UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

ANTONIO CRISTIANO MACIEL DA SILVA JUNIOR

COMPARAÇÃO ENTRE OS PRINCIPAIS FRAMEWORKS JAVASCRIPT DE FRONT-END PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB

QUIXADÁ

2022

ANTONIO CRISTIANO MACIEL DA SILVA JUNIOR

COMPARAÇÃO ENTRE OS PRINCIPAIS FRAMEWORKS JAVASCRIPT DE FRONT-END PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB

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

Orientador: Prof. Dr. Victor Aguiar Evangelista de Farias

QUIXADÁ

2022

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

S578c Silva Junior, Antonio Cristiano Maciel da.
Comparação entre os principais frameworks javascript de front-end para o desenvolvimento de aplicações web / Antonio Cristiano Maciel da Silva Junior. – 2022.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2022.
Orientação: Prof. Dr. Victor Aguiar Evangelista de Farias.

1. JavaScript (Linguagem de programação de computador). 2. Framework (Arquivo de computador). 3. Desempenho. I. Título.

CDD 005.1

ANTONIO CRISTIANO MACIEL DA SILVA JUNIOR

COMPARAÇÃO ENTRE OS PRINCIPAIS FRAMEWORKS JAVASCRIPT DE FRONT-END PARA O DESENVOLVIMENTO DE APLICAÇÕES WEB

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

Aprovada em: ____/ ____/ ____.

BANCA EXAMINADORA

Prof. Dr. Victor Aguiar Evangelista de Farias (Orientador)
Universidade Federal do Ceará (UFC)

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

Prof. Me. Júlio Serafim Martins
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

À minha mãe, tudo é por você.

AGRADECIMENTOS

Primeiro agradeço a mim mesmo, por não ter desistido e ter tido forças nessa jornada.

À minha mãe, Cleane Alves de Lima, por ter batalhado desde sempre sozinha ao meu lado, obrigado por ter se esforçado tanto, estamos nos formando juntos.

Aos meus amigos da PTF, que me motivaram diversas vezes e me fizeram acreditar que tudo era possível, vocês estão no meu coração.

À minha namorada Fauziany que esteve comigo desde o começo da faculdade que sempre me apoiou nos momentos mais complicados e agora comemoramos juntos esse momento.

Ao Victor Aguiar Evangelista de Farias, meu orientador, por ter me aconselhado e ajudado durante este trabalho.

À Júlio Serafim Martins e Jefferson de Carvalho Silva, minha banca avaliadora, pela disponibilidade em participar da banca desse trabalho, e pelas suas colaborações e sugestões.

A Universidade Federal do Ceará (UFC) e a todos os professores do campus Quixadá, que me auxiliaram nessa caminhada, foi um prazer fazer parte desse campus.

Muito Obrigado!

“É mais fácil se desculpar do que pedir permissão.”

(Grace Hopper)

RESUMO

Atualmente, as aplicações *web* estão se tornando mais complexas e exigindo cada vez mais recursos, sendo necessário ter um ótimo tempo de desempenho. Os *frameworks* surgiram para facilitar esses fatores, que para os desenvolvedores, são os mais cruciais. Diversos *frameworks* estão disponíveis no mercado, cada um possui suas características e motivações para uso em uma determinada aplicação *web*. Portanto, o objetivo deste trabalho é propor uma comparação dos *frameworks* para JavaScript mais utilizados para desenvolvimento *front-end*, com base nos fatores que levam à escolha de um *framework*. Foram utilizados os *frameworks* mais populares que estão presentes no mercado atual para realizar a comparação. Para isso, primeiro, investigou-se através de uma pesquisa de campo, as experiências de desenvolvedores *front-end* com os *frameworks* deste presente trabalho. Assim, foi desenvolvido uma aplicação *web* utilizando os *frameworks* do trabalho, para realizar uma comparação de tempo de compilação, tempo de renderização, tempo de *build* e tempo de execução de CRUD (Create, Read, Update e Delete) com cada *framework*. As análises que foram realizadas com os *frameworks* resultaram na realização de um comparativo envolvendo os *frameworks*, trazendo as principais características de cada *framework*, uma comparação sobre a documentação, configuração de ambiente, suporte a outras linguagens, curva de aprendizagem, mercado de trabalho e comunidade.

Palavras-chave: JavaScript (Linguagem de programação de computador). Framework (Arquivo de computador). Desempenho.

ABSTRACT

Currently, web applications are becoming more complex and demanding more and more resources, being necessary to have a great performance time. Frameworks emerged to facilitate these factors, which for developers are the most crucial. Several frameworks are available in the job market, each one has its characteristics and motivations for use in a particular web application. Therefore, the objective of this research is to propose a comparison of the most used JavaScript frameworks for front-end development, based on factors that lead to the choice of a framework. We used the most popular frameworks that are present in the current job market to perform the comparison. For this, first, it was investigated through field research, the experiences of front-end developers with the frameworks of this present research. Thus, an web application was developed using the frameworks of the work, to perform a comparison of compile-time, rendering, build time, and CRUD runtime (Create, Read, Update and Delete) with each framework. The analyzes that were realized with the frameworks resulted in the comparison involving the frameworks, bringing the main characteristics of each framework, a comparison of the documentation, environment configuration, support to other languages, learning curve, job market, and community.

Keywords: JavaScript (Computer programming language). Framework (Computer archive). Performance.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

1 INTRODUÇÃO

Atualmente, devido à infinidade de aplicativos que o JavaScript serve e a variedade de necessidades de programação, os *frameworks* de JavaScript foram desenvolvidas a fim de facilitar o trabalho de programadores *web* (GIZAS, 2012). É importante não apenas utilizar o *framework* que atende às necessidades de seu projeto, mas que também provê um código de alta qualidade e boa performance (MARIANO, 2017). De acordo com GIZAS (2012), desenvolvedores admitiram que fatores cruciais que levam à escolha de um *framework* são a capacidade de manutenção e o suporte ativo da comunidade.

Os desenvolvedores hoje em dia costumam usar mais de um *framework* e bibliotecas para cumprir seus trabalhos especialmente ao desenvolver aplicativos *web* complexos e em grande escala (MARIANO, 2017). Os *frameworks client-side* (Lado do Cliente) possuem vantagens em relação aos demais *frameworks* por serem simples, ágeis, de desenvolvimento rápido e capacidade de empacotar a aplicação e distribuir em vários dispositivos (WILLIAMSON, 2015). As aplicações atuais estão exigindo cada vez mais recursos e, assim, portanto, o desempenho do JavaScript no navegador é um dos o problemas de usabilidade mais importante no desenvolvimento (ZAKAS, 2010). Com a existência de vários *frameworks* atualmente que podemos utilizar, cada um com suas características e motivos para utilizar em um projeto *web* e para saber qual e quando utilizar um com tantas informações, a escolha para a utilização ou aprendizagem gera dúvidas na comunidade, com o desenvolvedor sabendo diferenciar os *frameworks*, para ajudar na etapa inicial de um projeto.

Sendo assim, o presente trabalho se propõe a realizar uma comparação entre os *frameworks* mais populares para *front-end* e mais utilizados pela a comunidade de desenvolvedores com o intuito de auxiliar qual seria a melhor escolha para seu projeto *web*. Através de critérios de escolha como ter uma comunidade ativa, uma grande utilização no mercado atual e curva de aprendizagem, foram definidos React, Angular e Vue.js como os *frameworks* deste presente trabalho. O trabalho consiste também em uma uma comparação envolvendo arquitetura, ferramentas e desempenho. Será realizado uma pesquisa na comunidade com os desenvolvedores *front-end* para entender as experiências de cada desenvolvedor quais motivos atualmente os levam a escolher um *framework* e as ferramentas mais utilizadas por eles. Posteriormente, será realizado uma pesquisa teórica com cada um dos *frameworks* para realizar as comparações de arquitetura e ferramentas. Para a comparação de desempenho será definido um projeto teste, que será um *To do List* e desenvolvido usando cada um dos *frameworks* da pesquisa, por fim,

será realizado testes de desempenhos relacionado a execução de tarefas pré-definidas. Com os resultados que serão obtidos, será apresentado um estudo comparativo com todos os tópicos da pesquisa, comparando cada um dos *frameworks* e sugerindo dicas de quando usar e para que tipos de projetos os *frameworks* avaliados devem ser utilizados no desenvolvimento de aplicação *web*.

O objetivo principal deste trabalho é propor uma comparação dos *frameworks* JavaScript mais utilizados para desenvolvimento *front-end* presentes no mercado com base em fatores cruciais que levam à escolha de um framework. Para isto, foi necessário compreender as experiências dos desenvolvedores com os *frameworks front-ends* mais utilizados na comunidade e uma comparação a nível de ferramentas e desempenho.

Este trabalho está estruturado da seguinte forma: no Capítulo 2 apresenta o objetivo geral e os objetivos específicos deste trabalho. no Capítulo 3 apresenta a fundamentação teórica para o entendimento completo do contexto deste trabalho. No Capítulo 4, são apresentados os trabalhos relacionados, bem como suas semelhanças e diferenças com o presente trabalho. No Capítulo 5, são apresentados as metodologias utilizadas na condução deste trabalho. O Capítulo 6 apresenta os resultados obtidos. Por fim, o 7 apresenta a conclusão e trabalhos futuros.

2 OBJETIVOS

Este Capítulo apresenta o objetivo geral e os objetivos específicos deste trabalho.

2.1 Objetivo geral

Propor uma comparação de *frameworks* JavaScript presentes no mercado com base em fatores cruciais que levam à escolha de um *framework*.

2.2 Objetivos específicos

1. Compreender as experiências dos desenvolvedores com os *frameworks front-ends* mais utilizados na comunidade.
2. Comparar os *frameworks* a nível de ferramentas de testes, gerenciamento de estados, design, curva de aprendizagem e acessibilidade.
3. Comparar os *frameworks* a nível de desempenho de tempo de compilação, tempo renderização, tempo de *build*, tempo de execução de CRUD *(Create, Read, Update e Delete)*.

3 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados os principais conceitos necessários para o entendimento deste trabalho. A Seção 3.1 conceitua sobre JavaScript, com uma breve descrição de sua história. A Seção 3.2 descreve uma introdução sobre *frameworks* e o motivo do porque eles existem. A Seção 3.3 conceitua e apresenta uma introdução sobre a biblioteca React, com uma descrição sobre sua arquitetura, uma lista de ferramentas que podem serem utilizadas. A Seção 3.5 conceitua e apresenta uma introdução sobre o *framework* Vue.js, como uma descrição sobre sua arquitetura, uma lista de ferramentas que podem serem utilizadas. A Seção 3.4 conceitua e apresenta uma introdução sobre o *framework* Angular, como uma descrição sobre sua arquitetura, uma lista de ferramentas que podem serem utilizadas.

3.1 JavaScript

JavaScript é a linguagem de programação da *web*, foi criada pela Netscape em parceria com a Sun Microsystems, com a finalidade de fornecer um de adicionar interatividade a uma página web (SILVA, 2010). Segundo Flanagan (2004) a maioria dos sites modernos usam JavaScript, e todos os navegadores em computadores, *tablets*, *smartphones* incluem interpretadores JavaScript.

JavaScript geralmente é implementado junto com HTML *(HyperText Markup Language)* e CSS3 *(Cascading Style Sheets)* em *client-side* (lado do cliente) para criar aplicações web interativa e criativas. De acordo com Mariano (2017) a evolução do JavaScript foi abordada com o surgimento de JSF *(JavaScript Frameworks)*, houve uma grande mudança na forma como os desenvolvedores desenvolvem *software* hoje em dia.

3.2 Client-side Frameworks

A definição de *framework* pode variar de acordo com a pesquisa, existindo assim, várias definições na literatura. Segundo (ALVIM, 2008) *framework* é um conjunto de classes que colaboram entre si de modo a prover um reúso abrangente, de grandes blocos de comportamento.

