Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

GUILHERME SCOTTI

ANÁLISE COMPARATIVA DE FRONT-ENDS DE FRAMEWORKS BASEADOS EM JAVASCRIPT

Belo Horizonte
2019

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

ANÁLISE COMPARATIVA DE FRONT-ENDS DE FRAMEWORKS BASEADOS EM JAVASCRIPT

por

GUILHERME SCOTTI

Monografia de Final de Curso

Prof. Roberto da Silva Bigonha
Orientador

Belo Horizonte
2019

GUILHERME SCOTTI

ANÁLISE COMPARATIVA DE FRONT-ENDS DE FRAMEWORKS BASEADOS EM JAVASCRIPT

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do certificado de Especialista em Informática.

Área de concentração: Engenharia de Software

Orientador: Prof. Roberto da Silva Bigonha

Belo Horizonte

2019

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Scotti, Guilherme Alves da Cruz.

S425a Análise comparativa de front-ends de frameworks baseados em javascript / Guilherme Alves da Cruz Scotti. Belo Horizonte, 2019.
40 f.: il.; 29 cm.

Monografia (especialização) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Roberto da Silva Bigonha.

1. Computação. 2. Engenharia de software. 3. Java Script (Linguagem de programação de computador). 4. Front-End. I. Orientador. II. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ESPECIALIZAÇÃO EM INFORMÁTICA: ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE

ANÁLISE COMPARATIVA DE FRONT-ENDS DE FRAMEWORKS BASEADOS EM JAVASCRIPT

GUILHERME ALVES DA CRUZ SCOTTI

Monografia apresentada aos Senhores:

Prof. Roberto da Silva Bigonha
Orientador
DCC - ICEx – UFMG

Prof. Clarindo Isaías Pereira da Silva e Pádua
DCC - ICEx - UFMG

Belo Horizonte, 25 de junho de 2019

AGRADECIMENTOS

Roberto Bigonha, pelo apoio incondicional para o desenvolvimento deste trabalho.

Minha mãe, por ser minha primeira e grande mestra.

Camila Schmith, pelo carinho e companheirismo e foco no desenvolvimento da monografia.

RESUMO

No início das aplicações web, os conteúdos gerados no servidor eram leves e simples de serem retornados para o navegador, porém atualmente, existe uma demanda alta por melhores funções e usabilidade das aplicações, criando espaço para novas tecnologias e arquiteturas. Sendo assim, foi necessário criar um novo tipo de abordagem no desenvolvimento web, que evolui para um novo tipo de aplicação conhecido como Single Page Application (SPA) que transfere parte da lógica das aplicações para o *client-side (front-end)*.

A proposta deste trabalho é analisar e comparar os principais padrões para projeto de software e frameworks JavaScript de aplicações SPA *front-end*, identificando suas características e comparando os principais fatores que influenciam a manutenibilidade, compatibilidade com os navegadores e boa performance. Para isso, serão analisados os três frameworks JavaScript mais populares do mercado, e será desenvolvido uma aplicação SPA por framework a fim de realizar uma análise comparativa.

Palavras-chave: JavaScript, Front-End, MVC, FLUX, React, AngularJs, VueJs.

ABSTRACT

At the beginning of the time of web applications, the content produced on the server was light and simple to be returned to the browser, but today there is a high demand for better functions and usability of this applications, creating space for new technologies and architectures. Therefore, it was necessary to create a new type of web development approach, in which it was called the Single Page Application (SPA) development that transfers part of the application logic to the client-side (front-end).

The purpose of this work is to analyze and compare the design pattern and JavaScript frameworks for front-end SPA applications, identifying their characteristics and comparing the main factors that influence the maintainability, compatibility with browsers and good performance. For this, we will analyze the three most popular JavaScript frameworks in the market, and an SPA application will be developed per framework in order to perform a comparative analysis.

Keywords*:* JavaScript, Front-End, MVC, FLUX, Design Pattern, React, AngularJs, VueJs.

.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AJAX	*Asynchronous JavaScript And XML*. Método assíncrono de transferência.
Angular	JavaScript Framework mantido pelo Google, sucessor do AngularJs.
Back-End	Desenvolvimento de aplicações no lado do servidor.
Client-Side	Desenvolvimento de aplicações no lado do cliente ou Front-End.
CSS	*Cascading Style Sheets.* Arquivo responsável pelo layout do HTML
DOM	*Document Object Model.* Arquitetura de componentes dentro de um HTML
Framework	Ambiente de desenvolvimento de software reusável.
Front-End	Desenvolvimento de aplicações no lado do cliente.
HTML	*Hypertext Markup Language.* Documento da página Web
HTTP	*HyperText Transfer Protocol.* Protocolo de transferência da Web.
JavaScript	Linguagem de programação de alto nível usada na Web
MVC	Model-View-Controller. Padrão de projeto de software.
React	Biblioteca JavaScript mantida pelo Facebook.
SPA	*Single Page Application.* Arquitetura de aplicações WEB de uma página.
TypeScript	Biblioteca de tipografía opcional para JavaScript.
Vue	JavaScript Framework Progressivo criado por Evan Y.
WWW	*World Wide Web.* Internet

SUMÁRIO

1. INTRODUÇÃO

O desenvolvimento de aplicações Web é aprimorado constantemente em todos os aspectos, desde a arquitetura e as metodologias empregadas até as linguagens de programação que o compõem. A arquitetura dessas aplicações está deixando de ser composta por grandes quantidades de códigos *server-side*, onde todo o processamento é feito no servidor, cuja abordagem já foi eficiente, pois os HTMLs eram simples e estáticos, sem a necessidade de grandes processamentos para realizar requisições.

Assim, foi necessário criar um novo tipo de abordagem no desenvolvimento web, que evolui para um novo tipo de aplicação conhecido como *Single Page Application* (SPA, ou aplicação de página única) que transfere parte da lógica das aplicações para o *client-side (front-end)*. Uma das características mais importantes dessas aplicações é realizar uma comunicação assíncrona com a base de dados, atualizando o conteúdo da página ou componente sem recarregar a página por inteiro. (MIKOWSKI, M.; POWELL, J., 2013).

Para sustentar essa nova arquitetura Web, foi necessário uma linguagem robusta para executar as lógicas no *client-side*, assim surge a utilização do JavaScript como a linguagem de programação mais utilizada no *front-end* (W3TECHS,2019) .

Dessa forma, é cada vez maior a demanda por softwares de qualidade, e a aplicação de padrões de projetos e frameworks no desenvolvimento de sistemas tem se tornado cada vez mais indispensável (MAURYA, L.; SMARAK, S., 2012).

Para Voutilainen (2014, p. 10, tradução nossa), framework é uma grande coleção de funções e ferramentas que facilitam o desenvolvimento. Os frameworks também podem ser definidos como o esqueleto de uma aplicação, ou uma abstração de soluções padronizadas. Ao utilizar um framework, o desenvolvedor pode aprimorar a eficácia do desenvolvimento, entregando softwares de qualidade.

Segundo Lavanya, Ramachandran e Mustafa (2010, p. 5455, tradução nossa), para desenvolver uma aplicação, o fator decisivo para escolha do framework normalmente é a familiaridade do desenvolvedor, o que não pode ser considerado como um fundamento correto para a escolha do framework , no entanto, um fator que frequentemente é ignorado é o desempenho, especialmente para grandes aplicações complexas e escaláveis. Dentro desse cenário, como desenvolver aplicações front-end baseadas em JavaScript, mantendo uma arquitetura ou framework que garanta a manutenibilidade, compatibilidade com os navegadores e boa performance?

1.1. OBJETIVO GERAL DO TRABALHO

O objetivo deste trabalho é analisar e comparar os principais padrões para projeto de software e frameworks JavaScript de aplicações *front-end*, identificando suas arquiteturas e características principais. Após essa análise, apresenta-se uma aplicação em cada framework comparando três aspectos comuns:

- Performance de renderização dos componentes, ou tempo de carregamento;
- Compatibilidade entre navegadores;

- Tamanho do pacote de código gerado na compilação do framework.

O estudo será delimitado em frameworks SPA escrito em JavaScript. Mesmo que a maioria das aplicações tenham um *back-end,* o foco do estudo será somente no *front-end*.

1.2. ESTRUTURA DO TRABALHO

O texto está dividido em duas partes, a primeira apresentada nos capítulos 2, 3 e 4 é a parte descritiva e conceitual onde será explicado como as aplicações Web evoluíram até a criação das SPAs. Além disso, exemplificam-se os padrões de projetos para aplicações *front-end* JavaScript e seus respectivos frameworks, para entender melhor a necessidade da utilização de frameworks e padrões de projeto para o desenvolvimento de aplicações *front-end* em JavaScript.

A segunda parte, descrita nos capítulos 5 e 6, contempla o desenvolvimento de uma aplicação SPA em cada um dos frameworks analisados, para em seguida realizar a análise comparativa entre as aplicações.

2. APLICAÇÕES WEB

O Capítulo 2 inclui uma introdução ao desenvolvimento Web, explicando como foi a evolução das arquiteturas tradicionais para as aplicações SPAs desenvolvidas em JavaScript. Explica-se o motivo dessa evolução, quais foram os requisitos para a criação dessas aplicações SPA e quais as diferenças das arquiteturas.

