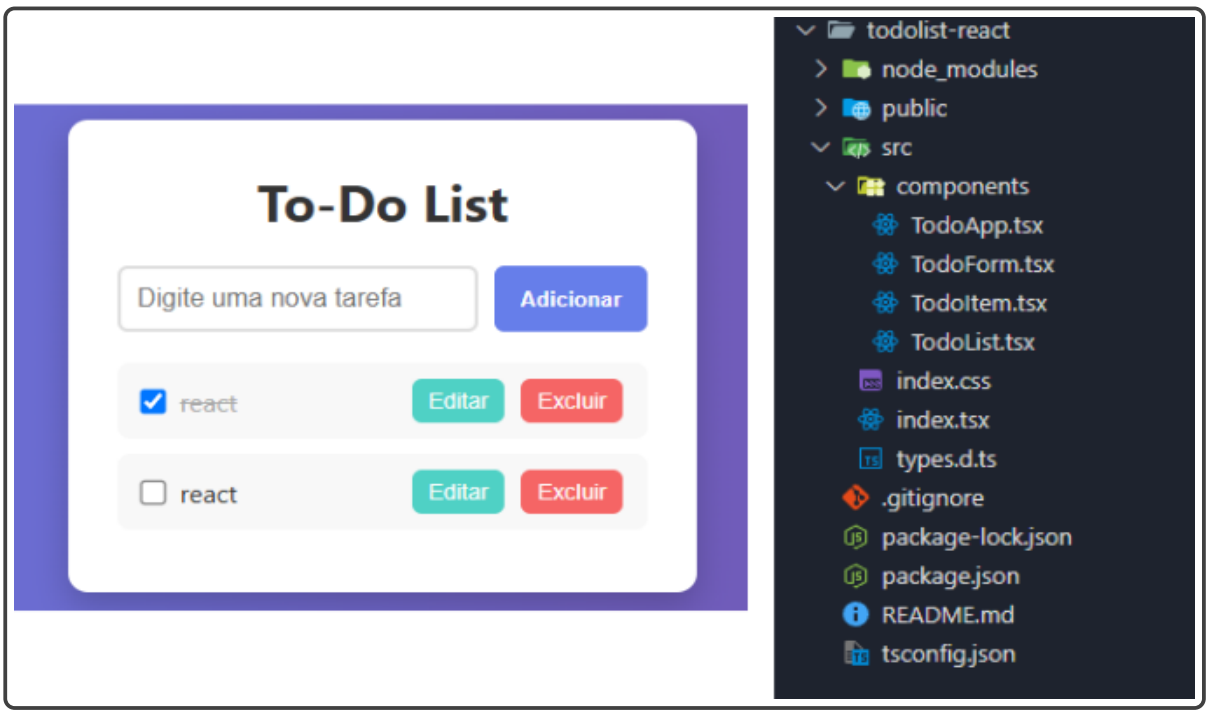
Essas funcionalidades permitem avaliar aspectos cruciais de cada *framework*, como manipulação de estado, renderização dinâmica, tratamento de eventos e ciclo de vida dos componentes, que são elementos diretamente relacionados ao desempenho e à curva de aprendizado. É importante frisar que essa aplicação não tem implementação de *back-end*, já que os *frameworks* tratados neste trabalho são de *front-end*.

A aplicação foi implementada separadamente em cada tecnologia por um único desenvolvedor (o autor), seguindo as práticas recomendadas e a documentação oficial de cada *framework*, o que permite controlar variáveis externas e facilita a avaliação subjetiva do processo de aprendizado. As imagens das interfaces feitas com os *frameworks React*, *Angular* e *Vue* podem ser vistas respectivamente na Figura 2, Figura 3 e Figura 4.

Ao lado da interface em cada imagem está a estrutura de pastas que evidencia uma característica relevante para a curva de aprendizado. Essa característica é a separação de arquivos necessária para cada *framework*. O *framework Angular*, por exemplo, necessita que as partes de estilo, *template* e *script* estejam separadas em arquivos diferentes, levando o desenvolvedor a escrever mais código para integrar as três partes citadas.

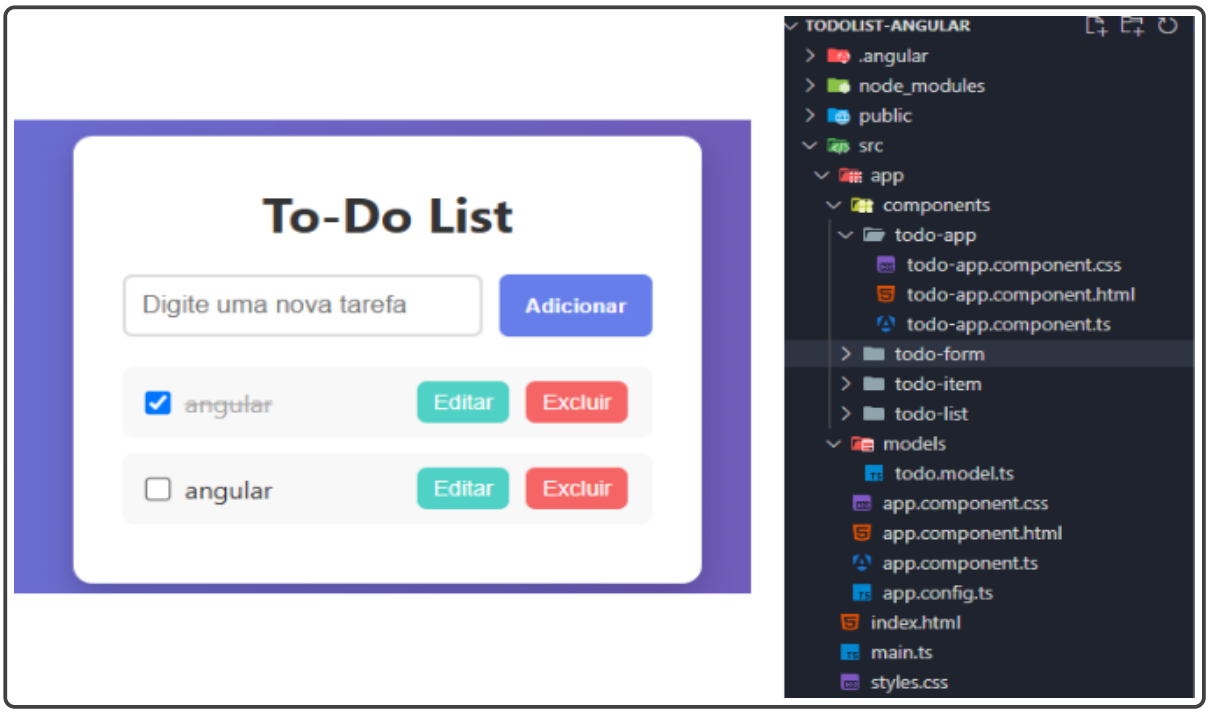
Esse aspecto está diretamente ligado à verbosidade do código, entendida como a quantidade de linhas e estruturas necessárias para implementar uma funcionalidade. Já a modularidade refere-se à divisão do código em componentes independentes e reutilizáveis, o que favorece a organização e manutenção. A legibilidade, por sua vez, diz respeito à clareza e facilidade de compreensão do código por um desenvolvedor, influenciada pela sintaxe, estrutura e consistência do *framework*.