Existem muitas bibliotecas de *frameworks* JavaScript disponíveis para os desenvolvedores de software trabalharem, cada um é único em sua própria maneira, enquanto muitos farão algumas das mesmas coisas, mas muitas vezes de forma diferente. Com base no que foi citado, os *frameworks* tendem a facilitar os desenvolvedores na construção de aplicações.

3.3 ReactJs

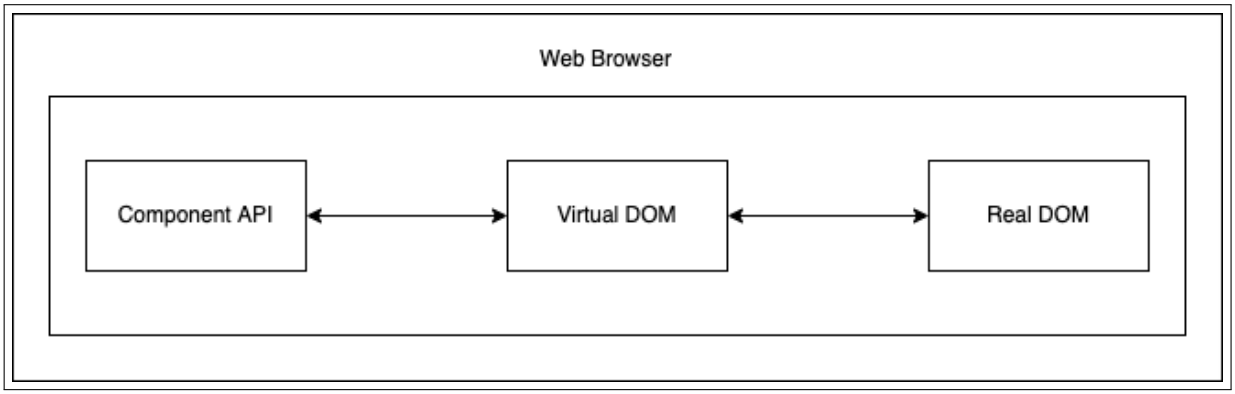
React é uma biblioteca de UI *(User Interface)* desenvolvida no Facebook para facilitar a criação de componentes reutilizáveis, com estados e interativos (KUMAR; SINGH, 2016). De acordo com a documentação[1] oficial do React, a seguir está a definição, React é uma biblioteca para construção de interfaces de usuário modulares.

3.3.1 Arquitetura

O React permite o desenvolvimento de aplicativos grandes e complexos baseados na *web* que podem alterar seus dados sem atualizações de páginas subsequentes, definindo sua arquitetura principal como V *(View)* do padrão MVC *(Model-View-Controller)* (AGGARWAL, 2018). De acordo com VIPUL A. M.; SONPATKI (2016) o React cria representações abstratas de *Views*, se dividindo em partes da visão nos componentes.

Possui por padrão o tipo de renderização SSG *(Static Site Generation)*, mas com o NextJS[2] é capaz de criar de sites SSR *(Server-side rendering)*, também possuindo liberdade de escrita com TypeScript o que facilita as declarações de tipos nas variáveis do projeto.

Figura 1 – Arquitetura do React



Fonte: Elaborado pelo o autor.

Para entender a Figura 1 precisamos sabe que arquitetura do React tem três camadas: API *(Application Programming Interface)* de definição de componente, módulo de gerenciamento de estado de componente na memória (também conhecido como módulo DOM (*Document Object Model)* virtual) e módulo de renderização de elemento DOM real. A terceira camada da arquitetura do React é conhecida como renderizador React e é implementada como um pacote

[1] https://pt-br.reactjs.org/
[2] https://nextjs.org/

separado para a biblioteca React. Podemos nomear a arquitetura do React como um renderizador de componentes porque ele renderiza diferentes elementos com componentes JSX *(JavaScript XML)*.

O React é baseado na ideia de que a manipulação de DOM é uma operação cara e deve ser minimizada. Ele também reconhece que otimizar a manipulação do DOM manualmente resultará em uma grande quantidade de código clichê, que é um erro, enfadonho e repetitivo. O React resolve isso dando ao desenvolvedor um DOM virtual para renderizar em vez do DOM real. Ele encontra a diferença entre o DOM real e o DOM virtual e conduz o número mínimo de operações DOM necessárias para atingir o novo estado (VIPUL A. M.; SONPATKI, 2016).

3.3.2 Ferramentas

Esta Seção trata em listar ferramentas que auxiliam no desenvolvimento de aplicações utilizando o React, tais como de acessibilidade, testes unitários, design e gerenciamento de estados.

1. Jest
   - Um *framework* de teste em JavaScript projetado para garantir a correção de qualquer código JavaScript.
2. Cypress
   - Um *framework* de teste de ponta a ponta para automação de teste para aplicações *web*.
3. Testing Library
   - Uma solução para testes de componentes.
4. Material UI
   - Uma biblioteca que permite importar e usar diferentes componentes para criar uma interface de usuário em aplicações *web*.
5. Styled Components
   - Permite você utilizar *CSS-in-Js* em suas aplicações *web*.
6. Redux
   - Uma biblioteca JavaScript de código aberto para gerenciar o estado do aplicativo.
7. WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications).
   - Uma especificação técnica para *web*, para facilitar a questão de acessibilidade nas aplicações.

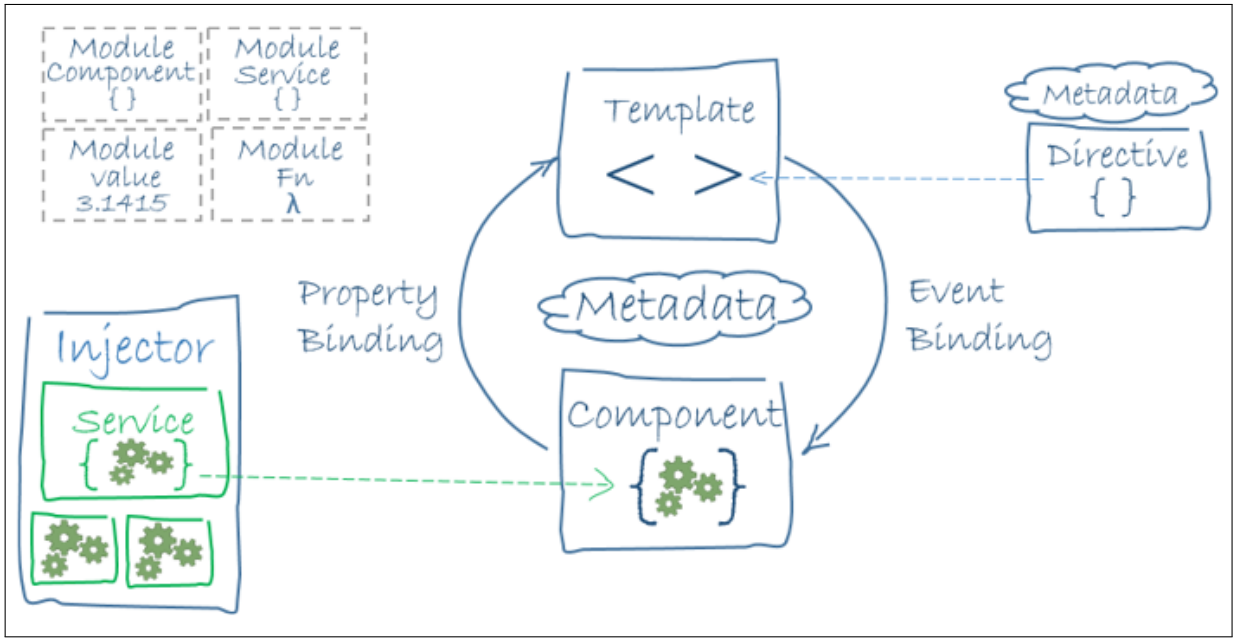
3.4 Angular

Criado por Miško Hevery e Adam Abrons em 2009, o AngularJS é um *open-source* (código aberto), *framework* JavaScript do *client-side* (lado do cliente), que promove uma alta produtividade experiência em desenvolvimento web (BRANAS, 2014). Angular é um *framework* estrutural para aplicativos da web dinâmicos, permitindo o uso de HyperText Markup Language (HTML) como sua linguagem de modelo e permite estender a sintaxe do HTML para expressar os componentes do seu aplicativo de forma clara e sucinta (ANGULARJS, 2009).

3.4.1 Arquitetura

Seus autores declararam que, o Angular adota o MVW *(Model-View-Whatever)* (ANGULARJS, 2009), porém, o Angular possui o ngController que é definido como o C *(controller)* de MVC *(Model-View-Controller)*, ou seja, assim fica a critério do desenvolvedor escolher qual melhor padrão se encaixa. Independentemente do nome, o benefício mais importante é que o *framework* fornece uma separação clara das preocupações entre o aplicativo camadas, fornecendo modularidade, flexibilidade e testabilidade (BRANAS, 2014).

Figura 2 – Arquitetura do Angular



Fonte: (ANGULARJS, 2009)

A Figura 2 mostra como cada parte estão relacionadas. Juntos, um componente e um modelo definem uma vista angular, um decorador em uma classe de componente adiciona os metadados, incluindo um ponteiro para o modelo associado. As diretivas e a marcação de

vinculação no modelo de um componente modificam as visualizações com base nos dados e na lógica do programa. O injetor de dependência fornece serviços para um componente, como o serviço de roteador que permite definir a navegação entre as visualizações (ANGULARJS, 2009).

Angular tem suporte ao TypeScript nativo e possui um serviço equiparado ao NextJS[3] (React) e NuxtJS[4] (Vue.js) trabalhando com SSR *(Server-side rendering)*, ou seja, renderização é feita pelo o lado do servidor, se chama Angular Universal (ANGULARJS, 2009).

3.4.2 Ferramentas

Esta Seção trata em listar ferramentas que auxiliam no desenvolvimento de aplicações utilizando o Angular. Segundo ANGULARJS (2009), ele possui por padrão as ferramentas de testes Karma e Jasmine, porém, podendo ter acesso a outras ferramentas, mas precisando configurar manualmente.

Angular também possui total compatibilidade com *ARIA*. Segundo (ANGULARJS, 2009) o Angular possui uma biblioteca de Design de Componentes, onde possui pacotes voltados para a acessibilidade, tais como, Live Announcer e cdkTrapFocus, assim aumentando os elementos nativos de HTML e facilitando a criação de componentes mais acessíveis. A lista de ferramentas é listada a seguir:

1. Jest
   - Um *framework* de teste em JavaScript projetado para garantir a correção de qualquer código JavaScript.
2. Cypress
   - Um *framework* de teste de ponta a ponta para automação de teste para aplicações *web*.
3. Testing Library
   - Uma solução para testes de componentes.
4. Angular Material Design
   - Infraestrutura de componentes de interface do usuário e componentes de design de materiais para aplicativos em Angular para dispositivos móveis e *desktop*.
5. AntDesign

[3] https://nextjs.org/
[4] https://nuxtjs.org/pt/

- Uma biblioteca de design que permite importar e usar diferentes componentes para criar interface em aplicações *web*.