2.1. EVOLUÇÃO

A *World Wide Web* (WWW), que foi criada em 1989 por Tim Berners-Lee, possuía o objetivo de facilitar o compartilhamento de informações entre computadores, o que, na época, não era uma tarefa simples de ser realizada (SEGAL,1995).

Com a evolução da Web, foi necessário a criação de metodologias e ferramentas para o desenvolvimento de sites que compõem a internet, no qual chamamos de desenvolvimento Web. Uma arquitetura tradicional desse desenvolvimento consiste em um cliente e um servidor, conforme Figura 1. O cliente envia solicitações *HyperText Transfer Protocol* (HTTP) para o servidor da Web, que por sua vez executa uma comunicação com o banco de dados e retorna um documento *Hypertext Markup Language* (HTML) e outro *Cascading Style Sheets* (CSS) para o navegador do usuário (VOUTILAINEN, 2017).

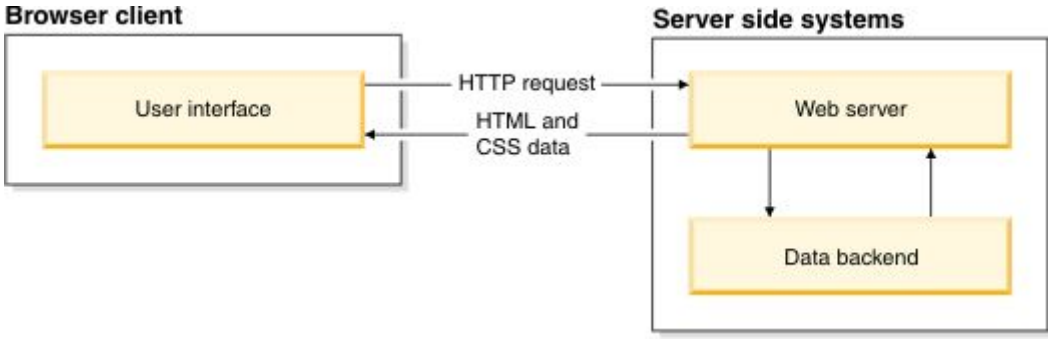

Figura 1 - Web application data transfer model.

Fonte: VOUTILAINEN, 2017 p. 4.

Em relação a arquitetura tradicional da Web, o *server-side* fica responsável por todo o processamento das solicitações e comunicação com o banco de dados, retornando um conjunto de documentos (HTML e CSS) imutáveis para o *client-side*. Porém, também existe processamento no *client-side,* mesmo que pouco. Geralmente utiliza-se JavaScript para manipular o *Document Object Model* (DOM), e criar visualizações interativas ou exibir alertas. HTML DOM é uma estrutura em árvore, ou modelo, que o navegador usa para acessar diferentes elementos dispostos nas páginas Web (VOUTILAINEN, 2017).

O servidor tradicionalmente também tem outras tarefas, além de retornar documentos HTML. Uma aplicação tradicional geralmente consiste em várias páginas HTML e uma rota de navegação. Quando o usuário clica em um link na página, ele redireciona o navegador para um arquivo HTML diferente existente no servidor. A rota de navegação é um processo no qual cada URL corresponde a uma página HTML ou conteúdo específico. Usando URLs e rotas de navegação, os usuários podem marcar ou compartilhar um link e navegar em sites usando os botões voltar e avançar (VOUTILAINEN, 2017).

De acordo com Mesbah, A. e Deursen, A (2007, p. 181, tradução nossa) “Apesar de sua enorme popularidade, as aplicações tradicionais Web sofreram com a baixa interatividade e performance de tempo de resposta para os usuários finais”. Essa baixa performance acontece porque, quando um usuário troca de página, demora-se um tempo para o navegador recuperar o novo documento HTML do servidor, além do tempo em que o servidor leva para processar a nova requisição.

Devido a esses fatos, uma parte da lógica e processamento da aplicação foi transferido para o *client-side*, assim liberando o servidor de processar diversas requisições. Dentro desse conceito, surgem as SPAs, melhorando significativamente a experiência do usuário quanto à performance e interatividade (VOUTILAINEN, 2017).

2.2. SINGLE PAGE APPLICATIONS

Existe uma convenção de que não há uma definição certa do conceito de SPA. Segundo Mesbah, A. e Deursen, A. (2007, p. 181, tradução nossa) “ Uma interface SPA é composta de componentes individuais independentes que podem ser atualizados ou substituídos, de tal forma que a página não necessite de ser atualizada a cada ação do usuário”. Porém, segundo Mikowski, M. e Powell, J. (2013, p. 4, tradução nossa) “SPA é uma aplicação que é entregue ao navegador, que não necessita recarregar a página durante o uso”. Dessa forma, percebemos que o ponto mais importante das aplicações SPA é a atualização parcial ou completa dos componentes sem recarregar a página.

As aplicações Web tradicionais e as SPAs operam de maneira diferente em termos de transferência de dados, uma vez que uma aplicação SPA possui apenas uma única página. Quando um cliente requer uma página do servidor, ele recebe o site completo em apenas uma transferência de dados. O conteúdo recebido geralmente contém um documento HTML parcialmente completo. Ele é complementado dinamicamente no final do processamento no cliente, utilizando a linguagem JavaScript. Em outras palavras, a página visível é criada por uma série de funções JavaScript, que manipulam e complementam o HTML DOM em tempo de execução. Nenhuma solicitação adicional é necessária. Assim, a mudança de página em um SPA não é realmente uma mudança de página, mas sim a substituição do conteúdo da página atual por um novo conteúdo. Este conteúdo é gerado por funções JavaScript (VOUTILAINEN, 2017) .

Em uma aplicação SPA, a primeira resposta do servidor inclui um site completo HTML, JavaScript e CSS. Devido a esse fato, a quantidade de dados na primeira requisição ao servidor é naturalmente grande, mas em um tempo hábil tolerável devido às altas velocidades atuais de banda larga. Alguns dados precisam ser recuperados após o carregamento inicial da página, esse tipo de dado é solicitado via *Asynchronous JavaScript And XML* (AJAX). As requisições AJAX são métodos assíncronos que permitem comunicar-se com o servidor sem recarregar a página, geralmente, os dados são transferidos como um objeto *JavaScript Object Notation* (JSON). JSON é um formato de objeto JavaScript comum, leve e legível por humanos. Essa arquitetura pode ser exemplificada na Figura 2.

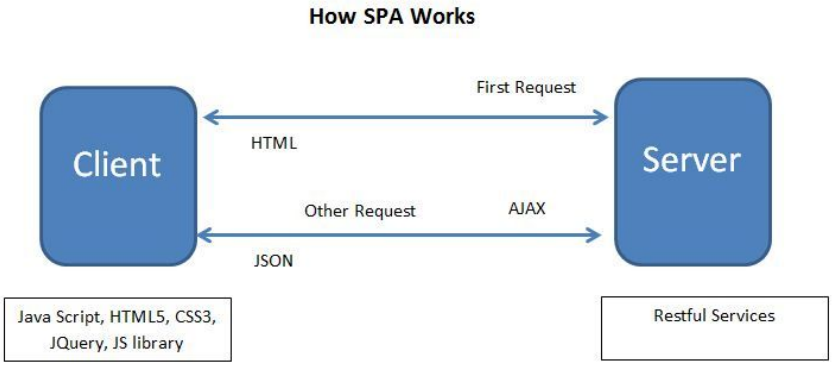

Figura 2 - Transferência de dados entre cliente e servidor.

Fonte: BAJAJ 2014.

A lógica desenvolvida no *server-side* é muito mais simples em aplicações SPA, uma vez que o servidor normalmente é um *Representational state transfer (*RESTful) *Web Service* puro. O servidor recebe menos atribuições, e o banco de dados não é mais responsável pela lógica de negócios, conforme exemplificado na Figura 3.

