UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM ENGENHARIA DE SOFTWARE

FRANCISCO EMERSON VIEIRA DE ALMEIDA

UM COMPARATIVO ENTRE FRAMEWORKS JAVASCRIPT PARA DESENVOLVIMENTO DE APLICAÇÕES FRONT-END

QUIXADÁ

2018

FRANCISCO EMERSON VIEIRA DE ALMEIDA

UM COMPARATIVO ENTRE FRAMEWORKS JAVASCRIPT PARA DESENVOLVIMENTO DE APLICAÇÕES FRONT-END

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Orientadora: Prof. Ma. Antonia Diana Braga Nogueira

Coorientador: Me. Zarathon Lopes Viana

QUIXADÁ

2018

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)

A446c Almeida, Francisco Emerson Vieira de.
Um comparativo entre frameworks javascript para desenvolvimento de aplicações front-end / Francisco Emerson Vieira de Almeida. – 2018.
42 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2018.
Orientação: Prof. Ma. Antonia Diana Braga Nogueira.
Coorientação: Me. Zarathon Lopes Viana.

1. Frameworks (Programa de Computador) . 2. Benchmarking (Administração) . 3. Javascript (Linguagem de programação de computador) . I. Título. CDD 005.1

FRANCISCO EMERSON VIEIRA DE ALMEIDA

UM COMPARATIVO ENTRE FRAMEWORKS JAVASCRIPT PARA DESENVOLVIMENTO DE APLICAÇÕES FRONT-END

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Aprovado em: ____/_____/_____.

BANCA EXAMINADORA

Prof. Ma. Antonia Diana Braga Nogueira (Orientadora)

Universidade Federal do Ceará – UFC

Me. Zarathon Lopes Viana (Coorientador)

Universidade Federal do Ceará - UFC

Prof. Dra. Paulyne Matthews Jucá

Universidade Federal do Ceará - UFC

Prof. Me. Victor Aguiar Evangelista de Farias

Universidade Federal do Ceará - UFC

Dedico este trabalho aos meus amigos, familiares e meus orientadores pelo suporte durante o desenvolvimento do mesmo.

AGRADECIMENTOS

Aos meu orientadores, Zarathon Maia e Diana Braga, em especial à Diana Braga pelo apoio durante minha graduação, pelos conselhos e paciência com minha pessoa.

“Sempre incluí minorias em minhas histórias, muitas vezes como heróis. Vivemos em uma sociedade diversificada - na verdade, um mundo diverso, e precisamos aprender a viver em paz e com respeito uns pelos outros.”

(Stan Lee)

RESUMO

Com o avanço da internet e seu número de usuários, os sistemas *web* precisam ser desenvolvidos com menor o tempo e esforço possível, com isto, surgiram os *frameworks*, que prometem solucionar essa questão. O presente trabalho apresenta um comparativo entre *frameworks javascript* para desenvolvimento de aplicações front-end. Foi feito uma pesquisa para descobrir os *frameworks* mais populares, e em seguida, foi proposto um projeto teste para que fosse desenvolvido utilizando cada um dos *frameworks* escolhidos. Foram definidos casos de testes para realizar e obter resultados de *bechmark* dos projetos desenvolvidos com os *frameworks*.Os dados coletados na análise foram consumo de memória, cpu e tempo de execução dos métodos de CRUD(Create, Read, Update e Delete).

Palavras-chave: Frameworks. Benchmark. Javascript.

ABSTRACT

With the advancement of the internet and its number of users, web systems need to be developed with the least time and effort possible, with this, the frameworks have emerged, which promise to solve this issue. The present work presents a comparison between javascript frameworks for the development of front-end applications. A research was done to discover the most popular frameworks, and then, a test project was proposed to be developed using each of the chosen frameworks. Was defined test cases to perform and obtain bechmark results of the projects developed with the frameworks. The data collected in the analysis were memory consumption, cpu and execution time of the CRUD (Create, Read, Update and Delete) methods.

Keywords: Framework. Benchmark. Javascript.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com o contínuo crescimento da internet e dos dispositivos que se conectam a ela, o mercado tem buscado diversas maneiras de lançar aplicações que atendam a crescente demanda e, como resultado, novas tecnologias surgiram junto da busca de levar uma melhor experiência para os usuários finais. As aplicações ficaram cada vez mais dinâmicas devido ao *javascript*, e seu uso tornou-se essencial para facilitar no desenvolvimento de software.

O trabalho de melhorar as aplicações, levou ao surgimento de *frameworks*, que buscam ajudar na construção de produtos de qualidade e com o menor esforço de trabalho. Alguns *frameworks* são criados com base no padrão de arquitetura chamado MVC (*Model-View-Controller*), um padrão que visa separar uma aplicação em camadas de modelo, visão e controle.

Nos últimos anos, o *javascript* ganhou força com o surgimento do *node.js* e NPM. O que antes era apenas uma linguagem do lado cliente, tornou-se um ecossistema usado tanto no *front-end* como no *back-end* (PINHO, 2017).

Os *frameworks javascript* do lado cliente possuem vantagens em relação aos demais web *frameworks* por serem simples, ágeis, de desenvolvimento rápido e capacidade de empacotar a aplicação e distribuir em vários dispositivos (WILLIAMSON, 2015).

Tendo em vista as diversas opções de *frameworks* disponíveis no mercado, o presente trabalho propõe um estudo comparativo entre os quatro *frameworks javascript* para *front-end* mais usados pela comunidade de desenvolvedores, com o intuito de auxiliar a comunidade na escolha de qual deles se adequa melhor ao seu projeto Web.

O trabalho consiste em uma pesquisa para a escolha dos *frameworks*. Em seguida, será definido um projeto teste, que será desenvolvido usando cada um dos *frameworks* escolhidos e, por fim, será realizado um *benchmark* para obter os resultados comparativos. Mediante os resultados que serão obtidos, será apresentado pontos positivos e negativos de cada *framework* e também será sugerido dicas de quando usar e para que tipo de projetos os *frameworks* avaliados devem ser usados no desenvolvimento de aplicações Web.

2 TRABALHOS RELACIONADOS

Nesta seção, serão descritos os estudos que possuem relação com o trabalho proposto.

2.1 Análise comparativa entre frameworks de front-end para aplicações web ricas visando reaproveitamento do back-end

Em Amaral e Neris (2016), é apresentada uma análise comparativa entre bibliotecas *Javascript* para o desenvolvimento do *front-end* de interfaces Web ricas e responsivas com o objetivo de avaliar o comportamento delas no apoio ao reaproveitamento do *back-end* (camada de negócio).

A análise leva em consideração duas bibliotecas, que foram selecionadas por apresentarem uma quantidade maior de componentes responsivos do que as demais disponíveis no mercado, além de serem *open sources*. Outro critério para a seleção dos *frameworks* foi o fato de que os mesmos funcionam na maioria dos navegadores dos dispositivos disponíveis no mercado.

Amaral e Neris (2016) desenvolveram um sistema de catálogo *cloud* para fins de análise comparativa. Para que a análise comparativa pudesse ser realizada, os autores selecionaram alguns critérios de avaliação, que foram:

- Reponsividade: Capacidade de um site se adaptar a diferentes resoluções de telas;
- Compatibilidade entre navegadores: Garantia de funcionamento em qualquer navegador.
- Suporte/documentação: Documentação bem escrita;
- Esforço de implantação: Esforço de tempo para colocar um sistema em produção.

Amaral e Neris (2016) concluem que a existência de diversas bibliotecas para desenvolvimento de RIAs(*Rich Internet Application*) dificulta a escolha do desenvolvedor sobre qual biblioteca utilizar em sua aplicação. Mas que, com a análise comparativa realizada, tendo como base um sistema de baixa complexidade, acredita-se ter auxiliado uma escolha mais precisa sobre qual biblioteca, dentre as analisadas, atendam aos requisitos de desenvolvimento de uma aplicação web.

A análise comparativa proposta neste trabalho, também tem foco em *frameworks* para front-end, mas considera-se *frameworks* específicos da linguagem *javascript*. Neste, há

diferença no que se refere ao método de escolha dos *frameworks*: enquanto Amaral e Neris (2016) escolhem os *frameworks* pela quantidade de componentes, o presente trabalho os escolheu por sua popularidade na comunidade de desenvolvimento.

2.2 Análise comparativa entre dois Frameworks MVC para a Plataforma Java EE: JSF e VRaptor

Couto e Foschini (2016) propõem um estudo comparativo entre dois *frameworks* MVC para a plataforma Java EE: JSF e VRaptor. O JSF possibilita desenvolver aplicações baseadas em componentes de forma ágil, abstraindo os detalhes do protocolo HTTP. O VRaptor é um *framework* baseado em ações, que permite desenvolver aplicações de forma ágil sem precisar abstrair ou inibir os detalhes do protocolo HTTP. A proposta dos autores foi apresentar as principais características, vantagens e desvantagens através de um estudo comparativo entre os dois *frameworks*.

Dadas as características técnicas de cada *framework*, o estudo comparativo feito em Couto e Foschini (2016), entre JSF e VRaptor aborda critérios como:

- Curva de aprendizagem: Esforço medido em tempo no que se refere ao aprendizado;
- Tamanho da comunidade: Número de pessoas que fazem parte da comunidade;
- Aceitação pelo mercado: Refere-se ao seu uso pelo mercado;
- Documentação: Documentação acessível e de boa escrita;
- Arquitetura REST: Estilo arquitetural de sistemas.

