Estudo Comparativo de Tecnologias de Desenvolvimento front-end para Web

Angular, React e Vue

Matheus de Souza
Universidade do Vale do Itajaí
Itajaí, SC, Brasil
matheussouza@edu.univali.br

Eduardo Alves da Silva
Universidade do Vale do Itajaí
Itajaí, SC, Brasil
eas@univali.br

ABSTRACT

There are several JavaScript technologies intended to assist in the construction of web systems user interfaces. Choose the most suitable for a new project can be a difficult task. Three of these technologies have gained prominence: Angular, Vue and React. All focused on the front-end development of web applications. In order to facilitate the process of decision making about which technology is the most suitable in a new project, this work establishes a comparative study of the three most used JavaScript technologies currently and to highlight the advantages and disadvantages of each one. This work adopted performance, size and support for different browsers to carry out an experimental comparative study. An application was developed as a use case and replicated in each of the technologies, in order to analyze the development process and the results under the same set of tests. A software to perform the tests in an automated way was implemented to collect the performance results using the Google Chrome browser. It was possible to identify which technology is most suitable in each test scenario. For example, the Angular framework performed better in 8 out of 10 scenarios evaluated, despite having a longer startup time and build size of the application compared to React and Vue. It is estimated that Angular loads more information in the initialization process to make the state of the application "more prepared" for user interactions.

KEYWORDS

Javascript, Front-end, Angular, React, Vue

1 INTRODUÇÃO

A popularidade que a Internet adquiriu nos últimos 20 anos contribuiu para que as soluções de software, que antes somente existiam em ambientes Desktop, fossem cogitadas na web. Segundo Deitel [1], os usuários desejam aplicações que possam ser executadas de qualquer lugar. Já os programadores desejam portabilidade para que as aplicações sejam executadas em diferentes plataformas sem que necessitem modificações. Tais anseios podem ser atingidos de maneira satisfatória por meio de aplicações web.

Quando se pensa em uma linguagem de programação para web, logo se pensa em JavaScript. Essa associação ocorre porque grande parte do sucesso da web se dá em virtude dessa linguagem, pois como descreve Goodman [2], o Javascript é uma tecnologia para melhoria da web, podendo transformar uma página de conteúdo estático em uma experiência atraente, interativa e inteligente. Assim, a JavaScript é uma das linguagens atuais que possibilita a existência de interfaces de usuário na web tão completas quanto interfaces clássicas de Desktop.

Contudo, a linguagem JavaScript por si só pode não ser suficiente para atender todas as demandas do mercado de desenvolvimento web atual. É inviável que desenvolvedores comecem o trabalho do zero a cada vez que um projeto é iniciado. Para reduzir o retrabalho, bibliotecas de programação e frameworks são utilizados no processo de desenvolvimento de software.

Diariamente, torna-se mais comum o uso de tecnologias, frameworks e bibliotecas, especialmente no desenvolvimento front-end, ou seja, na parte da aplicação que fornece a interface ao usuário. Existem diversas opções disponíveis, as quais aumentam a cada dia e tornam o processo de escolha cada vez mais difícil.

Um dos grandes problemas na adoção errônea de uma tecnologia de programação pode ser o custo de desenvolvimento ou retrabalho de uma equipe na atualização de um sistema, o qual costuma ser descoberto tardiamente em um projeto. Além disso, o número de tecnologias JavaScript para front-end aumentou e aumenta consideravelmente com o passar do tempo.

Para facilitar a escolha do projetista de software, um estudo comparativo que evidencie o desempenho e utilização de recursos computacionais das principais tecnologias front-end para JavaScript foi feito. Segundo Hotframeworks [3], as tecnologias JavaScript para criação de interfaces de usuário mais utilizadas são AngularJS, React, Angular e Vue, respectivamente. Portanto, uma mesma aplicação web foi implementada com as três tecnologias mais populares: Angular, React e Vue, e um conjunto de testes foi relizado. O AngularJS foi desconsiderado no estudo, pois é uma versão mais antiga do Angular e tende a ser menos expressiva com o passar do tempo.

2 REFERENCIAL TEÓRICO

Os principais assuntos relacionados a este trabalho são apresentados nesta seção. Inicialmente, uma contextualização sobre a evolução das aplicações web é descrita para posicionar o leitor e destacar a importância do JavaScript nesse tipo de plataforma. Na sequência, a linguagem JavaScript é brevemente apresentada e o principal motivo de sua ampla adoção em aplicações web é explanado. Posteriormente, Aplicações de Página Única (SPA - Single Page Aplication) são abordadas, pois são adotadas amplamente no desenvolvimento de aplicações web no presente momento. Por fim, as tecnologias alvo deste estudo são caracterizadas, Angular, React e Vue.

2.1 Aplicações Web

Segundo Conallen[4], as aplicações web se originam de sites simples. Antigamente, os sites se limitavam a um sistema de hipermídia distribuído que permitia que documentos fossem acessados por

meio de um navegador web. Esse sistema é chamado de hipermídia, pois os documentos podem estar interligados, o que permite que o usuário navegue entre documentos através de links.

De acordo com Flanagan[5], uma página web simples é basicamente construída a partir de três tecnologias: HTML (HyperText Markup Language – Linguagem de Marcação de Hipertexto), CSS (Cascading Style Sheets – Folhas de estilo em cascata) e JavaScript. Usa-se o HTML para especificar o conteúdo da página. Para especificar a apresentação dessas páginas, utiliza-se o CSS. E, para definir o comportamento delas, o JavaScript é utilizado.

Na década de 90, as páginas web limitavam-se a conteúdos HTML estáticos. A falta de interatividade das páginas web proporcionava uma pobre experiência para os usuários e era uma das principais justificativas delas não evoluírem para aplicações superiores as de Desktop.