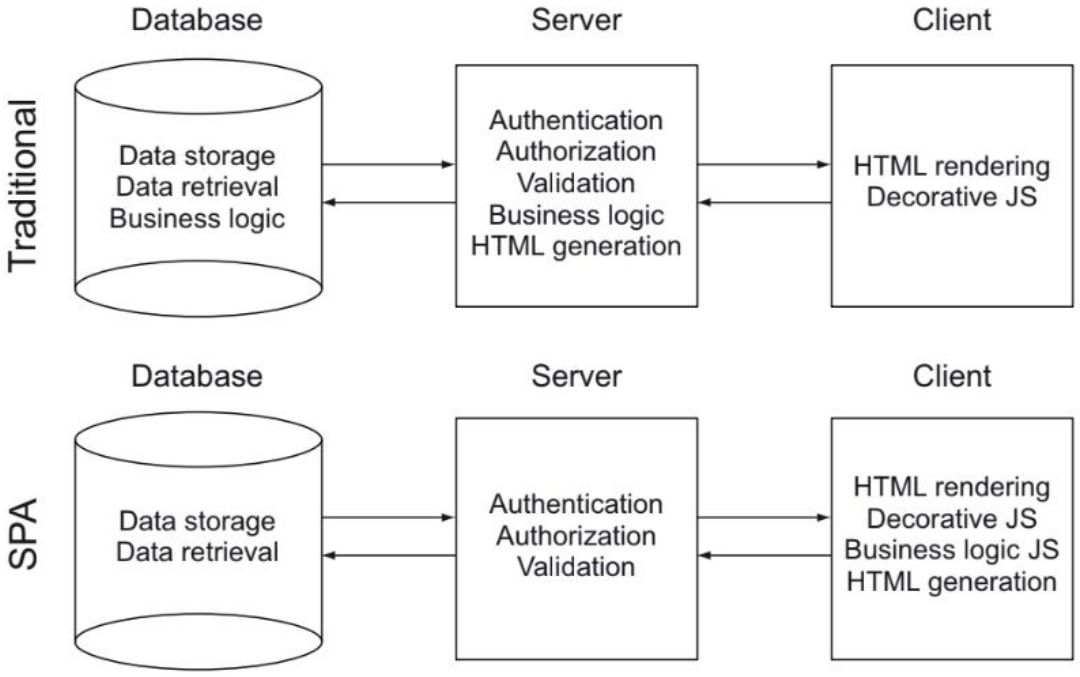

Figura 3 - SPA vs Aplicações Web Tradicionais.

Fonte: MIKOWSKI, J. e POWELL, J. 2014, p. 8.

Para as aplicações SPA atingirem todo esse potencial, foi necessária uma linguagem que suportasse todos os requisitos dessa arquitetura e, segundo a W3Techs (2019), “JavaScript é utilizado por 95,2% de todos os navegadores, se tornando a linguagem de programação mais utilizada no *client-side*”. Dessa forma, o JavaScript é hoje a linguagem de mercado mais utilizada para desenvolver SPAs, além de ter a maior compatibilidade entre navegadores.

2.3. JAVASCRIPT

JavaScript é uma linguagem de programação criada por Brendam Eich em 10 dias, lançada inicialmente com o nome de Mocha em maio de 1995, em homenagem ao fundador da Netscape. Ainda nesse ano, a Netscape queria anunciar a parceria feita com a Sun Microsystems, empresa criadora da linguagem de programação JAVA. Com vista a aproveitar toda publicidade/notícias na imprensa, em dezembro desse mesmo ano o nome foi alterado de Livescript para JavaScript. Este novo nome gerou bastante confusão, porque cria a ilusão de que essa nova linguagem seria baseada em JAVA, o que não é verdade (SEVERANCE, 2012).

Em 1996 JavaScript foi convertido em um padrão chamado ECMAScript criado pela ECMA, assim o padrão foi modificado e evoluindo até a versão atual ECMAScript 9. Porém esse padrão não foi projetado para ser capaz de gerenciar dados por si próprio, mas está de acordo com a especificação que esse gerenciamento seja realizado pelo ambiente computacional que cerca o padrão. Isso mostra que o JavaScript é originalmente usado apenas como uma maneira de "animar páginas da Web" (MALMSTRÖM , 2014).

Ao longo dos anos uma das melhorias implementadas sobre o padrão ECMAScript foi o AJAX (Asynchronous JavaScript and XML), que permitia aumentar a interatividade da página através de pedidos feitos ao servidor sem carregar uma nova página ou recarregar a mesma página. Apesar de todos os benefícios, foram detectados problemas no JavaScript, pois não havia um compilador universal, devido ao fato de ser uma linguagem ser compilada nos navegadores Web, que muitas vezes não eram compatíveis (DUARTE, 2015).

Devido ao problema de compatibilidade entre os navegadores, a qualidade dos softwares era comprometida. Porém, qualidade de software é um conceito com diversos atores relacionados ao sistema; um desses atores é a manutenibilidade do software, que define o quão fácil um sistema pode escalar seus componentes ou como o sistema pode se adaptar a mudanças de requisitos e ambientes, também está relacionado a vida útil do software (SOMMERVILLE, 2016).

Visando uma melhor manutenibilidade e qualidade dos softwares desenvolvidos, começaram a surgir bibliotecas para solucionar o problema de compatibilidade dos navegadores e para que o JavaScript não ficasse mais lento em alguns navegadores de internet. Uma delas foi a JQuery que teve bastante sucesso no aprimoramento do desenvolvimento Web, porque trouxe maior compatibilidade entre navegadores de internet. Posteriormente, surgiram os frameworks, que são estruturas para desenvolvimento de *softwares* robustos. (DUARTE, 2015).

Devido a suas vantagens de interatividade, compatibilidade entre navegadores e manutenibilidade, o JavaScript já é a linguagem de programação mais utilizada para desenvolvimentos *front-end*. Muitas aplicações que eram desenvolvidas para sistemas operacionais foram evoluindo e hoje são desenvolvidas como aplicações SPA.

Conclui-se que, as aplicações Web evoluíram devido à necessidade de aplicações mais interativas e dinâmicas. Para atingir essas demandas, novas linguagens de programação foram criadas, e a arquitetura tradicional da Web foi transformada, mudando a lógica e processamento de dados das aplicações para o *client-side,* surgindo as SPAs.

3. FRAMEWORKS JAVASCRIPT

O Capítulo 3 define o que é um framework e o diferencia de uma biblioteca. Também delimita o escopo do projeto entre os principais frameworks Angular, React e Vue, analisando suas principais características e diferenças.

3.1. DEFINIÇÃO E DELIMITAÇÃO DO ESCOPO

A definição de framework JavaScript pode variar na literatura dependendo da situação. Para Johnson, R. (1997, p. 10, tradução nossa) “framework é um padrão reusável representado por classes abstratas, sendo também um esqueleto de uma aplicação que pode ser customizável pelos desenvolvedores”.

O termo framework é usado quando o controle do fluxo de execução do programa passa da responsabilidade do desenvolvedor para a responsabilidade do framework. Isso separa os frameworks JavaScript das bibliotecas. Uma biblioteca oferece apenas um conjunto ou subconjunto de funções que auxiliam a execução da aplicação, já um framework, gerencia todo o fluxo de dados e execução do aplicativo (VOUTILAINEN, 2014).

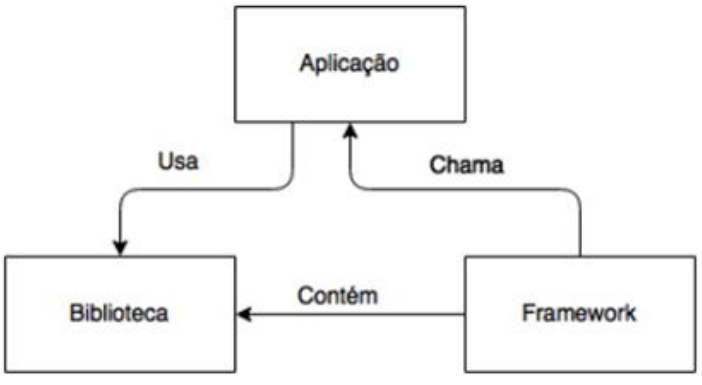

Figura 4 - Diferença biblioteca e framework.

Fonte: DUARTE, 2015 p. 15.

A Figura 4 representa a associação entre bibliotecas, aplicações e os frameworks. Verifica-se que um framework pode conter várias bibliotecas, e que o framework controla o código da aplicação. Por sua vez, uma aplicação pode usar bibliotecas, mesmo que estas não pertençam a um framework (DUARTE, 2015).

Devido à grande quantidade de frameworks JavaScript, foi necessário escolher apenas os 3 mais utilizados para o trabalho. Porém, é difícil categorizar os 3 melhores frameworks de mercado, uma vez que as mudanças nos frameworks são contínuas e novos frameworks entram no mercado todo ano.

De acordo com Hotframeworks (2018) os três frameworks mais utilizados em 2018 são React, Angular e Vue nessa ordem. Porém, a Stateofjs (2018) considera que a ordem dos mais utilizados é React, Vue e Angular, conforme Figura 5. Baseado nos dados apresentados acima foram selecionados os frameworks React, Angular e Vue para o presente trabalho.

