INTERFACE TECNOLÓGICA

VISÃO GERAL SOBRE O GERENCIAMENTO DE ESTADO NO VUE.JS COM A BIBLIOTECA VUEX

OVERVIEW ABOUT STATE MANAGEMENT IN VUE.JS WITH THE VUEX LIBRARY

Lucas Galhardo Lima – lima.galhardo@gmail.com
Erick Eduardo Petrucelli – erick.petrucelli@fatectq.edu.br
Felipe do Espírito Santo – felipe.santo@fatectq.edu.br
Faculdade de Tecnologia de Taquaritinga (FATEC) – SP – Brasil

RESUMO

Vue.js é um *framework* JavaScript progressivo para construção de interfaces visuais para o usuário usando JavaScript. Vuex é uma das bibliotecas oficiais deste *framework*, uma adaptação do Vue.js para o que é chamado de conceito Flux, que tem como objetivo o gerenciamento de estado entre componentes dentro do projeto. Juntos, trabalham para que o desenvolvedor possa administrar, de forma fluida e organizada, a comunicação de dados dentro do projeto, armazenando-os de forma centralizada, permitindo acessá-los dentro dos diversos componentes do projeto. O presente artigo inicia abordando brevemente a história do Vue.js e seu funcionamento. Em seguida, apresenta a utilização da Vuex em um pequeno projeto de exemplo, demonstrando como o Vuex é configurado, como funciona sua *store* e seus principais elementos, bem como realizar a organização/modularização de código com Vuex. Por fim, o artigo aborda alguns pontos positivos e negativos de utilizar este tipo de arquitetura.

Palavras-chave: Vue.js. Vuex. Gerenciamento de Estado. Desenvolvimento Front-End.

ABSTRACT

Vue.js is a progressive JavaScript framework for constructing visual interfaces using JavaScript. Vuex is one its official libraries, an adaptation of Vue.js to what is called the Flux concept, which aims to manage state between components within the project. Together, they work so that the developer can manage, in a fluid and organized way, the communication of data within the project, storing them centrally, allowing to access them within the various components of the project. This article begins by briefly discussing the history of Vue.js and its operation. Next, it presents the use of Vuex in a small example project, demonstrating how Vuex is configured, how its store works along with its main elements, as well as performing the organization/code modularization with Vuex. Finally, the article addresses some positive and negative aspects of using this type of architecture.

Keywords: Vue.js. Vuex. State Management. Front-End Development.

1 INTRODUÇÃO

No desenvolvimento *front-end* moderno, *frameworks* JavaScript são fortemente baseados em "componentizar" as interfaces, não demora muito até que o desenvolvedor possa passar por muitas complicações, principalmente devido a múltiplos dados de estado da aplicação espalhados entre diversos componentes e interações entre eles (VUE.JS, 2019).

Em 2014, o Facebook propôs a arquitetura Flux, com um fluxo de dados unidirecional dos componentes para um gerenciador central de estado da aplicação (FACEBOOK INC., 2014). Tal arquitetura se transformou na biblioteca Redux para o React do Facebook.

> Quando criamos um SPA complexo, não podemos lidar com os dados da aplicação apenas criando variáveis globais, pois serão tantas que chega um momento no qual não é possível mais controlá-las. Além da eventual "bagunça", em aplicações maiores, manter o correto estado de cada ponto do sistema torna-se um problema para o desenvolvedor (SCHMITZ; GEORGII, 2016, p. 93).

Essa "bagunça" é o principal problema que a arquitetura Flux se propõe a resolver, e tenta fazer isso da forma simples e organizada.

Assim, com o advento Vue.js, surgiu o Vuex para abordar tal problema. Curiosamente, como se trata da mesma inspiração arquitetural, é possível utilizar Redux com Vue.js também. Mas o Vuex surgiu mais adaptado às especificidades deste *framework*, sendo uma alternativa mais natural e produtiva (KYRIAKIDIS; MANIATIS; YOU, 2017). Como tais autores comentam, React e Vue compartilham muitas semelhanças: como manipulam os componentes em memória com DOM Virtual, como proporcionam reatividade de dados para a interface e como focam em oferecer um pequeno conjunto de funcionalidades, deixando para bibliotecas de apoio abordarem problemas comuns, como o gerenciamento de estado.