6. NgRx Store
   - Fornece gerenciamento de estado reativo para aplicações Angular inspirado no Redux.

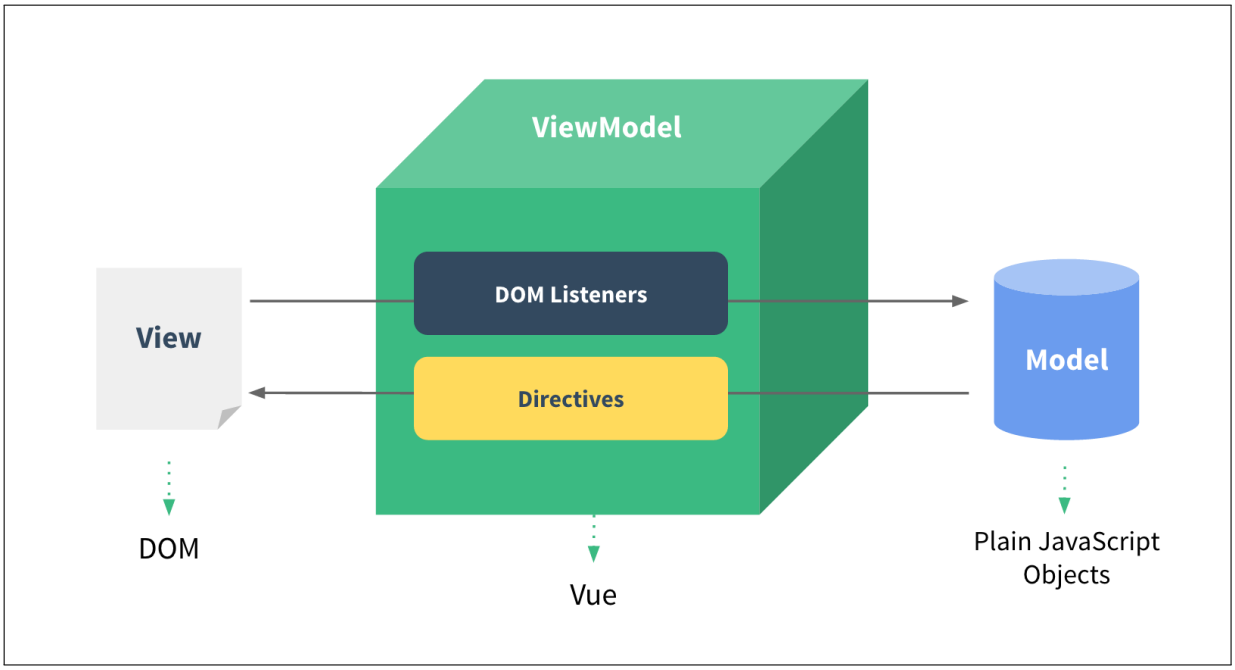
3.5 Vue.js

Criado por Evan You em 2013, o Vue.js é um *framework open-source* (código aberto) e progressivo para a construção de aplicações *web*, focado para ter poder de criação em SPA (*Single-Page Applications*) quando usado em conjunto com ferramentas modernas e bibliotecas de apoio. De acordo com Kyriakidis e Maniatis (2016) o Vue.js possui um grande ecossistema de *plugins* e ferramentas que estendem seus serviços básicos.

3.5.1 Arquitetura

Vue.js está focado sobre o ViewModel, onde é utilizado o padrão MVVM *(Model-View-ViewModel)*, diferente do que vimos na Seção 3.4.1 com o Angular, o Vue.js apresenta uma conexão entre a View e o Model com um ViewModel (VUEJS, 2013), pode-se observar na Figura 3 o que foi dito anteriormente.

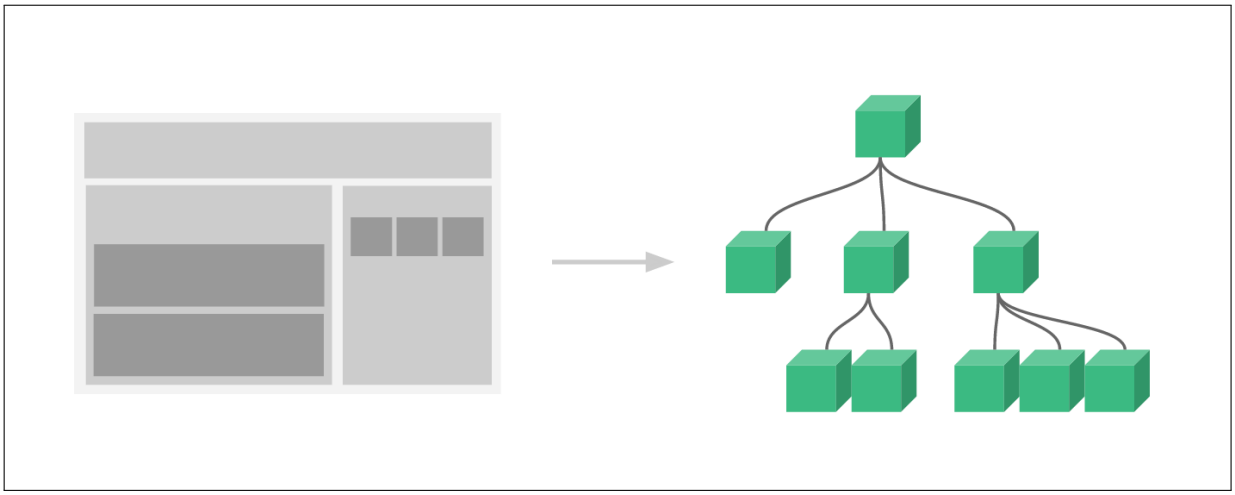
Figura 3 – Arquitetura do Vue.js



Fonte: (VUEJS, 2013)

Vue.js tem um documentação[5] detalhada e exemplos em sua página inicial, desde como para começar. As aplicações com Vue.js são feitas com componentes reutilizáveis, chamadas de Vue Instance (Instâncias de Vue.js), sendo comum uma aplicação ser organizada em árvore de componentes aninhados, como ilustrado na Figura 4. Os componentes podem ser registrado como global na aplicação, Tendo o componente registrado globalmente, o torna disponível para todos os subcomponentes de aquela árvore de componentes da instância Vue.js (VUEJS, 2013).

Figura 4 – Interface de usuário visual e árvore de componentes



Fonte: (VUEJS, 2013)

Equiparando ao React e Angular, o Vue.js possui o NuxtJs[6] para trabalhar com aplicações utilizando a renderização SSR (Server-side rendering), ou seja, a renderização é feita pelo o lado do servidor, por padrão é utilizado a renderização SSG e suporte ao TypeScript nativo a partir da versão 3.

3.5.2 Ferramentas

Esta Seção trata em listar ferramentas que auxiliam no desenvolvimento de aplicações utilizando o Vue.js, tais como de acessibilidade, gerenciamentos de estados e testes unitários.

1. Jest
   - Um *framework* de teste em JavaScript projetado para garantir a correção de qualquer código JavaScript.
2. Cypress

[5] https://vuejs.org/
[6] https://nuxtjs.org/

   - Um *framework* de teste de ponta a ponta para automação de teste para aplicações *web*.
3. Testing Library
   - Uma solução para testes de componentes.
4. Vuex
   - Um padrão de gerenciamento de estado e biblioteca para aplicativos Vue.js.
5. Chakra UI
   - Uma biblioteca de componentes simples, modular e acessível que fornece os blocos de construção necessários para construir seus aplicativos.
6. Vuetify
   - Uma biblioteca de interface do usuário com componentes de material lindamente artesanais.
7. WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications).
   - Uma especificação técnica para *web*, para facilitar a questão de acessibilidade nas aplicações.

4 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos relacionados, que abordam a comparação entre *frameworks* para desenvolvimento *front-end*. A proposta dos trabalhos, suas semelhanças e diferenças comparadas com o presente trabalho, são detalhadas a seguir.

4.1 *Um comparativo entre frameworks JavaScript para desenvolvimento de aplicações front-end*

Em ALMEIDA (2018) é apresentado uma análise comparativa entre os *frameworks* mais populares no mercado. Foi realizado uma pesquisa para saber quais os *frameworks* mais populares do mercado e realizado um teste de *benchmark* com um projeto realizado em cada um dos *frameworks* da pesquisa. Os dados coletados para a análise foram consumo de memória, CPU e tempo de execução dos métodos de CRUD *(Create, Read, Update e Delete)*.

Os resultados de ALMEIDA (2018) foram muito satisfatórios, com poucas diferenças entre os *frameworks* no geral. O Vue.js e Angular mostraram-se muito eficazes em quase todos os casos de testes, enquanto ReactJs e Angular acabaram tendo resultados com números mais elevados no consumo de recursos.

Assim como a pesquisa realizada por ALMEIDA (2018), o presente trabalho busca realizar comparações entre os *frameworks* mais populares no mercado. Entretanto, o trabalho citado anteriormente buscou comparar tópicos apenas de *benchmark*, enquanto o presente trabalho além de uma comparação de *benchmark*, leva em consideração pontos essenciais na escolha de um *framework* de acordo com desenvolvedores, como o uso de ferramentas que auxiliam no desenvolvimento. Complementando o que foi definido para trabalhos futuros na pesquisa de (ALMEIDA, 2018), o presente trabalho traz outras formas de avaliação de *performance*.

4.2 *Análise comparativa entre frameworks front-end baseados em JavaScript para aplicações web*

Em Ferreira e Zuchi (2018) é apresentado uma listagem dos principais *frameworks* de JavaScript do mercado mostrando as características mais marcantes de cada um, a fim de auxiliar os desenvolvedores de software na escolha daquele que mais se adéqua às necessidades e expectativas de seu projeto.

Realiza uma comparação de arquitetura, seguido pelo fator documentação e suporte da comunidade, em seguida, mostra a compatibilidade de cada um dos *frameworks* com os principais navegadores do mercado atual, o tamanho do pacote de arquivos e o tempo de renderização, por fim, colhe informações mais assertivas para identificar aquele que mais atende aos principais requisitos do projeto a ser desenvolvido.

Os resultados de Ferreira e Zuchi (2018) obteve que, por se tratar de *frameworks* de mesma linguagem base, foi possível observar grandes similaridades no código de cada um. Todos eles, possuem uma vasta comunidade e são compatíveis com os principais navegadores de internet do mercado atual. Por sua vez, o React trouxe vantagens diante dos concorrentes, mas não é o mais vantajoso quanto a *performance*, sendo o Vue.js, o mais veloz. Ao final da pesquisa, foi possível concluir que cada *framework* pode trazer determinada vantagem ao projeto, cabe ao desenvolvedor definir qual índice é o mais importante para sua aplicação.

Assim como a pesquisa realizada por Ferreira e Zuchi (2018), o presente trabalho busca realizar comparações entre os *frameworks* mais populares no mercado. Temos um semelhança quanto aos tópicos da pesquisa, porém quanto a maneira de execução temos uma diferença, o presente trabalho realiza uma pesquisa com os desenvolvedores para entender os motivos que o levam a escolha de um *framework* para um projeto ou estudo. Os tópicos de desempenho Ferreira e Zuchi (2018) são limitados a tamanho do pacote de arquivos e tempo de renderização, já o presente trabalho realiza testes mais profundos utilizando tarefas pré-definidas em um sistema de *To Do List* capaz de adicionar, remover e editar tarefas para organizar o dia a dia, assim, comparando tempo de cada *framework*, como, tempo *build*, compilação e também os tipos de renderização e características mais completas de cada *framework*.

4.3 *What leads developers towards the choice of a JavaScript framework?*

Em Pano *et al.* (2016) é feito um estudo qualitativo com entrevistas semiestruturadas, ao contrário dos estudos de Ferreira e Zuchi (2018), ALMEIDA (2018) que compararam os *frameworks* mais populares, o objetivo principal é saber quais as decisões que levam a seleção de um *framework*. Os fatores analisados foram a performance, esforço, influência social, concorrentes, tamanho da comunidade, condições facilitadoras e valor. Foram utilizados uma combinação de quatro atores, cliente, desenvolvedor, equipe e líder da equipe.

Os resultados de Pano *et al.* (2016) mostraram a falta de pesquisas para resolver os problemas relatados quanto a seleção de um *framework*. Com os fatores do estudo, descobriram

que eles são apenas a ponta do iceberg para compreender a adoção e muitas vezes são avaliados de maneiras que não são significativas para os desenvolvedores. Por fim, oferecem fatores desejáveis e sugestões que podem conduzir o desenvolvimento de futuras *frameworks* JavaScript.