Couto e Foschini (2016) concluem que escolher entre um ou outro *framework* é uma decisão cautelosa que deve levar em consideração muitos fatores, como maturidade do time, prazo do projeto, usabilidade e experiência do usuário, dispositivos que acessarão a aplicação e integrações com outros softwares e parceiros. O JSF, de acordo com as características apresentadas, é uma escolha mais confortável para times de desenvolvedores que não possuem tanta experiência com os padrões e tecnologias do paradigma Web ou que não possuem um profissional focado em design e experiência do usuário. O VRaptor, por sua vez, é uma boa opção para aplicações que necessitam de maior liberdade para a view, exigindo que o time tenha experiência com o protocolo HTTP e os demais detalhes de infraestrutura presentes no paradigma Web.

Couto e Foschini (2016) propõem como trabalho futuro a possibilidade de explorar,

por meio de um estudo comparativo, as características do MVC 1.0 e do JSF, ambos como especificações oficiais da plataforma Java EE.

O presente trabalho também tem como proprosta a análise de *frameworks*, mas diferente do que foi proposto por Couto e Foschini (2016), foi feito uma pesquisa para a escolha dos *frameworks* a serem analisados, com base em critérios definidos na seção 4.1. Outro diferencial do presente trabalho, é que os *frameworks* escolhidos serão aqueles da linguagem *javascript* e que sejam para front-end.

2.3 Frameworks e Bibliotecas JavaScript

Duarte (2015) apresenta um conjunto de plataformas e bibliotecas *javascript* que existem no mercado. O autor faz um breve resumo da história do *javascript*, suas características e evolução. Também faz um breve resumo sobe padrões de software, MVC, MVVM e fala sobre inversão de controle. Destaca ainda as diferenças entre biblioteca e *framework.* Aborda sobre plataformas do lado cliente em *javascript* e Isomórficas.

Em Duarte (2015) são descritos três cenários de desenvolvimento web que são importantes para proporcionar o contexto de avaliação dos *frameworks* e bibliotecas analisadas.

No capítulo quatro do texo de Duarte (2015) são apresentadas as características dos *frameworks* pesquisados, para se entender melhor o funcionamento e aplicabilidade desses *frameworks*. As características que foram apresentadas são típicas de plataformas do lado cliente, como:

- Two way binding: Sincronismo de dados entre *model-view/view-model*;
- Routing: Recurso para criar rotas;
- Templates Engines: Manipulação de HTML de forma dinâmica;
- Ferramentas de Teste: Ferramentas para testes de aplicação;
- Padrão de Arquitetura: Padrão de arquitetura de sistemas;
- Curva de aprendizagem: Esforço medido em tempo no que se refere ao aprendizado;
- Tamanho: Tamanho do framework medido em kb/mb;
- Comunidade: Número de pessoas que fazem parte da comunidade;
- Suporte do Navegador de Internet Mínimo: Refere-se da compatibilidade mínima de seu funcionamento em um navegador;
- Isomórfica: Que pode ser executado tanto no lado cliente quanto do lado servidor;

- Suporte Comunitário: Pessoas da comunidade engajadas no que se refere a tirar dúvidas.

As informações de suporte comunitário tiverem como referência duas plataformas que são: referência em controle de versão e de ajuda. São elas:

- Github;
- StackOverflow.

A escolha das plataformas a serem usadas no estudo em Duarte (2015) baseou-se em dois objetivos:

- Apresentar diferentes plataformas com ideais consequentemente diferentes
- Plataformas populares no mercado e algumas outras não tão conhecidas.

Em Duarte (2015), a comparação das plataformas e bibliotecas teve como objetivo mostrar a relação que existe entre estas plataformas, as suas características e os cenários em que estas plataformas podem ser usadas. A avaliação do autor foi de que existem inúmeras plataformas e bibliotecas no mercado, tornando a escolha de uma plataforma em detrimento de outra uma tarefa difícil e em muitos casos subjetiva. E, devido a esta dificuldade, decidiu-se realizar um inquérito com desenvolvedores de aplicações web. O inquérito constituiu-se de um conjunto de questões de múltipla escolha relacionadas com o desenvolvimento de aplicações web para cada um dos vários cenários propostos no trabalho. Foi enviado para 50 desenvolvedores na área, dos quais 27 responderam, dando um total de 54% da amostragem. Os resultados obtidos através do inquérito realizado aos desenvolvedores na área focou-se em três cenários, que foram:

- Cenário SPA: Aplicações Web de uma página, dinâmicas e com o mínimo de atualizações na página;
- Cenário Isomórfico: *Framework* que possibilita rodar um mesmo código tanto no cliente como no servidor
- Widget: Componente de interface gráfica.

Duarte (2015) conclui que existem inúmeras plataformas e bibliotecas no mercado, tornando a tarefa de escolha quase inconcebível. Devido a esta dificuldade de escolha, considerou-se neste estudo, que a melhor solução seria avaliá-las em relação aos vários cenários já previamente estabelecidos. Os cenários foram avaliados em relação à sua aplicabilidade em relação a um conjunto de características importantes para os mesmos, o que permitiu abranger ao máximo as necessidades do mercado e favorecendo uma melhor avaliação de cada uma das plataformas, porque estas também foram desenvolvidas a pensar em cenários diferentes.

O trabalho proposto também tem como objetivo escolher as plataformas a serem comparadas de acordo com sua popularidade através das plataformas que são referências em controle de versão e ajuda em desenvolvimento de software, e também mostrar as caraterísticas dos *frameworks*. O diferencial do presente trabalho em relação ao de Duarte (2015) é que o foco de estudo é sobre os três *frameworks javascript* para *front-end* e os resultados da análise comparativa se dará através de um *bechmark.*

3 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados os principais conceitos necessários para o desenvolvimento deste trabalho, e como estes conceitos impactam seu desenvolvimento.

3.1 Frameworks

Existem muitas definições para *Frameworks* na literatura. Segundo Fayad, Schmidt e Johnson (1999), *Framework* é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação.

Para ALVIM (2008), *Framework* é um conjunto de classes que trabalham de forma integrada e extensível, procurando dar bases de soluções reutilizáveis para domínios específicos de problemas.

Já Schmidt et al. (2013), definem um *framework* como um software parcialmente completo projetado para ser instanciado. O framework define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los.

Com base no que foi dito pelos autores citados, os *Frameworks* permitem que funcionalidades já implementadas possam ser reutilizadas pelos desenvolvedores na construção de se suas aplicações, com um menor esforço de trabalho e tempo gastos no mesmo.

3.2 Document Object Model(DOM)

Segundo (NICOL et al., 2001), O DOM(Document Object Model) é uma interface de programação para documentos HTML(HyperText Markup Language). Ele define como os documentos são acessados e manipulados. É a partir do DOM que programadores podem criar documentos, criar, atualizar ou até deletar elementos de uma estrutura HTML.

É a API do DOM que é usada para manipular o DOM. O navegador re-renderiza as partes do DOM que foram manipuladas. Manipular o DOM exige experiência, pois se mal manipulado, ele pode renderizar nós desnecessários e isso pode acarretar em baixo desempenho (PRUSTY, 2016).

Alguns *frameworks* possuem suas estratégias para manipular o DOM. O ReactJs manipula através de um recurso chamado de Virtual DOM, que possui uma melhor performance em relação ao DOM convencional. O Angular manipula o DOM através do *two-way binding*(ligação bidirecional), que permite uma ligação de duas vias entre o DOM e os objetos

no *Javascript*. (PONTES, 2018).

O *framework* VueJs, assim como o ReactJs, manipula o DOM através do DOM Virtual (INCAU, 2017).

O Virtual DOM é uma cópia do DOM real. Quando um componente recebe novas propriedades ou seu estado é alterado, o *framework* verifica a diferença entre o estado anterior do DOM virtual com o novo estado e então atualiza o DOM real apenas nos nós que sofreram modificações.

O *two-way binding* lida com a comunicação bidirecional entre o componente e o DOM, ou seja, do componente para o DOM e do DOM para o componente. O componente pode sofrer alterações que vão modificar o DOM, assim como o DOM pode sofrer alterações que podem modificar o componente.

3.3 Front-end

O fron-end está relacionado ao visual de uma página web. Segundo EIS (2015) as linguagens de front-end são divididas em HTML, CSS e Javascript. E com essas linguagens, as páginas web ganham vida e permitem uma melhor experiência do usuário através de uma interface bem desenvolvida. É através do front-end que o usuário iterage com as aplicações web e inserem entradas que o back-end da aplicação irá processar de acordo com as regras de negócio da aplicação.

Para citrus7 (2017) front-end é a primeira camada na qual o usuário se depara ao se acessar um site, uma intranet ou um sistema web. É no front-end onde se encontra a interface de navegação e design da página, compostas por HTML, CSS E *Javascript*.

O front-end trabalha junto ao designer para poder construir as interfaces de usuário, de forma que os usuários possam ter uma boa experiência de usabilidade. É do front-end a responsabilidade de receber dados de entrada dos usuários e enviar para o servidor, onde esses dados serão processados.

3.4 Benchmark

Segundo o Wikipédia (2017), *benchmark* na computação é uma forma de avaliar o desempenho relativo de um objeto, executando uma série de testes padrões sobre ele, afim de obter informações do funcionamento do objeto sob diferentes condições.

Para Bouckaert et al. (2010), *benchmark* é o ato de medir e avaliar o desempenho computacional, protocolos de rede, dispositivos e etc. Tem como objetivo: permitir uma comparação equitativa entre diferentes soluções, ou entre os desenvolvimentos subsequentes de um sistema de teste.