Assim, o presente artigo busca introduzir o Vue.js e o Vuex como solução ao gerenciamento de estado para aplicações "componentizadas", demonstrando brevemente trechos de código de um projeto de exemplo.

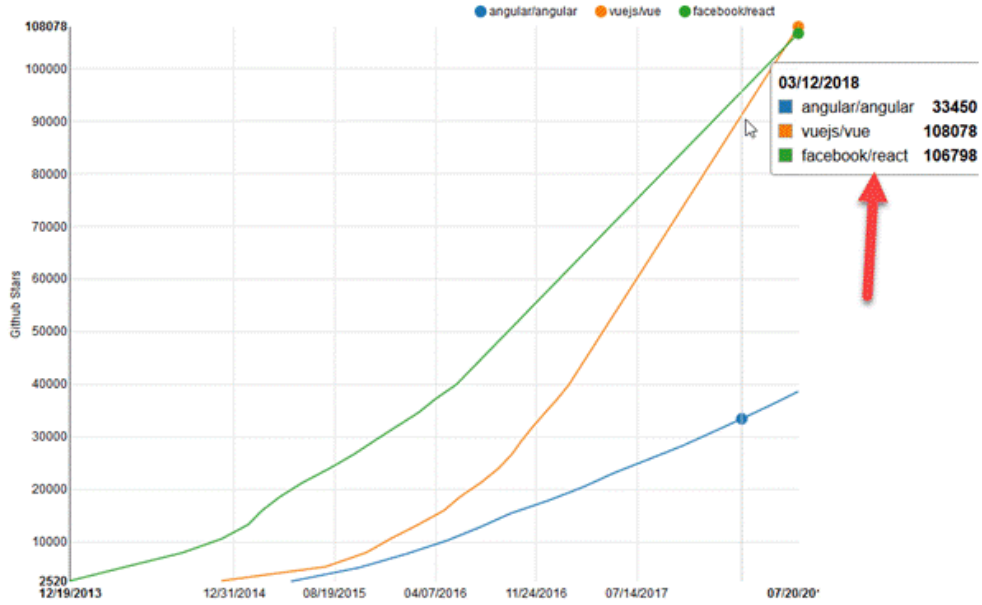
2 FUNDAMENTAÇÃO SOBRE VUE.JS

"Vue.js é um *framework* progressivo para a construção de interfaces de usuário. Ao contrário de outros *frameworks* monolíticos, Vue foi projetado desde a sua concepção para ser adotado incrementalmente" (VUE.JS, 2019). Ainda em sua documentação oficial, é dito que "a biblioteca principal é focada exclusivamente na camada visual (*view layer*), sendo fácil adotar e integrar com outras bibliotecas ou projetos existentes".

Conforme é possível observar em You (2014), Vue.js surgiu como um projeto particular e cresceu para se tornar um *framework* popular e de fácil compreensão. Vue.js foi criado depois de seu autor, Evan You, trabalhar para a Google utilizando o *framework* AngularJS, tendo sido fortemente inspirado por este, mas, também, inspirado em muitos conceitos que estavam surgindo naquele período no React. Ao perceber que não existia nenhuma biblioteca de prototipagem rápida, Evan decidiu criar sua própria, com a proposta de ser simples, flexível, e altamente escalável (GALDINO, 2017).

Hoje, o Vue.js se encontra em sua segunda versão (com a terceira em desenvolvimento, mas sem data definida de lançamento enquanto este artigo é escrito). Está em constante crescimento na comunidade e recebendo cada vez mais recursos. Em meados de 2018, passou os dois principais *frameworks* alternativos, Angular e React, em número de estrelas em seu repositório de código-fonte na plataforma GitHub, um indicativo de como os desenvolvedores gostam de trabalhar com ele, conforme é possível observar na Figura 1.