Figura 2 – Imagem da interface feita com *React*



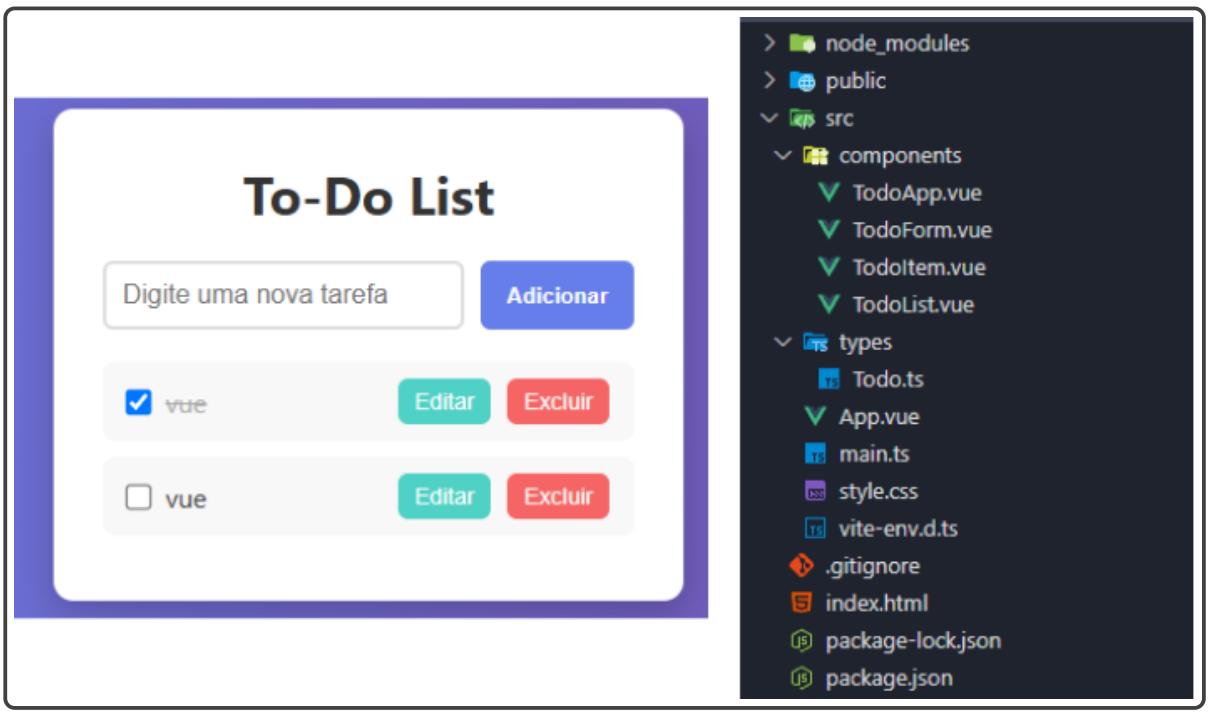
Fonte: elaborado pelo autor (2025)

Figura 3 – Imagem da interface feita com *Angular*



Fonte: elaborado pelo autor (2025)

Figura 4 – Imagem da interface feita com *Vue*



Fonte: elaborado pelo autor (2025)

4.2 Análise da Curva de Aprendizado

A curva de aprendizado foi avaliada com base na experiência prática do autor durante a implementação da aplicação *To-do List* em *React*, *Angular* e *Vue*. O objetivo é compreender as dificuldades e facilidades iniciais associadas a cada *framework*, especialmente no contexto de um desenvolvedor que o utiliza pela primeira vez. Para garantir uma análise consistente, foram definidos quatro critérios qualitativos de avaliação:

- Clareza da documentação oficial: avaliação da facilidade de compreensão dos tutoriais, exemplos de código e explicação de conceitos fundamentais (componentes, estado, diretivas).
- Complexidade da configuração inicial: análise do processo de criação do projeto, necessidade de ferramentas adicionais, uso de *CLI* e tempo necessário para obter um ambiente funcional.
- Verbosidade e organização do código: comparação da quantidade e estrutura do código necessário para implementar as mesmas funcionalidades em cada *framework*, conside-

rando legibilidade, simplicidade e modularidade. Um código menos verboso e mais modular tende a ser mais legível e fácil de manter.

- Tempo de desenvolvimento: registro do tempo total gasto em cada etapa (configuração, implementação das funcionalidades, depuração e ajustes), medido em horas.

