Artículo de investigación

Un acercamiento teórico y práctico a la implementación de Flux

Nazario Luis Ayala Frasnelli[1*] y Antonio David Ruiz Diaz Medina[2]

[1, 2] Facultad de Ciencias y Tecnología, Universidad Nacional de Canindeyú, Paraguay.

*Autor correspondiente: Nazario Luis Ayala Frasnelli; nazarioluisaf@gmail.com

Recibido: 21/08/2020 **Aceptado:** 09/08/2021

Resumen

En el desarrollo de una aplicación web existen varios aspectos a ser considerados, y entre los más importantes se encuentra la arquitectura. Con este trabajo se pretende realizar un acercamiento teórico y práctico a la arquitectura Flux, más específicamente se analizó la librería Vuex. Esto nace a partir de las propias dudas de los investigadores sobre la implementación de la arquitectura mencionada; y con el afán de que el documento pueda ayudar a otros desarrolladores que se encuentren en la misma situación. Vuex es una librería que permite incorporar de una forma relativamente sencilla los conceptos de arquitectura Flux a los proyectos desarrollados utilizando el framework Vue.js. En el documento se describen las principales características de la librería Vuex, además se presenta un análisis de dos implementaciones de una misma aplicación donde en una se utilizó la librería propuesta y en otra no. En el documento se platea entre otras cosas, que la complejidad de los proyectos es uno de los factores determinantes a la hora de optar por la integración de la librería Vuex.

Palabras clave: aplicaciones web, vuex, flux.

Abstract

In the development of a Web Application, there are several aspects to be considered; and architecture is among the most important ones. This work aims to make a theoretical and practical approach to Flux Architecture; for which this work analyzed the Vuex library. This stems from the researchers' own doubts about the implementation of the aforementioned architecture so that this research can help other developers on the matter.

Vuex is a library that allows to incorporate in a simple way the concepts of Flux architecture to the projects developed using the Vue.js framework. The document describes the main features of the Vuex library; and contains an analysis of two implementations of the same application, in which the proposed library was used in one and not in the other. The document states, among other things, that the complexity of the projects is one of the determining factors when opting for the integration of the Vuex library.

Key words: Web applications, Vuex, Flux.

1. Introducción

Cuando se desarrolla un sistema de información se deben tener en cuenta varios factores que permitan garantizar la escalabilidad, rendimiento y mantenibilidad del mismo. Estos aspectos no se deben dejar de lado, es más cobran aún mayor relevancia, en el desarrollo web [1] donde incluso aparecen desafíos adicionales junto a las nuevas especificaciones.

Es común que los desarrolladores se encuentren ante problemas de diferente complejidad, muchos de los cuales son comunes y aparecen con regularidad. Es en este escenario en el que aparecen los patrones de diseño que ofrecen soluciones hasta cierto punto estandarizadas, pero esto no implica que exista un manual específico de implementación que se ajuste a todos los casos. Si bien con frecuencia pueden encontrarse ejemplos prácticos, la única manera de entender un patrón y a su vez validar si es aplicable a un problema concreto es implementarlo por cuenta propia [2]. Esto se aplica a las diversas áreas del desarrollo de aplicaciones, incluyendo en la construcción de componentes web.

Cuando una aplicación web por su tamaño o complejidad cuenta con varios componentes interconectados, la comunicación ente estos se vuelven muy compleja y algo desordenada. La arquitectura Flux plantea una solución a este problema mediante la aplicación del concepto de gerenciamiento centralizado de estados [3]. En esta arquitectura varios componentes acceden a una misma fuente de datos llamada *store* y de esta manera se evita que los componentes deban intercambiar mensajes entre ellos.

Vuex es una librería que permite incorporar de una forma relativamente sencilla los conceptos de arquitectura Flux a los proyectos desarrollados utilizando el *framework* Vue.js. La misma proporciona un depósito o *store* centralizado para todos los componentes de una aplicación [4].