Figura 1 – Número de estrelas no GitHub entre Vue.js, React e AngularJS



Fonte: Spec India (2019).

Outro indício desta boa reputação encontra-se em Galdino (2017), citando uma pesquisa realizada entre diversos desenvolvedores JavaScript, onde 89% deles aprovaram o Vue.js.

Dentre as principais características deste *framework*, o Vue.js permite que você reutilize componentes de uma forma bem simples em qualquer parte de sua aplicação, sem a necessidade de criar modelos especiais ou coleções. Basta criar um componente simples dentro de um *template* e, depois, importá-lo onde deseja utilizá-lo em seu projeto (KYRIAKIDIS;

MANIATIS; YOU, 2017). Ao mesmo tempo em que criar e reutilizar componentes é simples, é uma das principais causas do problema de gerenciamento de estado da aplicação.

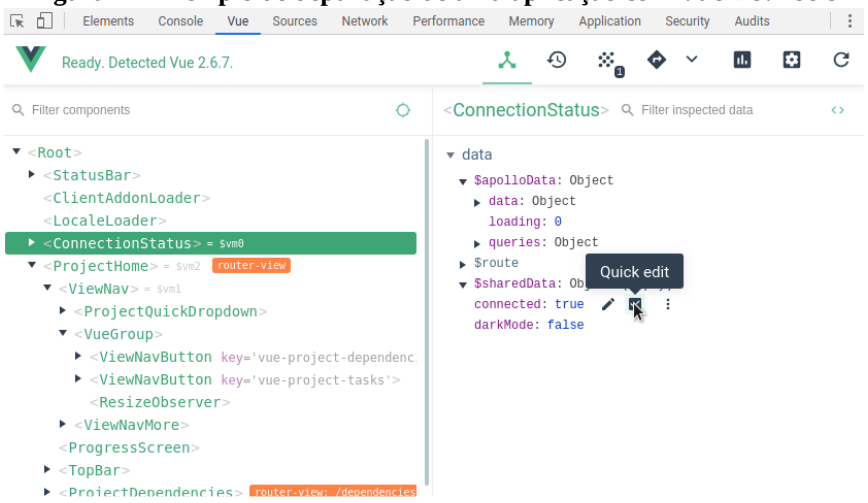
3 FUNDAMENTAÇÃO SOBRE VUEX

De acordo com a documentação oficial do Vuex (2019), trata-se de “um padrão de gerenciamento de estado + biblioteca para aplicativos Vue.js”.

Segundo Schmitz e Georgii (2016), uma forma simples de compreender o Vuex dentro do um projeto é através de seu conceito de *store*. A Vuex serve como uma “loja” de dados para sua aplicação, permitindo que todos os componentes tenham acesso a um determinado estado (ou *state*), pois todos os estados ficam centralizados dentro dessa loja.

Vuex também permite uma boa organização do projeto, permitindo ainda modularizar a *store*, para organizar quais estados pertencem a uma determinada entidade da aplicação. Vuex também se integra com a extensão para navegadores do Vue.js, chamada Vue DevTools, exibida na Figura 2, oferecendo recursos interessantes, como a depuração através do histórico de estado, também conhecida como *time travel* (VUEX, 2019).