Durante a implementação, o autor manteve um diário de bordo (ou *development log*), onde foram anotadas:

- Dificuldades encontradas (erros comuns, falta de exemplos, sintaxe confusa)
- Recursos utilizados (documentação oficial, tutoriais, fóruns como Stack Overflow)
- Impressões subjetivas sobre a intuitividade da tecnologia
- Momentos de produtividade ou bloqueio

Esse registro foi feito em tempo real, logo após cada sessão de desenvolvimento, para minimizar vieses de memória. Ao final da implementação em cada *framework*, os dados foram consolidados em uma tabela comparativa, permitindo uma análise crítica e fundamentada da curva de aprendizado.

Essa abordagem, embora baseada na experiência individual, busca simular o processo de adoção por um desenvolvedor real que precise aprender uma nova tecnologia em um contexto profissional. Assim, os resultados podem oferecer *insights* valiosos sobre a acessibilidade, produtividade inicial e suporte oferecido por cada *framework*, especialmente para iniciantes ou equipes em processo de migração tecnológica.

4.3 Análise de Desempenho

Após a implementação da aplicação *To-do List* em *React*, *Angular* e *Vue*, foi realizada uma análise de desempenho para comparar o comportamento de cada *framework* em cenários práticos e controlados. A escolha das métricas visa garantir uma avaliação objetiva, reprodutível e tecnicamente fundamentada, permitindo identificar diferenças relevantes no desempenho de cada tecnologia. As métricas adotadas para avaliação são as seguintes:

- Tempo de inicialização: tempo necessário para que a aplicação seja carregada e esteja interativa em um navegador. Através da aba de performance das ferramentas do desenvolvedor no *Chrome* é possível gravar dados sobre a performance de uma página em sua inicialização (St. Marthe; Emelianova, 2025). Dentro desses dados está incluso o tempo total de inicialização, que é dado em milissegundos, mas será convertido para segundos neste trabalho. Essa métrica reflete a eficiência do processo de compilação e da iniciali-

zação do ambiente.

- Uso de memória: consumo de memória durante a execução da aplicação. Através da aba de memória das ferramentas do desenvolvedor no *Chrome* é possível criar um *snapshot* de *heap* que mostra a distribuição de memória pelos objetos *Javascript* da página e dos nós *DOM* relacionados (Kearney; Emelianova, 2025). Essa métrica é dada em *megabytes*. Essa métrica mostra o quão eficiente e inteligente é a forma como os *frameworks* utilizam os recursos do sistema.
- Uso de CPU: consumo dos recursos do sistema durante a execução. Através do painel de monitoramento das ferramentas do desenvolvedor no *Chrome* é possível ter acesso à porcentagem de uso da CPU durante o uso da página (St. Marthe, 2025). Os valores serão registrados durante a renderização inicial e a média da porcentagem de uso durante esse período será usada como métrica. Essa métrica foi escolhida visando avaliar o impacto de cada *framework* na performance do dispositivo.

Todos os testes foram realizados no mesmo ambiente físico e de *software*, com as seguintes características padronizadas:

- Sistema operacional: Windows 11
- Navegador: *Google Chrome*, com extensões desativadas
- *Hardware*: Máquina com configurações fixas. CPU Intel i5, 16GB de RAM e SSD de 256GB
- Condições de rede 3G simuladas por meio do *Throttling* do *Chrome DevTools*.

Cada métrica foi medida 10 vezes para cada *framework*, e o valor médio foi registrado para minimizar variações aleatórias. Os resultados serão analisados quantitativamente e sintetizados em observações concisas na apresentação final.

5 RESULTADOS

Neste Capítulo serão discutidos os resultados dos testes descritos no Capítulo 4. Primeiro serão discutidos os resultados da curva de aprendizado, depois os resultados de desempenho e por fim uma discussão geral cruzando os resultados de curva de aprendizado e desempenho.

5.1 Curva de aprendizado

A avaliação da curva de aprendizado foi conduzida com base na experiência prática do autor durante a implementação da aplicação *To-do List* em *React*, *Angular* e *Vue*. O objetivo foi identificar as facilidades e dificuldades iniciais enfrentadas ao adotar cada *framework*, simulando o processo de aprendizado por um desenvolvedor em contato pela primeira vez com a tecnologia. Os critérios analisados foram clareza da documentação oficial, complexidade da configuração inicial, verbosidade do código e tempo de desenvolvimento. Os dados foram registrados em um diário de bordo e sintetizados em anotações qualitativas, garantindo uma análise objetiva e reprodutível.

Para ajudar na análise da quantidade de código foi criado um *script python* para a contagem de linhas e caracteres dos arquivos criados para cada *framework*. Cada *framework* tem arquivos de configuração e *boilerplate* diferentes que são gerados automaticamente na criação do projeto. Por essa razão os únicos arquivos incluídos na análise de quantidade de código são os arquivos usados para a criação dos componentes.

5.1.1 *React*

O uso de *React* apresentou um processo de aprendizado acessível, especialmente após a familiarização com conceitos centrais como componentes funcionais e *hooks*. A ferramenta `create-react-app` simplificou significativamente a configuração inicial, permitindo que o ambiente de desenvolvimento fosse preparado com poucos comandos e sem intervenção manual em arquivos de configuração. As anotações gerais podem ser vistas no Quadro 2.

Quadro 2 – Anotações sobre a curva de aprendizado em *React*

Critério	Anotações
Documentação	Documentação clara e bem estruturada, com exemplos práticos, mas sem suporte oficial em português.
Configuração Inicial	Processo automatizado e simples com `create-react-app`.
Quantidade de Código	Código conciso e modular, com boa divisão entre componentes. Junção de estilo e *script* junto aos componentes diminui a quantidade de arquivos.
Tempo de Desenvolvimento	Implementação concluída em 10 horas, com curva inicial moderada e produtividade crescente após domínio dos conceitos.

Fonte: Fonte: elaborado pelo autor (2025).