Assim como a pesquisa realizada por Pano *et al.* (2016), o presente trabalho busca contribuir aos desenvolvedores motivações e decisões que levam a escolha de um *framework*. Entretanto, o trabalho citado anteriormente realizou apenas uma pesquisa com atores pré-definidos em uma abordagem semiestruturada, enquanto o presente trabalho além de uma pesquisa com desenvolvedores, realizamos uma pesquisa sobre arquitetura e uso de ferramentas para cada um dos *frameworks*, além de criar um projeto base para realizar testes de performance com atividades pré-definidas para cada um dos *frameworks*. Complementando o que foi definido para trabalhos futuros na pesquisa de Pano *et al.* (2016), o presente trabalho traz outras formas de avaliação e classificação de um *framework*.

4.4 Comparação dos trabalhos

A tabela apresenta uma comparação entre os trabalhos relacionados e o presente trabalho. É possível visualizar quatro critérios de comparação: As características foram definidas através de um estudo sobre os trabalhos relacionados e assim foram escolhidos, pesquisa com desenvolvedores, testes de *benchmark*, popularidade dos *frameworks* no mercado, pesquisa teórica de cada um dos *frameworks* desta pesquisa.

Quadro 1 – Comparação dos trabalhos relacionados com o presente trabalho

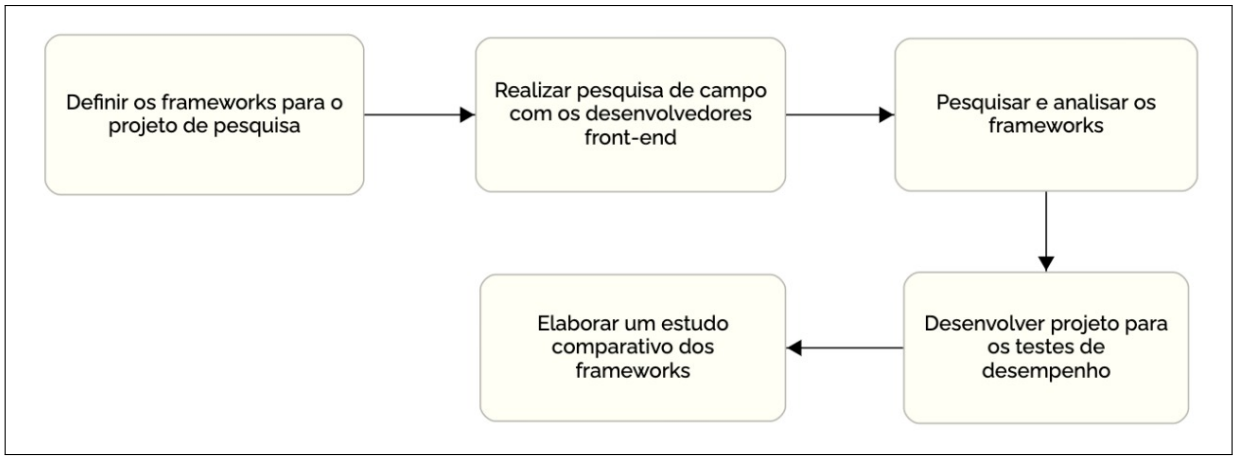
Características	(ALMEIDA, 2018)	(FERREIRA;ZUCHI, 2018)	(PANO, 2016)	Presente trabalho
Pesquisa com desenvolvedores ou atores	Não	Não	Sim	Sim
Testes de desempenho através de um projeto com os *frameworks*	Sim	Sim	Não	Sim
Utilização dos *frameworks* mais populares do mercado	Sim	Sim	Não	Sim
Pesquisa teórica de cada um dos *frameworks* (Arquitetura e Ferramentas)	Não	Não	Não	Sim

Fonte – Elaborado pelo o autor.

5 METODOLOGIA

Este Capítulo apresenta os procedimentos metodológicos e os procedimentos metodológicos foram estabelecidos pelo o autor deste presente trabalho. Os procedimentos metodológicos são representados na Figura 5 e descritos a seguir.

Figura 5 – Fluxo de execução dos procedimentos metodológicos



Fonte: elaborada pelo autor.

5.1 Definir os *frameworks* para o projeto de pesquisa

Neste passo, foi realizada uma pesquisa para definir quais os *frameworks* seriam pesquisados, analisados e comparados. Essa seleção seguiu alguns critérios e um número de *frameworks* a serem comparados, foi estabelecido que seriam escolhidos três *frameworks*. Esses critérios de escolha são listados a seguir.

1. Popularidade entre os desenvolvedores *front-end*.
2. Comunidade ativa.
3. Utilização no mercado atual.
4. Curva de aprendizagem.

5.2 Realizar pesquisa de campo com os desenvolvedores *front-end*

Este passo consistiu em realizar uma entrevista através de um formulário [1] com os desenvolvedores *front-end* da comunidade do mercado atual. As perguntas são direcionadas para consolidação das informações da Seção 5.1.

[1] https://forms.gle/o9TCspu2QnNtXNVf8

As perguntas são direcionadas aos desenvolvedores *front-end* de qualquer nível de senioridade, através de perguntas diretas, foi definido alguns resultados esperados após analisar as respostas do formulário, como, consolidar a lista de *frameworks* mais utilizadas, saber motivações para a escolha de um *framework*, filtrar uma lista de ferramentas utilizadas e o nível de satisfação utilizando o *framework* escolhido.

5.2.1 Compreender as motivações para a escolha de um framework atualmente e filtrar a lista de ferramentas mais utilizadas pelos desenvolvedores da pesquisa

Este passo consistiu em analisar as respostas do formulário da Seção 5.2 e filtrar a lista de ferramentas mais utilizadas pelos os desenvolvedores. Assim, foram coletados motivos que os desenvolvedores estão utilizando para a escolha de um *framework* atualmente, sendo para estudo, trabalho ou projetos pessoais. Filtrar a lista de ferramentas que são mais utilizadas pelos os desenvolvedores na pesquisa de campo será feita de modo que podemos associar as ferramentas aos *frameworks* da pesquisa, por ser uma pergunta aberta, teremos várias ferramentas, após a filtragem podemos associar as ferramentas aos *frameworks* da pesquisa. Com esses dados foi possível consolidar um catálogo com uma comparação ferramental de cada um dos *frameworks*.

5.3 Pesquisar e analisar dos *frameworks*

Este Capítulo consistiu nos detalhes sobre a pesquisa e análise dos *frameworks*, primeiro foi definido os tópicos da pesquisa e posteriormente realizado a pesquisa e a análise. Foram definidos duas partes para esse passo, primeiro podemos acompanhar na Seção 5.3.1 a definição dos tópicos da pesquisa para cada um dos *framework* desse presente trabalho e na Seção 5.3.2 é descrito como será realizado e analisado os dados da pesquisa.

5.3.1 Definir tópicos da pesquisa sobre os frameworks

Este passo consistiu em definir quais assuntos seriam comparados sobre os *frameworks*, garantindo critérios no qual os desenvolvedores seguem para a escolha de um *framework*. O tópicos da pesquisa são listados a seguir.

- Introdução
- Arquitetura
- Ferramentas

Com a análise na Seção 5.2.1, concluímos que esses tópicos são os mais importantes para a escolha de um *framework*.

5.3.2 Realizar e analisar a pesquisa sobre os frameworks

Este passo consistiu em uma pesquisa aprofundada sobre os *frameworks* escolhidos, seguindo os tópicos da Seção 5.3.1 foram realizadas pesquisas teóricas sobre cada um dos *frameworks*. As análises realizadas comparando as características de cada um dos *frameworks*. As principais análises realizadas nesse passo são listadas a seguir.

- Comparar arquitetura
- Comparar a compatibilidade do uso de ferramentas
- Quando usar os *frameworks*

Além disso, foram realizados análises para verificar o grau de concordância entre a pesquisa e os desenvolvedores.

5.4 Desenvolver projeto para os testes de desempenho

O projeto para os testes de desempenho consistiu em um aplicação desenvolvida com cada um dos *frameworks* da pesquisa, foi dividido esta seção em quatro partes, na Seção 5.4.1 é mostrado como foi definido o escopo da aplicação a ser desenvolvida, requisitos e qual aplicação vai ser desenvolvida, na Seção 5.4.2 é descrita a lista dos testes que serão realizados e como vai ser realizado os testes de *benchmark*.

5.4.1 Definir escopo e implementar a aplicação com os frameworks da pesquisa

Este passo consistiu em definir os requisitos para a aplicação e desenvolve-la utilizando os três *frameworks* escolhidos neste presente trabalho. A aplicação foi um *To Do List* para auxiliar os seus utilizadores na organização de tarefas a fazer, possuindo os seguintes requisitos.

- Adicionar tarefa
- Remover tarefa
- Editar tarefa
- Listagem das tarefas

5.4.2 Realizar e compreender os resultados dos testes de benchmark do projeto

Este passo consistiu em realizar testes pré-definidos na aplicação desenvolvida na seção 5.4.1. Foi utilizado uma lista de ferramentas para auxiliar nos testes de *benchmark*, foi definido pelo o autor os seguintes testes.

- Tempo de compilação
- Tempo de renderização
- Tempo de *build*
- Tamanho final do *Bundle*
- Tamanho de Linhas de código
- Tempo de execução de CRUD (Create, Read, Update, Delete)

Após o passo de realização dos testes, foi realizado uma análise dos resultados, com base na análise dos resultados dos testes de *benchmark* em cada aplicação dos *frameworks* que foi desenvolvido na Seção 5.4.1, foi possível identificar as características de cada *framework* no tópico de desempenho, auxiliando a criação do catálogo na Seção a seguir.

5.5 Elaborar um estudo comparativo dos *frameworks*

Este passo consistiu na elaboração de um estudo comparativo dos *frameworks*, após realizar todos os processos das seções anteriores, foi construído um estudo comparativos com as características e informações de cada um dos *frameworks* obtidos ao longo deste presente trabalho. O estudo comparativo contém uma comparação de arquitetura, ferramentas, quais tiveram melhores resultados nos testes de desempenho realizado na Seção 5.4.2, exemplos de quando se usar cada um dos *frameworks*, uma comparação de arquitetura contendo informações de padrões que são utilizados dentro da arquitetura de cada *framework*, tipos de renderização que são aceitas, utilização de TypeScript nativo ou não, uma lista de ferramentas que auxiliam os desenvolvedores no uso com cada um dos *frameworks*.

6 RESULTADOS

Neste Capítulo, são apresentados os resultados do presente trabalho.

6.1 Definição dos *frameworks* para o projeto de pesquisa

Para consolidar os critérios descritos na Seção 5.1, buscou-se no site StateOfJs [1] a listagem dos *frameworks* que mais atendiam aos requisitos, assim, foram escolhidos os três *frameworks* mais utilizados em 2020, React, Angular e Vue.js, como vimos no Capitulo 1 alguns desenvolvedores gostam de utilizar mais de um *framework*, o Quadro 2 mostra em porcentagem o uso de cada um dos *frameworks*.

Quadro 2 – Top 3 *frameworks* mais utilizadas em 2020 - StateOfJs

Frameworks - StateOfJs	
Nome	Porcentagem de Uso
React	80%
Angular	56%
Vue.js	49%

Fonte: elaborada pelo autor.

6.2 Pesquisa de campo com os desenvolvedores *front-end*

A etapa da pesquisa de campo com os desenvolvedores foi realizada para entender quais são os *frameworks* mais utilizados na comunidade e avaliar as experiências dos desenvolvedores, como o uso de ferramentas e motivos que levaram a escolha de um *framework*. Com um total de 16 respostas na pesquisa realizada na Seção 5.3.2, as perguntas e resultados da pesquisa é listada abaixo.

1. Qual o seu nível de senioridade no *front-end?**
2. Quais desses *frameworks* você trabalha atualmente?*
3. O que te levaria a escolher uma *framework* hoje?*
4. Com qual você teve o primeiro contato?*
5. Por qual motivo você escolheu a *framework* escolhida para ser a primeira?*
6. Se você trocou de *framework* durante a vida profissional, qual o motivo?
7. Tendo em vista a sua *framework* de trabalho atualmente, liste algumas ferramen-

[1] https://2020.stateofjs.com/en-US/

tas que você utiliza no dia a dia. Ex: Ferramentas para Testes, Bibliotecas, IDE, etc*

8. O quão satisfeito você se sente com a tecnologia em que trabalha hoje em dia?*

6.2.1 Qual o seu nível de senioridade no front-end?

A partir desta pergunta queríamos filtrar o nível de senioridade com as respostas das perguntas a seguir para analisar algumas diferenças de motivações de escolhas de *framework* ou ferramentas de acordo com sua experiência.