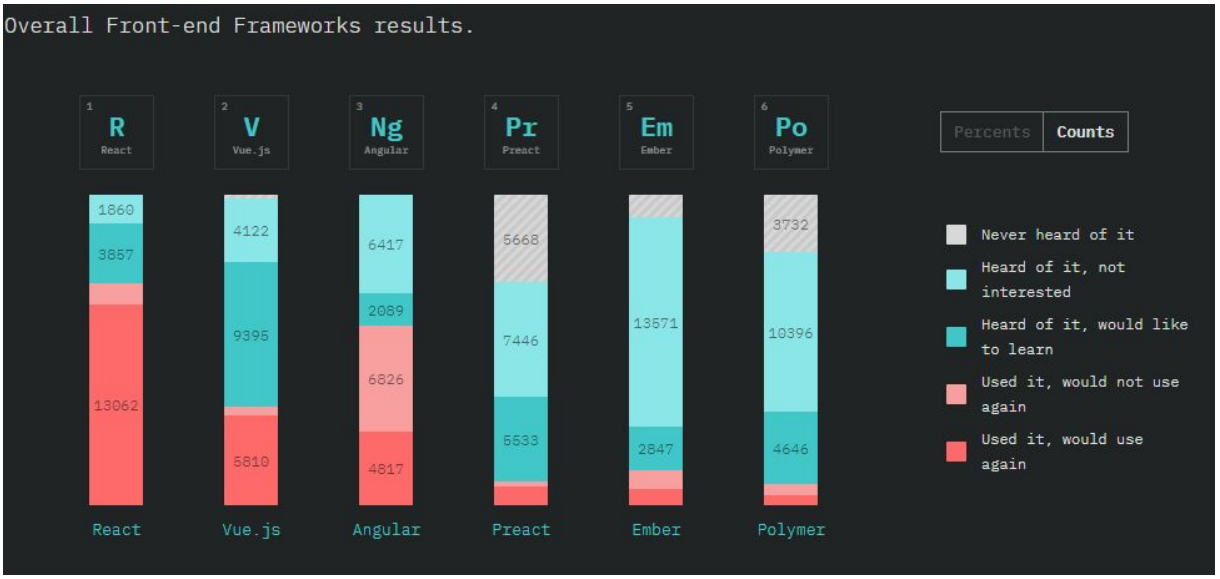

Figura 5 - Frameworks mais utilizados em 2018

Fonte: STATEOFJS, 2018

3.2. REACT

React é uma biblioteca JavaScript para criar *views* declarativas da arquitetura *Model-View-Controller* (MVC). Como mencionado na seção acima, o termo "biblioteca" neste contexto significa que não se destina a cobrir todas as áreas da aplicação, mas fornecer uma plataforma com uma coleção de funções que o desenvolvedor usa.

React é mantido pelo Facebook e foi lançado pela primeira vez em março de 2013, por Jordan Walke. O Facebook usa o React fortemente em seus sites, no próprio Facebook e no Instagram e WhatsApp, mas outras empresas também utilizam esse framework, como Yahoo, Airbnb, Sony entre outras (FACEBOOK, 2013).

O React não usa templates HTML, todo o código do documento HTML é definido em JavaScript e renderizado no *client-side*. React também introduz o uso do DOM virtual. Pode-se pensar no *virtual* DOM como uma página virtual que se consegue modificar e alterar a página real. Esse conceito mostra ser extremamente eficiente porque a página final só iria receber as alterações depois de um algoritmo ter verificado quais iriam ser essas alterações.

O React foi projetado para criar interfaces de usuário interativas e responsivas, e para isso utiliza um padrão de projeto novo chamado de FLUX que será abordado no Capítulo 4. Entre os três frameworks a serem avaliados, o React é a opção com menos controle sobre o código desenvolvido, uma vez que ele é uma biblioteca e, portanto, as funções implementadas são deixadas a critério do desenvolvedor.

3.3. VUE

O Vue é um framework de código aberto, como React and Angular. Inicialmente, foi lançado em fevereiro de 2014, por Evan You. É um framework progressivo, de fácil acesso, versátil e muito eficiente, tendo uma curva de aprendizado menos acentuada em comparação aos outros frameworks abordados neste trabalho (VUE, 2019).

Pode-se dizer que o Vue é uma versão aprimorada dos dois maiores frameworks JavaScript de mercado React e Angular. Ele possui uma biblioteca principal focada apenas na camada da *view,* facilmente integrável com outros frameworks ou bibliotecas de mercado. Por outro lado, o Vue também é capaz de desenvolver SPAs robustos e escaláveis (VUE, 2019).

Vue utiliza muitos recursos do React, como o *virtual* DOM e o padrão de projeto Flux. Dessa forma, o Vue e o React são similares quando comparados no quesito performance. Também, devido a similaridade é um bom framework para oferecer soluções robustas para aplicações complexas e escaláveis.

3.4. ANGULAR

A primeira versão de ANGULAR era denominada AngularJS, que foi inicialmente desenvolvido em 2009 por Miško Hevery e Adam Abrons como um *software* por trás de um serviço de armazenamento JSON online, que teria preço estimado por megabyte, para as empresas. Os dois desenvolvedores decidirem abandonar a ideia comercial e distribuir o AngularJS como uma plataforma *open-source*. Abrons deixou o projeto, mas Hevery, que trabalha na Google, continuou o seu desenvolvimento e manteve a plataforma em conjunto com alguns colegas do Google: Igor Minár e Vojta Jína (ANGULAR,2019).

Assim a plataforma AngularJS foi criada, focada no *client-side* e integrada com servidores que fornecem interface em REST/JSON. Após alguns anos enfrentando diversos problemas arquiteturais, o AngularJS foi totalmente reescrito, e em 2016 lançado o framework de JavaScript Angular.

Angular é construído inteiramente sobre o TypeScript, e usá-lo no desenvolvimento é recomendado, embora não seja obrigatório. O TypeScript é um superconjunto de JavaScript, que adiciona tipificação estática ao código. Uma das mais importantes caraterísticas e vantagens desta plataforma é o uso de *dependency injection* como forma de adotar a inversão

de controle, permitindo interligar partes de código e módulos, mantendo o código com baixo acoplamento.

Esse framework foi desenvolvido utilizando uma arquitetura de componentes em árvore, onde o componente pai verifica constantemente se existe uma mudança nos componentes filhos, toda essa arquitetura é baseada no padrão de projeto MVC*,* implementando uma comunicação *Two Way Data Binding,* que será abordado no Capítulo 4.

3.5. ANÁLISE COMPARATIVA

Levando em consideração que existe uma grande quantidade de frameworks no mercado, a delimitação do trabalho entre os três frameworks mais populares ajuda a destacar suas principais diferenças. Também foi possível verificar que o React não possui todas as características necessárias para ser denominado framework, sendo na verdade um conjunto de bibliotecas.

Pela observação dos frameworks analisados, conclui-se que existem similaridades nas suas funções, uma vez que são baseados em JavaScript. Porém, cada framework manipula o DOM de uma forma única. Além disso, o Angular apresenta um padrão de projeto baseado no MVC, diferente do Vue e React que apresentam uma arquitetura *One Way Data Binding* oriundas do padrão de projeto FLUX, que será abordado no Capítulo 4.

4. PADRÕES DE PROJETO

Capítulo 4 explica o que são os padrões de projeto e as arquiteturas MVC e FLUX que são utilizadas pelos frameworks analisados, demonstrando como as arquiteturas funcionam e suas diferenças.

4.1. DEFINIÇÃO

Padrões são um conjunto de soluções para um problema recorrente em comum em determinado contexto. Essa idéia foi introduzida pelo arquiteto Christopher Alexander que salientava,

> *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice* [1](ALEXANDER, 1979, p. 10).

Os padrões descrevem templates de soluções para problemas que podem ser aplicados em diferentes situações, que facilitam o reuso correto de arquiteturas escolhendo técnicas de design corretas que não comprometem a reusabilidade, além de aprimorar a documentação e manutenção de sistemas existentes, fornecendo uma especificação explícita das interações entre classe e objeto (GAMMA et al, 1994).

Cada padrão fornece um conjunto predefinido de componentes que fazem parte de uma solução genérica para uma infinidade de problemas. Esses componentes podem ser

[1] Cada padrão de projeto descreve um problema recorrente no ambiente, e um conjunto de soluções para esse problema, que de certa forma você pode reutilizar diversas vezes, sem que seja necessário fazer da mesma forma duas vezes. Tradução livre pelo autor.

combinados como blocos de construção, e podem ser utilizados para o desenvolvimento de projetos mais complexos. Dessa forma, ao aplicar um padrão para resolver algum problema, existe uma economia de tempo quando comparado à busca de soluções específicas (BUSCHMANN et al ,1996).

Os padrões possuem quatro principais elementos (GAMMA et al, 1994):

- O nome do padrão geralmente descreve o problema de design, suas soluções e consequências em poucas palavras.
- O problema descreve quando aplicar o padrão, além de descrever o contexto.
- A solução descreve os elementos necessários para solucionar o problema do padrão, seus relacionamentos, responsabilidades e colaborações.
- A consequência são os resultados e as alterações aplicadas nos padrões de projeto.

Porém, segundo Buschmann (1996, p. 10, tradução nossa) “os padrões são estruturas que proveem papéis específicos para problemas específicos, não existindo padrões semelhantes, uma vez que cada padrão é único em sua forma, independente da área de aplicação”, e possuem três elementos comuns, conforme Figura 6.