Según lo expuesto por Galhardo, Petrucelli y Espírito Santo en su artículo [5] publicado este año en revista Interface Tecnológica Vuex es una opción muy válida para el manejo de estados de aplicaciones. Esta solución presenta algunas ventajas tales como un claro flujo de datos en los estados y una clara división de responsabilidades. Por otro lado en el documento se plantea la existencia de cierto grado de repetición de código, aspecto que se puede agravar en proyectos de mayor envergadura.

En este trabajo se pretende realizar un acercamiento a la arquitectura Flux y a su implementación en proyectos web desarrollados según las especificaciones *web components*. Esto nace a partir de las propias dudas de los investigadores sobre la implementación de la arquitectura mencionada; y con el afán de que el documento pueda ayudar a otros desarrolladores que se encuentren en la misma situación.

En primera instancia se buscó responder a ciertas preguntas tales como: ¿cómo funciona este patrón de arquitectura?, ¿cuándo es aplicable?, ¿cuándo no es necesario?, ¿cuáles son sus limitaciones?, ¿qué beneficios ofrece? y otras. Por otro lado, además de un mero acercamiento teórico, se realizaron pruebas prácticas de implementación en proyectos basados en *web components*. De manera más específica, se trabajó con Vue.js como *framework*, y se utilizó la librería Vuex para la implantación de la arquitectura Flux.

2. Materiales y Métodos

2.1. Caracterización de la librería Vuex

Para tener una comprensión general de las características y del funcionamiento de la librería se procedió a la realización de un análisis de las documentaciones oficiales de la librería.

2.2. Desarrollo de aplicaciones de prueba

Para la prueba de la librería se procedió al desarrollo de aplicaciones con las mismas funcionalidades. Donde la primera fue desarrollada sin utilizar la librería Vuex y la segunda utilizando dicha librería para incorporar los conceptos la arquitectura Flux.
Las aplicaciones desarrolladas consistían en buscadores de álbumes de música utilizando la API de Deezer. Las funcionalidades principales fueron las de listar y agregar a una carpeta de favoritos las canciones de los álbumes encontrados, así como escuchar una *preview* de las mismas.

2.3. Criterios para evaluar la librería

En el presente trabajo se analiza la utilización de la librería Vuex como alternativa para la implementación de la arquitectura Flux en proyectos Front-end desarrollados con Vue.js.
Ahamed, Pezewski y Pezewski en [6] plantean algunos criterios para la evaluación de *frameworks* entre los cuales se puede citar lexibilidad, requerimientos de implementación**,** requerimientos del lenguaje, complejidad, entre otros. Algunos de estos criterios son de gran interés para la selección de un *framework* o librería [7]. Para la evaluación de la librería Vuex se tomaron los siguientes criterios:

- **Flexibilidad:** adaptabilidad a proyectos de diferente tamaño y complejidad.
- **Requerimientos de implementación:** se evalúa si la implementación está restringida a una plataforma.
- **Requerimientos de la tecnología:** restricción de implementación en diferentes lenguajes de programación o tecnologías.

- **Complejidad:** nivel de dificultad en la implementación en proyectos.
- **Madurez:** se analiza la estabilidad de la librería, y si cuenta con una con buena documentación y una sólida comunidad.

3. Resultados y Discusión

3.1. Caracterización de la librería Vuex

Es importante comenzar describiendo la estructura básica de la arquitectura Flux la cual cuenta de tres partes. En primer lugar, se encuentra el estado que es una única fuente centralizada de datos accesible para cualquier componente que los necesite, en segundo lugar, están las acciones que son las formas en la que un estado puede ser modificado y por último están las vistas que se encargan de mapear y visualizar los estados en los componentes [4].

La librería Vuex agrega algunos elementos extras para la optimización de la arquitectura. A continuación, se describen los elementos básicos de esta librería:

- **Estado:** la librería Vuex maneja un único árbol de estados, que recibe el nombre de *store*, para una aplicación, es decir, que solo existe una única fuente de donde se extraen los datos. En la documentación oficial se dice que existe una única fuente de verdad [8].
- ***Getters*:** en los casos que es necesario obtener un valor calculado en base a un estado o si se deben realizar operaciones como el filtrado de listas, se pueden crear métodos capaces de encapsular dichas operaciones y que las hagan accesibles para ser utilizadas en cualquier lugar en el que se requiera. A estos métodos se los conoce como accesores [8].
- **Mutaciones:** son el único medio para modificar los estados de una aplicación, en específico son métodos que se encargan de modificar los valores almacenados en los estados. Las mutaciones son síncronas y son ejecutadas por las acciones mediante una función especial denominada *commit* [8].
- **Acciones:** contienen lógica de negocio y son capaces de ejecutar funciones asíncronas. Esto último por ejemplo se utiliza para comunicación con un servidor externo. Las acciones no pueden modificar los estados de manera directa, pero pueden ejecutar mutaciones para ello. Las acciones se ejecutan mediante una función especial denominada *dispatch* [8].
- **Módulos:** a medida que las aplicaciones crecen en tamaño, la cantidad de estados contenidos en el *store* va aumentando y se vuelve más difícil su administración. Para lidiar

con este problema, Vuex permite la división de *store* en módulos, y estos pueden encapsular sus propios estados, *getters*, mutaciones y acciones [8].

3.2. ¿Cuándo utilizar Vuex?

Técnicamente hablando, esta librería puede ser integrada a cualquier proyecto, aunque en ciertos casos cuando una aplicación es pequeña o muy simple la incorporación de conceptos de manejo centralizado de estados puede resultar engorrosa. Es decir, es probable que introduzca un grado de complejidad innecesaria al proyecto. En estos casos trabajar de la manera tradicional resultaría una opción muy válida [4].

En contraposición a lo mencionado en el párrafo anterior, cuando se construyen aplicaciones de un tamaño mayor, el manejo de los estados dentro los componentes se vuelven cada vez más complejo. Es en este punto donde entra en juego esta librería, permitiendo manejar los estados de la aplicación de manera externa y centralizada. Esto permite una gestión óptima y escalable de los estados [4]. Además, la comunicación entre componentes que necesitan transferir estados a otros se vuelve cada vez más compleja, lo cual hace difícil el seguimiento y comprensión del flujo de los datos.

Se puede decir en este caso que para aquellos proyectos cuya propia complejidad los vuelven difíciles de escalar y mantener son los que necesitan aplicar la arquitectura Flux, y específicamente dentro del contexto del presente trabajo implica la utilización de la librería Vuex.

3.3. Aplicación de prueba desarrollada sin utilizar Vuex

En este caso se trabajó exclusivamente con los componentes de Vue.js, en los cuales se aglomera las plantillas, la definición de los componentes, el manejo de los estados y la lógica de negocios. Además, el paso de los estados se realiza mediante parámetros, lo cual significó un aumento de la complejidad a medida que aumentó la cantidad de componentes que se comunican entre sí. En la siguiente figura, se presenta la implementación de una plantilla de Vue.js, en la misma se puede apreciar cómo se realiza el paso de estados.

```
<template>
  <div class="row">
    <Header :titulo="datosCancion.title" icono="play_circle_outline"/>
    <Reproductor v-if="datosCancion.title" :cancion="datosCancion"/>
  </div>
</template>
```

Figura 1. Plantilla Vue.js sin implementar de la librería Vuex

En esta implementación toda la lógica relacionada al manejo de los estados se encuentra encapsulada en el componente, y estos estados están representados por las propiedades del mismo. Incluso se puede observar como en el mismo componente se incluye la lógica para recuperar los datos de un servidor externo utilizando una petición *http*.

```
export default {
  name: 'Cancion',
  props: {
    id:{},
    cancion:{
      type:Object,
      default:function () {
        return {title:''}
      }
    },
    album:{}},
  data(){
    return {
      datosCancion: Object.assign(this.cancion,{album:this.album})
    }
  },
  components: {
    Header,Reproductor
  },
  mounted() {
    if (this.cancion.title){
      this.$route.meta.breadCrumbs.album.to = `/album/${this.album.id}`;
    }
    else {
      this.$http.get(`track/${this.id}`).then((response) => {
        this.datosCancion = response.data;
        this.$route.meta.breadCrumbs.album.to = `/album/${this.datosCancion.album.id}`;
      })
    }
```

Figura 2. Lógica de componente Vue.js sin implementar de la librería Vuex

3.4. Aplicación de prueba desarrollada utilizando la librería Vuex

Para este caso se desarrolló la aplicación utilizando la librería Vuex, es decir, se aplicaron los principios de manejo centralizado de estados de la arquitectura Flux. Se realizó una división de responsabilidades dentro del proyecto, manejando los estados de manera externa, y además se crearon archivos encargados de gestionar las peticiones *http* (esto último responde a la decisión de realizar una división más estricta de responsabilidades).