Quadro 3 – Qual o seu nível de senioridade no *front-end?*

Senioridade	Número de Desenvolvedores
Júnior	3
Pleno	6
Sênior	6

Fonte: elaborada pelo autor.

6.2.2 Quais desses frameworks você trabalha atualmente?

A partir desta pergunta conseguimos consolidar os *frameworks* mais utilizados para este presente trabalho, os resultados pode ser visto a seguir, vemos que os resultados são equiparados ao da Seção 6.1.

Quadro 4 – Quais desses *frameworks* você trabalha atualmente?

Framework	Número de Desenvolvedores
React	12
Angular	3
Vue	1
Svelte	0

Fonte: elaborada pelo autor.

6.2.3 O que te levaria a escolher um framework hoje?

A partir desta pergunta conseguimos perceber as motivações dos desenvolvedores a escolherem um *framework*, consolidando as expectativas que tínhamos na Seção 5.1 onde definimos critérios para definição dos *frameworks*, filtramos as respostas por ordem das que mais se repetiram, por ser uma pergunta aberta, dentre as 16 respostas obtidas.

1. Curva de aprendizagem - 9 respostas

2. Tamanho da comunidade do *framework* - 4 respostas
3. Facilidade na criação de projetos pelo uso de bibliotecas - 2 respostas
4. Performance - 1 respostas
5. Acessibilidade - 1 respostas
6. Vagas no mercado - 1 respostas
7. Manutenção de código a longo prazo - 1 respostas
8. Conhecimento comum do time - 1 respostas
9. Facilidade na realização de *deploy* da aplicação - 1 respostas
10. Facilidade de uso do TypeScript - 1 respostas
11. Execução no *client-side* - 1 respostas

6.2.4 Com qual você teve o primeiro contato?

Esta pergunta foi para obter qual *framework* estava sendo mais escolhido pelos os desenvolvedores para ser o primeiro de sua carreira.

Quadro 5 – Com qual você teve o primeiro contato?

Framework	Número de Desenvolvedores
React	8
Angular	5
Vue	3
Svelte	0

Fonte: elaborada pelo autor.

6.2.5 Por qual motivo você escolheu o framework escolhido na resposta anterior para ser o primeira?

Esta pergunta foi para compreender os motivos que os desenvolvedores levaram para escolher o primeiro *framework* para estudar ou trabalhar, é listado a seguir em ordem de mais utilizados, dentre as 16 respostas de múltipla escolha.

1. Curva de aprendizagem - 7 respostas
2. Vagas no mercado - 5 respostas
3. Comunidade ativa - 4 respostas
4. Popularidade - 2 respostas

6.2.6 Se você trocou de framework durante a vida profissional, qual o motivo?

Esta pergunta foi para entender os motivos de troca de *framework* durante a vida profissional, vimos na Seção 6.2.4 o aumento do uso de Angular e Vue.js para o primeiro *framework*, porém, na Seção 6.2.2 esse dados são diferentes porque tiverem desenvolvedores que trocaram de *framework*, as respostas desses motivos são listados a seguir, dentre as 8 respostas obtidas.

1. Vagas no mercado - 5 respostas
2. Regras de negócio na empresa - 2 respostas
3. Explorar novas tecnologias (estava atuando com *C Sharp*) - 1 respostas

6.2.7 Tendo em vista a sua framework de trabalho atualmente, liste algumas ferramentas que você utiliza no dia a dia.

Esta pergunta foi para filtrar as ferramentas que os desenvolvedores da pesquisa de campo estão utilizando no dia a dia para fazer uma comparação com as ferramentas encontradas na pesquisa sobre os *frameworks* na Seção 3, é listado as ferramentas em ordem de mais utilizados, dentre as 78 respostas obtidas.

1. Visual Studio Code - 8 respostas
2. Jest - 8 respostas
3. Cypress - 6 respostas
4. Styled Components - 4 respostas
5. Vuex - 4 respostas
6. React Hook Form - 3 respostas
7. Redux - 3 respostas
8. Material UI 3 respostas
9. WebStorm - 2 respostas
10. Chakra UI - 2 respostas
11. NextJS - 2 respostas
12. NuxtJS - 2 respostas

6.2.8 O quão satisfeito você se sente com a tecnologia em que trabalha hoje em dia?

Esta pergunta foi para consolidar o nível de satisfação entre os desenvolvedores que trocaram de *framework* durante a carreira ou continuaram na mesma do inicio da carreira profissional ou estudantil.

Quadro 6 – O quão satisfeito você se sente com a tecnologia em que trabalha hoje em dia?

Nível	Porcentagem
5 - Muito Satisfeito	72%
4 - Satisfeito	18.8%
3 - Indiferente	6.3%
2 - Insatisfeito	0%
1 - Muito Insatisfeito	0%

Fonte: elaborada pelo autor.

Ao final da pesquisa foi analisada as respostas de acordo com a senioridade e foi observado que a mudança de *framework* na carreira de um Sênior ou Pleno está entre o cliente da empresa que estava atualmente pedindo para trabalhar com um determinado *framework* e oportunidades de mudança de emprego. O React e o Angular foram os mais comentados no momento da mudança de *framework*.

6.3 Pesquisa e avaliação sobre os *frameworks*

6.3.1 Arquitetura

Para a experiência deste presente trabalho, podemos descrever a arquitetura principal de React como V *(View)* do padrão MVC *(Model-View-Controller)* como é visto na Seção 3.3.1, construindo aplicações com o tipo de renderização SSG *(Static Site Generation)* por padrão, mas conta com disponibilidade de utilizar o tipo de renderização SSR *(Server-side rendering)*, utilizando NextJs e possui também uma liberdade de escrita com TypeScript.

O Angular não deixou explícito qual arquitetura é utilizada por padrão em suas aplicações, porém por possuir o ngController podemos definir que o Angular utiliza o padrão MVC *(Model-View-Controller)*, igualmente ao React, o Angular tem suporte ao TypeScript só que de forma nativa e consegue construir aplicações com o tipo de renderização SSR *(Server-side rendering)*, utilizando o Angular Universal. Concluímos que o Angular é mais preparado para aplicações grande e de empresas como bancos pela a forma que sua arquitetura é estruturada.

Em sua documentação, é afirmado que o Vue.js utiliza o padrão MVVM *(Model-*

View-ViewModel), onde é apresentado uma conexão entre a View e o Model, por isso é utilizado o termo ViewModel. O Vue.js possui por padrão o tipo de renderização SSG *(Static Site Generation)*, mas com o NuxtJS é possível criar aplicações utilizando o tipo de renderização SSR *(Server-side rendering)*, possuindo também disponibilidade de utilizar TypeScript.

Concluímos que através de seu arquitetura, o Vue.js se diferencia do React por ser mais rápido, enquanto no React quando um estado da aplicação muda, ele aciona também a renderização de toda a árvore de componentes filhos, já no Vue.js, as dependências de um componente são automaticamente observadas durante sua renderização, desta forma o sistema sabe precisamente quais componentes precisam ser renderizados quando o estado muda.

6.3.2 Ferramentas

Os *frameworks* deste presente trabalho quando falamos em ferramentas para testes unitários foram encontrados vasta semelhança entre eles, todos os *frameworks* possuem uma diversidade para realização de testes em suas aplicações, podemos concluir que os *frameworks* estão bem seguros quanto a realização de testes, é listado alguns exemplos a seguir: Jest, Cypress, Testing Library.

Cada *framework* possui uma biblioteca exclusiva para gerenciamentos de estados que possuem as mesmas funções, mas com nomes diferentes, o React trabalha com o Redux, Vue.js utiliza o VueX e o Angular tem seu próprio gerenciador também chamado NgRx Store. Sobre a questão de ferramentas para auxiliar acessibilidade, todos os *frameworks* deste presente trabalho tem um suporte com o WAI-ARIA *(Web Accessibility Initiative - Accessible Rich Internet Applications)* utilizando as técnicas de HTML padrão para as aplicações *web*.

E, para a conclusão das listas de ferramentas disponíveis no mercado atual para os *frameworks* deste presente trabalho, todos os *frameworks* possuem uma lista de ferramentas vasta para auxiliar no desenvolvimento de aplicações *web*, React e Vue.js possuem uma biblioteca de Design UI exclusiva ainda não disponível para o Angular, o ChakraUI, mas possuem várias em comum como AntDesign e Material Design que possuem documentações para todos os *frameworks*.

O React e Vue.js é facilmente possível adicioná-lo em um projeto já existente com apenas uma *tag script* dentro de uma página *HTML*, mas a principal vantagem de criar um projeto com React é que utilizando apenas o *Node.js* e o NPM *(Node Package Manager)* ou Yarn com o seguinte comando: *npx create-react-app nome-do-projeto* é criado um primeiro projeto

com React, já com Vue.js pode criar o primeiro projeto utilizando de sua CLI, mas na própria documentação avisa que para desenvolvedores iniciantes não é recomendado especialmente se ainda não está familiarizado com ferramentas de *build* baseadas em *Node.js*. O Angular é bem parecido quanto aos outros *frameworks*, é necessário ter instalado o *Node.js* para posteriormente instalar a CLI do Angular e depois apenas seguir o passo a passo para criar o primeiro projeto.

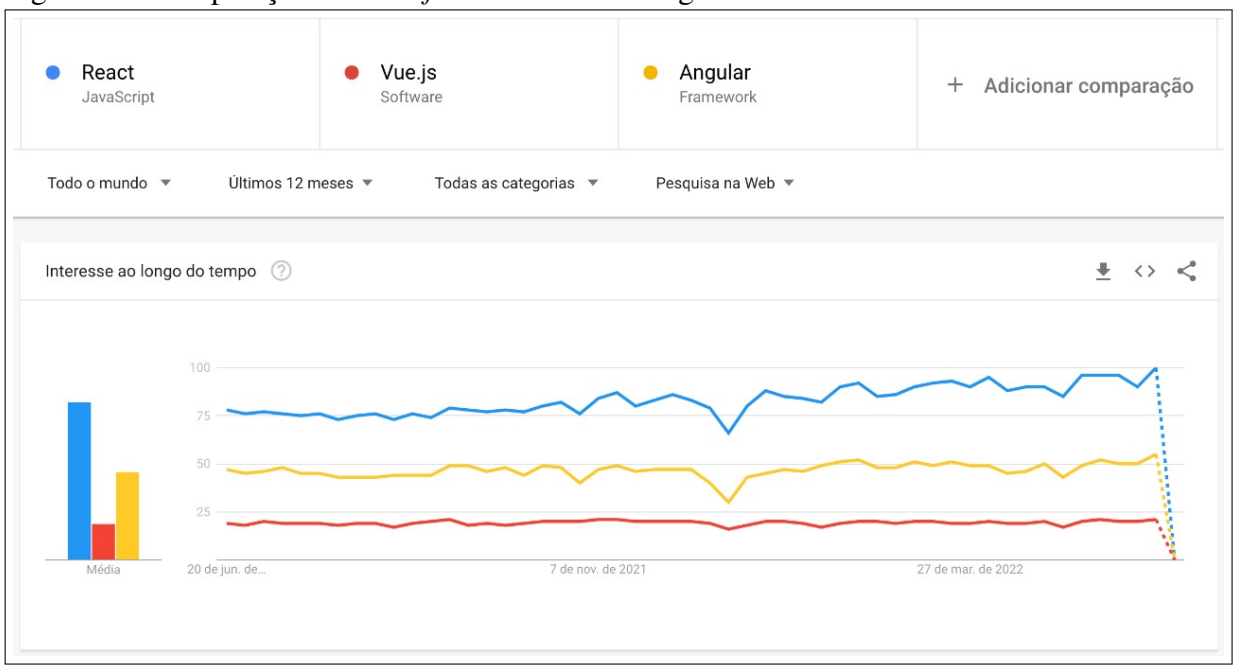
Figura 6 – Comparações entre os *frameworks*

Popularidade	React	Vue	Angular
Pesquisas no StackOverFlow	390,996 resultados (até o dia 27 de maio de 2022)	122,379 resultados (até o dia 27 de maio de 2022)	262,505 resultados (até o dia 27 de maio de 2022)
Repositório no Github	14,992 commits, 38.9k forks, 189k estrelas, 1,558 contribuidores, mais de 10 milhões de usuários	3,312 commits, 32.2k forks, 196k stars, 326 contribuidores	24,572 commits, 21.5k forks, 81.6k stars, 1,556 contribuições, mais de 2 milhões de usuários
Documentação em número de linguagens	17	8	4
Virtual DOM	Sim	Sim	Não
Typescript	Sim	Sim	Sim
Primeira Release	2013	2014	2016

Fonte: elaborada pelo autor.