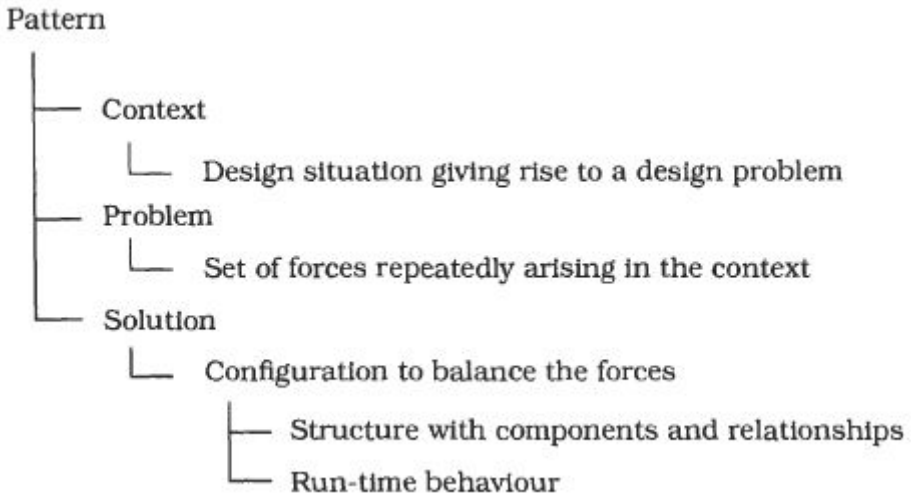

Figura 6 - Arquitetura dos padrões.

Fonte: BUSCHMANN et al, p. 11 (1996).

Segundo BUSCHMANN et al (1996), na Engenharia de Software os padrões podem ser categorizados levando em conta a similaridade entre os padrões e o nível de abstração. Assim os padrões são divididos em três categorias *Architectural Patterns, Design Patterns* e *Idioms*. Porém, neste trabalho abordaremos apenas os padrões de arquitetura, que corresponde ao padrão adotado pelos frameworks analisados, que são definidas como:

> *"An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them"* [2](BUSCHMANN et al, 1996, p. 12).

Os padrões de arquitetura são templates estruturais para a criação de sistemas de software. Dessa forma, a escolha de um padrão arquitetural é de suma importância. Dentre os padrões

[2] Uma arquitetura de software expressa um esquema estrutural organizado para softwares. Ele fornece um conjunto predefinido de subsistemas, especifica suas responsabilidades e inclui regras e diretrizes para organizar as relações entre eles. Tradução livre pelo autor.

arquiteturais serão abordadas as arquiteturas MVC e FLUX, que são utilizadas pelos frameworks analisados .

4.2. MVC

O MVC, que foi criado para introduzir a modularidade e reusabilidade aos desenvolvimentos orientados a objeto do Smalltalk, onde os objetos de diferentes classes assumem as operações relacionadas ao domínio da aplicação, e exibem o estado da aplicação através das ações dos usuários. O modelo MVC consiste na separação da aplicação em três grupos, conforme Figura 7 (GLEEN, KRASNER e STEPHEN, 1988).

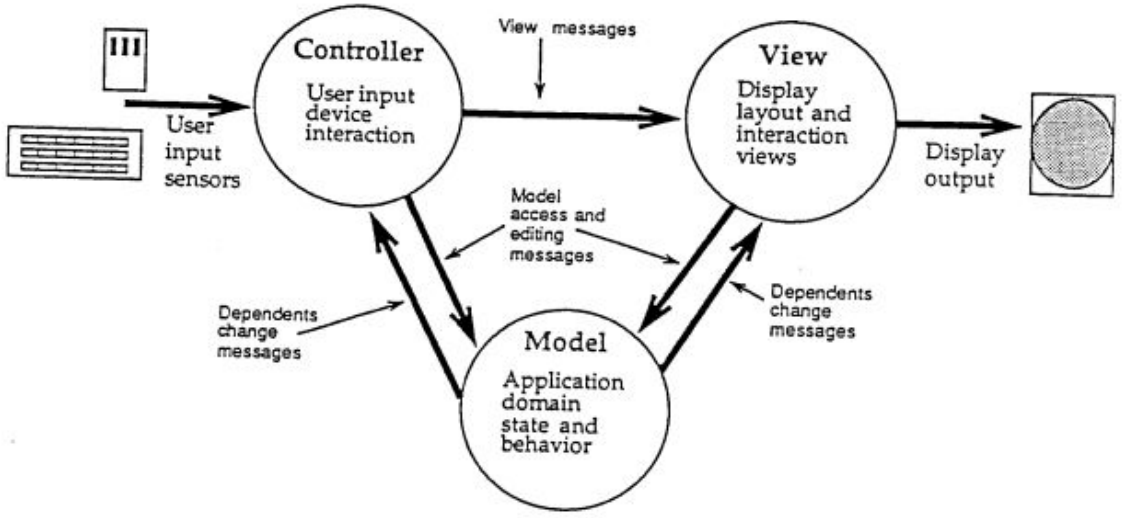

Figura 7 - Model View Controller em aplicações do SmallTalk.

Fonte: GLEEN, KRASNER e STEPHEN, p. 6 (1988).

O *Model* é o núcleo da aplicação, onde é implementada e encapsulada toda estrutura lógica central da aplicação. A *View*, lida com todas as aplicações gráficas, ela é responsável por

disponibilizar as interfaces para interação com o usuário, requerendo dados do *Model.* (GLEEN, KRASNER e STEPHEN , 1988).

Cada *View* está interligada a um *Controller*, que é responsável por tratar a entrada propagada pela *View* e direcioná-lo ao *Model* correto, servindo como um intermediário entre a *View* e o *Model*. Essa separação permite que uma *View* se comunique com múltiplos *Models.* Dessa forma, caso o estado do *Model* seja alterado por alguma aplicação, todas as *Views* que utilizam esse *Model* serão notificadas, atualizando as informações exibidas (BUSCHMANN et al,1996).

O MVC estabelece um protocolo de notificação e inscrição entre os *Models* e as *Views.* A *View* tem que refletir sempre o estado atual do *Model*, ou seja, sempre que houver alguma modificação nos dados do *Model*, a *View* dependente desses dados será notificada, e consequentemente atualizada. A partir dessa abordagem é possível definir um relacionamento de múltiplas *Views* para o mesmo *Model,* conforme Figura 8 (GAMMA et al, 1994).

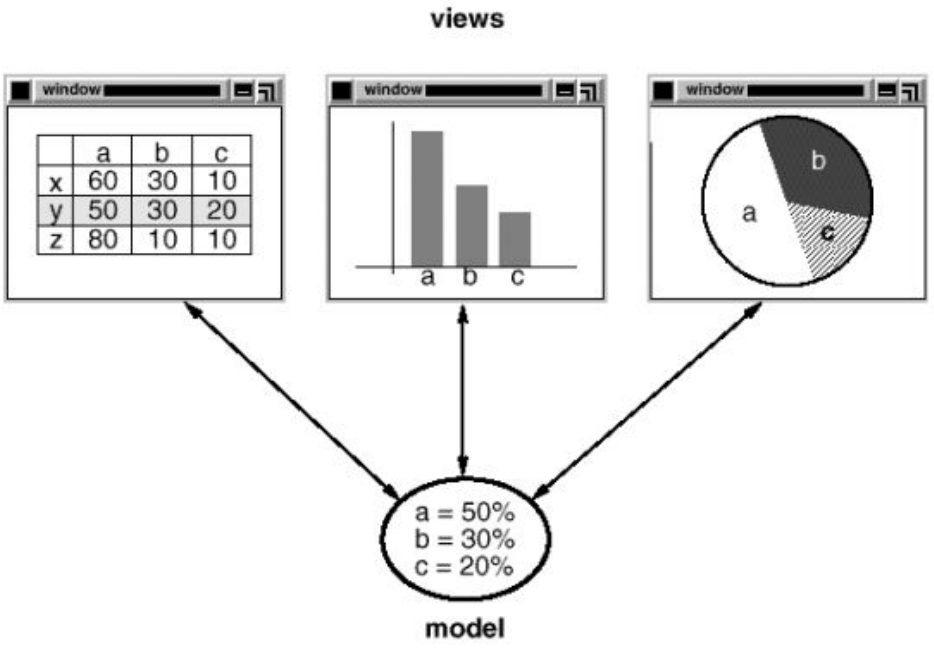

Figura 8 - Decomposição dos dados em múltiplas visões

Fonte: GAMMA et al, p. 15 (1994).

O padrão MVC fornece um fluxo de dados bidirecional, conforme observado na Figura 9; quando uma *View* recebe uma notificação para se atualizar, ela pode realizar outras ações que refletem em outros *Models* indefinidamente.

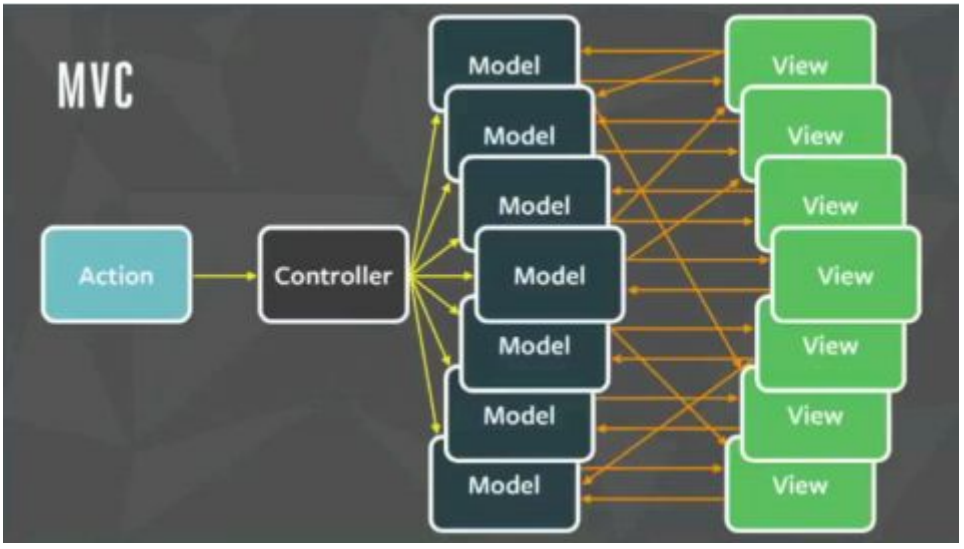

Figura 9 - Action loop MVC.