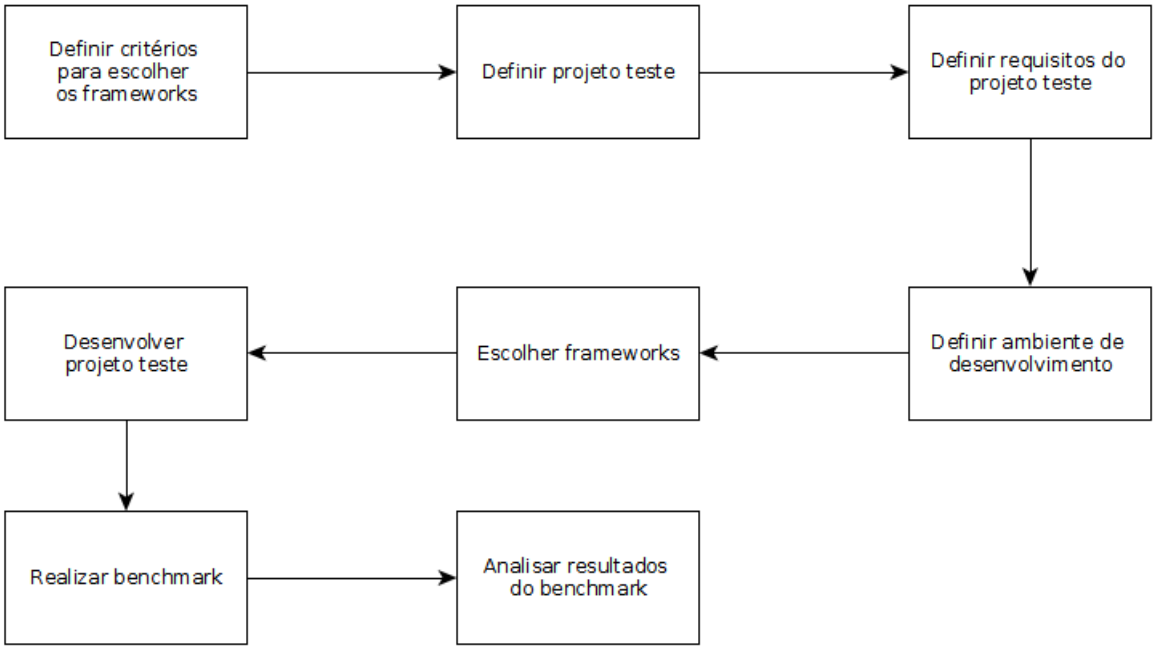
O *benchmark* que será realizado neste trabalho será para avaliar o desempenho dos *frameworks* em virtude dos casos de testes que serão aplicados no projeto teste que será desenvolvido com cada um dos *frameworks* escolhidos. Será utilizado o navegador Google Chrome e sua ferramenta de desenvolvedor para executar e avaliar cada caso de teste realizado.

O *benchmark* levará em consideração o tempo de execução, consumo de memória e CPU para cada caso de teste. Os *frameworks* serão avaliados individualmente.

4 MATERIAIS E MÉTODOS

Na Figura 1, são apresentados os procedimentos metodológicos que foram utilizados neste trabalho. Nas seções abaixo é apresentado o detalhamento dos passos ilustrados na figura.

Figura 1 – Procedimentos Metodológicos



Fonte – Elaborada pelo autor

4.1 Critérios de escolha dos *frameworks*

A fim de que o estudo comparativo pudesse ser realizado, foram estabelecidos alguns critérios de escolha dos *frameworks*. Mas primeiramente foi preciso definir o número de *frameworks* a serem comparados, e por escolha do autor, foi estabelecido que seriam escolhidos três *frameworks*.

Os critérios adotados para definir os *frameworks* a serem analisados foram:

- Popularidade;
- Comunidade.

Para se conseguir os dados referentes aos critérios definidos, buscou-se sites especializados em controle de versão e ajuda com desenvolvimento de software. E os escolhidos foram:

- Github;

- StackOverflow.

4.2 Projeto Teste

O projeto teste é a aplicação que foi desenvolvida utilizando os três *frameworks* escolhidos através da pesquisa realizada anteriormente. A aplicação é um sistema de agenda telefônica que foi utilizada para a análise dos *frameworks*.

4.3 Requisitos do Projeto Teste

O objetivo do sistema de agenda telefônica é de auxiliar os seus utilizadores na organização das informações de seus contatos.

- Requisito - 01: Adicionar Contato
- Requisito - 02: Remover Contato
- Requisito - 03: Buscar Contato
- Requisito - 04: Editar Contato

4.4 Ambiente de Desenvolvimento

A Tabela 1 a seguir contém a descrição das ferramentas que serão utilizadas para o desenvolvimento do projeto teste.

Tabela 1 – Tabela de Ferramentas

Ferramentas			
Nome	Tipo	Versão	Descrição
WebStorm	IDE	2018.2.5	Ferramenta para a codificação do projeto teste.
Google Chrome	Navegador	70.0.3538.77	Navegador para executar o projeto teste.
Git	Sistema de Controle de Versão	2.17.1	Ferramenta utilizada para controlar/gerenciar as mudanças do projeto teste.
Linux Mint	Sistema Operacional	19 Tara	Sistema Operacional utilizado para executar as ferramentas de desenvolvimento.
Javascript	Linguagem de Programação	ES6	Linguagem por trás dos frameworks a serem comparados.
Selenium IDE	Testes	3.4.4	Ferramenta para a automatização de testes de UI.

Fonte – elaborada pelo autor

4.5 Escolha dos Frameworks

Definido os critérios de escolhas dos *frameworks*, foi feita uma pesquisa nos sites escolhidos na seção 4.1. No Github, foi levado em consideração o número de estrelas no repositório do *framework*, já no StackOverflow, foi levado em consideração o número de questões e seguidores. Foi constatado uma pequena disparidade nos resultados em relação as duas plataformas. No *Github*, o *react* ocupa a primeira colocação em popularidade, mas no *Stackoverflow*, o *angular.js* tem popularidade maior.

A Tabela 2 contém o resultado obtido na plataforma de controle de versão Github. Na Tabela 3, pode-se verificar o resultado obtido na plataforma de ajuda com desenvolvimento de software StackOverflow.

Tabela 2 – Top 3 Frameworks no Github

Frameworks - Github	
Nome	Data de Acesso
ReactJs	28 de maio de 2017
Angular	28 de maio de 2017
VueJs	28 de maio de 2017

Fonte – elaborada pelo autor

Tabela 3 – Top 3 Frameworks no StackOverflow

Frameworks - StackOverflow			
Nome	Nº de Questões	Nº de Seguidores	Data de Acesso
ReactJs	43,295	29,2K	28 de maio de 2017
Angular	233,048	103,05K	28 de maio de 2017
VueJs	6,558	3,9K	28 de maio de 2017

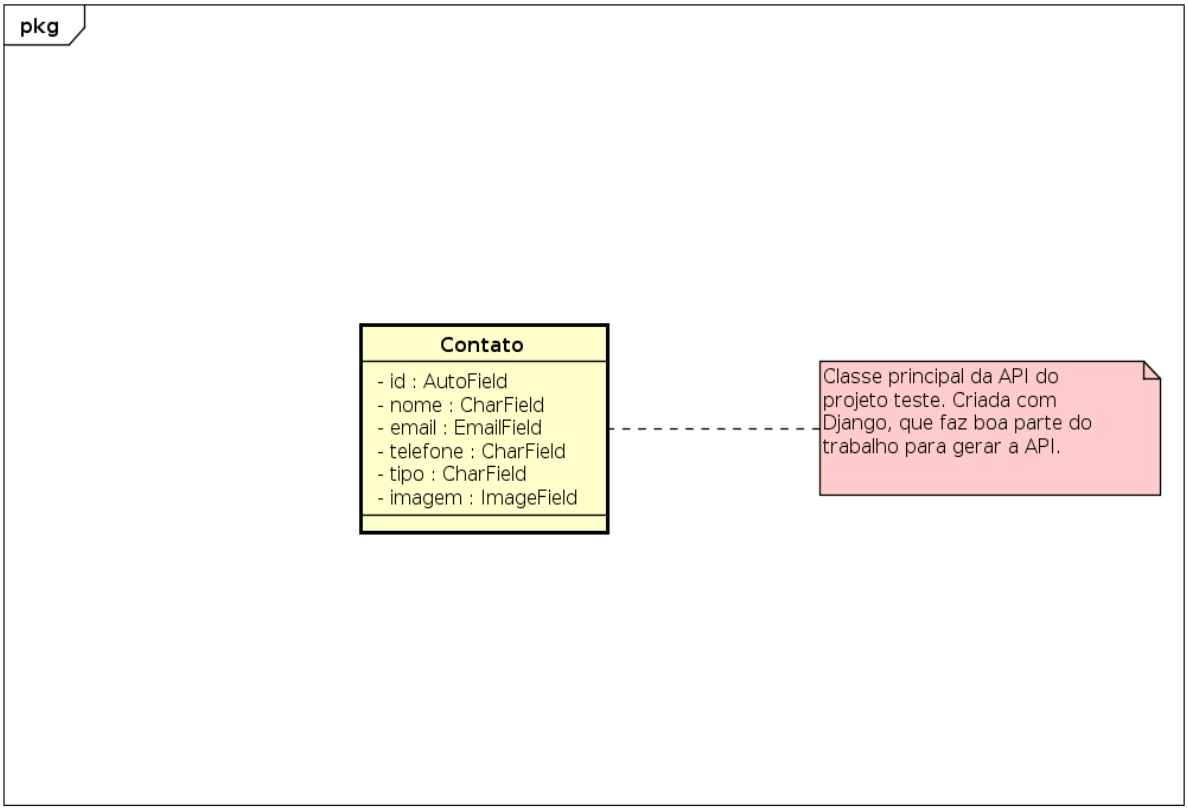
Fonte – elaborada pelo autor

5 RESULTADOS

Este capítulo apresenta os resultados e descreve os testes realizados para avaliar os *frameworks* escolhidos na seção 4.5. Os testes seguiram o que foi descrito na seção 3.4.