Apesar da qualidade da documentação, a ausência de tradução para o português pode representar uma barreira para alguns desenvolvedores. A sintaxe TSX e o modelo baseado em estado exigem um tempo de adaptação, mas uma vez compreendidos, permitem um desenvolvimento ágil. Em geral, *React* mostrou-se uma opção equilibrada entre poderio técnico e acessibilidade.

A análise quantitativa do código desenvolvido em *React* revela um total de 118 linhas e 3.447 caracteres distribuídos em quatro arquivos principais, conforme apresentado na Tabela 1. A média de 29,5 linhas por arquivo indica um código conciso e bem modularizado, alinhado à filosofia de componentes leves e reutilizáveis. O arquivo `TodoApp.tsx` é o mais extenso, com 39 linhas, seguido por `TodoItem.tsx` com 32 linhas, refletindo o quão compactos esses componentes são na aplicação.

Tabela 1 – Análise de arquivos: *React*

Arquivo	Linhas	Caracteres
`TodoApp.tsx`	39	1121
`TodoForm.tsx`	25	633
`TodoItem.tsx`	32	1103
`TodoList.tsx`	22	590
Total	**118**	**3447**
Média por arquivo	**29.5**	**861.75**

Fonte: Fonte: elaborado pelo autor (2025).

5.1.2 *Angular*

Angular apresentou uma curva de aprendizado mais acentuada, principalmente devido à sua arquitetura robusta e ao alto nível de configuração inicial. Embora o uso do *CLI* (`ng new` e `ng generate`) automatize a criação de componentes e serviços, a quantidade de arquivos gerados por padrão pode gerar sobrecarga cognitiva para iniciantes. As anotações gerais podem ser vistas no Quadro 3.

Quadro 3 – Anotações sobre a curva de aprendizado em *Angular*

Critério	**Anotações**
Documentação	Completa e técnica, mas densa e com curva de leitura elevada. Sem suporte em português.
Configuração Inicial	Automatizada, mas gera muitos arquivos e pastas, aumentando a complexidade.
Quantidade de Código	Código mais verboso, com necessidade de anotações (*decorators*) e tipagem rígida além de necessitar separar estilo, *template* e *script* em arquivos separados.
Tempo de Desenvolvimento	Implementação levou 14 horas, com dificuldades iniciais em entender a estrutura e o fluxo de dados.

Fonte: Fonte: elaborado pelo autor (2025).

A tipagem forte e o uso de *TypeScript* são vantagens para manutenção, mas aumentam a barreira inicial. A documentação, embora de alta qualidade, exige tempo para absorção. *Angular* mostrou-se mais adequado para equipes com experiência prévia em desenvolvimento estruturado, mas pode desestimular iniciantes.

A análise da quantidade de código em *Angular* (Tabela 2) mostra um total de 134 linhas e 4.060 caracteres, distribuídos em 12 arquivos, incluindo separação estrita entre *HTML*, *CSS* e *TypeScript*. A média de apenas 11,17 linhas por arquivo é enganosa, pois reflete a necessidade de separação da lógica de implementação dos componentes imposta pela arquitetura, com múltiplos arquivos por componente. Essa separação aumenta a verbosidade e o número de arquivos, contribuindo para a complexidade percebida durante o aprendizado. Componentes como `todo-item.component.ts` (29 linhas) e `todo-app.component.ts` (34 linhas) concentram a lógica principal, enquanto arquivos de estilo e *template* são frequentemente vazios ou mínimos.

Tabela 2 – Análise de arquivos: *Angular*

Arquivo	Linhas	Caracteres
`todo-app.component.css`	0	0
`todo-app.component.html`	11	293
`todo-app.component.ts`	34	1020
`todo-form.component.css`	0	0
`todo-form.component.html`	4	183
`todo-form.component.ts`	21	555
`todo-item.component.css`	0	0
`todo-item.component.html`	6	214
`todo-item.component.ts`	29	883
`todo-list.component.css`	0	0
`todo-list.component.html`	11	234
`todo-list.component.ts`	18	678
Total	**134**	**4060**
Média por arquivo	**11.17**	**338.33**

Fonte: Fonte: elaborado pelo autor (2025).

5.1.3 *Vue*

Vue se destacou como a tecnologia com a curva de aprendizado mais suave. O comando `npm init vue@latest` oferece um processo de criação de projeto personalizável e intuitivo, permitindo selecionar apenas os recursos necessários. A estrutura de arquivos é limpa e a sintaxe é direta, facilitando a compreensão mesmo para quem está começando. As anotações gerais podem ser vistas no Quadro 4.

Quadro 4 – Anotações sobre a curva de aprendizado em *Vue*

Critério	Anotações
Documentação	Bem clara, com guias passo a passo e suporte oficial em português.
Configuração Inicial	Simples e rápida, com opções para diferentes níveis de complexidade.
Quantidade de Código	Código conciso e bem organizado, com baixa verbosidade e alta legibilidade. Também junta estilo, *script* e componente em um arquivo só.
Tempo de Desenvolvimento	Implementação concluída em 8 horas, com progressão rápida desde o início.

Fonte: Fonte: elaborado pelo autor (2025).

A combinação de documentação acessível, configuração leve e baixa complexidade

do código torna *Vue* especialmente indicado para projetos com prazos curtos ou equipes em processo de capacitação. O suporte oficial em português é um fator decisivo para desenvolvedores brasileiros, reduzindo significativamente o tempo de aprendizado.

Quanto à quantidade de código, o *Vue* apresentou o menor volume total, com 114 linhas e 2.909 caracteres (Tabela 3), distribuídos em quatro arquivos. A média de 28,5 linhas por arquivo reflete um equilíbrio entre concisão e clareza. O arquivo `TodoItem.vue` é o mais extenso (41 linhas), seguido por `TodoApp.vue` (38 linhas), indicando uma concentração semelhante à do *React*. A arquitetura de arquivos únicos (*Single File Component*) permite reunir *template*, *script* e estilo em um único arquivo, reduzindo a fragmentação e facilitando a manutenção.