Uma aplicação web é desenvolvida e estendida a partir de um sistema web para adicionar funcionalidade de negócio [4]. Logo, o que diferencia uma aplicação web de um simples site é que a interação do usuário com o sistema afeta o estado do negócio, ou seja, contribui para a solução de um problema.

Uma das principais tecnologias que contribuiu para transformar páginas web estáticas em aplicações web robustas foi a linguagem JavaScript.

2.2 Javascript

De acordo com a W3C[6], JavaScript é a principal linguagem de programação presente na web para manipulação de informação dentro do navegador. Ela é responsável por resolver o problema de pouca ou nenhuma interatividade que existia nas aplicações web do passado. Antes do seu surgimento, por mais que existissem tecnologias para geração de páginas dinâmicas no back-end, elas possuíam diversas limitações.

A JavaScript foi qualificada por Flanagan [5] como: uma linguagem de alto nível, dinâmica, interpretada e não tipada, conveniente para estilos de programação orientados a objetos e funcionais. Ela é definida como de alto nível por trabalhar diretamente com o conteúdo do navegador. Dinâmica, basicamente porque os tipos de suas variáveis não precisam ser definidos no início do programa. Interpretada, pois não possui uma etapa de compilação (o código é avaliado em tempo de execução). E não tipada, pois as variáveis são fracamente tipadas.

A JavaScript, como o nome sugere, é uma linguagem de scripting. Uma linguagem de scripting é comumente definida como uma linguagem de programação que permite ao programador controlar uma ou mais aplicações de terceiros [7]. Ou seja, a linguagem JavaScript foi feita para controlar e modificar alguns comportamentos contidos nos navegadores web.

Hoje em dia, um dos grandes usos do JavaScript na elaboração de páginas e sistemas web está relacionado a interfaces mais dinâmicas e responsivas com interações diretas com os navegadores. Para obter esse resultado, empresas, times de desenvolvimento, programadores e comunidade têm focado em prover uma experiência de usuário agradável e as aplicações de página única vem sendo amplamente adotadas para atingir este objetivo.

2.3 Aplicações de Página Única

Segundo Wasson[8], aplicações de página única (SPA - Single Page Application) são aplicações web que carregam uma única página HTML no front-end e permanecem atualizando esta página conforme a interação do usuário. As aplicações de página única utilizam Ajax para criar interações fluídas que não recarregam constantemente. É o Ajax que possibilita atualizações incrementais de conteúdo na interface de uma aplicação web sem necessidade de recarregar uma página inteira [9]. No entanto, isso significa que boa parte do trabalho é carregado para o lado do cliente (no navegador), onde os dados são processados em JavaScript.

Quando o usuário muda de página em uma SPA, na verdade a página não é alterada, mas sim seu conteúdo é substituído por um novo conteúdo proveniente de funções JavaScript. Basicamente, uma aplicação de página única carrega toda a aplicação na memória durante a primeira requisição, e posteriormente, beneficia-se do Ajax para carregar os dados necessários dinamicamente.

As aplicações web que foram desenvolvidas nesse trabalho utilizaram o padrão SPA.

2.4 Tecnologias JavaScript para front-end

Ao utilizar a linguagem JavaScript no desenvolvimento da parte cliente de aplicações web, o uso de tecnologias que acelerem e facilitem o processo de desenvolvimento é indispensável. Dentre as opções disponíveis no mercado, destacam-se Angular, React e Vue. O termo tecnologia foi utilizado neste trabalho para abstrair as definições usadas ao se referir aos frameworks (Angular e Vue) ou bibliotecas (React) de forma geral.

2.4.1 Angular. O Angular é um framework JavaScript desenvolvido pela Google e que tem por objetivo desenvolvimento de aplicações multiplataforma, isto é, que são capazes de funcionar tanto em dispositivos móveis quanto em aparelhos desktop. O Angular foi um dos primeiros frameworks deste tipo a se tornar popular. Sua primeira versão, conhecida por AngularJS, é, ainda hoje, a mais utilizada no mundo [3], ocupando a segunda posição no ranking, atrás somente do React. Em contrapartida, as versões atuais possuem recursos mais modernos, o que possibilita um desempenho final mais eficiente.

O Angular é desenvolvido em Typescript, mas permite a codificação em Javascript, Dart e em Typescript. Conforme descrevem Faria e Afonso [10], os principais elementos do Angular são componentes, templates, diretivas, roteamento, módulos, serviços, injeção de dependências e ferramentas de infraestrutura que automatizam tarefas, como a de executar os testes unitários de uma aplicação.

2.4.2 React. O React é uma biblioteca JavaScript da empresa - Facebook. Seu foco é facilitar o processo de criação de interfaces de usuário [11]. Atualmente é uma das tecnologias JavaScript mais populares. De acordo com Rambeau, Benitte e Greif.[12], 64,8% dos desenvolvedores já o utilizaram e voltariam a utilizar a biblioteca.

Hunt et al. [13] afirmam que a React consegue essencialmente abstrair o DOM (Document Object Model - Modelo de Documento por Objetos), simplificando assim, o modelo de programação, enquanto melhora o desempenho. Isso é possível por meio da utilização do DOM "virtual" ou VDOM. O VDOM é definido como um

conceito de programação onde uma representação ideal, ou "virtual", da interface do usuário é mantida em memória e sincronizada com o DOM 'real' [11].