Figura 2 – Exemplo de depuração de uma aplicação com Vue DevTools

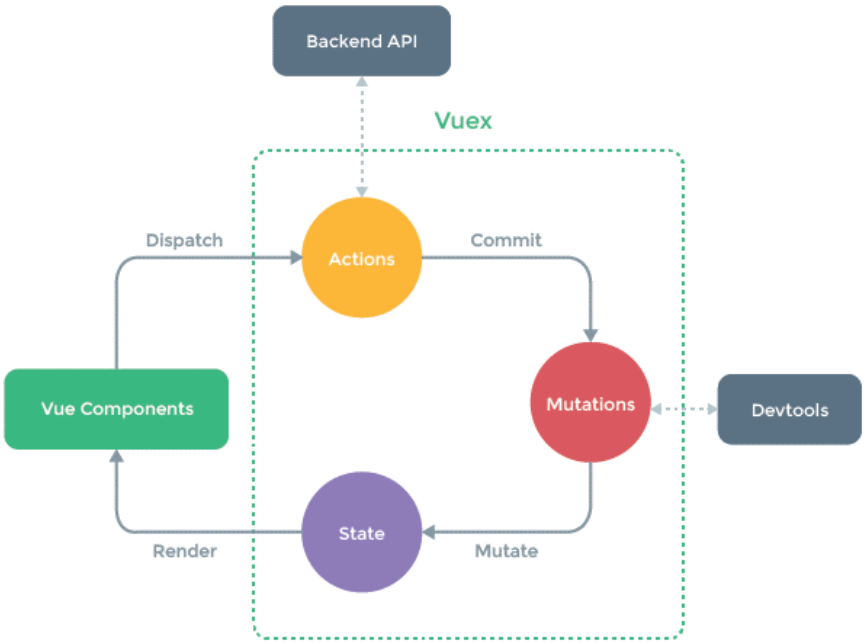


Fonte: Vue DevTools (2019).

Segundo Vuex (2019), a arquitetura empregada pelo Vuex possui o seguinte fluxo: os componentes podem disparar ações (*actions*), que por sua vez podem ou não fazer requisições a APIs no *back-end* (opcionalmente, dependendo da aplicação). Tal ação pode realizar um *commit* nos dados, o que irá chamar uma mutação (*mutation*). Neste momento, o evento é lançado no Vue DevTools e poderá ser observado. Por fim, a *mutation* efetivamente armazena

o dado no *state*. Esse *state* poderá ser acessado por meio de um *getter* em qualquer componente da aplicação. O resumo desta arquitetura pode ser observado na Figura 3, a seguir.

Figura 3 – Elementos arquiteturais do Vuex



Fonte: Vuex (2019).

É relevante ressaltar problemas que um desenvolvedor normalmente enfrenta ao tentar gerenciar estados sem a Vuex, conforme pontua a documentação oficial:

> Para o problema um, passar propriedades pode ser entediante para componentes profundamente aninhados e simplesmente não funcionam para componentes irmãos. Para o problema dois, frequentemente nos encontramos recorrendo a soluções como alcançar referências diretas da instância pai/filho ou tentar alterar e sincronizar várias cópias do estado por meio de eventos. Ambos os padrões são frágeis e levam rapidamente a um código não-sustentável. (VUEX, 2019)

Vuex também possui um grande potencial de escalabilidade em projetos grandes, mas sua utilização vem com o custo de mais conceitos teóricos e códigos potencialmente mais repetitivos. É uma escolha de prós e contras entre produtividade de curto e longo prazo.

4 Exemplo de Utilização

O projeto de exemplo desenvolvido pelo autor deste artigo, trata-se de uma aplicação que realiza envio de notificações *push* (*push notifications*). Entretanto, nem a lógica de envio de notificações e nem a criação dos componentes visuais são o foco neste trabalho. Apenas para

introduzir a utilização do Vuex, a Figura 4 exibe um trecho do componente que faz o envio da notificação *push*, utilizando a biblioteca de gerenciamento de estado em parte da lógica.

Figura 4 – Componente de envio de notificação *push* usando Vuex

```
<script>
  import {mapActions} from 'vuex'
  export default {
    data(){
      return {
        novosms:{
          phone:'',
          text:'',
          isPush: false,
          captura: false,
        }
      }
    },
    methods: {
      ...mapActions(['sendSms']),
      async send () {
        let resp = await this.sendSms(this.novosms)
        this.$eventHub.emit('snackBar', {message:resp.mensagem})
        this.novosms = {
          phone:'',
          text:'',
          isPush: false,
          captura: false,
        }
      }
    }
  }
</script>
```

Fonte: próprio autor.