Tabela 3 – Análise de arquivos: *Vue*

Arquivo	**Linhas**	**Caracteres**
`TodoApp.vue`	38	952
`TodoForm.vue`	18	413
`TodoItem.vue`	41	1087
`TodoList.vue`	17	457
Total	**114**	**2909**
Média por arquivo	**28.5**	**727.25**

Fonte: Fonte: elaborado pelo autor (2025).

5.1.4 Análise Comparativa

Ao comparar os três *frameworks*, é possível identificar perfis distintos de acessibilidade. *Vue* demonstrou a curva de aprendizado mais favorável, com documentação clara, configuração simples e menor tempo de implementação. *React* apresentou um equilíbrio entre simplicidade e complexidade, sendo acessível após uma curva inicial moderada. *Angular*, por outro lado, exigiu mais tempo e esforço cognitivo, com uma estrutura mais complexa e maior verbosidade, o que pode desestimular novos desenvolvedores.

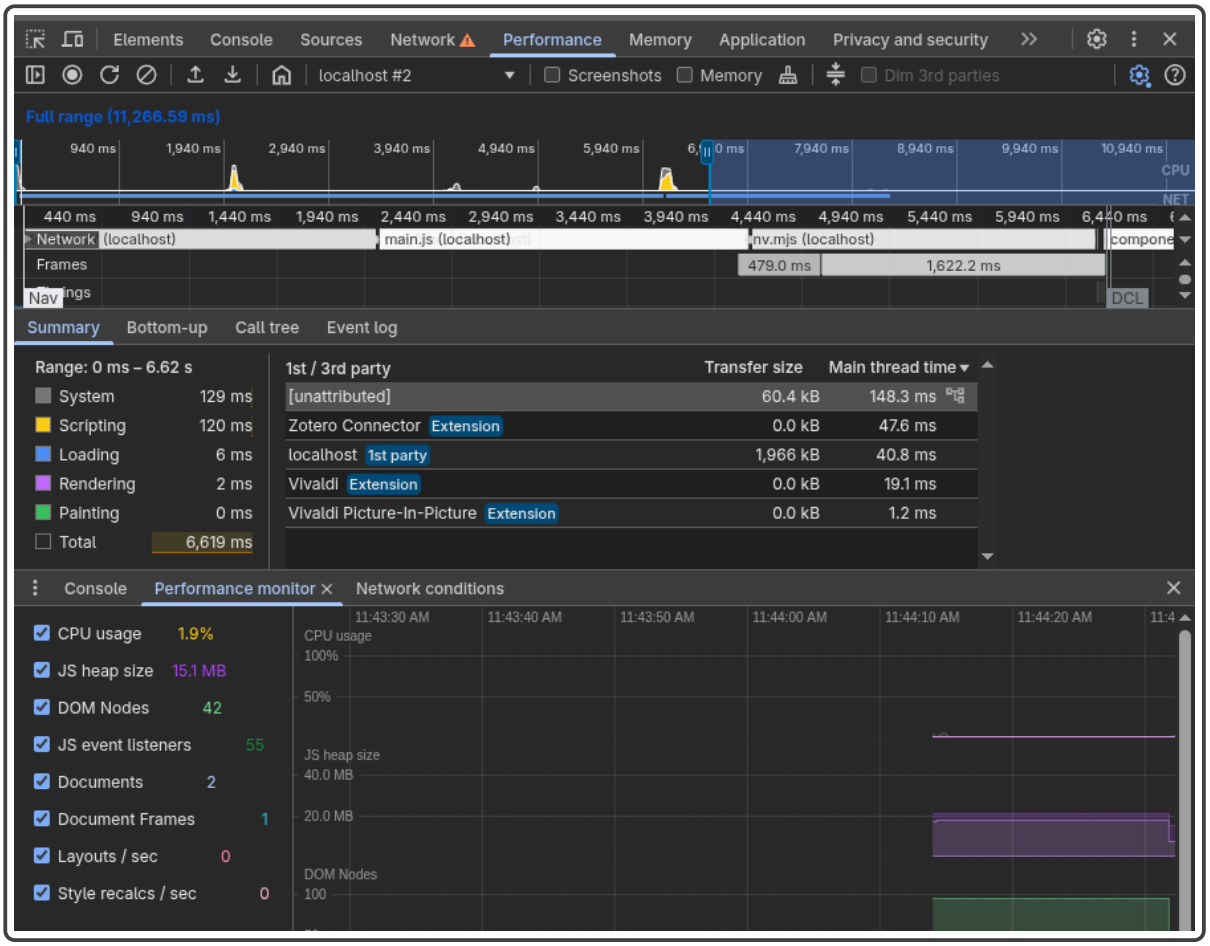
Esses resultados indicam que, para contextos que priorizam produtividade imediata e facilidade de adaptação, *Vue* é a opção mais vantajosa. *React* é adequado para equipes que buscam flexibilidade e ecossistema robusto, enquanto *Angular* se destina a projetos de larga escala com requisitos de manutenibilidade e tipagem rigorosa.

5.2 Desempenho

Nesta Seção é apresentada a síntese dos valores de desempenho obtidos dos testes feitos com os *frameworks*. Os dados estão dispostos nas Tabelas 4, 5 e 6, uma para cada *framework*. Dentro dessas tabelas estão valores de tempo de inicialização, uso de memória e uso de CPU. Estão inclusos os valores das 10 execuções dos testes com a média dos valores na última linha.