2.4.3 Vue. You[14] descreve o Vue como um framework progressivo para construção de interfaces de usuário. Este framework é focado em componentização, ou seja, a aplicação final é formada por pequenos componentes que são incrementados. Ele possui somente o essencial para funcionar no seu núcleo, deixa a cargo do seu utilizador, a responsabilidade de adicionar mais bibliotecas conforme for necessário. Ele também é capaz de ser usado para desenvolver sofisticadas aplicações de página única.

3 DESENVOLVIMENTO

A proposta do trabalho foi explorar alguns dos fatores que são considerados na escolha de uma tecnologia JavaScript para front-end, com base no estudo realizado por Pano e Graziotin[15]. O intuito é evidenciar o comportamento de cada uma das tecnologias com uma aplicação prática (real). Para esse fim, cenários de verificação identificados no trabalho de Voutilainen [16] foram utilizados como base e foram desenvolvidos em cada tecnologia. Desse modo, foi possível perceber as diferenças de cada tecnologia durante o desenvolvimento e o respectivo impacto no resultado.

Nem todos os fatores dos trabalhos relacionados foram adotados. Optou-se por focar no comparativo do desempenho das tecnologias, por ser um critério quase sempre considerado no desenvolvimento de aplicações. Ainda assim, outros fatores são apontados como desejáveis, devido suas importâncias.

3.1 Fatores escolhidos para a comparação

Existem diversos fatores que podem ser considerados na hora de se escolher uma tecnologia JavaScript. Foram analisados os 15 fatores levantados no trabalho de Pano e Grazziotin[15]. Desse conjunto, os fatores testados com as tecnologias foram: desempenho, tamanho e suporte aos diferentes navegadores.

3.1.1 Desempenho. Para medir o desempenho, uma aplicação idêntica foi desenvolvida com as três tecnologias. O objetivo foi servir de base para a coleta de dados na avaliação. A aplicação consiste em uma SPA que permite ao usuário realizar operações em uma tabela (ex. inserir várias linhas de uma vez). Desse modo, o desempenho individual de cada tecnologia pode ser verificado, uma vez que, trabalhar com grandes quantidades de elementos no DOM pode ser uma tarefa pesada e dependendo da forma de trabalho de cada um, as diferenças nos resultados podem ser perceptíveis.

Os seguintes cenários de teste foram adotados com o intuito de coletar métricas de tempo de execução e memória utilizada ao manipular o DOM, assim como coletar o tempo de inicialização da aplicação:

- `Inserção de linhas`: Tempo dispendido na inserção de 1.000 linhas depois que a página iniciou;
- `Substituir todas as linhas`: Tempo dispendido na exclusão de 1.000 linhas e, após, reinserção de novas 1.000 linhas;
- `Atualização parcial`: Tempo dispendido para atualizar o texto de uma linha em cada dez em uma tabela com 10.000 linhas;
- `Selecionar uma linha`: Tempo dispendido para destacar uma linha em resposta ao clique do usuário com grande volume de informações na tabela;
- `Trocar linhas de posição`: Tempo dispendido para trocar duas linhas de posição, em uma tabela com 1.000 linhas;
- `Remover linha`: Tempo dispendido para remover uma linha localizada na metade da tabela em uma tabela com 10.000 linhas;
- `Criar várias linhas`: Tempo dispendido para adicionar 10.000 linhas;
- `Adicionar linhas em uma tabela grande`: Tempo dispendido para adicionar 1.000 linhas de uma vez em uma tabela de 10.000 linhas;
- `Limpar linhas`: Tempo dispendido para limpar uma tabela com 10.000 linhas;
- `Ordenar a tabela`: Tempo dispendido para realizar a ordenação de dados de forma crescente em uma tabela com 10.000 linhas.
- `Memória utilizada`: Memória utilizada em todos os testes listados acima;
- `Tempo de inicialização`: Tempo que o navegador leva para carregar a página, analisar o código JavaScript e renderizar a página.

3.1.2 Tamanho. Com base nas aplicações desenvolvidas para testar o desempenho, duas métricas serão extraídas em relação ao tamanho:

- `Tamanho da aplicação` comprimida, após processo de build;
- `Quantidade de linhas` necessárias para realizar a mesma tarefa em cada uma das tecnologias.

3.1.3 Suporte à diferente navegadores. O suporte das tecnologias em diferentes navegadores foi mandatório. A aplicação desenvolvida deve rodar, sem nenhum erro no console, nas versões mais recentes dos navegadores Google Chrome (versão 88), Mozilla Firefox (versão 86) e Microsoft Edge (versão 89).

3.2 Aplicação implementada

O desenvolvimento iniciou com a implementação das operações no Vue e, posteriormente, no Angular e React. Dessa forma, foi possível focar em desenvolver uma aplicação para atender todos requisitos estabelecidos no projeto (conjunto de operações e compatibilidade) e decidir exatamente como deveria ser o comportamento de cada funcionalidade. A codificação foi feita nas outras duas tecnologias replicando o mesmo comportamento.

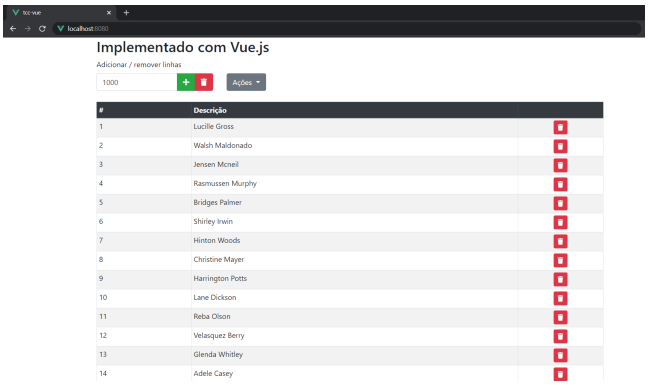
Para se aproximar de uma aplicação real, utilizou-se da biblioteca Bootstrap versão 4[1] para estilizar a página. O resultado do layout definido é apresentado na Figura 1.