Para iniciar a utilização do Vuex, antes foi necessário configurá-lo no projeto Vue.js que pode ser feito instalando a biblioteca. A forma mais comum de fazer isso é utilizar o comando **`npm install vuex –save`** em um terminal na pasta do projeto. Em seguida, a utilização mais simples seria importar o Vuex no arquivo "*main.js*" do projeto e, utilizando o comando **`Vue.use(Vuex)`**, atribuir o Vuex à árvore de bibliotecas carregadas pelo Vue.js. Em seguida, define-se um novo armazenamento através do comando **`const store = new Vuex.Store(VuexStore)`** e se atribui este novo *store* à construção da instância Vue.js, através de um comando como: **`new Vue({el: '#app', store, render: h=>h(App)})`**. Todos estes passos estão claramente apresentados na documentação oficial, em Vuex (2019).

Concluída a configuração inicial, é possível ter uma pasta "*store*" no projeto *e, dentro dela,* ter vários módulos separados. Para isso, utiliza-se uma pasta "*modules*" com um único arquivo denominado "*index.js*", o qual foi implementado conforme a Figura 5 a seguir.

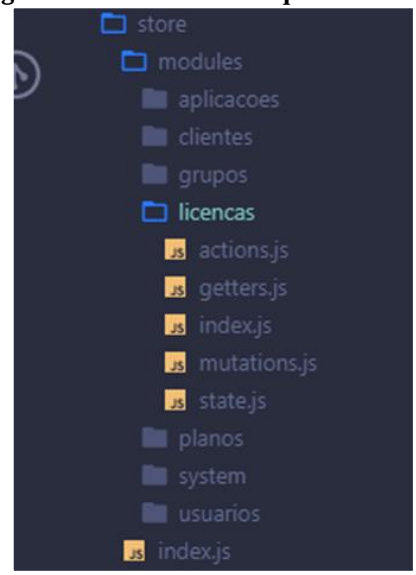
Figura 5 – Módulos no *index.js* da *store* Vuex

```
import system from './modules/system/index.js'
import root from './modules/root/index.js'
import creditos from './modules/creditos/index.js'
import dashboard from './modules/dashboard/index.js'
import mailing from './modules/mailing/index.js'
import mensagens from './modules/mensagens/index.js'
import torpedos from './modules/torpedos/index.js'
import usuario from './modules/usuario/index.js'
import protaxi from './modules/protaxi/index.js'
import push from './modules/push/index.js'
import pxtalk from './modules/pxtalk/index.js'
import zendesk from './modules/zendesk/index.js'

import keepAlive from './keepAlive'

export default {
  modules: {
    system,root,creditos,dashboard,mailing,mensagens
    },
    plugins: [keepAlive]
}
```

Fonte: próprio autor.

Tais comandos *import* estão sendo realizados para que a *store* "enxergue" os módulos e possa realizar a comunicação com eles e entre eles. Cada caminho corresponde a um arquivo de módulo criado na pasta "*modules*", com seu respectivo arquivo "*index.js*". Além deste arquivo principal, cada módulo normalmente possui os arquivos "*actions.js*", "*getters.js*", "*mutations.js*" e "*state.js*". Embora esta estrutura possa ser modificada dependendo das demandas de cada projeto, seguir desta forma deixa mais organizado, por seguir a nomenclatura da própria arquitetura empregada pelo Vuex. A Figura 6 apresenta os arquivos e um dos módulos do projeto de exemplo.

Figura 6 – Módulo com arquivos internos



Fonte: próprio autor.

Enfatiza-se que a modularização no Vuex é algo completamente opcional, mas neste projeto foi muito utilizada, por ser uma aplicação com diversos domínios de entidades bem definidos. A seguir, na Figura 7, observa-se o código de "*actions.js*" um dos módulos.