Como ya se mencionó anteriormente la librería Vuex permite separar el manejo de estados por módulos. En el ejemplo presentado, se declaran los estados de la aplicación, los *getters* que realizan ciertas operaciones, y las mutaciones que permiten modificar los estados.

```
1   import CancionService from "@/services/CancionService";
2   import FavoritoService from "@/services/FavoritoService";
3
4   const state = {
5       canciones:[],
6       cancionSeleccionada:null,
7       favoritos:FavoritoService.getFavoritos()||{}
8   }
9   const getters = {
10      esFavorito:(state) => (cancion) =>{
11          return state.favoritos[cancion.id] !== undefined;
12      },
13      getFavoritosCopy:(state) => {
14          return JSON.parse(JSON.stringify(state.favoritos))
15      }
16  }
17  const mutations = {
18      setAlbum:(state, album) => state.album = album,
19      setCanciones:(state, canciones) => state.canciones = canciones,
20      setFavoritos:(state, canciones) => state.favoritos = canciones,
21      setCancionSeleccionada:(state, cancion) => state.cancionSeleccionada = cancion,
22  }
```

Figura 3. Módulo de manejo de estados desarrollado con la librería Vuex

Continuando con el módulo presentado, en la siguiente figura se pueden visualizar las acciones que encapsulan la lógica de negocio. En este caso por ejemplo se recuperan los datos del servidor y se ejecutan mutaciones para modificar los estados.

```
24  const actions= {
25      recuperarPorAlbum: async function ({commit},album) {
26          let canciones = await CancionService.recuperarPorAlbum(album)
27          commit("setCanciones",canciones)
28      },
29      favoriteChange: ({commit,getters}, cancion) => {
30          if(getters.esFavorito(cancion)) commit('setFavoritos', FavoritoService.removeFavorito(cancion))
31          else commit('setFavoritos', FavoritoService.addFavorito(cancion))
32      },
33      seleccionarCancion: ({commit},cancion) => commit("setCancionSeleccionada",cancion),
34      recuperarPorId: async function ({commit},id) {
35          let cancion = await CancionService.recuperarPorId(id)
36          commit("setCancionSeleccionada",cancion)
37      },
38  }
39  export default {
40      namespaced: true,
41      state,
42      getters,
43      mutations,
44      actions,
45  };
```

Figura 4. Acciones del módulo de manejo de estados desarrollado con la librería Vuex

De esta manera las responsabilidades asociadas a la gestión de los estados, que en un modelo tradicional estarían a cargo del mismo componente, son delegadas al *store* de la

aplicación. En este escenario los componentes se encargan de manera exclusiva al acceso y visualización de los estados, así como de la ejecución de las acciones del *store*.

src > components > DetalleCancion.vue > {} "DetalleCancion.vue" > template

```vue
<template>
  <div class="row">
    <Header v-if="cancionSeleccionada" :titulo="cancionSeleccionada.title" icono="play_circle_outline"/>
    <Reproductor v-if="cancionSeleccionada"/>
  </div>
</template>
<script>
import Header from '@/components/headers/Header.vue'
import Reproductor from '@/components/cards/Reproductor.vue'
import { mapState, mapActions } from 'vuex';
export default {
  name: 'Cancion',
  props: ['id'],
  components: {
    Header,Reproductor
  },
  computed:{
    ...mapState('cancion',['cancionSeleccionada'])
  },
  methods:{
    ...mapActions('cancion',['recuperarPorId'])
  },
```

Figura 5. Componente Vue.js con la implementación de la librería Vuex

En la imagen anterior se puede apreciar cómo se despoja al componente de mucha de la lógica de negocios requerida para su funcionamiento, pues se delega esta responsabilidad al *store* de la aplicación. De esta manera, la codificación se vuelve más limpia y optimizada. Además, mediante el manejo centralizado de estados se reduce la complejidad que presenta cuando varios componentes comparten datos.