A Figura 2 contém a modelagem do projeto teste que foi desenvolvido com cada um dos *frameworks* escolhidos.

Figura 2 – Diagrama de classe da agenda telefônica.



Fonte – elaborada pelo autor

Foi desenvolvido uma API REST com o *framework* Django, que facilitou desenvolver o *Back-end* do projeto, que contém os dados consumidos pelo projeto teste proprosto neste trabalho. A API foi desenvolvida seguindo o diagrama de classes da figura 2.

A Tabela 4 possui os casos de teste que foram elaborados neste trabalho, onde cada caso servirá de base para avaliar os *frameworks*.

Tabela 4 – Tabela de Casos de Teste

Casos de Teste	
Casos de Teste	Descrição
CT01	Criar 100 contatos
CT02	Editar/Atualizar 100 contatos
CT03	Deletar 100 contatos
CT04	Carregar aplicação com todos os contatos
CT05	Abrir 10 instâncias da aplicação sem contatos
CT06	Abrir 10 instâncias da aplicação com 100 contatos

Fonte – elaborada pelo autor

A tabela 5 contém o nome e a versão dos *frameworks* usados neste trabalho. Foi adicionado o AngularJs no trabalho,para fins de análise das duas versões do *framework.*

Tabela 5 – Frameworks utilizados

Frameworks - Github	
Nome	Versão
ReactJs	15.6.2
Angular	7
AngularJs	1.7.5
VueJs	2.5.17

Fonte – elaborada pelo autor

5.1 Execução dos Testes

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

- **Processador**: Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz
- **Memória(RAM)**: 12GB
- **Hard drive**: 500GB

A execução dos casos de testes seguiu a ordem que está na Tabela 5 e com cada um dos *frameworks* por vez. Os dados colhidos para alimentar a análise do *benchmark* foram:

- **Consumo de memória**
- **Consumo de cpu**
- **Tempo de execução de cada método**

Para a obtenção dos dados de memória e CPU foi usado o gerenciador de tarefas do navegador Google Chrome.

Figura 3 – Gerenciador de tarefas do Google Chrome.

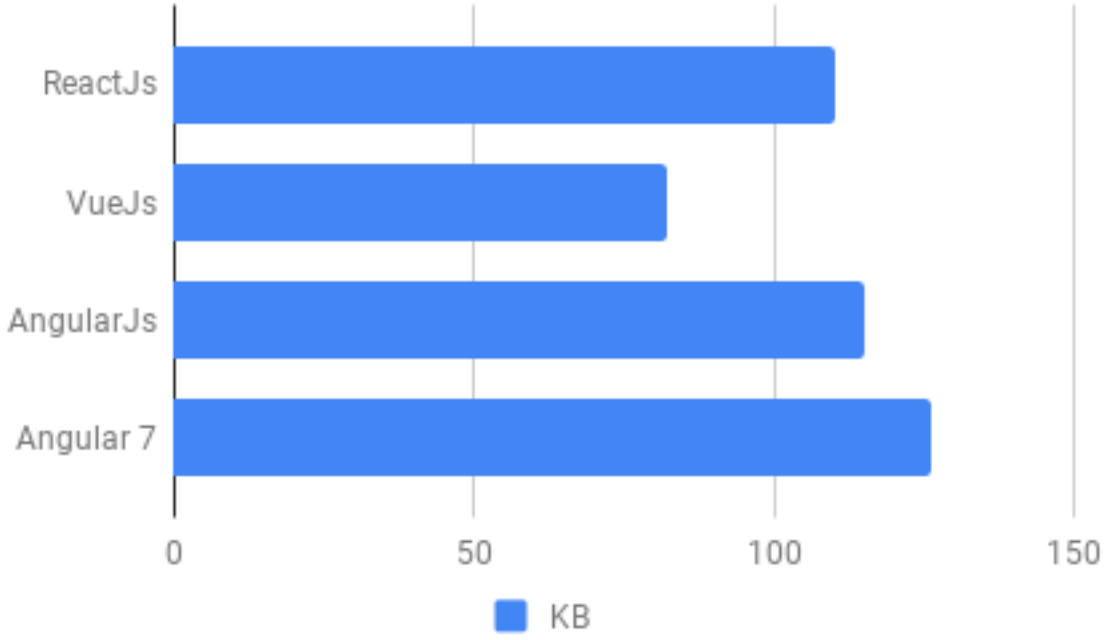
Fonte – Elaborada pelo autor

O *benchmark* foi executado no Google Chrome, como definido na tabela 1. Os testes foram automatizados com a ferramenta Selenium IDE no navegador. Primeiramente foi feito a execução manual dos casos de testes, com a ferramenta Selenium gravando cada passo da execução. Por fim, foi gerado um *script* de automação para cada caso de teste.

O projeto teste e os *scripts* de automação estão hospedados no Github. Link para o repositório: <https://github.com/mensonones/TCC2_ProjetoTeste>

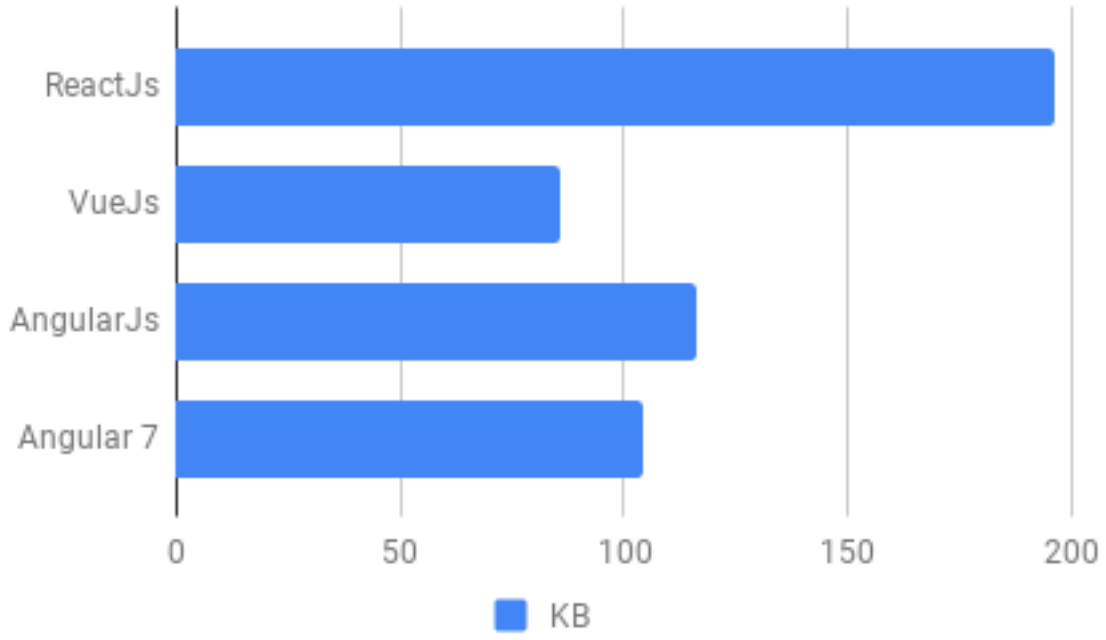
As Figuras 4, 5, 6, 7 contém os resultados dos testes para o consumo de memória RAM. O teste levou em consideração o pico de consumo de memória por cada *framework*.

[CT01]
Figura 4 – Resultado dos testes para consumo de memória.



Fonte – elaborada pelo autor

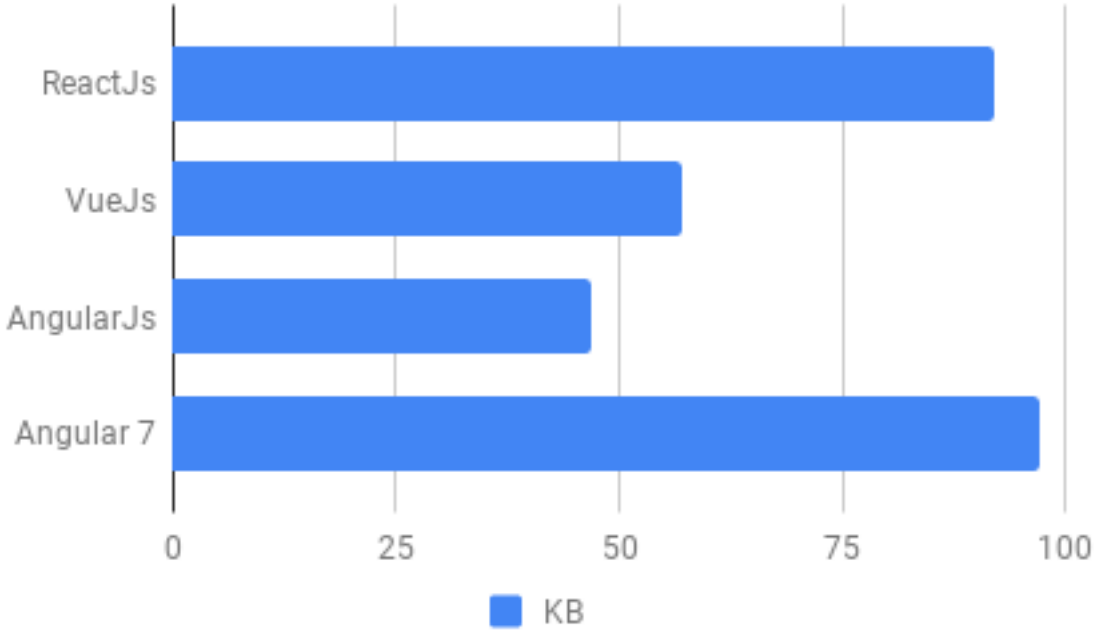
[CT02]
Figura 5 – Resultado dos testes para consumo de memória.