As únicas diferenças na implementação de uma tecnologia para a outra no layout apresentado na Figura 1 são: (i) o título da página; (ii) ícone da página; e (iii) nome da tecnologia após o texto de cabeçalho "Implementado com".

Os dados populados na tabela são oriundos de uma lista com cem nomes aleatórios geradas por ferramenta web [2] para compor

[1]Bootstrap é uma biblioteca front-end que tem como objetivo facilitar a criação e estilização de interfaces web
[2]http://listofrandomnames.com/

Figura 1: Layout da aplicação desenvolvida.



a coluna "descrição". Quando uma linha é adicionada, obtém-se um nome de forma aleatória nessa lista, portanto, há repetição de nomes.

Com relação às operações de ordenação da tabela, a função padrão de ordenação do JavaScript foi utilizada, a qual executa o algoritmo de ordenação definido pelo navegador utilizado. No caso do Google Chrome, o algoritmo utilizado é o QuickSort para vetores maiores que 10 elementos.

Para a funcionalidade de troca de linhas, troca-se a primeira com a última linha. Essa escolha é baseada no trabalho relacionado de Krause [17].

Para identificar a conclusão de uma operação, um feedback no estilo toast[3] foi inicialmente implementado. No entanto, com a necessidade de extrair informações de eventos específicos de renderização relacionados à pintura da tabela nos testes, identificou-se que o feedback visual interferia diretamente nos resultados pela temporização utilizada para a sua exibição. Logo, o feedback visual foi removido e o usuário não é informado sobre o término das operações solicitadas.

Com a implementação na primeira tecnologia, ao replicar nas outras duas foi possível comparar a forma com que cada funcionalidade pode ser implementada. O comparativo é apresentado na próxima seção.

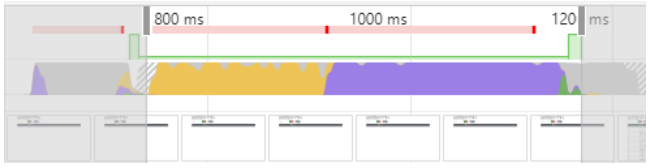
4 EXPERIMENTOS E RESULTADOS

Após o desenvolvimento das aplicações, os testes de desempenho foram realizados com suporte da ferramenta de análise do Google Chrome. Trata-se da ferramenta *DevTools*, acessível pelo console de desenvolvimento do navegador.

Na aba performance (desempenho) da DevTools, há a opção de iniciar uma gravação de desempenho sobre o conjunto de atividades realizadas no uso do navegador. Durante a gravação, o navegador registra tudo o que a aplicação realiza, todas as chamadas de funções, eventos, renderização de elementos na tela, entre outros processos. Quando a gravação é encerrada, um relatório com indicadores dos processos avaliados e quanto tempo cada um durou é apresentado ao usuário.

[3]Mensagem que é exibida com pequena duração, 2 a 5 segundos sobre algum conteúdo da página

Figura 2: Linha do tempo de desempenho, gerada ao adicionar mil linhas em uma tabela utilizando Angular



A Figura 2 representa a linha do tempo, na qual são listados todos os eventos ocorridos na ordem que foram executados, durante a gravação de desempenho. A área demarcada se refere aos momentos de início e fim da operação de inserir linhas na tabela. A operação começa com um clique do usuário no botão para adicionar as linhas e termina com a pintura/renderização da tabela na tela/navegador.

O principal objetivo de demarcar a linha do tempo é definir o intervalo de análise. Ou seja, identificar o tempo que o usuário espera para ter um feedback visual a partir da sua ação na aplicação. Somente quando o evento de pintura é concluído que o usuário tem a resposta visual.

O gráfico da Figura 2 está dividido em três áreas, diferenciadas por cores. A primeira área, em amarelo, representa o tempo em que o evento de clique foi manipulado (executou os scripts necessários para realizar a operação). Logo após, é apresentada uma área em roxo que representa a etapa de renderização dos elementos na tela. Por último, uma pequena área em verde, a qual representa o momento em que as linhas da tabela estavam sendo pintadas na tela.

4.1 Software para automação dos testes

Visando automatizar o processo de testes e não depender das interações do usuário com a aplicação, o processo de captura de eventos foi automatizado no Google Chrome. A biblioteca Selenium Webdriver foi adotada para facilitar as chamadas à API (Application Programming Interface - Interface de Programação de Aplicação) do Chromedriver.

A biblioteca Selenium Webdriver [18] é de código-fonte aberto e permite que os eventos de desempenho gerados pelo navegador sejam coletados. Essa biblioteca reúne uma série de métodos que permitem interações com o navegador programaticamente. Por exemplo, a biblioteca permite navegar para uma página específica ou procurar um botão na tela e disparar um clique. A biblioteca Selenium Webdriver foi utilizada em conjunto com o framework de testes para JavaScript Mocha [19]. Isso porque o uso dos dois permite o enfileiramento de testes para a execução sequencial em uma única ativação.

4.2 Resultados obtidos

Nesta seção, os resultados obtidos com a execução das operações predefinidas são apresentados. As análises realizadas para cada operação são tempo de execução e memória utilizada. Alguns comportamentos foram mais explorados em busca de identificar uma tendência, como o caso da operação de adicionar quantidades cada vez maiores de linhas na tabela e ordenar a tabela com diferentes

quantidades de linhas. Por fim, o tempo de inicialização da aplicação e o tamanho do build são comparados entre as tecnologias.