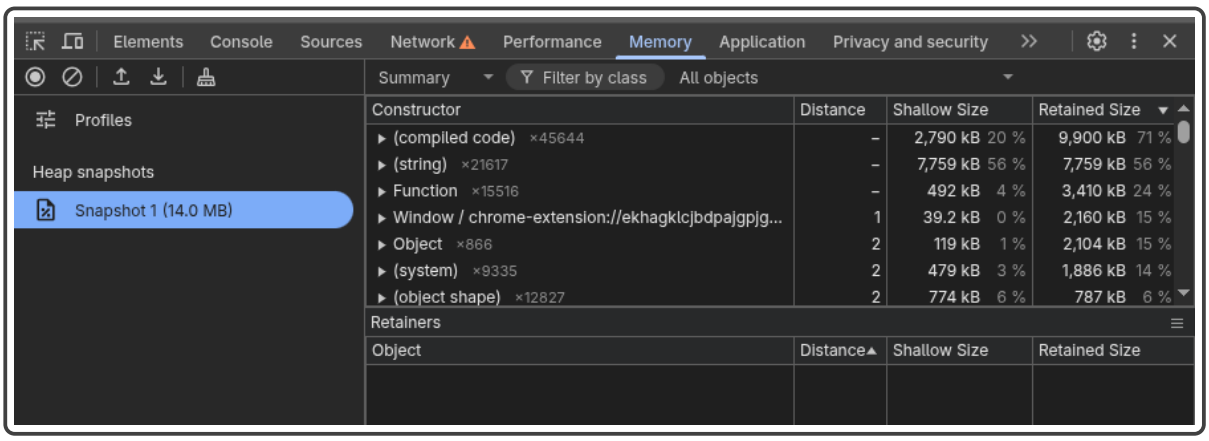
Como citado na Seção 4.3, os dados foram coletados usando as ferramentas do desenvolvedor do *Chrome*. A Figura 5 e Figura 6 mostram respectivamente a aba de performance e memória para uma execução de testes com a versão da aplicação feita com *Angular*. É importante frisar que os valores de tempo estão elevados devido ao *throttling* usado para simular uma rede 3G.

Figura 5 – Imagem da aba de performance do *Chrome DevTools*



Fonte: elaborado pelo autor (2025)

Figura 6 – Imagem da de memória do *Chrome DevTools*



Fonte: elaborado pelo autor (2025)

Tabela 4 – Valores de desempenho medidos para *React* (10 execuções)

Execução	**Tempo**	**Memória**	**CPU**
T1	4,0 s	10 MB	1,3 %
T2	4,3 s	9 MB	1,4 %
T3	3,9 s	10 MB	1,5 %
T4	4,2 s	10 MB	1,4 %
T5	4,4 s	8 MB	1,5 %
T6	4,1 s	10 MB	1,3 %
T7	4,3 s	10 MB	1,4 %
T8	4,0 s	10 MB	1,3 %
T9	4,2 s	9 MB	1,4 %
T10	4,1 s	12 MB	1,5 %
Média	**4,13 s**	**9,8 MB**	**1,4 %**

Fonte: Fonte: elaborado pelo autor (2025).

Tabela 5 – Valores de desempenho medidos para *Angular* (10 execuções)

Execução	**Tempo**	**Memória**	**CPU**
T1	6,0 s	14 MB	2,3 %
T2	6,3 s	15 MB	2,4 %
T3	6,1 s	14 MB	2,3 %
T4	6,4 s	15 MB	2,5 %
T5	6,0 s	14 MB	2,2 %
T6	6,2 s	14 MB	2,4 %
T7	6,3 s	15 MB	2,4 %
T8	6,1 s	14 MB	2,3 %
T9	6,2 s	15 MB	2,5 %
T10	6,2 s	14 MB	2,4 %
Média	**6,18 s**	**14,5 MB**	**2,38 %**

Fonte: Fonte: elaborado pelo autor (2025).

Tabela 6 – Valores de desempenho medidos para *Vue* (10 execuções)

Execução	**Tempo**	**Memória**	**CPU**
T1	12,0 s	7 MB	1,6 %
T2	12,5 s	7 MB	1,7 %
T3	11,8 s	8 MB	1,5 %
T4	12,6 s	7 MB	1,6 %
T5	12,1 s	7 MB	1,6 %
T6	12,4 s	8 MB	1,7 %
T7	12,3 s	7 MB	1,6 %
T8	12,0 s	7 MB	1,5 %
T9	12,2 s	7 MB	1,7 %
T10	12,3 s	7 MB	1,7 %
Média	**12,22 s**	**7,4 MB**	**1,62 %**

Fonte: Fonte: elaborado pelo autor (2025).

Os dados coletados revelam diferenças claras no desempenho entre os três *frameworks*. *React* destacou-se com o menor tempo médio de inicialização (4,13 s), menor consumo de memória (9,8 MB) e menor uso de CPU (1,4 %), demonstrando eficiência e leveza no ambiente de desenvolvimento. *Angular* apresentou o pior desempenho geral, com tempo médio de inicialização elevado (6,18 s), alto consumo de memória (14,5 MB) e uso intenso de CPU (2,38 %), reflexo de sua arquitetura completa e processo de detecção de mudanças mais

custoso. *Vue* mostrou um comportamento equilibrado em memória (7,4 MB), mas surpreendeu com um tempo de inicialização elevado (12,22 s), possivelmente devido à configuração de *build* utilizada.

Esses resultados indicam que *React* e *Vue* são mais adequados para ambientes que priorizam agilidade e baixo consumo de recursos, enquanto *Angular*, apesar de robusto, pode impactar negativamente a produtividade inicial. A seguir a conclusão feita a partir desses dados:

- *React*: Excelente desempenho geral, com inicialização rápida e baixo uso de CPU e memória.
- *Angular*: Alto custo de inicialização e uso elevado de recursos, o que pode afetar a experiência em máquinas menos potentes.
- *Vue*: Desempenho equilibrado e previsível, com boa eficiência de recursos e tempo de resposta ágil.

5.3 Discussão Geral

A análise comparativa entre *React*, *Angular* e *Vue* revelou perfis distintos em termos de desempenho e curva de aprendizado, indicando que a escolha do *framework* mais adequado depende diretamente do contexto de uso, dos objetivos do projeto e do perfil da equipe de desenvolvimento.