Fonte – elaborada pelo autor

[CT03]

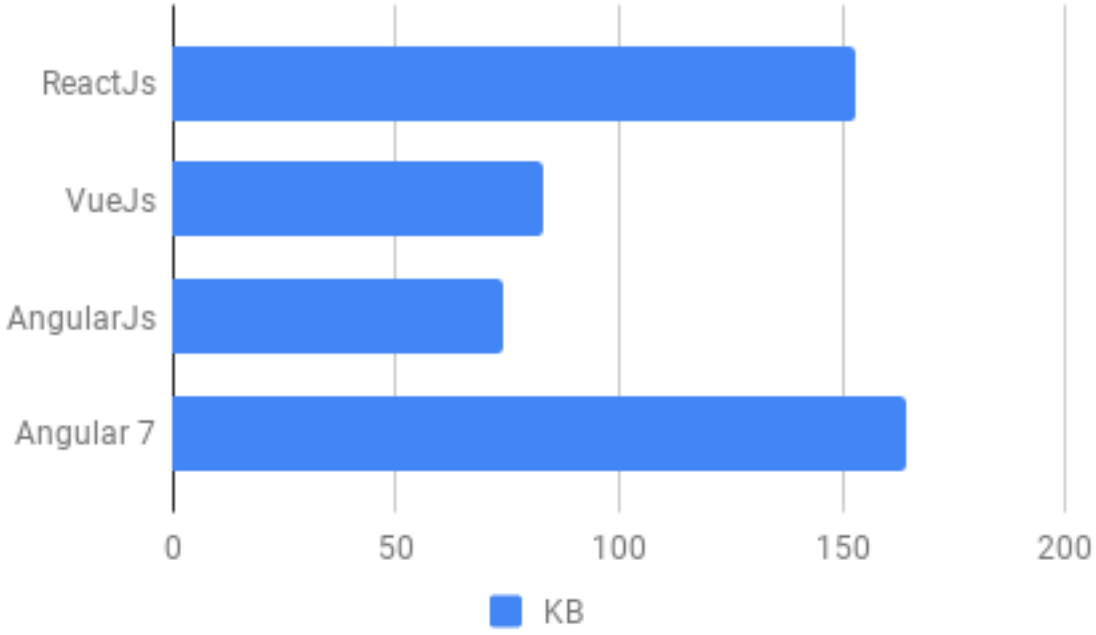
Figura 6 – Resultado dos testes para consumo de memória.



Fonte – elaborada pelo autor

[CT04]

Figura 7 – Resultado dos testes para consumo de memória.



Fonte – elaborada pelo autor

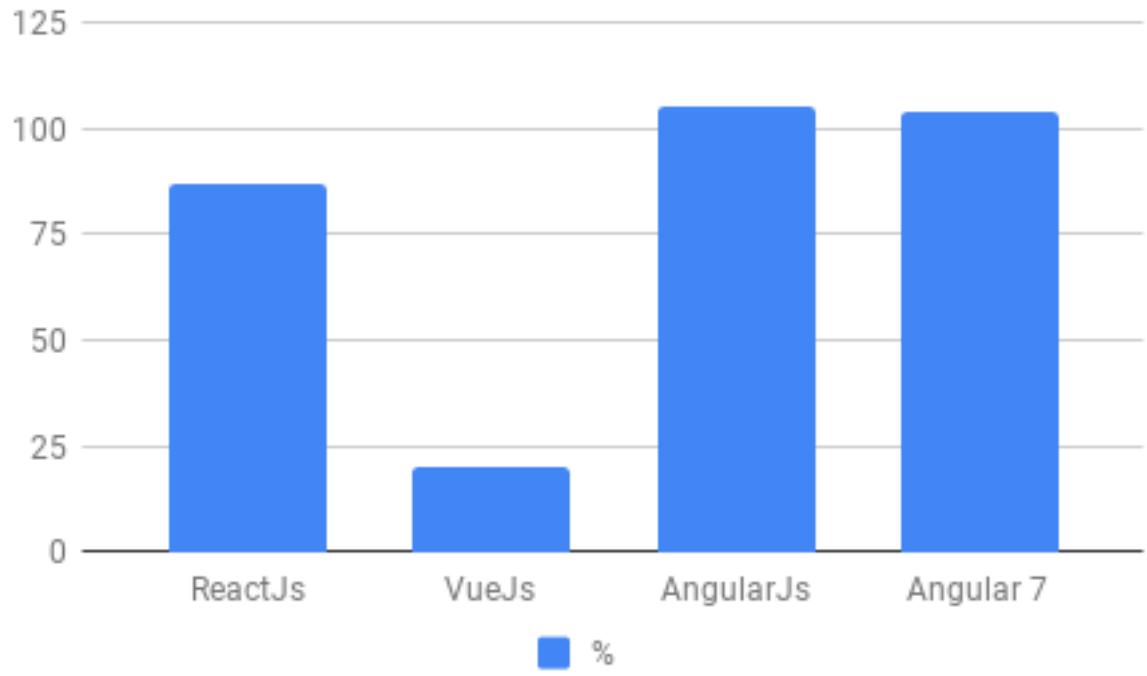
Como pode se constatar no gráfico do**CT1** (criar 100 contatos), o ReactJs, AngularJs e Angular tiverem um consumo um pouco acima de 100, enquanto o VueJs teve um melhor

desempenho por ter consumido menos RAM. No **CT2** (atualizar 100 contatos), o gráfico mostra que o ReactJs teve um maior consumo de RAM, já o AngularJs e Angular tiveram um consumo parecido, enquanto VueJs continuou com o menor consumo. Já o **CT03** (deletar 100 contatos), ReactJs e Angular tiveram os maiores consumos, chegando próximo de 100, enquanto VueJs teve um consumo menor em relação ao ReactJs e Angular, seu consumo ainda foi maior que o do AngularJs, que teve o menor consumo. E por último, no **CT04** (carregar a aplicação com todos contatos), ReactJs e Angular continuaram com os maiores consumo de RAM, o VueJs e o AngularJs com os menores consumos, mas o AngularJs com o menor consumo no geral.

As Figuras 8, 9, 10 e 11 ilustram os resultados obtidos no testes para consumo de CPU. Como foi no teste para obter o consumo de RAM, levou-se em consideração o pico de consumo de CPU.

[CT01]

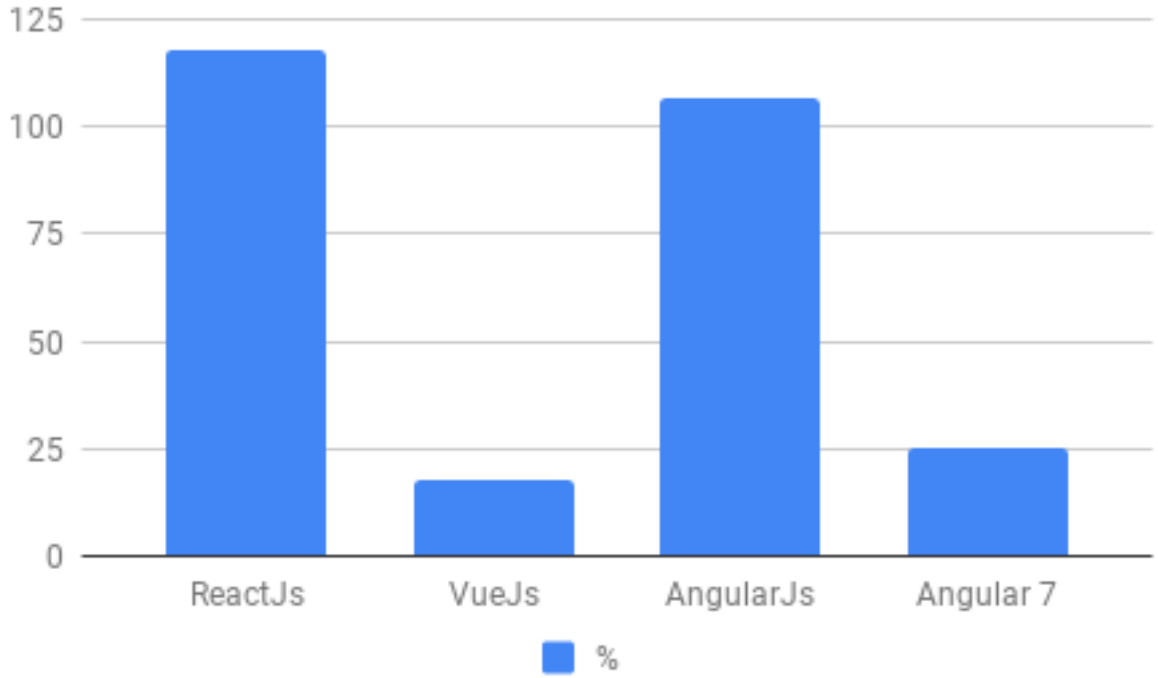
Figura 8 – Resultado dos testes para consumo de CPU.