4.2.1 Tempo de execução. Na Tabela 1 os resultados do tempo de execução nas 10 (dez) operações testadas são apresentados. O tempo absoluto que cada operação levou em cada tecnologia é mostrado em milissegundos (ms) na Tabela 1. Além disso, o percentual de quão pior foi o tempo de uma tecnologia em relação a que obteve o melhor desempenho (realizou a operação em menor tempo) também é apresentado na mesma tabela.

Destaca-se que dos 10 cenários de teste (operações analisadas), o Angular apresentou menor tempo em 8 (oito), ou seja, foi superior em desempenho em 80% dos cenários. Entretanto, nas operações 7 e 9, o Angular apresentou o pior tempo para realizar as operações e com uma diferença significativa em relação ao React e Vue. Por significativo, entende-se perceptível ao usuário pela diferença maior que 1 (um) segundo.

No teste 7, o Angular teve um desempenho cerca de 18 vezes pior que o React, o mais rápido neste cenário. Já no teste 9, o Angular foi quase três vezes mais lento que o Vue que dispendeu o menor tempo na operação. Pode-se observar que nos testes 4 e 5, que envolvem realçar uma linha em reposta ao click do usuário, o Angular teve um desempenho muito melhor do que as outras duas tecnologias. Para o teste 4, o Vue foi 20,5 vezes mais lento que o Angular, e o React foi 14,55 vezes mais devagar. Já no teste 5, o Vue e o React tiveram tempos semelhantes, porém foram, quase 14 e 12 vezes mais lentos que o Angular, respectivamente.

Destaca-se também o teste 10, onde o React teve o pior resultado, cerca de 22 vezes pior que o Angular e o Vue em torno de 8 vezes pior. No geral, o Angular teve os melhores tempos nos testes individuais e o React e o Vue apresentaram tempos de execução mais próximos um do outro. Isso pode ter ocorrido devido a forma com que o Angular observa as mudanças do modelo e pela estratégia que ele utiliza internamente para manipular o DOM.

4.2.2 Memória utilizada. Nos testes de memória utilizada, um comportamento similar aos testes de tempo de execução foi observado, com o Angular tendo os melhores resultados na maioria dos testes e o React e Vue apresentando resultados próximos. Isso pode ser visto na Tabela 2.

Verificou-se no somatório total de ocupação de memória das 10 operações que o Angular foi a tecnologia que menos utilizou memória, seguido pelo React, que acabou utilizando 53% a mais de memória que o primeiro. O Vue foi a tecnologia com a maior utilização de memória, cerca de 78% a mais que o Angular na visão geral (soma dos tempos das 10 operações).

No segundo teste, o Angular e o React utilizaram praticamente a mesma quantidade de memória, mas o Vue acabou utilizando quase o dobro. Isto pode estar relacionado à maneira como o Vue coloca os objetos em memória antes de atualizar o DOM.

No quinto teste, as três tecnologias utilizaram uma quantidade de memória muito próxima. Aparentemente, há uma possível estrutura de cache em comum nas tecnologias que fez com que um clique na primeira linha da tabela não apresentasse diferença significativa no uso de recurso computacional.

Fazendo uma relação entre os resultados obtidos no teste de tempo de execução e memória utilizada, observou-se que no teste 7, o Angular obteve o pior resultado nos dois cenários. Já no teste 9, Vue acabou usando mais memória que os demais, porém, obteve o melhor tempo de execução. Algo parecido ocorreu com o Angular no primeiro teste.

Na busca por compreender o comportamento de cada tecnologia com diferentes volumes de dados, dois cenários foram explorados em maior ênfase, a adição e a ordenação da tabela com diferentes quantidades de linhas. Esses dois cenários são apresentados na sequência e representam a tendência de desempenho de cada tecnologia.

4.2.3 Progressão na adição de linhas. Com o intuito de verificar se há linearidade no comportamento das tecnologias conforme o aumento no número de linhas adicionadas em uma tabela vazia, testes de inclusão de mil à cem mil linhas foram realizados. O resultado obtido é apresentado no gráfico da Figura 3.

Figura 3: Progressão na adição de linhas

Observando o gráfico da Figura 3, identifica-se que tanto o Angular quanto o Vue apresentam um crescimento linear com o aumento no número de linhas sendo adicionadas na tabela. No entanto, o React apresentou um crescimento com tendência exponencial. Analisando um intervalo menor, é possível verificar que a partir de 40 mil linhas, o React começa a apresentar uma disparidade mais alta no tempo de execução.

Enquanto o Angular e o Vue mantiveram um aumento de aproximadamente três segundos conforme o número de linhas aumentou, o tempo do React aumentou por volta de 5 segundos de 30 mil para 40 mil linhas, e em torno de 6 segundos de 40 mil para 50 mil linhas. Esse tempo continuou aumentando até chegar em 1 minuto e 8 segundos com 100 mil linhas adicionadas, enquanto o Angular e o Vue alcançaram dispenderam 18 segundos e 29 segundos, respectivamente. Isso representa uma diferença significativa de tempo para o usuário e pode ser um fator decisivo na escolha da tecnologia.