Em relação ao desempenho, *React* destacou-se como a solução mais eficiente no ambiente testado, apresentando o menor tempo médio de inicialização (4,13 s), baixo consumo de memória (9,8 MB) e uso moderado de CPU (1,4 %). Essas características o tornam especialmente adequado para ambientes de desenvolvimento ágeis, onde produtividade e tempo de resposta são críticos. *Angular*, embora robusto, apresentou o maior consumo de recursos (14,5 MB de memória e 2,38 % de CPU), o que pode impactar negativamente a experiência em máquinas com configurações modestas. Já *Vue*, apesar de leve em memória (7,4 MB), surpreendeu com um tempo de inicialização elevado (12,22 s), possivelmente devido à configuração de build ou ao uso de ferramentas não otimizadas, como o *legacy build* em vez do *Vite*.

No que diz respeito à curva de aprendizado, *Vue* foi o *framework* que ofereceu a experiência mais acessível. Sua documentação clara, suporte oficial em português, configuração simples e código conciso permitiram que a implementação fosse concluída em apenas 8 horas (o menor tempo entre os três). Esse conjunto de fatores o posiciona como a melhor opção para iniciantes, equipes em capacitação ou projetos com prazos curtos. *React* mostrou-se uma

alternativa equilibrada, apesar de não ter suporte em português, sua documentação de qualidade e ferramentas como `create-react-app` facilitam a adoção, especialmente para quem já tem familiaridade com *JavaScript* e *TypeScript*. *Angular*, por outro lado, exigiu 14 horas de desenvolvimento e apresentou uma curva de aprendizado mais acentuada, devido à complexidade da arquitetura, verbosidade do código e densidade da documentação, características que, embora sejam vantajosas em projetos de larga escala, podem desestimular novos desenvolvedores.

Esses resultados indicam um *trade-off* claro entre desempenho e acessibilidade. *React* combina bom desempenho com uma curva de aprendizado razoável, sendo uma escolha versátil para diversos cenários. *Vue* prioriza acessibilidade e produtividade inicial, mas pode apresentar limitações de desempenho dependendo da configuração. *Angular*, por sua vez, oferece robustez e escalabilidade, mas com custos elevados em termos de recursos e tempo de aprendizado.

Assim, a seleção do *framework* deve considerar não apenas aspectos técnicos, mas também fatores humanos e organizacionais. Para projetos educacionais, *startups* ou equipes com pouco tempo para aprendizado, *Vue* se mostra a opção mais vantajosa. Para projetos corporativos de médio a grande porte, onde manutenibilidade, tipagem forte e arquitetura bem definida são essenciais, *Angular* pode ser a escolha mais apropriada. Já para projetos que buscam equilíbrio entre desempenho, flexibilidade e ecossistema amplo, *React* emerge como a solução mais versátil e amplamente recomendada.

É importante ressaltar que os resultados deste estudo são baseados em um ambiente controlado e na experiência individual do autor. Embora os dados tenham sido coletados de forma sistemática, a subjetividade da curva de aprendizado e as particularidades do ambiente de teste (*hardware*, configuração de *build*, versões das ferramentas) podem influenciar os resultados. Estudos futuros poderiam ampliar a amostra de desenvolvedores ou testar cenários mais complexos, como aplicações com roteamento, autenticação e integração com APIs.

Em resumo, não há um *framework* melhor universalmente, mas sim aquele que melhor se alinha aos requisitos técnicos, temporais e humanos do projeto. Este trabalho oferece um panorama prático e detalhado que pode auxiliar gestores, desenvolvedores e pesquisadores na tomada de decisões mais informadas no contexto do desenvolvimento *web* moderno.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve como objetivo comparar os *frameworks* de desenvolvimento *web* *React*, *Angular* e *Vue* em duas dimensões principais: desempenho e curva de aprendizado. Para isso, foi desenvolvida uma aplicação *To-do List* em cada tecnologia, seguida de uma avaliação prática e sistemática dos resultados.

Quanto ao desempenho, *React* destacou-se com o melhor equilíbrio entre tempo de inicialização, uso de memória e CPU, sendo a opção mais eficiente no ambiente testado. *Angular* apresentou alto consumo de recursos, enquanto *Vue*, apesar de leve em memória, mostrou tempo de inicialização inesperadamente alto, indicando dependência crítica da configuração de build.

Na análise da curva de aprendizado, *Vue* foi o *framework* mais acessível, com documentação clara, suporte em português e menor tempo de implementação (8 horas). *React* mostrou-se uma alternativa equilibrada, com boa produtividade após uma curva inicial moderada. *Angular* exigiu mais tempo (14 horas) e esforço cognitivo, devido à sua arquitetura complexa e verbosidade.

Como contribuição, esse estudo oferece uma visão prática e detalhada que pode auxiliar desenvolvedores iniciantes e estudantes na escolha do *framework* mais adequado ao contexto do projeto, seja ele educacional, corporativo ou ágil. Os resultados reforçam que não há uma solução única ideal, mas sim escolhas que dependem de fatores técnicos, humanos e organizacionais.

Como trabalhos futuros, sugere-se a ampliação da amostra de desenvolvedores, a inclusão de cenários mais complexos (como roteamento e autenticação) e a realização de testes em diferentes ambientes de *hardware*, para validar a reprodutibilidade dos resultados.

REFERÊNCIAS

ALMEIDA, Pedro H Marques *et al.* Análise comparativa das características de performance dos javascript frameworks react e vue. pt, 2022. Disponível em: https://bib.pucminas.br/pergamumweb/vinculos/000004/00000433.pdf. Acesso em: 1 ago. 2025.

CARNIATO, Ryan. **JavaScript Framework TodoMVC Size Comparison**. en. [*S. l.: s. n.*], out. 2021. Disponível em: https://dev.to/this-is-learning/javascript-framework-todomvc-size-comparison-504f. Acesso em: 2 ago. 2025.