Figura 7 – *Actions* de um módulo Vuex

```
export default {
    async getUltimosSms({commit, getters}) {
        let {data} = await getters.getApi.get("message")
        commit('SET_ULTIMAS_MENSAGENS', data)
    },

    async sendSms ({commit, getters}, sms) {
        let {data} = await getters.getApi.post("message", sms)
        return data
    },

    async arquivos_saveArquivo ({commit, getters}, formData) {
        const config = { headers: { 'Content-Type': 'multipart/form-data' } };
        let { data } = await getters.getApi.post("message/file", formData, config)
        return data
    }
}
```

Fonte: próprio autor.

Neste trecho, há a implementação das ações de um módulo, inclusive utilizando recursos do JavaScript moderno (das especificações ECMAScript 2015 e posteriores), como *async/await* para execução assíncrona das ações. Neste exemplo, a ação realização *commit* de dados através de *mutation* e, também utilizar *getters* mapeados para retornarem o endereço do *back-end* da aplicação, um mecanismo de utilização de APIs com Vuex também opcional. Portanto, a seguir, na Figura 8, observa-se a breve implementação de uma mutação deste módulo:

Figura 8 – *Mutations* de um módulo Vuex

```
export default {
    'SET_ULTIMAS_MENSAGENS': (state, data) => {
        state.ultimasMensagens = data
    }
}
```

Fonte: próprio autor.

Neste exemplo, observa-se a *mutation* que foi acionada pelo *commit* da *action* exibida na Figura 7. Normalmente, uma *mutation* recebe dois parâmetros de entrada (que se encontram dentro dos parênteses): *state*, para permitir acesso aos dados do estado que se deseja modificar, e *data* para receber os próprios dados que devem ser operados na mutação. Um *state* trata-se, na verdade, de um simples objeto JavaScript e cada módulo pode ter o seu.

A seguir, na Figura 9, observa-se um método *getter* de exemplo relativo a este mesmo módulo, o qual poderia ser utilizado por diversos componentes da aplicação, para que pudessem acessar os dados deste *state* sem precisar conhecer os detalhes internos de sua estrutura.

Figura 9 – *Getters* de um módulo Vuex

```
export default {
    getUltimasMensagens: state => state.ultimasMensagens
}
```

Fonte: próprio autor.

Na Figura 10, é possível observar a utilização de um método *getter* em meio a código de um componente, através da função de apoio *mapGetters* que poderia ser importada no início do código do componente, com o comando **`import { mapGetters } from 'vuex'`**.

Figura 10 – *mapGetters* em utilização no código de um componente

```
computed:{
    ...mapGetters(['getChildren']),
    getTabs()
    {
        return this.getChildren.filter( m =>
            this.tabs.find( t =>
                t == m.rota && m.acoes.filter( a =>
                a.status == true ).length > 0
            )
        )
    }
},
```

Fonte: próprio autor.

Vale ressaltar que o *mapGetters* recebe um *array* contendo os nomes dos métodos *getter* que forem necessários para o componente, independente do módulo em que ele se encontra. Para os desenvolvedores que preferirem deixar um pouco de lado a organização, em troca de uma codificação mais direta, também é possível acessar um método *getter* diretamente a partir do código de um componente, por exemplo: **`this.$store.getters.getExemplo`**. Embora não seja uma forma de utilização ruim, reduz um pouco a organização que o Vuex oferece.

De forma similar, também é possível invocar os métodos das *actions* a partir de qualquer componente desejado, de duas maneiras: utilizando o método de apoio *mapActions*, ou acessando diretamente a ação desejada através do método *dispatch* existente na *store*, por exemplo: **`this.$store.commit('listaChamados')`**. A Figura 11 demonstra a primeira

abordagem, utilizando *mapActions* para carregar todas as ações desejadas no código de um componente e, logo depois, invocando algumas destas ações.