3.5. Evaluación de la librería Vuex

Tabla 1. Evaluación de la librería utilizando por criterios

Criterios	Evaluación	Observación
Flexibilidad	Flexible	Su implantación es viable en proyectos de diferente complejidad y tamaño.
Requerimientos de implementación	Adaptable	Es aplicable a proyectos web, móviles e incluso de escritorio. Esto se consigue utilizando herramientas como Vue Native.
Requerimientos de la tecnología	Cerrado	La librería Vuex solo se puede utilizar en proyectos desarrollados con Vue.js
Complejidad	Relativamente complejo	En proyectos pequeños puede sumar complejidad innecesaria.
Madurez	Maduro	El proyecto cuenta con una sólida comunidad que lo respalda y mantiene. Además, la librería se encuentra ampliamente documentada

4. Conclusiones

Los aspectos de arquitectura son ineludibles en todo proyecto de software. Y en el contexto abordado en este trabajo implica que los proyectos desarrollados puedan escalar en tamaño sin que la complejidad vuelva insostenible el mantenimiento de los mismos. De manera indistinta a cuál sea la tecnología en la que se desarrolle un proyecto, la implementación de una arquitectura determinada depende de varios factores entre los cuales se puede mencionar del mismo.

En el escenario de prueba planteado en el presente trabajo la librería Vuex ofreció una manera sencilla de incorporar los conceptos de la arquitectura Flux al proyecto desarrollado utilizando Vue.js.

Con la implementación de la librería Vuex se elimina el complejo esquema de paso de estados entre componentes mediante la centralización del manejo de estados. De esta manera sin importar el tamaño de la aplicación el manejo de los estados no varía en complejidad, si bien es válido considerar que para proyectos pequeños podría resultar más sencilla la implementación tradicional.

Siguiendo con lo planteado en el párrafo anterior, si bien puede resultar compleja la implementación de la arquitectura Flux para proyectos pequeños, se debe tener en cuenta la posibilidad de que eventualmente los mismos pueden crecer. De este modo se reafirma la necesidad de realizar un análisis sobre su incorporación antes de comenzar el proceso de construcción de la aplicación.

Conflicto de interés: Los autores declaran que no existe ningún conflicto de interés con respecto a la publicación de este artículo.

Referencias

[1] S. K. Mukhiya and K. Hoang Hung, “An Architectural Style for Single Page Scalable Modern Web Application,” vol. 5, no. 4, pp. 6–13, 2018, [Online]. Available: https://www.ijrra.net/Vol5issue4/IJRRA-05-04-02.pdf.

[2] M. Fowler, “Patterns,” *IEEE Softw.*, vol. 20, no. 2, pp. 56–57, 2003, doi: 10.1109/MS.2003.1184168.

[3] Comunidad Vue.js, “State Management. VueJs Docuemtation,” *vuejs.org*, 2016. https://vuejs.org/v2/guide/state-management.html (accessed Sep. 10, 2019).

[4] Comunidad Vue.js, “What is Vuex?,” *vuejs.org*, 2016. https://vuex.vuejs.org/.

[5] L. Galhardo Lima, E. E. Petrucelli, and F. do Espírito Santo, “Visão geral sobre o Gerenciamento de Estado no Vue.js com a biblioteca Vuex,” *Rev. Interface Tecnológica*, vol. 16, no. 1, pp. 56–66, 2019.

[6] S. I. Ahamed, A. Pezewski, and A. Pezewski, “Towards framework selection criteria and suitability for an application framework,” in *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, 2004, vol. 1, pp. 424-428 Vol.1, doi: 10.1109/ITCC.2004.1286492.

[7] E. Johansson and J. Söderberg, “Evaluating performance of a React Native feature set,” 2018, [Online]. Available: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1215795&dswid=4286.

[8] Comunidad Vue.js, “Vuex guide,” *vuejs.org*, 2015. https://vuex.vuejs.org/guide/.