CHEN, Quan Liang; SHIMOMURA, Takao. Automatic Generation of Web Applications from Visual High-Level Functional Web Components. **Advances in Software Engineering**, v. 2009, n. 1, p. 879725, 2009. DOI: https://doi.org/10.1155/2009/879725. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1155/2009/879725. Acesso em: 1 ago. 2025.

FERREIRA, Haline Kelly; ZUCHI, Jederson Donizete. Análise comparativa entre frameworks frontend baseados em javascript para aplicações web. **Revista Interface Tecnológica**, v. 15, n. 2, p. 111–123, 2018. Disponível em: https://revista.fatectq.edu.br/interfacetecnologica/article/view/502. Acesso em: 1 ago. 2025.

FLANAGAN, D. **JavaScript: The Definitive Guide**: Master the World's Most-used Programming Language. [*S. l.*]: O'Reilly Media, Incorporated, 2020. ISBN 978-1-4919-5202-3. Disponível em: https://books.google.com.br/books?id=b5G4zAEACAAJ. Acesso em: 1 ago. 2025.

GOOGLE. **Angular**. en. [*S. l.: s. n.*]. Disponível em: https://angular.dev/. Acesso em: 1 ago. 2025.

KAUR, Daljit; KAUR, Parminder. Ramification of hustling the web application development. **International Journal of Engineering and Technology(UAE)**, v. 7, p. 6–9, maio 2018. DOI: 10.14419/ijet.v7i2.30.13454.

KEARNEY, Meggin; EMELIANOVA, Sofia. **Gravar snapshots de heap | Chrome DevTools**. pt-BR-x-mtfrom-en. [*S. l.: s. n.*]. Disponível em: https://developer.chrome.com/docs/devtools/memory-problems/heap-snapshots?hl=pt-br. Acesso em: 2 ago. 2025.

KOLLA, Sireesha; GANTA, Srujan. Evolving Trends in Mobile Web Application Development from Desktop Roots to Browser-Based Solutions. **International Journal of Multidisciplinary Research and Growth Evaluation**, v. 06, p. 1493–1496, jun. 2025. DOI: 10.54660/.IJMRGE.2025.6.3.1493-1496.

KRISHNA, A. A Survey on Current Technologies for Web Development. **International Journal of Engineering Research and**, v. V9, jun. 2020. DOI: 10.17577/IJERTV9IS060267.

MARCOTTE, Ethan. **Responsive web design**. New York, NY: A Book Apart, 2011. (Brief books for people who make websites, 4). ISBN 978-0-9844425-7-7.

META. **React**. en. [*S. l.: s. n.*]. Disponível em: https://react.dev/. Acesso em: 1 ago. 2025.

MULLINS, Cheri. Responsive, mobile app, mobile first: untangling the UX design web in practical experience. *In:* PROCEEDINGS of the 33rd Annual International Conference on the Design of Communication. Limerick, Ireland: Association for Computing Machinery, 2015. (SIGDOC '15). ISBN 9781450336482. DOI: 10.1145/2775441.2775478. Disponível em: https://doi.org/10.1145/2775441.2775478. Acesso em: 1 ago. 2025.

NGUYEN, Luong Anh Tuan *et al.* Design and Implementation of Web Application Based on MVC Laravel Architecture. en. **European Journal of Electrical Engineering and Computer Science**, v. 6, n. 4, p. 23–29, ago. 2022. ISSN 2736-5751. DOI: 10.24018/ejece.2022.6.4.448. Disponível em: https://www.ejece.org/index.php/ejece/article/view/448. Acesso em: 1 ago. 2025.

PINTO, Luana Alves; HOFFMANN, Sandy; URIARTE, Luiz Ricardo. Análise comparativa entre as tecnologias de front-end React, Angular e Vue. pt. **Revista de extensão e iniciação científica da UNISOCIESC**, v. 10, n. 1, 2023. Number: 1. ISSN 2358-4432. Disponível em: https://dalfovo.com/ojs/index.php/reis/article/view/398. Acesso em: 1 ago. 2025.

ST. MARTHE, Dale. **Painel do monitor de desempenho | Chrome DevTools**. pt-BR-x-mtfrom-en. [*S. l.: s. n.*]. Disponível em: https://developer.chrome.com/docs/devtools/performance-monitor?hl=pt-br. Acesso em: 2 ago. 2025.

ST. MARTHE, Dale; EMELIANOVA, Sofia. **Painel de desempenho: analise o desempenho do seu site | Chrome DevTools**. pt-BR-x-mtfrom-en. [*S. l.: s. n.*]. Disponível em: https://developer.chrome.com/docs/devtools/performance/overview?hl=pt-br. Acesso em: 2 ago. 2025.

STACK OVERFLOW. **Technology | 2025 Stack Overflow Developer Survey**. en. [*S. l.: s. n.*]. Disponível em: https://survey.stackoverflow.co/2025/technology#1-web-frameworks-and-technologies/. Acesso em: 1 ago. 2025.

SUTCLIFFE, Alistair. Designing for User Experience and Engagement. *In:* [*s. l.: s. n.*], maio 2016. p. 105–126. ISBN 978-3-319-27444-7. DOI: 10.1007/978-3-319-27446-1_5.

TONG, Jason; JIKSON, Ricky Rivaldo; GUNAWAN, Alexander Agung Santoso. Comparative Performance Analysis of Javascript Frontend Web Frameworks. *In:* 2023 3rd International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS). [*S. l.: s. n.*], 2023. p. 81–86. DOI: 10.1109/ICE3IS59323.2023.10335250.

VUE.JS. **Vue.js**. en-US. [*S. l.: s. n.*]. Disponível em: https://vuejs.org/. Acesso em: 1 ago. 2025.

VYAS, Rishi. Comparative Analysis on Front-End Frameworks for Web Applications. **International Journal for Research in Applied Science and Engineering Technology**, v. 10, p. 298–307, jul. 2022. DOI: 10.22214/ijraset.2022.45260.