Na Figura 6 é visto que o React se destaca nas pesquisas no StackOverFlow com 390,996 mil resultados, o Vue.js com 122,379 mil resultados ficou com o pior resultado e o Angular com 262,505 mil resultados. Observando as comparações realizadas, também é possível identificar que a comunidade do React é maior que os demais *frameworks* deste presente trabalho, com mais de 14,992 mil *commits* no Github e na documentação o React também se destaca por possuir o maior número de traduções em sua documentação com 17.

Figura 7 – Comparação entre os *frameworks* no Google Treads



Fonte: elaborada pelo autor.

Podemos visualizar na Figura 7 que a popularidade do React na comunidade de desenvolvimento é muito superior entre os *frameworks* deste presente trabalho, ele é o *framework* mais buscado nos últimos 12 meses de acordo com o Google Treads com uma média (82), como é visto também, o Vue.js ficou com a pior média (19) e o Angular ficou entre os dois *frameworks* (46), quanto mais perto de 100 representa um pico maior a popularidade, um valor de 50 significa que o termo teve metade da popularidade e 0 não havia dados suficientes.

Assim, consolidando as informações, concluímos que os *frameworks* deste presente trabalho possuem bastante semelhanças quanto ao uso de ferramentas, mas existindo diferenças em vários momentos durante a pesquisa que será consolidada ao final do teste de desempenho.

6.4 Projeto para os testes de desempenho

Este capítulo apresenta os resultados e descreve os testes realizados para avaliar os *frameworks* escolhidos na seção 6.1. Os testes seguiram o que foi descrito na seção 5.4.2.

Os casos de testes foram realizados em uma máquina com a seguinte configuração:

Modelo: MacBook Pro (16-inch, 2019)

Processador: Intel Core i7 2,6 GHz 6-Core

Memória(RAM): 16 GB 2667 MHz DDR4

Placa de Vídeo: AMD Radeon Pro 5300M 4 GB

6.4.1 Execução dos testes

O Quadro 7 possui os casos de teste que foram elaborados neste trabalho, onde cada caso servirá para avaliar os *frameworks.* O Quadro 8 contém o nome e a versão dos *frameworks* utilizados neste presente trabalho.

Quadro 7 – Tabela de Casos de Teste

Casos de Teste	Descrição
CT01	Adicionar Tarefa
CT02	Remover Tarefa
CT03	Editar Tarefa
CT04	Tempo de compilação
CT05	Tamanho e Tempo de *build*

Fonte: elaborada pelo autor.

Quadro 8 – Tabela com os *frameworks* utilizados

Nome	Versão
React	17.0.2
Vue	2.6.14
Angular	13.3.0

Fonte: elaborada pelo autor.

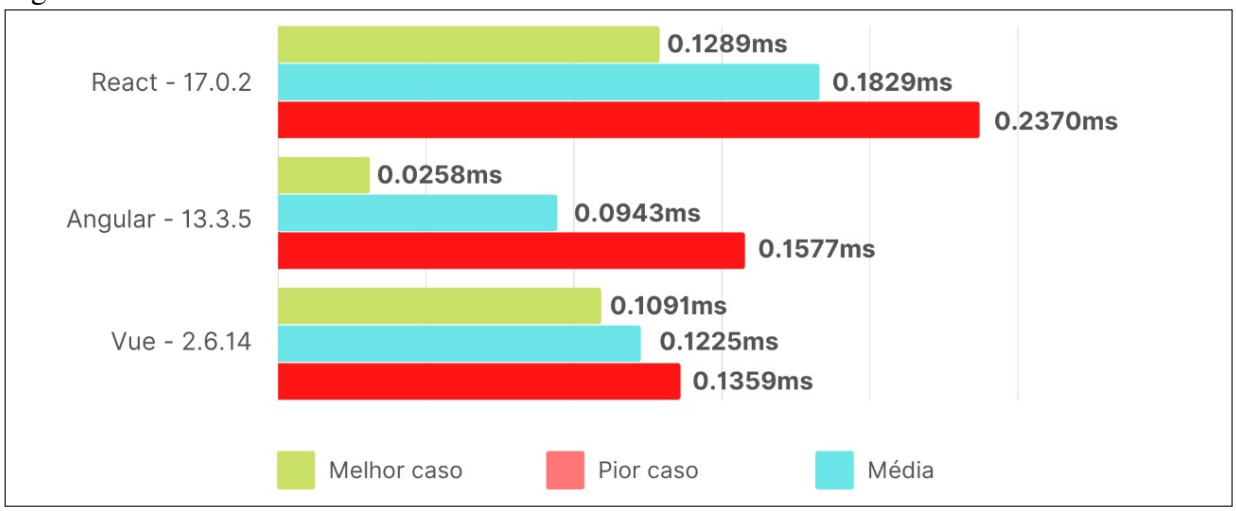
Os testes seguiram a sequência apresentada no Quadro 7 e com cada um dos *frameworks* por vez. Os dados colhidos para alimentar a análise de desempenho foram:

- **Tempo de execução**
- **Tempo de compilação**
- **Tamanho e Tempo de *build***
- **Números de linhas no código**

As análises foram feitas através do navegador Google Chrome, primeiramente foi feita uma execução manual dos casos de teste. O projeto teste de cada um dos *frameworks* pode ser acessada através: <github.com.br/cristianojr9/TCC-UFC>. As Figuras 8, 9, 10 contém os resultados para o tempo de execução de cada atividade definida.

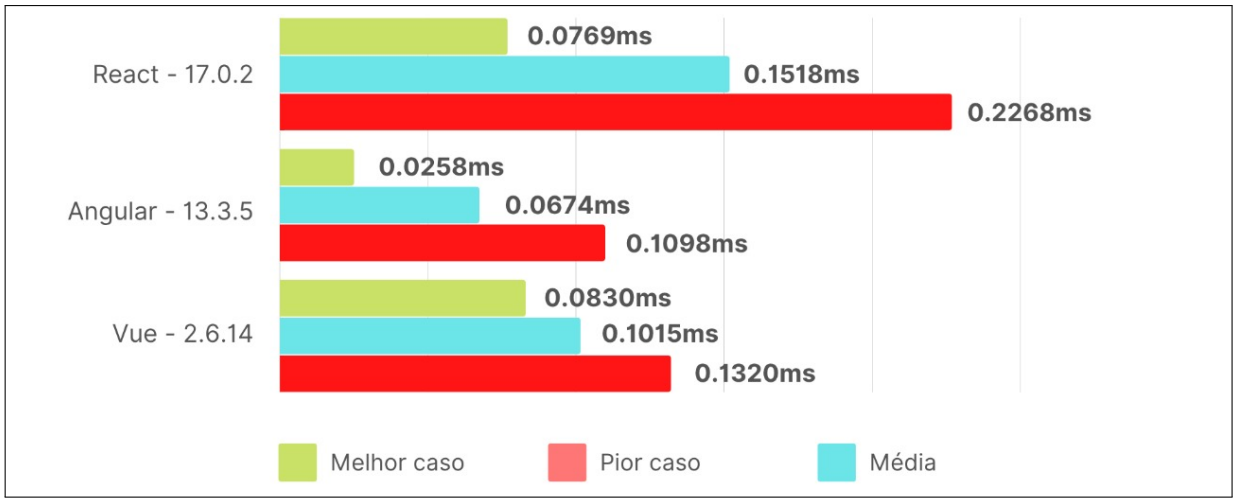
Foram realizados 5 testes em cada caso de teste, quando se trata de tempo de execução cada *framework* possuiu uma variação, como pode se constatar na Figura 8 (Adicionar tarefa), o Angular teve o melhor caso (0.0258ms) e uma média de (0.0943ms), enquanto o React teve os piores casos de tempo de execução do caso de teste (0.1289ms e 0.2370ms) e o por outro lado o Vue.js foi o mais consistente em relação aos resultados com uma boa média (0.1225ms)

Figura 8 – Resultados do teste CT01 - Adicionar Tarefa



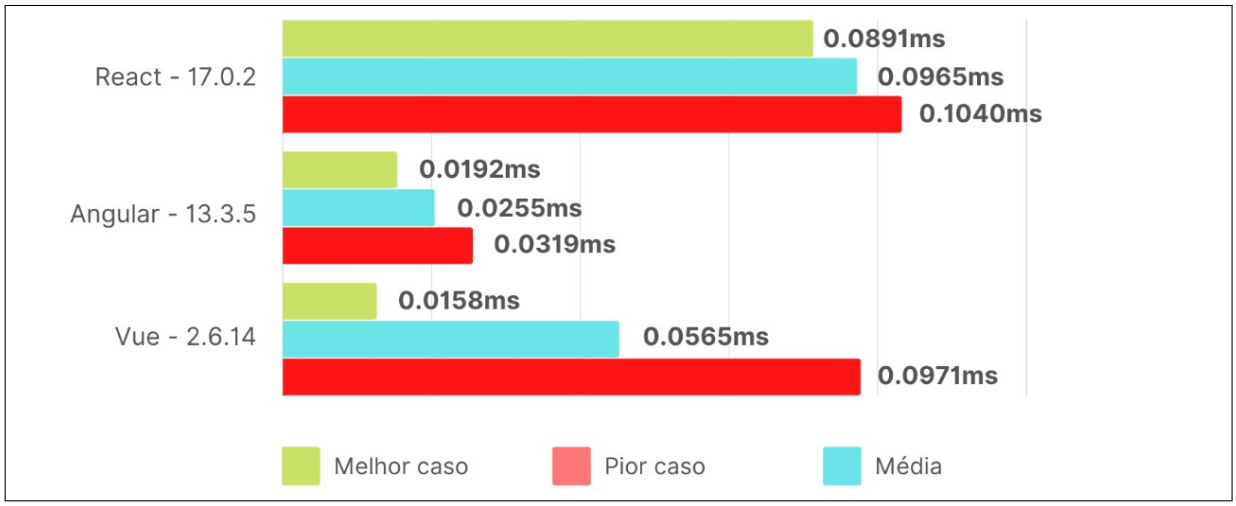
Fonte: elaborada pelo autor.

Figura 9 – Resultados do teste CT02 - Remover Tarefa



Fonte: elaborada pelo autor.