Fonte: FLUX, p. 1 (2013).

Dessa forma, um sistema em grande escala utilizando a arquitetura MVC está vulnerável à falhas de escalabilidade, uma vez que a adição de uma nova função pode ocasionar um efeito em cascata de updates, além de falhas na manutenibilidade do sistema.

4.3. FLUX

Devido a vulnerabilidade do fluxo de dados bi-direcional nos sistemas de grande escala, em 2013 o Facebook criou um novo padrão de arquitetura de software unidirecional chamado FLUX que funciona de forma similar ao MVC.

Segundo MARTTILA (2016) esse padrão de arquitetura é semelhante ao MVC onde o Dispatcher realiza o papel do *Controller*, e o *Store* realiza o papel do *Model*. Porém o FLUX dita como são realizadas as interações entre os pacotes, além de possuir componentes independentes e separados, conforme Figura 10.

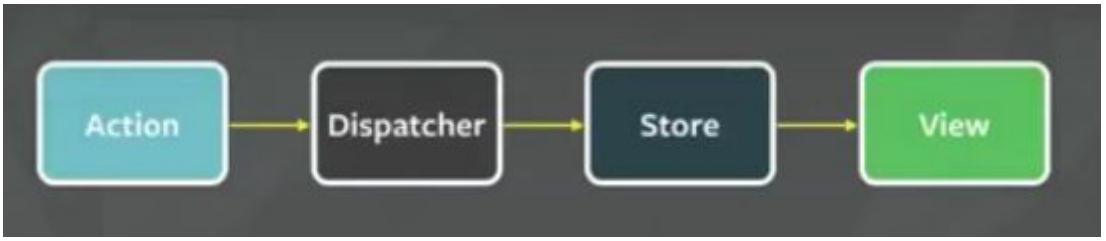

Figura 10 - Arquitetura FLUX.

Fonte: FLUX, acesso em Dezembro de 2017.

A *Store* é onde toda a lógica da aplicação é implementada e encapsulada, além de ser responsável pela camada de dados. As *Views* estão ligadas diretamente às *Stores,* que contêm os dados necessários para sua renderização. Assim, a cada atualização na camada de dados, as views são renderizadas novamente. (FACEBOOK, 2013).

Ao gerar uma ação em alguma aplicação, ela é enviada ao *Dispatcher* que é responsável por gerenciar todo o tráfego das *Actions* distribuindo-as para suas respectivas *Stores*. Além disso, o *Dispatcher* é responsável pelo fluxo de dados unidirecional, uma vez que ele segue o padrão de projeto *Singleton* que permite instanciar apenas uma ação por *Store* por vez, conforme Figura 10. (FACEBOOK, 2013).

4.4. ANÁLISE COMPARATIVA

A principal diferença das duas arquiteturas de projetos é a localização dos componentes responsáveis pela atualização do estado dos dados, além da direção do fluxo de dados. No padrão MVC, o fluxo de dados bidirecional permite que um dado seja instanciado em múltiplos componentes (*Models e Views)*. Já no FLUX, o dado é instanciado em apenas um componente (*Store)* e todos os outros componentes (*Stores* e *Views)* possuem apenas a referência desse dado, permitindo uma atualização dinâmica das *Views.*

5. MÉTODO

Este capítulo apresenta o método utilizado para desenvolver o protótipo, que foi utilizado nas análises comparativas entre os frameworks, além das métricas utilizadas na comparação.

Para realizar a análise comparativa, foi desenvolvido uma aplicação SPA simples com as características de *Create, Read, Update e Delete* (CRUD). Essa aplicação, foi replicada nos frameworks React, Angular e Vue, a fim de obtermos uma comparação correta entre eles.

A aplicação desenvolvida é um cadastro de pessoas contendo dois campos Nome e Email. A tela inicial irá exibir uma lista de cadastros previamente carregados no sistema. Também é possível editar essa lista de cadastro, usando um formulário, ou inserir um novo cadastro.

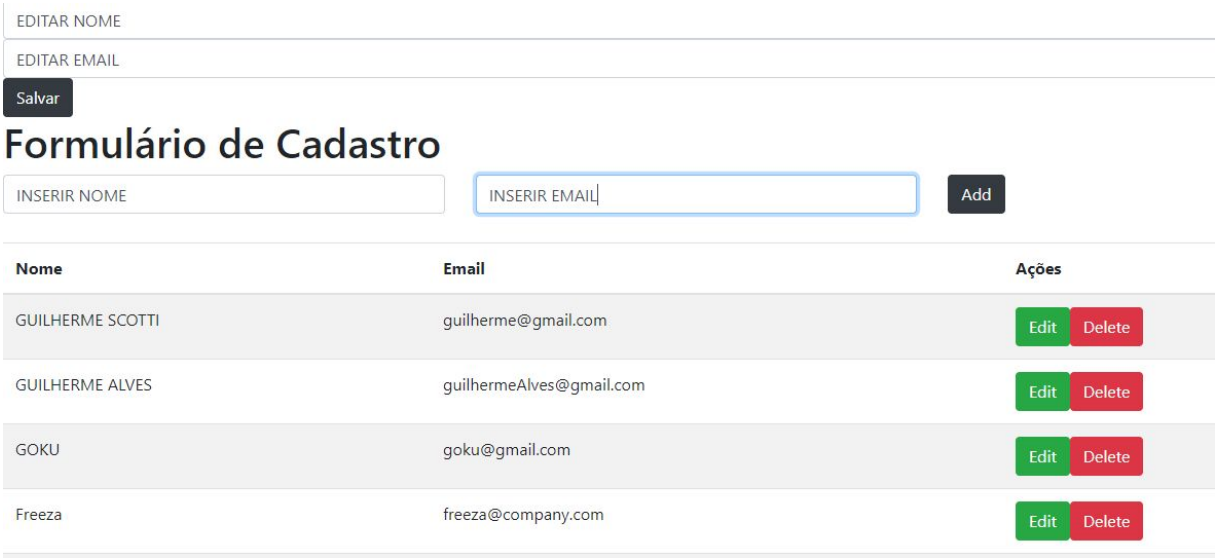

Figura 11 - Aplicação SPA desenvolvida.

Todas as aplicações foram desenvolvidas utilizando apenas JavaScript e recursos dos frameworks. Além disso, foi utilizada uma biblioteca de componentes CSS, chamada *Bootstrap*.

Neste trabalho, utilizou-se um método quantitativo para realizar a comparação entre os frameworks. O primeiro passo, antes de propor os métodos para a análise comparativa, foi realizado uma revisão bibliográfica para entender as similaridades e diferenças entre os frameworks, apontando as características que serão comparadas e por fim realizar a comparação prática dos protótipos na compatibilidade entre navegadores, tempo de carregamento e tamanho do pacote de código compilado..

Em uma aplicação Web tradicional os testes de performance são realizados normalmente no *server-side* da aplicação. Porém, como o escopo do trabalho são as aplicações SPAs, serão analisados os pontos cruciais que impactam a performance de uma SPA, sendo eles a compatibilidade entre navegadores, tempo de carregamento e tamanho do pacote de código compilado.

A compatibilidade entre navegadores é uma característica importante nas aplicações SPAs, uma vez que no início da popularização dessas aplicações, a incompatibilidade entre navegadores era o principal fator que impedia a utilização mais abrangente dessa arquitetura. Dessa forma, será analisado a versão mínima compatível entre os navegadores utilizando as informações disponíveis nas documentações dos frameworks.

O código das aplicações SPAs são compilados em conjuntos de arquivos JavaScript e esse conjunto de arquivo é enviado do servidor para o navegador do cliente. Assim, é crucial que o tamanho desse conjunto de arquivos seja o menor possível, possibilitando um menor uso da banda de internet o que é crucial para aplicações Web mobiles. Para realizar essa medida, foi realizado a compilação nativa dos frameworks para gerar o conjunto de arquivos de JavaScript, em seguida será mensurado o tamanho desse conjunto de arquivos.

Segundo Ninio (2006) a curva de aprendizagem consegue mensurar o esforço necessário para absorver conceitos e tecnologias. Um fator importante para potencializar a curva de aprendizagem dos frameworks é a comunidade que os cerca, uma vez que, caso o framework não tenha uma comunidade ativa que responde dúvidas o aprendizado será demorado e exaustivo, pois será despendido mais tempo para entender o framework do que para efetuar o desenvolvimento do código. Outro fator importante, é o nível de conhecimento do desenvolvedor com a linguagem JavaScript, uma vez que todos os frameworks são baseados em JavaScript.