Fonte – elaborada pelo autor

[CT02]

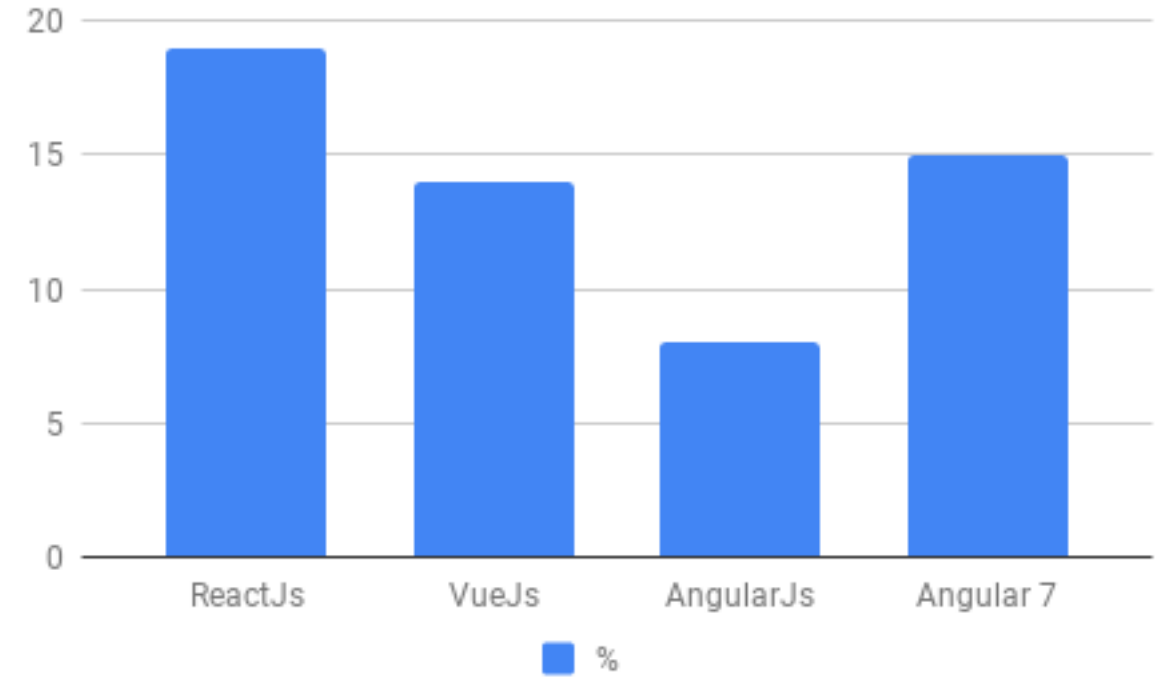
Figura 9 – Resultado dos testes para consumo de CPU.



Fonte – elaborada pelo autor

[CT03]

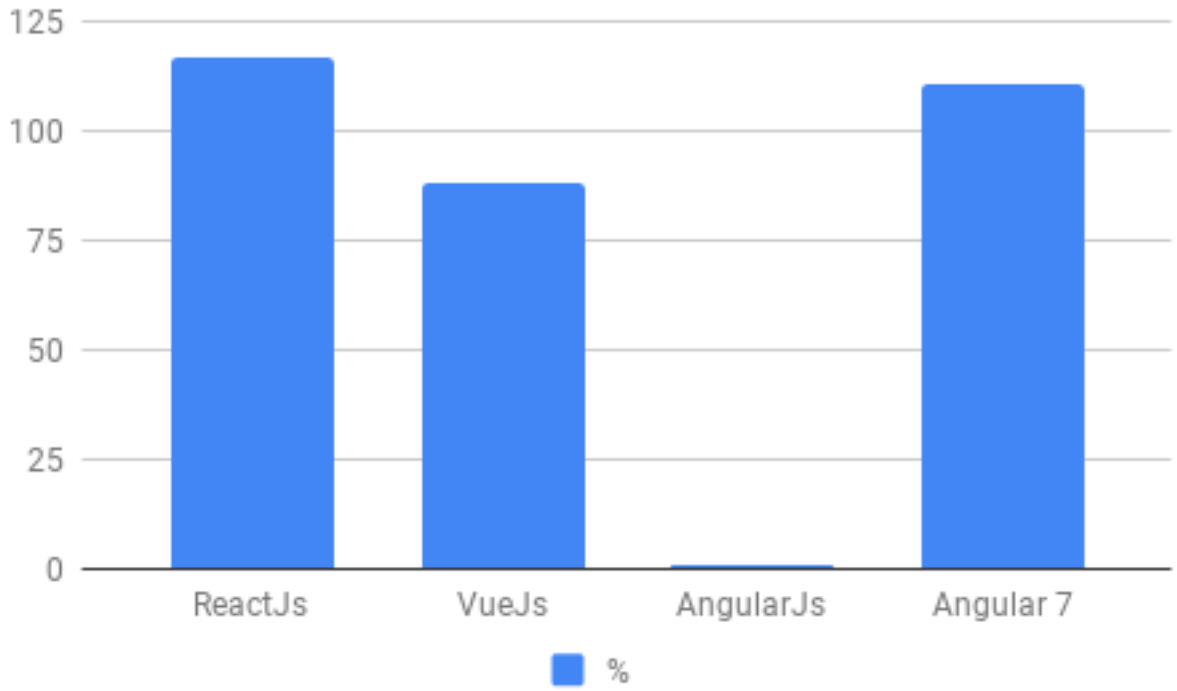
Figura 10 – Resultado dos testes para consumo de CPU.



Fonte – elaborada pelo autor

[CT04]

Figura 11 – Resultado dos testes para consumo de CPU.



Fonte – elaborada pelo autor

O gráfico mostra que, no **CT01** (criar 100 contatos), teve uma proximidade de consumo entre AngularJs e Angular, o ReactJs ficou com um valor próximo em consumo em relação aos dois já mencionados, e o VueJs que teve o menor consumo. No**CT02** (atualizar 100 contatos), o ReactJs teve o maior consumo, seguido do AngularJs, enquanto VueJs e Angular tiveram consumos menores, tendo o VueJs consumido menos que todos. Já no**CT03** (deletar 100 contatos), ReactJs continuou como maior consumidor, seguido por Angular e VueJs, enquanto o AngularJs teve o menor consumo neste caso. E por fim, no**CT04** (carregar a aplicação com todos contatos), o consumo foi um pouco idêntico ao caso de teste anterior para ReactJs, VueJs e Angular, já que o AngularJs teve um consumo muito inferior.

As Figuras 13, 14, 15, 16 mostram os resultados obtidos nos testes para obter os dados de tempo de execução. Foi levado em consideração o menor, maior e a média de tempo em cada caso de teste. Apenas no CT04 que foi necessário apenas o tempo de carregamento da aplicação. Para obter o tempo de execução de cada método, foi utilizado a função *console.time*. Um exemplo pode ser visto na figura 12.

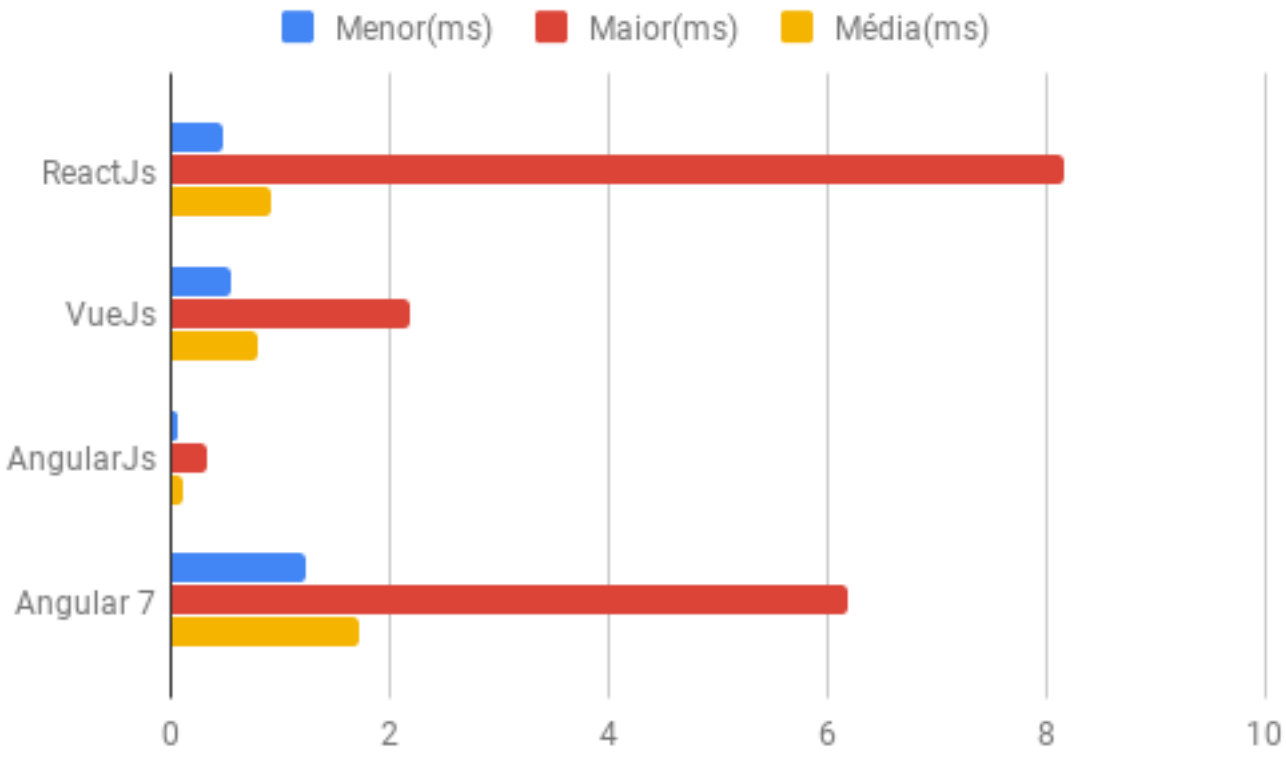
Figura 12 – Função console.time.

```
fetchData: function() {
    console.time("#carregarContatos");
    fetch('http://127.0.0.1:8000/api/', optionsget)
        .then(response => response.json())
        .then(json => this.data = json);
    console.timeEnd("#carregarContatos");
},
```