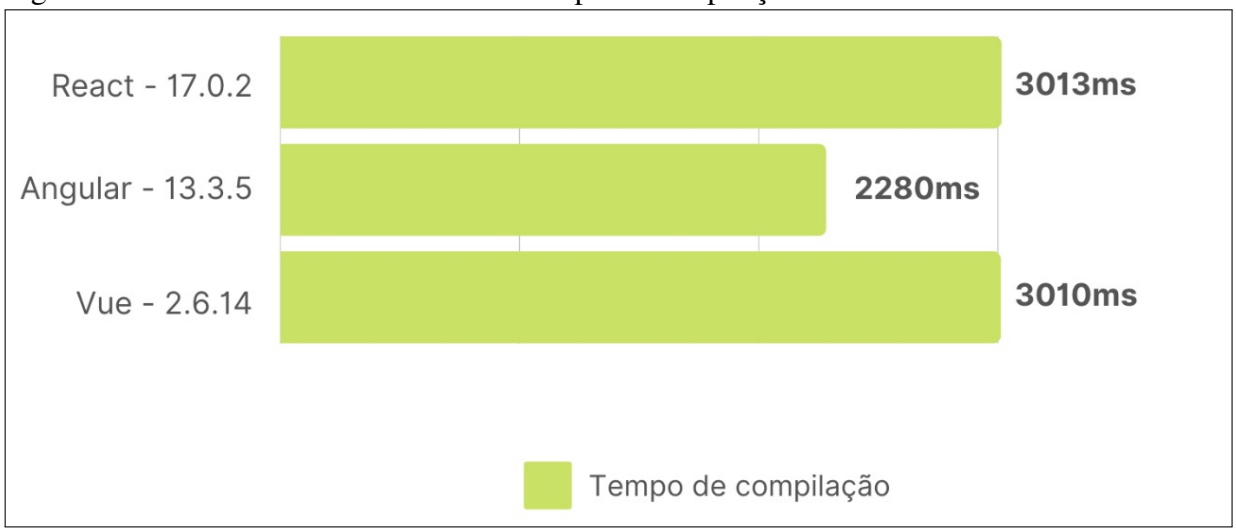
Figura 10 – Resultados do teste CT03 - Editar Tarefa



Fonte: elaborada pelo autor.

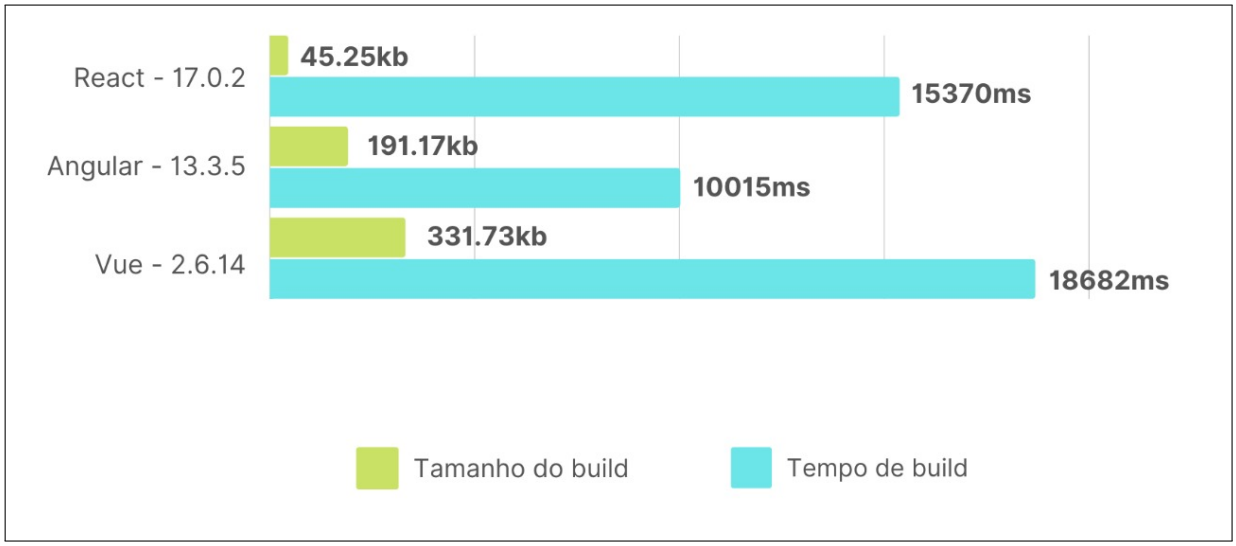
Observando a Figura 9 (Remover tarefa) o Angular se destaca novamente com o melhor tempo de execução na tarefa (0.0258ms) e o React com o pior caso (0.2268), é possível destacar que o Vue.js manteve o padrão da primeira tarefa e não teve grande diferença entre o melhor e pior caso com uma média de (0.1015ms). A Figura 10 apresenta uma visão geral dos resultados de CT03 (Editar tarefa), observa-se que o Vue.js teve o melhor desempenho em tempo de execução (0.0158ms), mas teve o segundo pior caso com (0.0971ms), o Angular manteve uma boa média com o segundo melhor tempo e o melhor pior tempo de execução (0.0192ms e 0.0319ms), o React obteve o pior desempenho de execução e a pior média entre os *frameworks* deste trabalho (0.0891ms e 0.1040ms).

Figura 11 – Resultados do teste CT04 - Tempo de compilação



Fonte: elaborada pelo autor.

Figura 12 – Resultados do teste CT05 - Tamanho e Tempo de *build*

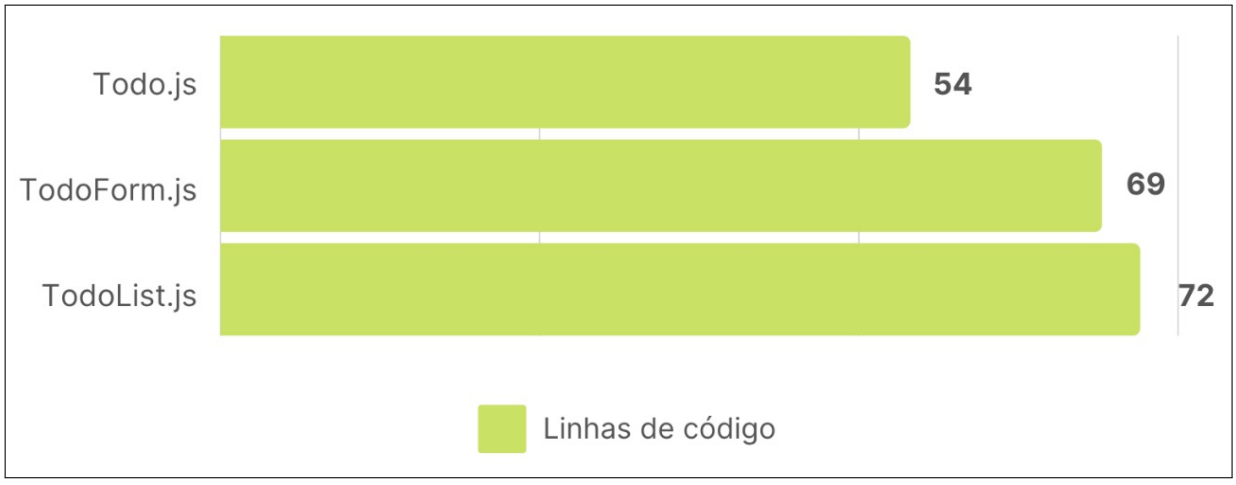


Fonte: elaborada pelo autor.

Quando se trata de tempo de compilação, tamanho e tempo de *build* a Figura 11 CT04 (Tempo de compilação) e Figura 12 CT05 (Tamanho e tempo do *build*) ilustram os resultados obtidos durante a execução destes casos de testes. No CT04 (Tempo de compilação) React e Vue.js obtiveram resultados parecidos com o Vue.js (3010ms) e React (3013), durante o desenvolvimento de um software esta diferença é quase imperceptível, já o Angular teve o melhor desempenho entre os *frameworks* (2280ms), neste caso já foi possível observa uma diferença. No CT05 (Tamanho do *build* e Tempo de *build*) o React se destacou com o tamanho de *build* menor que os *frameworks* deste presente trabalho (45.25kb) e o Angular com o melhor tempo de *build* (10015ms), já o Vue.js conseguiu obter os piores resultados em tempo de *build* (18682ms) e tamanho de *build* (331.73kb).

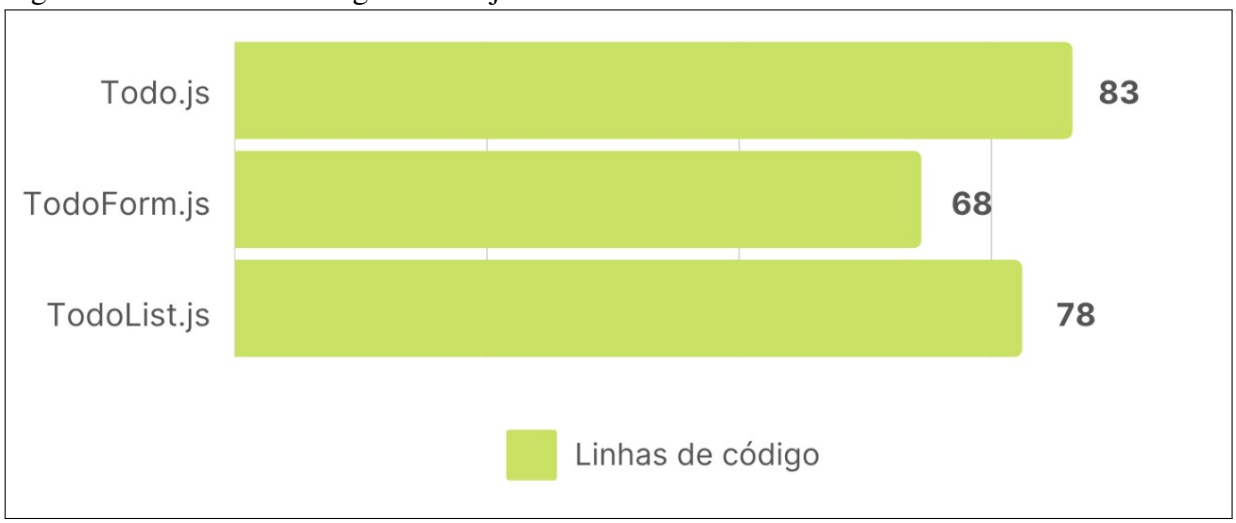
As figuras 13, 14, 15 ilustram as linhas de código dos componentes utilizados com os *frameworks* deste presente trabalho, foram selecionados Todo.js, TodoForm.js e TodoList.js para a comparação de linhas de código e apenas HTML e JavaScript entraram na contagem de linhas para ter um melhor comparação entre os *frameworks* deste presente trabalho, assim foi obtidos os seguintes resultados nos projeto em cada um dos *frameworks*. Com a arquitetura bem definida, o Angular tem o menor número de linhas de código em todos os componentes criados, o Angular utiliza um *service.js* onde é utilizado para o gerenciamento de estados (a lista de Todos) diferente do React e Vue.js, o compartilhamento de estados entre os componentes filhos é feito através de uma Lista Observável e compartilhado entre os componentes.

Figura 13 – Linhas de código no React



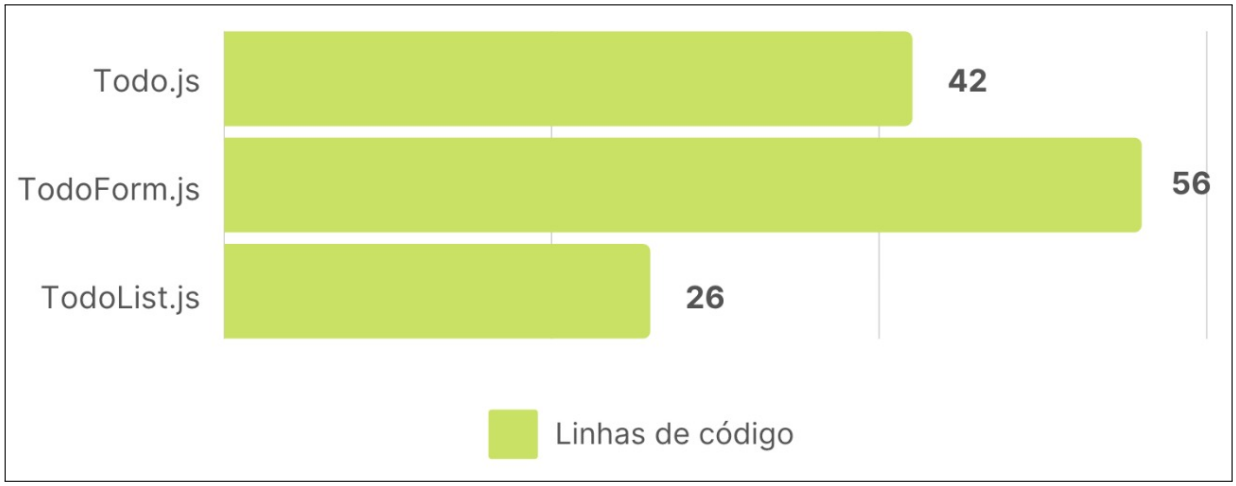
Fonte: elaborada pelo autor.