Para verificar a curva de aprendizado dos frameworks, foi analisado o quão ativa é a comunidade que os cerca, para ter a noção da quantidade de pessoas que utilizam esse framework e a possibilidade de apoio em caso de dúvidas ou bugs durante o desenvolvimento.

6. RESULTADOS

A característica mais marcante dos frameworks analisados é o padrão de projeto adotado. Uma vez que ele determina a localização dos componentes responsáveis pela atualização do estado dos dados, além da direção do fluxo de dados (unidirecional ou bidirecional). No padrão Angular, o fluxo de dados bidirecional permite que um dado seja instanciado em múltiplos componentes. Já no React e Vue, o dado é instanciado em apenas um componente e todos os outros componentes possuem apenas a referência desse dado, permitindo uma atualização dinâmica das *Views*.

Todas as características e métricas citadas acima foram organizadas de forma comparativa, possibilitando realizar a análise comparativa, conforme Tabela 1.

Características	**React**	**Vue**	**Angular**
Fluxo de dados	Unidirecional	Bidirecional	Bidirecional
Padrão de arquitetura	FLUX	-	MVC
Tamanho do pacote de arquivos JavaScripts compilados	684 Kb	520 Kb	508 Kb

Suporte mínimo ao navegador de internet (ECMAScript 2015)	EDGE:14 Chrome:58 Firefox:54 Safari:10	EDGE:14 Chrome:58 Firefox:54 Safari:10	EDGE:14 Chrome:58 Firefox:54 Safari:10

Tabela 1 - Comparação das características principais dos frameworks.

Apesar de as características principais dos frameworks JavaScripts serem genéricas, a forma como cada uma é executada pode variar, cabendo ao desenvolvedor escolher qual irá satisfazer melhor as necessidades da sua aplicação. Assim, dependendo da implementação de uma arquitetura de dados bidirecional, um sistema de grande escala poderá estar vulnerável a falhas de escalabilidade, uma vez que a adição de uma nova função pode ocasionar um efeito em cascata de atualizações.

Após análise da documentação dos frameworks, verificou-se que todos utilizam JavaScript versão ECMAScript 2015. Dessa forma, a exigência de compatibilidade de navegadores está contemplada nas versões mais atuais dos navegadores. Vale ressaltar que, existem bibliotecas não oficiais disponíveis na comunidade de desenvolvedores, que possibilitam compilar o código das aplicações para versões anteriores dos navegadores, como Internet Explorer 9.

Verifica-se que o fluxo de dados bidirecional (*Two Way Data Binding*) é utilizado apenas pelo Angular, que é o framework mais antigo estudado, e os demais frameworks utilizam o fluxo unidirecional de dados (*One Way Data Binding*). Essa mudança de paradigma nos frameworks novos sinaliza que os futuros frameworks devem seguir essa nova arquitetura.

Tendo em vista que não há grandes variações entre os tamanhos dos arquivos de código compilado, sendo a variação de 140 Kb insignificante para o influenciar na performance do framework, uma vez que para gerar modificações significativas no tamanho do código compilado, seria necessário o desenvolvimento de um protótipo robusto, com mais funções, o que é inviável devido ao tempo de desenvolvimento desse trabalho.

Uma forma de analisar as comunidades que cercam os frameworks é mensurar estatisticamente os sites especializados em ajuda aos desenvolvedores e versionamento de código. Foram analisados o site GitHub que é uma comunidade de versionamento de código *open source* e o Stack Overflow, que é o principal fórum de dúvidas de desenvolvimento internacional existente.

Para o GitHub, foram analisados as métricas de seguidores (*watchers*), utilização do framework por repositórios públicos e cópias do código original do framework (*fork*), conforme Tabela 2. Já para o Stack Overflow, foi utiliza a ferramenta *Stack Overflow Trends* nativa da plataforma, que obteve os resultados da Figura 12.

Métricas GitHub	React	Vue	Angular
Repositórios que utilizam o framework	2.124.365	898.101	-
Seguidores (*watchers*)	6.644	5.930	3.278
Cópia do código (*fork*)	24.329	20.513	13.430

Tabela 2 - Métricas das comunidades no GitHub.

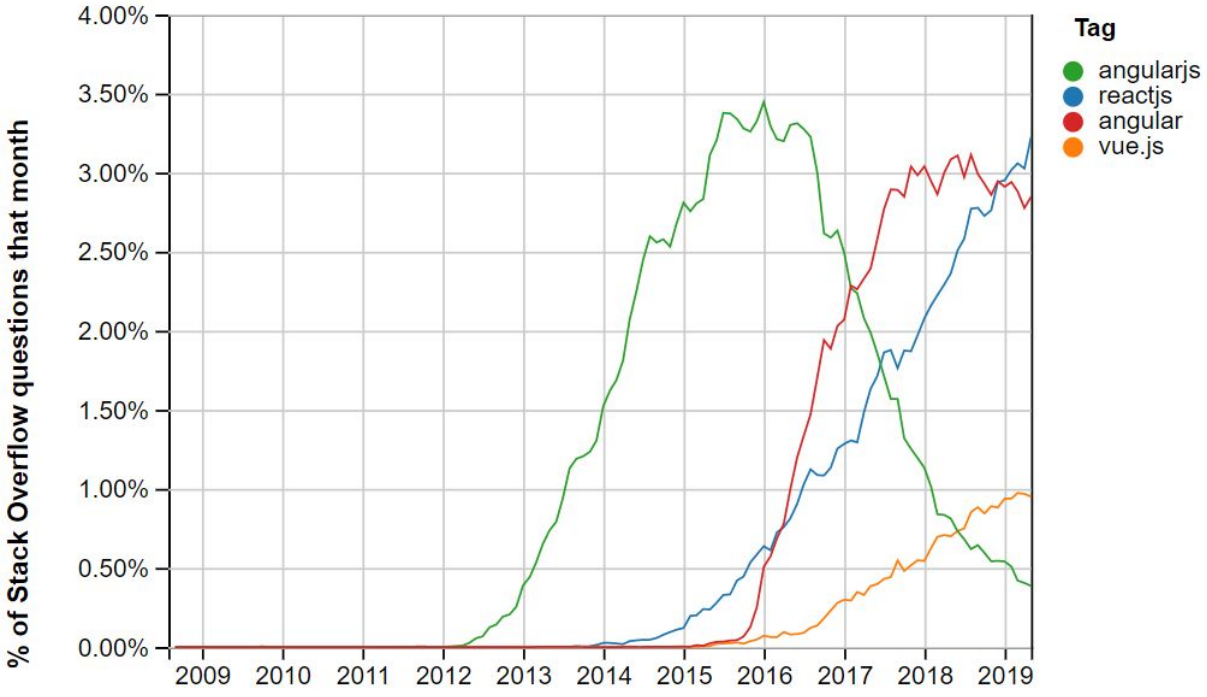

Figura 12 - Comparação do número de dúvidas registradas por framework.

Fonte: STACK OVERFLOW 2019

Conclui-se que, o Angular foi a maior comunidade entre os frameworks durante muitos anos, porém depois da reestruturação do framework em 2016, foi criado espaço para o desenvolvimento das outras comunidades como a do React e Vue. Atualmente, a comunidade do Angular continua grande, porém a maior e mais ativa comunidade de desenvolvedores é a do React, possibilitando uma menor curva de aprendizado do framework, devido a grande quantidade de tutoriais, e desenvolvedores dispostos a responderem dúvidas nos fóruns.

O tempo de renderização foi analisado com o recurso *Performance* do Chrome DevTools, que mede diversas métricas de performance como o (*DOM Content Loaded)* tempo de carregamento completo da página, ou seja, quando a página está pronta para interação com o usuário (GOOGLE, 2019).

Os tempos de carregamento analisados pela ferramenta do Google variam por fatores como alocação de memória e threads ativas no processador (GOOGLE, 2019). Dessa forma, foram coletadas 10 amostras do tempo de carregamento das aplicações e calculada a média, conforme Tabela 2. Além disso, para simular um sistema mais complexo, foi aumentado o número de cadastros já existentes na aplicação de 10 a 2000 cadastros pré carregados, conforme Figura 12.

Para realizar a comparação do tempo de carregamento da aplicação SPA, foi utilizado a seguinte máquina como servidor:

- Windows 10 pro;
- Intel Core i7-7500U CPU @2.70 GHZ

- 8GB RAM
- Google Chrome 74

Número de cadastros pré carregados	React	Vue	Angular
10	190ms	170ms	185ms
100	342ms	306ms	297ms
500	498ms	446ms	416ms
1000	822ms	736ms	725ms
2000	1450ms	1298ms	1115ms

Tabela 3 - Tempo de carregamento da aplicação variando o número de cadastros pré carregados.