Fonte – Elaborada pelo autor

[CT01]

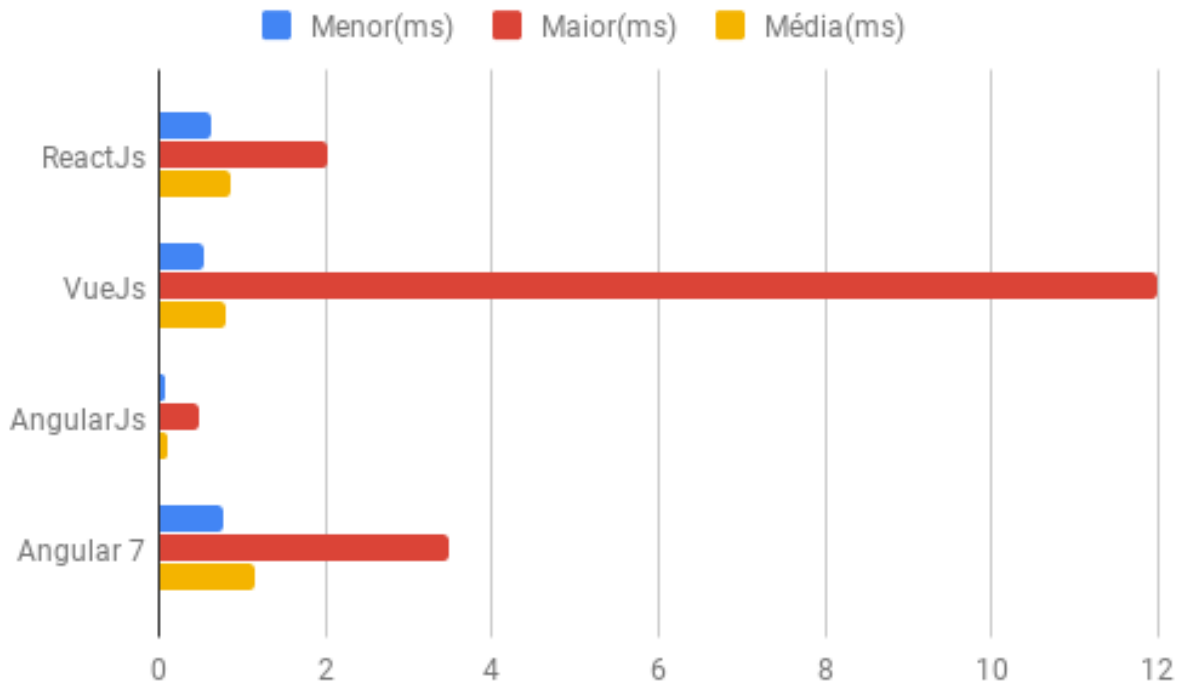
Figura 13 – Resultado dos testes para tempo de execução(ms).



Fonte – elaborada pelo autor

[CT02]

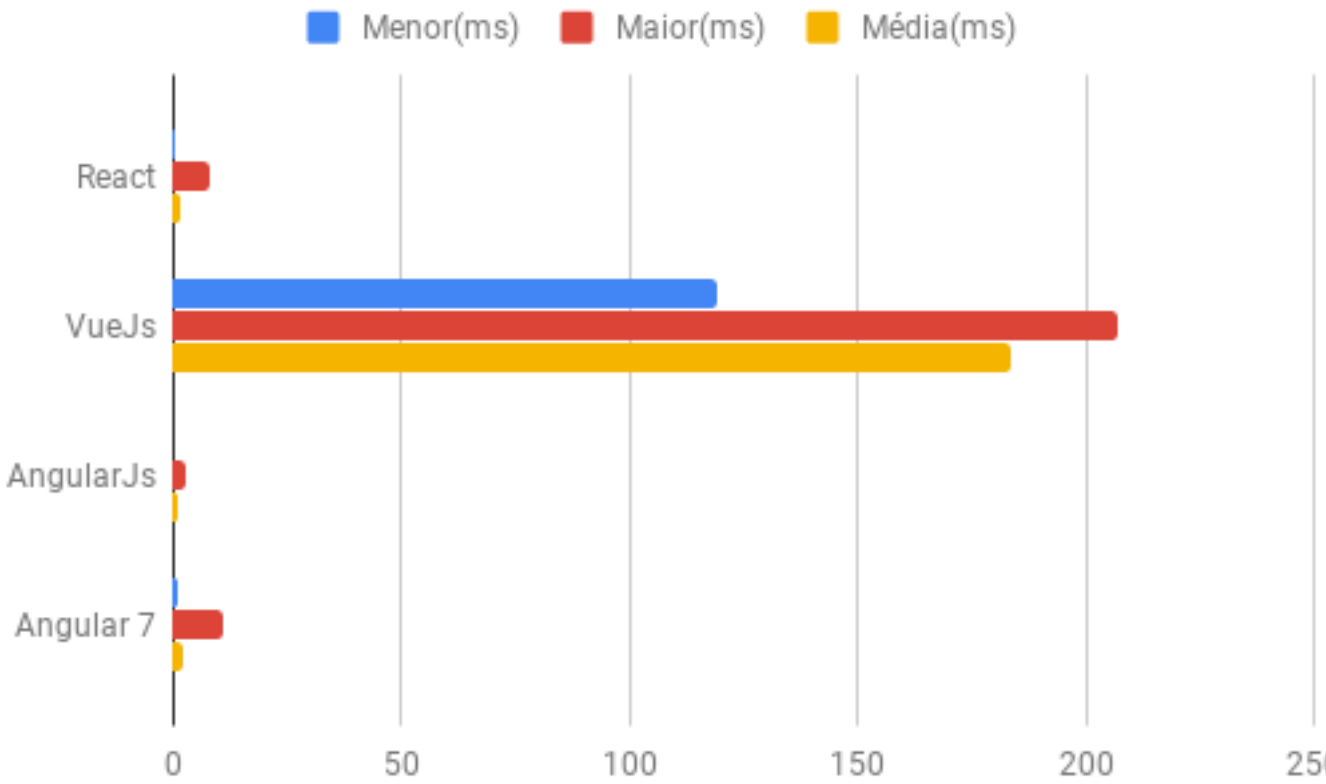
Figura 14 – Resultado dos testes para tempo de execução(ms).



Fonte – elaborada pelo autor

[CT03]

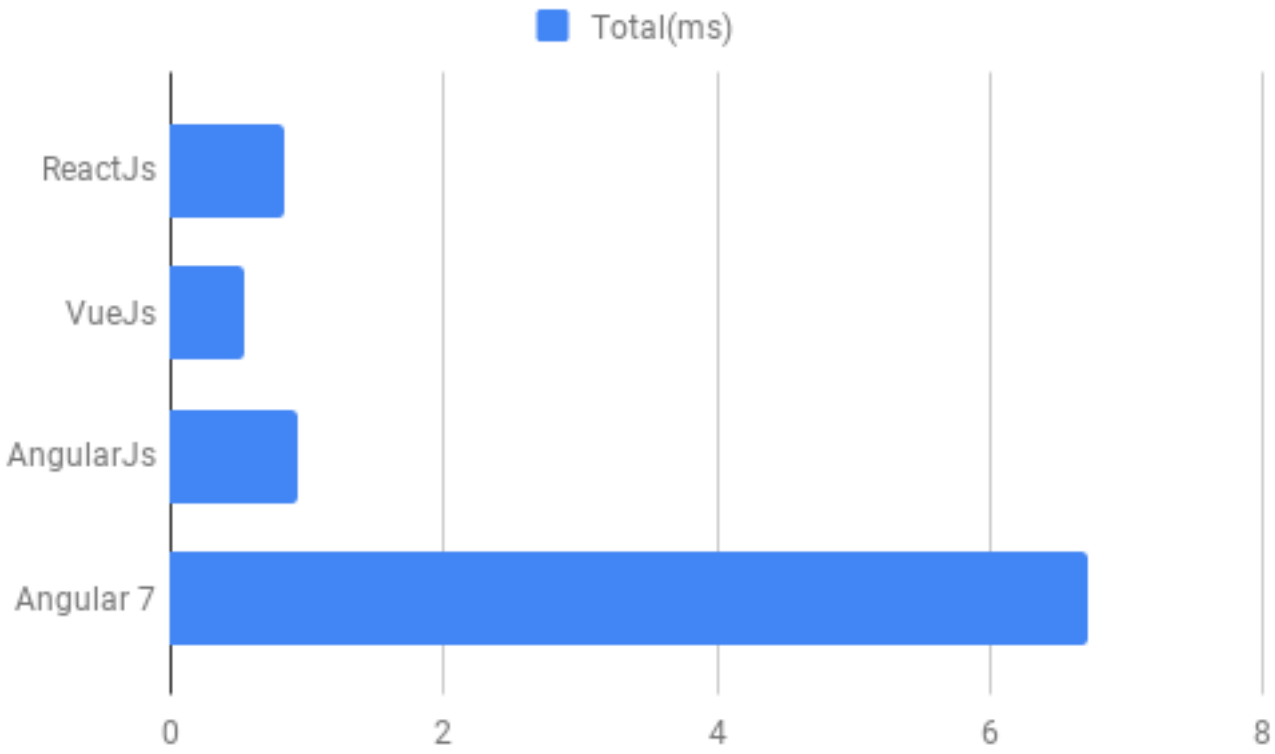
Figura 15 – Resultado dos testes para tempo de execução(ms).



Fonte – elaborada pelo autor

[CT04]

Figura 16 – Resultado dos testes para tempo de execução(ms).