Figura 14 – Linhas de código no Vue.js



Fonte: elaborada pelo autor.

Figura 15 – Linhas de código no Angular



Fonte: elaborada pelo autor.

6.5 Comparativo dos *frameworks*

Na etapa do comparativo foi considerado todos os resultados encontrados neste presente trabalho, com isso foi possível classificar critérios de avaliação para a comparação final dos *frameworks*, foram definidos: **Documentação, Configuração, Suporte a outras linguagens, Curva de aprendizagem, Mercado de trabalho e Comunidade.**

6.5.1 React

Documentação: Possui uma documentação que abrange cerca de 17 linguagens e contando com algumas em *backlog*.

Configuração: A sua configuração é fácil de ser fazer, basta adicionar o link da

biblioteca em sua aplicação ou possuir o Node.js instalado na máquina.

Suporte a outras linguagens: Possui suporte ao TypeScript, e utilizando o NextJs você tem suporte ao tipo de renderização SSR.

Curva de Aprendizagem: A curva de aprendizagem é bastante curta, tanto que desenvolvedores da pesquisa realizada apontam como o mais fácil de se aprender.

Mercado de trabalho: O mercado possui bastante vagas para este *framework*.

Comunidade: O React possui uma vasta comunidade, é possível ver vários projetos no Github, artigos realizados no Medium e pesquisas no StackOverFlow.

6.5.2 Vue.js

Documentação: Possui uma documentação que abrange cerca de 8 linguagens e destacando que existe em Português também.

Configuração: A sua configuração também é fácil de ser fazer, assim como o React basta adicionar o link da biblioteca em sua aplicação ou possuir o Node.js instalado na máquina, para projeto maiores ele possui o CLI próprio de configuração.

Suporte a outras linguagens: Possui suporte ao TypeScript e possui o Nuxtjs que é equivalente ao Nextjs do React, assim possuindo suporte ao tipo de renderização SSR.

Curva de Aprendizagem: A curva de aprendizagem também é bastante curta, com pouco conhecimento em JavaScript é possível criar uma aplicação com ele, seu ciclo de vida é facilmente entendido.

Mercado de trabalho: O mercado possui bastante vagas para este *framework* em grandes empresas.

Comunidade: O Vue.js possui uma vasta comunidade, foi visto que no Github o Vue.js tem mais contribuidores que o próprio React e assim é possuem vários projetos no Github utilizando o *frameworks*, e diversas pesquisas no StackOverFlow.

6.5.3 Angular

Documentação: Diferente dos outros *frameworks* da pesquisa, sua documentação tem suporte apenas a 4 idiomas, mas contém muitos detalhes.

Configuração: A configuração é pouco um mais complicada para iniciantes porque exija que tenham algumas interfaces de configuração utilizando o terminal.

Suporte a outras linguagens: Possui suporte ao TypeScript.

Curva de Aprendizagem: A curva de aprendizagem pode parecer ser curta, mas quando se trata de aplicações maiores tem um prazo para aprendizagem por possuir uma arquitetura diferente de seus concorrentes.

Mercado de trabalho: O mercado possui bastante vagas para este *framework* principalmente em bancos e empresas que tenham um sistema grande.

Comunidade: O Angular possui uma boa comunidade, mesmo não se destacando no Github é possível encontrar projetos de pesquisa e artigos na internet.

Após as comparações realizadas foi possível observar que os *frameworks* avaliados são ótimas opções para construção de aplicações web e cada um mostrou seus pontos positivos e negativos, mas deixam claro que desenvolvedores no inicio de sua carreira podem criar e desenvolver projetos em pouco tempo utilizando qualquer um dos *frameworks*. Em questão de desempenho o Angular e o Vue.js se destacam em relação ao tempo de execução dos casos de testes realizados, o React apesar de ficar atrás nos testes, não deixa de ser uma ótima opção, pois sua curva de aprendizagem se destaca em relação aos demais e o mercado atual está com várias vagas em relação aos outros *frameworks*. Vale ressaltar que o projeto realizado para os testes é uma pequena aplicação, em relação aos projetos em grandes empresas e que os *frameworks* talvez tenham melhores desempenho nessas aplicações com uma arquitetura melhor definida. Então para o uso em pequenas aplicações, o Vue.js é o mais recomendado após todos os testes realizados ele possui uma menor curva de aprendizagem e se destaca na parte de ferramentas.

7 CONSIDERAÇÕES FINAIS

Através deste trabalho, observou-se que está se tornando necessário o entendimento de diversos *frameworks* para o desenvolvimento de aplicações web, tem-se tornado importante não apenas utilizar um *framework* que atende às necessidades do projeto a ser desenvolvido, mas também um que provê um código de alta qualidade e bom desempenho. Dessa forma, um dos resultados gerais desse trabalho foi uma comparação entre os principais *frameworks* de JavaScript a nível de arquitetura e ferramentas, com o intuito de ajudar os desenvolvedores que pretendem entrar no mercado de desenvolvimento.

Além disso, foi realizado uma pesquisa com desenvolvedores *front-end* para entender quais características são levadas em consideração para o primeiro contato com um *framework* ou em motivos de mudanças no decorrer de suas carreiras. Com isso, foi feito uma pesquisa para descobrir quais os são os *frameworks* de JavaScript mais populares no mercado atualmente, em seguida foi definido um projeto para testes e então realizado a comparação de desempenho com cada um dos *frameworks* da pesquisa.

Os resultados mostraram-se muito satisfatórios, com poucas diferenças entre os *frameworks* em geral. O React acabou tendo os piores resultados nos testes de desempenho, mas conta com uma ótima comunidade para quem está iniciando no mundo da programação, enquanto Vue.js e Angular mostram-se bastante eficazes em quase todos os testes realizados.

Como trabalhos futuros pretende-se construir uma comparação de JavaScript com outras linguagens de programação, como Python utilizando Django[1], tanto em questões de arquitetura, como em desempenho. Além disso, é possível notar que existem novos *frameworks* sendo criados diariamente, aplicar uma comparação com *frameworks* novos é imprescindível para diversificar novas pesquisas, como trabalhar com Svelte[2] ou SolidJs[3]. Não foi utilizado apenas JavaScript neste trabalho, aplicar uma comparação entre trabalhar com *framework* e utilizando apenas HTML, CSS e JavaScript é importante para que seja possível coletar resultados para novos desenvolvedores que pensam utilizar ou não um *framework*.

[1] https://www.djangoproject.com/
[2] https://svelte.dev/
[3] https://www.solidjs.com/

REFERÊNCIAS

AGGARWAL, S. Modern web-development using reactjs. **International Journal of Recent Research Aspects**, [S.l], v. 5, n. 1, p. 133–137, 2018.

ALMEIDA, F. E. V. D. **Um comparativo entre frameworks javascript para desenvolvimento de aplicações front-end**. [S.l: s.n], 2018.

ALVIM, P. **Tirando o máximo do java ee 5 open-source com jCompanyc.** 3. ed. Belo Horizonte: Developer Suite: Powerlogic, 2008. ISBN 8590784800.

ANGULARJS. **What is Angular?** [S.l], 2009. Disponível em: https://docs.angularjs.org/guide/introduction. Acesso em: 15 nov. 2021.

BRANAS, R. **AngularJS Essentials** : design and construct reusable, maintainable, and modular web applications with AngularJS. Birmingham: Packt Publishing, 2014.

FERREIRA, H. K.; ZUCHI, J. D. Análise comparativa entre frameworks frontend baseados em javascript para aplicações web. **Revista Interface Tecnológica**, [S.l], v. 15, n. 2, p. 111–123, dez. 2018. Disponível em: https://revista.fatectq.edu.br/index.php/interfacetecnologica/article/view/502. Acesso em: 15 nov. 2021.

FLANAGAN, D. **JavaScript:** O Guia Definitivo. 6. ed. [S.l]: Bookman Editora, 2004. ISBN 9788565837484. Disponível em: https://books.google.com.br/books?id=zWNyDgAAQBAJ. Acesso em: 15 nov. 2021.

GIZAS, A. B. **Comparative evaluation of JavaScript frameworks**. Lyon, France: HPCLab, 2012.

KUMAR, A.; SINGH, R. K. Comparative analysis of angularjs and reactjs. **International Journal of Latest Trends in Engineering and Technology**, [S.l], v. 7, n. 4, p. 225–227, 2016.

KYRIAKIDIS, A.; MANIATIS, K. **The Majesty of Vue.js**. [S.l]: Packt Publishing, 2016. ISBN 9781787125209. Disponível em: https://books.google.com.br/books?id=Xp7cDgAAQBAJ. Acesso em: 15 nov. 2021.

MARIANO, C. L. **Benchmarking javascript frameworks**. Dublin: Dublin Institute of Technology, 2017.

PANO, A.; GRAZIOTIN, D.; ABRAHAMSSON, P. What leads developers towards the choice of a javascript framework? **ArXiv**, [S.l], 2016.

SILVA, M. S. **JavaScript-Guia do Programador:** guia completo das funcionalidades de linguagem JavaScript. [S.l]: Novatec Editora, 2010. ISBN 8575222481. Disponível em: https://books.google.com.br/books?id=BB9WDQAAQBAJ. Acesso em: 15 nov. 2021.

VIPUL A. M.; SONPATKI, P. **ReactJS by Example-Building Modern Web Applications with React**. [S.l]: Packt Publishing, 2016. ISBN 9781785282744. Disponível em: https://books.google.com.br/books?id=Ht3JDAAAQBAJ. Acesso em: 15 nov. 2021.

VUEJS. **Primeiros passos**. [S.l: s.n.], 2013. Disponível em: https://012.vuejs.org/guide/. Acesso em: 13 dez. 2021.

WILLIAMSON, K. **Introdução ao AngularJS:** um guia para o desenvolvimento com o AngularJS. [S.l]: Novatec Editora, 2015. ISBN 9788575224304. Disponível em: https://books.google.com.br/books?id=W4jDCAAAQBAJ. Acesso em: 13 nov. 2021.

ZAKAS, N. **High Performance JavaScript:** build faster web application interfaces. [S.l]: O'Reilly Media, 2010. ISBN 9781449388744. Disponível em: https://books.google.com.br/books?id=ED6ph4WEIoQC. Acesso em: 13 nov. 2021.

APÊNDICE A – *SCRIPTS* PARA A ANÁLISE DE DESEMPENHO EM TEMPO DE EXECUÇÃO

Código-fonte 1 – Função de Adicionar Tarefa

```
const handleSubmit = text => {
    console.time('addTodo');
    const newTodo: Todo = {
      text,
      isComplete: false,
      id: Math.random().toString(16)
    }
    const updatedTodos = [...this.todos$.getValue(),
       newTodo];
    this.todos$.next(updatedTodos);
    console.timeEnd('addTodo');
}
```

Código-fonte 2 – Função de Remover Tarefa

```
const removeTodo(id: string): void {
    console.time('removeTodo');
    const updatedTodos = this.todos$
      .getValue()
      .filter((todo) => todo.id !== id);
    this.todos$.next(updatedTodos);
    console.timeEnd('removeTodo');
}
```

Código-fonte 3 – Função de Editar Tarefa

```
const updateTodo(todoId, newTodo) {
    console.time('update');
```

```
    this.todos$.map((todo) => {
        if (todo.id === todoId) {
          todo.text = newTodo.text;
        }
        return todo;
      });
    console.timeEnd('update');
}
```