Figura 11 – *mapActions* nos *methods* de um componente

```
methods: {
    ...mapActions(['listaChamados', 'listaGrupos', 'getGroupsManager'])
},
mounted()
{
    this.tabs = this.$route.meta.subroutes
    this.$router.push(`/${this.getTabs[0].rota}`)
    this.listaChamados()
    this.getGroupsManager()
    //this.listaGrupos()
},
```

Fonte: próprio autor.

Novamente, ressalta-se que estes códigos, extraídos de um projeto real, apresentaram o fluxo comum de utilização dos elementos da arquitetura do Vuex, bem como um cenário de organização da aplicação através de *modules*. Entretanto, algumas variações nesta organização são válidas, como apresentado por Vuex (2019) e Schmitz e Georgii (2016).

5 CONSIDERAÇÕES FINAIS

Após introduzir brevemente o Vue.js e o problema do gerenciamento de estados em aplicações com *frameworks front-end* reativos, este trabalho apresentou conceitos fundamentais sobre o Vuex e demonstrou como emprega-lo em um projeto real com vários módulos.

A partir do exposto, observou-se que o Vuex efetivamente pode resolver o problema do gerenciamento de estados através de uma arquitetura consistente. Ainda traz benefícios como a fácil identificação de quais componentes estão modificando quais dados no *state*, inclusive permitindo a depuração através de *time travel*. Por outro lado, visualiza-se certa repetição de códigos, principalmente caso seja necessária a utilização de um mesmo módulo Vuex em componentes distintos. Em projetos grandes, com inúmeros módulos, pode se tornar confuso.

Ainda assim, a organização e a alta escalabilidade do Vuex são qualidades muito positivas, que o tornam bastante atrativo. O Vuex pode, até mesmo, ser utilizado em conjunto com outros *frameworks*, como o React, embora não seja um cenário comum e não tenha sido abordado neste trabalho. Assim, conclui-se que a utilização do Vuex é quase inquestionável em aplicações Vue.js com mais do que apenas alguns poucos componentes.

REFERÊNCIAS

FACEBOOK INC. **Flux**: Application Architecture For Building User Interfaces. Flux Documentation, 2014. Disponível em: <https://facebook.github.io/flux>. Acesso em: 12 abr. 2019.

GALDINO, F. **Vue.js Tutorial**. DevMedia, 2017. Disponível em: <https://www.devmedia.com.br/vue-js-tutorial/38042>. Acesso em: 10 fev. 2019.

KYRIAKIDIS, A; MANIATIS, K; YOU, E. **The Majesty Of Vue.js 2**. LeanPub, 2017

SCHMITZ, D; GEORGII P. D. **Vue.js na Prática**. LeanPub, 2016.

SPEC INDIA. **React vs Angular vs Vue.js**: A Complete Comparison Guide. Spec India Blog, 2018. Disponível em: <https://medium.com/front-end-weekly/react-vs-angular-vs-vue-js-a-complete-comparison-guide-d16faa185d61>. Acesso em: 12 abr. 2019.

VUE DEVTOOLS. **Browser DevTools Extension for Debugging Vue.js Applications**. Vue DevTools Repository, 2019. Disponível em: <https://github.com/vuejs/vue-devtools>. Acesso em: 12 abr. 2019.

VUE.JS. **Gerenciamento de Estado**. Guia do Vue.js, 2019. Disponível em: <https://br.vuejs.org/v2/guide/state-management.html>. Acesso em: 09 fev. 2019.

_____. **Introdução**. Guia do Vue.js, 2019. Disponível em: <https://br.vuejs.org/v2/guide>. Acesso em: 09 fev. 2019.

VUEX. **O Que é Vuex?** Documentação do Vuex, 2019. Disponível em: <https://vuex.vuejs.org/ptbr>. Acesso em: 09 fev. 2019.

YOU, E. **First Week of Lauching Vue.js**. Evan You Blog, 2014. Disponível em: <https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project>. Acesso em: 09 fev. 2019.