4.2.4 Progressão ao ordenar uma tabela. Presumindo que a ordenação de uma tabela seja uma tarefa comum em uma aplicação e o Angular apresente um desempenho quase três vezes pior que a tecnologia mais rápida nessa operação, decidiu-se aprofundar a investigação. Para isso, foram realizados os testes ordenando uma tabela com mil linhas até 10 mil linhas, aumentando mil linhas a cada teste. Foi escolhido um intervalo menor pois a ordenação

Tabela 1: Comparação do tempo de execução entre todas as tecnologias

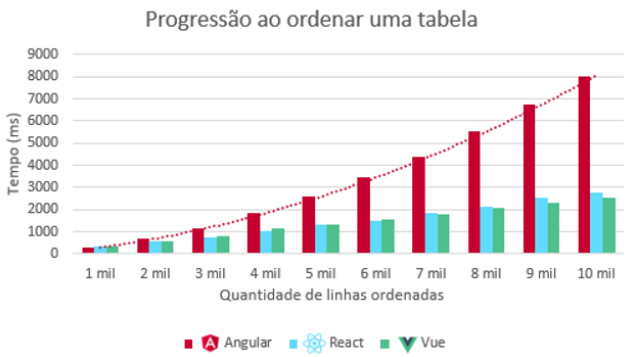
Ação	Angular		React		Vue	
	ms	%	ms	%	ms	%
1. Adicionar 1.000 linhas em uma tabela vazia	312,47	0	419,64	34	448,11	43
2. Adicionar 10.000 linhas em uma tabela vazia	2129,09	0	3383,51	59	3345,84	57
3. Adicionar 1.000 linhas em uma tabela com 10.000 linhas	315,39	0	817,70	159	1198,41	280
4. Clicar na linha 5.000 em uma tabela com 10.000 registros	37,49	0	583,04	1455	805,85	2050
5. Clicar na primeira linha de uma tabela com 1.000 linhas	9,36	0	121,34	1197	137,90	1374
6. Excluir a linha 5.000 em uma tabela com 10.000 registros	702,85	0	1227,87	75	1432,71	104
7. Limpar tabela com 10.000 linhas	4273,17	1853	218,77	0	251,45	15
8. Realizar atualização parcial em uma tabela com 10.000 registros	597,29	0	1105,87	85	1433,65	140
9. Realizar ordenação crescente em uma tabela com 10.000 registros	7727,14	198	2739,84	6	2594,69	0
10. Trocar linhas em uma tabela com 10.000 registros	105,69	0	2440,54	2209	998,57	845
Total	16209,9269	28	13058,1171	3	12647,1792	0

Tabela 2: Comparação da memória utilizada entre todas as tecnologias

Ação	Angular		React		Vue	
	MB	%	MB	%	MB	%
1. Adicionar 1.000 linhas em uma tabela vazia	6,92	30	5,31	0	7,20	36
2. Adicionar 10.000 linhas em uma tabela vazia	22,64	0	22,66	0	42,98	90
3. Adicionar 1.000 linhas em uma tabela com 10.000 linhas	24,42	0	45,28	85	46,71	91
4. Clicar na linha 5.000 em uma tabela com 10.000 registros	23,80	0	39,23	65	44,24	86
5. Clicar na primeira linha de uma tabela com 1.000 linhas	7,51	0	7,53	0	7,90	5
6. Excluir a linha 5.000 em uma tabela com 10.000 registros	23,84	0	39,48	66	44,33	86
7. Limpar tabela com 10.000 linhas	6,75	71	5,42	37	3,95	0
8. Realizar atualização parcial em uma tabela com 10.000 registros	23,24	0	38,66	66	43,61	88
9. Realizar ordenação crescente em uma tabela com 10.000 registros	21,65	0	38,88	80	43,60	101
10. Trocar linhas em uma tabela com 10.000 registros	23,18	0	38,59	66	43,59	88
Total	183,9542	0	281,0384	53	328,0991	78

das tabelas passa a ser muito custosa com mais de dez mil linhas. Ressalta-se que antes de ordenar a tabela, foram adicionadas linhas com elementos em uma ordem aleatória e cada teste foi executado dez vezes, obtendo o gráfico da Figura 4.

Figura 4: Progressão ao ordenar uma tabela

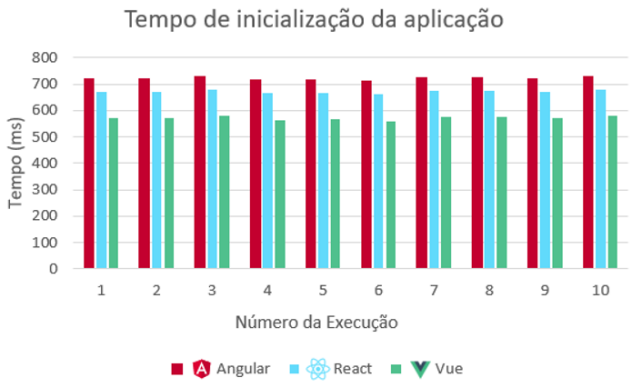


Analisando o gráfico da Figura 4, verifica-se que o React e o Vue tiveram um crescimento linear no tempo e não passaram dos três segundos para ordenar a tabela com até 10 mil linhas. O Vue experimentou menor tempo com maior número de linhas. Quanto ao Angular, dispendeu mais tempo do que os outros dois a partir de duas mil linhas. A partir de 2 mil linhas, o tempo do Angular foi cada vez pior em todos os testes com tendência exponencial, apresentando valores muito maiores que as outras duas tecnologias. Ao utilizar a aplicação, é evidente e perceptível ao usuário que a aplicação desenvolvida em Angular demora mais para ordenar a tabela.

Apesar do algoritmo de ordenação, em tese ser o mesmo, pois é definido pelo navegador utilizado (ex. quicksort no Google Chrome), há uma diferença expressiva no tempo dispendido pelo Angular para realizar esta tarefa. Não foi possível identificar o motivo desta diferença, especula-se que o angular reimplemente a função de ordenação do navegador e use outro algoritmo de ordenação (mas não foi possível confirmar). Essa pode ser uma estratégia de projeto do framework que vise a escolha do algoritmo de ordenação para pequenas quantidades de informações, pois dificilmente um usuário trabalhará com a visualização de 10.000 linhas em uma tabela, por exemplo.