Fonte – elaborada pelo autor

No **CT01**, levando em consideração o tempo menor, maior e média, pode-se observar que o ReactJs teve um pico de tempo de execução maior no geral, enquanto seu menor tempo e média ficaram um pouco parecidos com o do VueJs, que ficou atrás do AngularJs em desempenho de tempo, já o Angular teve o menor tempo de execução e a média maior que todos. O **CT02** mostra que o VueJs teve o maior pico de tempo de execução, mas com o seu menor e média de tempo relativamente baixas se comparados com os demais, enquanto que ReactJs e Angular tiveram seu menor tempo e média próximos, com a diferença que o Angular teve um pico de tempo maior, já o AngularJs teve os menores tempos de execução. O gráfico do **CT03** apresenta-se bem diferente, onde podemos ver que ReactJs, AngularJs e Angular tiverem tempos quase iguais, mas com o VueJs se apresentando com tempos maiores. E **CT04**, onde foi feito o carregamento dos contatos, o Angular apresentou o maior tempo, enquanto os demais ficaram com tempos bem parecidos.

O **CT05** (abrir 10 instâncias da aplicação sem contatos) demonstra como ficou o consumo de memória e CPU por cada um dos *frameworks*. Pode-se ver que o Angular teve um maior consumo, ReactJs ficou um pouco atrás, consumindo relativamente menos que o Angular,

já o VueJs e o AngularJs foram os que consumiram menos.

[ReactJs]

Figura 17 – Resultado do CT05.

Fonte – elaborada pelo autor

[VueJs]

Figura 18 – Resultado do CT05.

Fonte – elaborada pelo autor

[AngularJs]

Figura 19 – Resultado do CT05.

Fonte – elaborada pelo autor

[Angular]

Figura 20 – Resultado do CT05.

Fonte – elaborada pelo autor

No **CT06** (abrir 10 instâncias da aplicação com 100 contatos) não mudou muito se comparado ao CT05, ReactJs e Angular tiveram maiores consumos e AngularJs e VueJs com os menores consumos.

[ReactJs]

Figura 21 – Resultado do CT06.

Fonte – elaborada pelo autor

[VueJs]

Figura 22 – Resultado do CT06.

Fonte – elaborada pelo autor

[AngularJs]

Figura 23 – Resultado do CT06.

Fonte – elaborada pelo autor

[Angular]

Figura 24 – Resultado do CT06.

Fonte – elaborada pelo autor

5.2 Pontos Fortes e Fracos do Trabalho

Pontos Fortes

- Foram utilizados os *frameworks* mais populares neste trabalho;
- Foco do trabalho foi em desempenho dos *frameworks*, diferente dos trabalhos relacionados;
- Definição de casos de testes;
- Autor desenvolveu o projeto proposto.

Pontos Fracos

- Deveria ter desenvolvido um projeto teste com *Javascript* puro para fins de comparação;
- Não desenvolvido com as melhores práticas para cada *framework*;
- Número de contatos poderia ser maior ao invés de apenas cem.

5.3 Pontos Positivos e Negativos dos *Frameworks*

Para avaliar os Pontos Positivos e Negativos dos *frameworks*, foram definidos critérios como: **Documentação**, **Curva de Aprendizagem** e **Configuração**.

ReactJs

- **Documentação**: A Documentação é clara e objetiva, com bons exemplos de códigos, o que ajuda em um melhor atendimento do seu funcionamento;
- **Curva de Aprendizagem**: Com uma boa documentação e exemplos ricos em detalhes, a curva de aprendizagem é relativamente baixa;
- **Configuração**: Sua configuração básica e que foi utilizada no projeto, consistiu de adicionar o link da biblioteca na página do projeto.

VueJs

- **Documentação**: A Documentação do Vue também é clara, com bons exemplos e com a opção em português;
- **Curva de Aprendizagem**: A Curva de arendizagem é relativamente baixa, pois com pouco conhecimento em *Javascript* é possível trabalhar com ele;
- **Configuração**: Assim como no ReactJs, sua configuração básica foi adicionar o link da biblioteca na página do projeto;

AngularJs

- **Documentação**: Ótima documentação, bastante rica em detalhes e exemplos de código para os iniciantes;
- **Curva de Aprendizagem**: Curva de aprendizagem também é baixa, bastando saber també o mínimo de *Javascript* para poder começar a trabalhar com o mesmo;
- **Configuração**: Assim como no ReactJs e VueJs, sua configuração mais básica e usada no projeto foi adicionar o link da lib na página do projeto.

Angular

- **Documentação**: Documentação relativamente boa, mas poderia ser melhor, mais rica em exemplos, com mais detalhes para um melhor entendimento;
- **Curva de Aprendizagem**: Diferente de sua primeira versão, o Angular trabalha com *TypeScript*, que é uma forma de escrever *Javascript* e isso torna sua curva de aprendizagem um pouco maior;
- **Configuração**: Sua configuração pode parecer uma pouco complicada para iniciantes, pois é preciso instalar uma interface de comando para poder gerar um projeto que contém vários arquivos que podem confundir os inicantes.

Os *frameworks* avaliados são uma ótima opção para a construção de aplicações Web. Cada um mostrou-se capaz de responder bem ao seu propósito, de forma que mesmo com poucos conhecimentos sobre o *framework*, qualquer desenvolvedor possa criar seu projeto em pouco tempo, de maneira fácil e eficiente. O ReactJs, VueJs e AngularJs podem se sair melhor em projetos que exijam um bom desempenho, já que foram esses três que obtiveram melhores resultados nos testes. O Angular, apesar de ficar atrás nos testes, não deixa de ser uma boa escolha para aqueles com mais conhecimentos sobre o mesmo. Vale ressaltar que o projeto teste é uma pequena aplicação, e que apesar dos resultados mostrados, os *frameworks* talvez tenham um desempenho melhor em aplicações maiores e mais complexas, principalmente o Angular, que ficou atrás na questão de desempenho. Para uso em aplicações pequenas, como o projeto teste, o VueJs talvez seja o mais recomendado.

6 CONSIDERAÇÕES FINAIS

Com a popularização da internet e dos sistemas que rodam sob a mesma, tornou-se necessário diminuir o tempo de desenvolvimento de novas aplicações, devido a este surgiram os *frameworks*, que vieram para suprir essa necessidade.

O presente trabalho visou a comparação entre *frameworks Javascript* mais populares para a construção de aplicações *front-end*. Foi feito uma pesquisa para descobrir os *frameworks* mais populares, em seguida definido o projeto para testes e então realizado o *benchmark*.

Os resultados mostraram-se muito satisfatórios, com poucas diferenças entre os *frameworks* no geral. O VueJs e AngularJs mostram-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.

No contexto que foi aplicado os testes, não teve um projeto feito com *Javascript* puro para comparar os resultado e avaliar melhor a questão do desempenho dos *frameworks* escolhidos. Devido a isto, fica como trabalho futuro o desenvolvimento de um projeto com *Javascript* puro para melhor análise de resultados e utilizar outras formas de *benchmark*.

REFERÊNCIAS

ALVIM, P. **TIRANDO O MÁXIMO DO JAVA EE 5 OPEN-SOURCE com jCompany e Developer Suite**. [S.l.]: Powerlogic, 2008.

AMARAL, R. A. do; NERIS, V. P. de A. Análise comparativa entre frameworks de frontend para aplicações web ricas visando reaproveitamento do back-end. **Revista TIS**, v. 4, n. 1, 2016.

BOUCKAERT, S.; GERWEN, J.; MOERMAN, I.; PHILLIPS, S. C.; WILANDER, J. Benchmarking computers and computer networks. **EU FIRE White Paper**, 2010.

CITRUS7. **O QUE É FRONT-END E BACK-END?** 2017. Disponível em: https://citrus7.com.br/artigo/o-que-e-front-end-e-back-end/. Acesso em: 22 jun. 2017.

COUTO, R. S.; FOSCHINI, I. J. Análise comparativa entre dois frameworks mvc para a plataforma java ee: Jsf e vraptor. **Revista TIS**, v. 4, n. 3, 2016.

DUARTE, N. F. B. **Frameworks e Bibliotecas Javascript**. Dissertação (Mestrado) - Curso de Engenharia Informática — Instituto Superior de Engenharia do Porto, Porto, 2015.

EIS, D. **O caminho das pedras para ser um dev Front-End**. [S.l.]: Casa do Código, 2015.

FAYAD, M.; SCHMIDT, D. C.; JOHNSON, R. E. **Building application frameworks: object-oriented foundations of framework design**. [S.l.]: John Wiley & Sons, 1999.

INCAU, C. **Vue.js: Construa aplicações incríveis.** [S.l.]: Casa do Código, 2017.

NICOL, G.; WOOD, L.; CHAMPION, M.; BYRNE, S. Document object model (dom) level 3 core specification. **W3C Working Draft**, v. 13, p. 1–146, 2001.

PINHO, D. Martins de. **ECMAScript 6: Entre de cabeça no futuro do JavaScript**. [S.l.]: Casa do Código, 2017.

PONTES, G. **Progressive Web Apps: Construa aplicações progressivas com React**. [S.l.]: Casa do Código, 2018.

PRUSTY, N. **Modern JavaScript Applications**. [S.l.]: Packt Publishing, 2016.

SCHMIDT, D. C.; STAL, M.; ROHNERT, H.; BUSCHMANN, F. **Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects**. [S.l.]: John Wiley & Sons, 2013. v. 2.

WIKIPéDIA. **Benchmark (computação)**. 2017. Disponível em: https://pt.wikipedia.org/wiki/Benchmark_(computa%C3%A7%C3%A3o). Acesso em: 05 jul. 2017.

WILLIAMSON, K. **Introdução ao AngularJS: Um guia para o desenvolvimento com o AngularJS**. [S.l.]: Novatec, 2015.