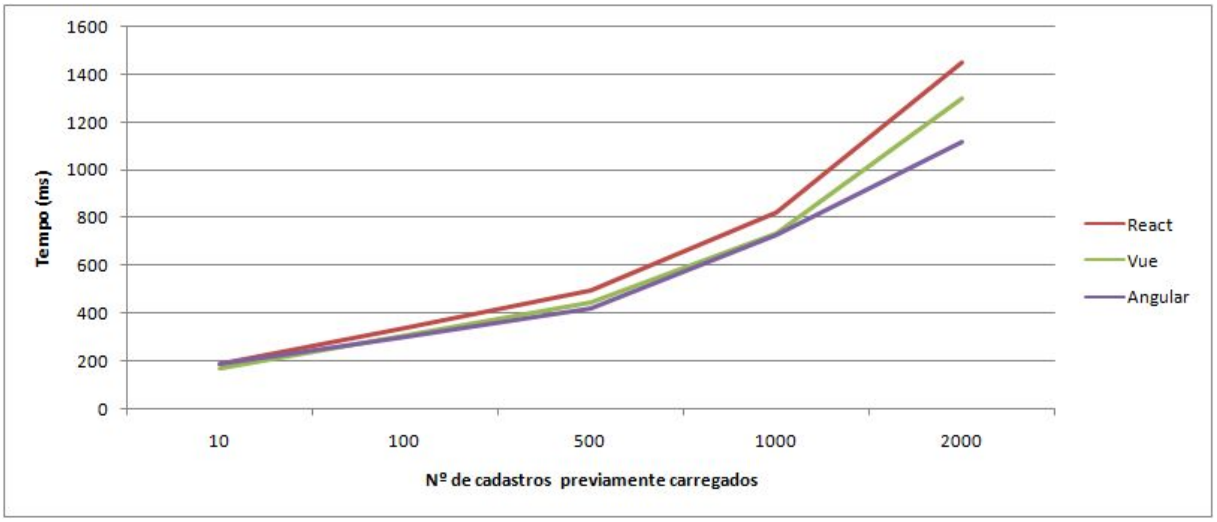

Figura 13 - Comparação do tempo de carregamento para 10-2000 cadastros pré carregados.

Observando o resultado da Figura 12, conclui-se que todos os frameworks JavaScript analisados possuem um bom tempo de resposta para aplicações SPA de pequeno porte.

Porém, o framework Angular apresentou os melhores resultados ao aumentar a complexidade da aplicação.

Conclui-se que, todos os frameworks analisados possuem similaridades na compatibilidade entre navegadores, e tamanho do pacote de arquivos compilados, uma vez que todos são baseados em JavaScript. Sendo assim, a métrica decisiva para a distinção dos frameworks foi o tempo de carregamento das aplicações, onde o Angular se destacou apresentando resultados em média melhores que os demais, principalmente após o aumento da quantidade de cadastros renderizados na tela inicial da aplicação.

7. CONCLUSÃO

Atualmente, o JavaScript é a melhor linguagem para se desenvolver uma aplicação SPA, porém levaram-se vários anos para que o JavaScript e as aplicações SPA se tornassem competitivas no mercado, ou possível de serem utilizadas e compatíveis a todos os navegadores.

Verificou-se que, a principal diferença entre os frameworks está no padrão de software adotado, o local onde é definida a localização dos componentes responsáveis pela atualização do estado dos dados, além da direção do fluxo de dados. No Angular, a bidirecionalidade dos dados permite que a informação seja instanciada em múltiplos componentes (*Models e Views)*. Já no FLUX (React e Vue), o dado é instanciado em apenas um componente (*Store)* e os demais (*Stores* e Views*)* possuem apenas a referência dessa informação, permitindo uma atualização dinâmica das *Views*.

Conclui-se que não há diferenças significativas entre os frameworks, tanto no tempo de carregamento, quanto no tamanho do arquivo final da Build ou a compatibilidade entre navegadores e isso se deve principalmente por serem frameworks oriundos da mesma linguagem. Porém, o fator decisivo para a análise comparativa dos frameworks foi o tempo de carregamento da página, onde o framework Angular se destacou apresentando os melhores resultados, mesmo após aumentar a carga inicial de dados da aplicação.

Portanto, apesar de as características principais dos frameworks JavaScripts serem genéricas, a forma como cada uma é executada pode variar, cabendo ao desenvolvedor escolher qual irá satisfazer melhor às necessidades da sua aplicação.

Propõe-se como sugestão para trabalhos futuros, estudar outras métricas importantes para a manutenibilidade do software, como performance e coesão da base de código, além de desenvolver uma aplicação mais robusta onde seria possível verificar as diferenças entre os frameworks.

8. REFERÊNCIAS

CHUAN, Y., WANG, H. Characterizing Insecure JavaScript Practices on the Web. In: 09 PROCEEDINGS OF THE 18TH INTERNATIONAL CONFERENCE ON WORLD WIDE WEB. Madrid, Espanha,2009,p. 964-965

LAVANYA, R., RAMACHANDRAN, V., e MUSTAFA, J. A Comparative Study on Internet Application Development Tools, in: INTERNATIONAL JOURNAL OF ENGINEERING SCIENCE AND TECHNOLOGY, 2010,p. 5452-5456

MIKOWSKI, M. e POWELL, J, single page web applications. Manning Publications, 2013.

MAURYA, L. e SMARAK, S. Maintainability Assessment Of Web Based Application, in: *Journal of Global Research in Computer Science*., vol. 3, no. 7, p. 30-35, Jul. 2012.

MESBAH, A. e DEURSEN, A. Migrating Multi-page Web Applications To Single-page AJAX Interfaces. in: *Software Maintenance and Reengineering. CSMR* ’07. 11th European Conference, 2007, p.181–190.

VOUTILAINEN, Jaakko. Evaluation of Front-end JavaScript Frameworks for Master Data Management Application Development. 50p. Information Technology – Metropolia University of Applied Sciences, 2017.

SEGAL, B. A Short History Of Internet Protocols at CERN, 1995. Disponível em: <https://bit.ly/2wqQPzr> Acesso em Maio de 2019

W3TECHS, Usage of client-side programming languages for websites, 2019. Disponível em: <https://bit.ly/2Mh1esa> Acesso em Maio de 2019.

SEVERANCE, Charles. JavaScript: Designing a Language in 10 Days. 8p. Information Technology – University of Michigan, Michigan, 2012.

MALMSTRÖM , Jaakko. Structuring modern web applications. 80p. Masters of Science in Engineering – NADA, Fev de 2014.

DUARTE, Nuno Filipe Brandão. Frameworks e Bibliotecas JavaScript. 70p. Mestrado em Engenharia Informática, Especialização em Arquiteturas Sistemas e Redes – Instituto superior de engenharia do porto (ISEP) , porto, Outubro de 2015.

SOMMERVILLE, Ian. Software engineering, 10th ed., Marcia Horton, M. Hirsch, and M. Goldstein, Eds. Pearson Education, Inc., 2016.

BAJAJ, Pankaj. Overview of Single Page Application (SPA), 2014. Disponível em: <https://bit.ly/2HJLN7F> Acesso em Maio de 2019.

HOTFRAMEWORKS, Measuring web framework popularity, 2018. Disponível em: <https://hotframeworks.com/> Acesso em Maio de 2019.
STATEOFJS, Overall Front-end Frameworks results, 2018. Disponível em: <https://2018.stateofjs.com/front-end-frameworks/overview//> Acesso em Maio de 2019.

JOHNSON, Ralph E. Components, frameworks, patterns. in: *Proceedings of the 1997 Symposium on Software Reusability*, SSR ’97, New York,
NY, USA, 1997. p. 10–17.

BUSCHMANN, et al. Pattern-Oriented Software Architecture-A System of Patterns. Wiley and Sons. 1996.

GAMMA, Erich. et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994, 395p.

MARTTILA, Riku . HANDLING UNIDIRECTIONAL DATA FLOW IN A REACT.JS APPLICATION. 04 de maio de 2016. 44p. Tese - Tampere University Of Technology.

GRENMYR, David. MAGNUSSON, Erik. An Investigation of Data Flow Patterns Impact on Maintainability When Implementing Additional Functionality. 2016, Tese – Linnéuniversitetet, Växjö – Suiça.

ALEXANDER C. The Timeless Way of Building. Oxford University Press. 1979

GLEEN E., KRASNER e STEPHEN. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-OrientedProgramming, 1(3):26–49, 1988.

FACEBOOK, Flux Documentation 2013. Disponível em: <https://facebook.github.io/flux/docs/in-depth-overview.html>. Acessado em dezembro de 2017.

ANGULAR, Angular Documentation 2019. Disponível em: <https://angular.io/docs>. Acessado em maio de 2019.

VUE, Vue Documentation. 2019. Disponível em: <https://vuejs.org>. Acessado em maio de 2019.

GOOGLE, Crhome Dev Tools, 2019. Disponível em: <https://developers.google.com/web/tools/chrome-devtools/> . Acessado em maio de 2019.

NINIO, Anat. Language and the learning curve: A new theory of syntactic development.Language and the learning curve: A new theory of syntactic development. Addison-Wesley, 2006.

STACK OVERFLOW, Stack Overflow Trends. Disponível em: <http://insights.stackoverflow.com/trends>. Acessado em julho de 2019.