4.2.5 Tempo de inicialização. É importante salientar que o tempo que o usuário teria que esperar até poder interagir com a tela, pode variar de acordo com a velocidade da sua conexão com a internet. Portanto, os resultados desta seção foram coletados com a aplicação hospedada na mesma máquina utilizada para testá-las, ou seja, os resultados não são influenciados por qualquer tráfego de rede. Foram realizadas dez execuções do software de teste automatizado desenvolvido, para medir o tempo de inicialização de cada tecnologia. Com os resultados, o gráfico da Figura 5 expressa o tempo dispendido na inicialização de cada tecnologia nas 10 execuções.

Figura 5: Tempo de inicialização da aplicação



Analisando o gráfico da Figura 5, é possível verificar que os resultados apresentaram um padrão de tempo nas dez execuções. O Angular teve o pior tempo de inicialização, média de 721 ms, seguido pelo React, média de 671 ms. O Vue apresentou o melhor tempo de inicialização, com a média de 571 ms. É preciso destacar que essa diferença de em torno de 100 ms entre o tempo das tecnologias, provavelmente é imperceptível ao usuário. Porém, esse tempo tende a aumentar de acordo com o tamanho da aplicação. Cabe ainda mencionar que o Vue inicializou uma aplicação idêntica à aplicação desenvolvida em Angular com cerca de 20% de tempo a menos, o que pode ter relação direta com o tamanho final do build de produção, relatado na sequência.

4.2.6 Tamanho do build de produção. Quando se está desenvolvendo uma aplicação com as tecnologias comparadas nesse trabalho, elas rodam em modo de desenvolvimento, o que facilita a depuração do código, mas em contrapartida afeta o desempenho. Para testar o máximo que a tecnologia pode oferecer, deve-se gerar um build de produção, o qual é totalmente focado em obter o melhor desempenho. O tamanho do build de produção pode interferir no tempo de carregamento de uma página, principalmente se houver uma conexão lenta com a internet, no lado do cliente.

Analisando a tabela 3, Angular produziu o build mais "pesado" (de maior tamanho), seguido pelo React e, por último, Vue, o mais leve. Analisando a quantidade de arquivos gerados, o React gerou o maior número de arquivos, seguido pelo Angular com uma pequena diferença. Já o Vue, foi o que gerou a menor quantidade de arquivos de produção, 17 a menos que o React. Considera-se que exista

Tabela 3: Comparação do tamanho do build para produção entre as tecnologias

Tecnologia	Num. de arquivos	Tam. dos arquivos(KB)
Angular	16	895
React	21	582
Vue	4	467

Tabela 4: Quantidade de linhas de código por tecnologia

Tecnologia	Quantidade de arquivos	Quantidade de linhas
Angular	10	741
React	5	750
Vue	3	720

uma correlação direta entre o tamanho do build com o tempo de inicialização da aplicação. Ou seja, pode-se afirmar que o tamanho do build de produção afeta diretamente o tempo de inicialização da aplicação.

4.2.7 Tamanho em linhas de código. Para calcular a quantidade de linhas de código úteis, a ferramenta CLOC (Counts lines of code) foi utilizada. Essa ferramenta analisa os arquivos e desconsidera comentários e linhas em branco na hora de realizar a contagem. Somente os arquivos criados pelo autor, localizados nas pastas "src" das tecnologias, foram considerados. Após executar a ferramenta, os resultados foram registrados na Tabela 4.

4.2.8 Suporte a diferentes navegadores. Como o software desenvolvido para testar as aplicações de forma automatizada foi focado no Google Chrome, as três aplicações desenvolvidas com as tecnologias foram testadas manualmente nos navegadores Mozilla Firefox versão 83.0 (64 bits) e Microsoft Edge versão 87.0.664.41 (64 bits). As aplicações executaram todas as operações sem apresentar nenhum erro e com o mesmo comportamento esperado resultante em cada navegador. Ou seja, identificou-se compatibilidade total entre os navegadores verificados.

5 CONCLUSÕES

Este estudo demonstrou que apesar das similaridades na forma de codificar uma aplicação web com o mesmo conjunto de funcionalidades e operações, os resultados experimentais apresentam tanto similaridades como discrepâncias em diferentes cenários de teste. Ou seja, mesmo com subsídios quantitativos de uma aplicação prática, a tarefa de decisão na adoção de uma tecnologia de desenvolvimento web não é uma tarefa trivial.

Com um conjunto de experimentos (cenários de teste) foi possível identificar que o Angular apresentou desempenho superior na maior parte dos cenários de teste. Por exemplo, o Angular obteve menor tempo de execução em 9 dos 12 cenários avaliados para aferir o tempo de execução. Também, utilizou menos memória em 8 dos 10 cenários em que a memória foi analisada. O que evidencia que para aplicações que envolvem intensa manipulação direta do DOM (ex. trabalhar com tabelas), o Angular é o mais adequado.

As exceções foram as operações de limpeza de tabela (ex. limpar uma tabela com dez mil linhas) e ordenação crescente de tabela (de 1.000 a 10.000 linhas). Nessas duas situações específicas, o Angular foi inferior e apresentou pior tempo de execução quando comparado com as outras 2 tecnologias. Inclusive, a diferença de tempo é perceptível ao usuário na realização dessas operações (vários segundos), o que pode ser um fator decisivo para não adotar o Angular quando houver necessidade de efetuar repetidas limpezas em tabelas com grandes volumes de dados apresentadas para os usuários. Já a React e o Vue tiveram resultados similares em quase todos os cenários com diferenças menores que um segundo (imperceptíveis aos utilizadores) na maioria dos testes.

A biblioteca React apresentou similaridades na forma de programar com os dois frameworks do estudo em virtude do padrão de codificação adotado. Porém, a React possibilita que a aplicação seja implementada com componentes funcionais. Entretanto, neste estudo não foi possível verificar as diferenças de codificação com esse padrão, pois os cenários de testes foram implementados apenas de forma imperativa, ou seja, de forma similar nas 3 tecnologias. Ou seja, identificou-se que o código escrito em JavaScript/TypeScript foi basicamente o mesmo nas três tecnologias, o que resultaria sempre em uma mesma complexidade cognitiva. Dessa forma, o cálculo da complexidade cognitiva não foi realizado por ter sido identificado como desnecessário pelo autor para o estudo e cenários de teste implementados.

Em relação a ocupação de memória na execução da aplicação, o Angular também obteve resultados superiores. O React apresentou quase a mesma ocupação de memória que o Angular em dois cenários, e o Vue teve os piores resultados em quase todos os testes.

No que tange ao tamanho de build de aplicação, o Vue resultou tanto em menor número de arquivos quanto menor tamanho total dos arquivos gerados para produção. Ou seja, em aplicações que demandam rápida inicialização e menor transferência de dados pela rede, o Vue é o mais apropriado. Isso pode ser determinante para aplicações que tendem a crescer com o tempo e exigem rápida inicialização. Além disso, foi possível identificar uma relação direta entre o tamanho final do build de produção, com o tempo de inicialização. O Angular foi a tecnologia que apresentou o pior tempo de inicialização possivelmente por possuir o maior build.

A aplicação desenvolvida foi testada no Google Chrome de forma automatizada e manualmente no Mozilla Firefox e Microsoft Edge. Todas as tecnologias demonstraram suporte total da aplicação e operações sem apresentar nenhum erro no console. A aplicação desenvolvida nas três tecnologias e a ferramenta que automatiza os testes pode ser verificada no repositório GitHub do autor[4].

Com os cenários experimentados, conclui-se que o framework Angular é o mais adequado quando houver necessidade de manipulação extensiva do DOM. Mesmo apresentando resultados inferiores em alguns cenários, o Angular foi superior na maioria, o que demonstra sua maturidade e equilíbrio no compromisso de obter o melhor desempenho/custo computacional no desenvolvimento front-end para web.

[4]https://github.com/MatheusSouzaCC/comparativo-tecnologias-frontend-js

REFERENCES

[1] Paul J. Deitel. *Ajax, Rich Internet Applications e desenvolvimento Web para programadores.* Pearson Prentice Hall, São Paulo, 2008.
[2] Danny Goodman. *Javascript: a bíblia.* Pearson Prentice Hall, Rio de Janeiro, 4th edition, 2001.
[3] Hotframeworks. Web framework rankings, 2018. URL https://hotframeworks.com/languages/javascript.
[4] Jim Conallen. *Desenvolvendo Aplicações Web com UML.* Pearson Prentice Hall, Rio de Janeiro, 2nd edition, 2003.
[5] David Flanagan. *Javascript: O guia definitivo.* Bookman, Porto Alegre, 4th edition, 2004.
[6] W3C. Javascript, 2011. URL https://www.w3.org/wiki/Javascript.
[7] CAELUM. Desenvolvimento web com html, css e javascript, 2019. URL https://www.caelum.com.br/apostila/apostila-html-css-javascript.pdf.
[8] Mike Wasson. Asp.net - single-page applications: Build modern, responsive web apps with asp.net, 2013. URL https://msdn.microsoft.com/en-us/magazine/dn463786.aspx.
[9] Mozilla Developers Network. Ajax: Documentação, 2021. URL https://developer.mozilla.org/pt-BR/docs/Web/Guide/AJAX.
[10] Thiago Faria Alexandre Afonso. *Fullstack Angular & Spring: Guia para se tornar um desenvolvedor moderno.* Algaworks Softwares, Treinamentos e Serviços Ltda, Minas Gerais, 2018.
[11] Facebook Inc. React: Uma biblioteca javascript para criar interfaces de usuário, 2019. URL https://pt-br.reactjs.org/.
[12] Michael Rambeau Raphaël Benitte, Sacha Greif. The state of javascript 2018: Front-end frameworks - overview, 2018. URL https://2018.stateofjs.com/front-end-frameworks/overview/.
[13] Pete Hunt, Paul O'Shannessy, Dave Smith, and Terry Coatta. React: Facebook's functional turn on writing javascript: A discussion with pete hunt, paul o'shannessy, dave smith and terry coatta. *Queue*, 14(4):96–112, August 2016. ISSN 1542-7730. doi: 10.1145/2984629.2994373. URL https://doi.org/10.1145/2984629.2994373.
[14] Evan You. Introduction: Vue.js, 2019. URL https://vuejs.org/v2/guide.
[15] Pekka Abrahamsson Amantia Pano, Daniel Graziotin. Factors and actors leading to the adoption of a javascript framework. *Empirical Software Engineering*, 23(6): 3503–3534, March 2018. doi: 10.1007/s10664-018-9613-x. URL http://dx.doi.org/10.1007/s10664-018-9613-x.
[16] Jaakko Voutilainen. Evaluation of front-end javascript frameworks for master data management application development, 2017.
[17] Stefan Krause. Js-framework-benchmark, 2019. URL https://github.com/krausest/js-framework-benchmark.
[18] Selenium. Webdriver, 2020. URL https://www.selenium.dev/documentation/en/webdriver/.
[19] Mocha. Mocha - the fun, simple, flexible javascript test framework, 2020. URL https://mochajs.org/.