PROYECTO DE SOFTWARE

Cursada 2020

TEMARIO

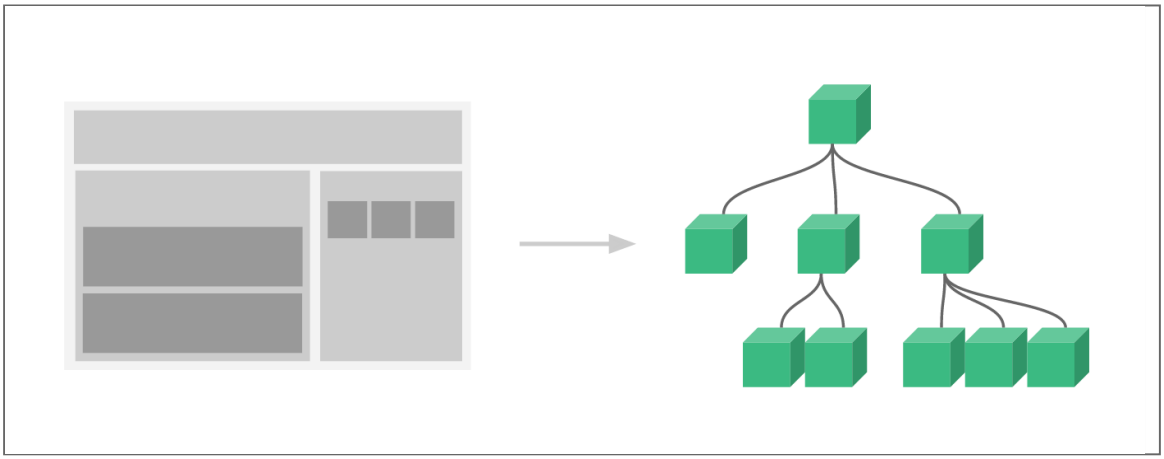
Vue.js

- Componentes
- Vue cli
- Ruteo
- Vuex

ARRANQUEMOS CON LAS PREGUNTAS DE REPASO

COMPONENTES VUE

- El sistema de componentes es un concepto importante.
- Nos permite construir aplicaciones grandes a partir de componentes:
    - Más pequeños
    - Reutilizables
    - Auto-contenidos



COMPONENTES VUE

- En Vue, una componente es una instancia Vue con opciones predefinidas.

```
Vue.component('todo-item', {
  template: '<li>This is a todo</li>'
})
```

- Se puede utilizar dentro del template de otra componente:

```
<ol>
  <!-- Create an instance of the todo-item
component -->
  <todo-item></todo-item>
</ol>
```

COMPONENTES VUE

- En una aplicación grande podríamos separar todo en componentes independientes.

```
<div id="app">
  <app-nav></app-nav>
  <app-view>
    <app-sidebar></app-sidebar>
    <app-content></app-content>
  </app-view>
</div>
```

EJEMPLO BÁSICO COMPONENTES VUE:

- Veamos **componentes-basico** y **componentes-multiple**.

```
<title>Componente</title>
<script src="https://unpkg.com/vue"></script>


<div id="components-demo">
  <button-counter></button-counter>
</div>
<script>
// Define a new component called button-counter
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
 template:'<button v-on:click="count++">You
clicked me {{ count }} times.</button>'
})
new Vue({ el: '#components-demo' })
</script>
```

PASANDO DATOS A UNA COMPONENTE CON PROPS:

- Veamos **componentes-props** y **componentes-multiple-apps**

```
  <title>Componente</title>
<script src="https://unpkg.com/vue"></script>

<div id="components-demo">
  <blog-post title="My journey with Vue">
</blog-post>
  <blog-post title="Blogging with Vue"></blog-
post>
  <blog-post title="Why Vue is so fun"></blog-
post></div>
<script>
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})

new Vue({ el: '#components-demo' })
</script>
```

- **Importante**: las props se pueden pasar únicamente hacia abajo en el árbol de componentes.

PARA PROYECTOS MÁS GRANDES: VUE CLI

- Crear un proyecto y seleccionar plugins:

```
vue create my-project
```

- También provee una interfaz web para realizar esto mismo:

```
vue ui
```

- Instalar dependencias con **npm** o **yarn** definidas en **package.json**:

```
npm/yarn install
```

- Levantar el servicio para desarrollo:

```
npm/yarn run serve
```

- Generar los archivos necesarios para producción:

```
npm/yarn run build
```

COMPONENTES SINGLE-FILE

- Para grandes proyectos las componentes utilizando **Vue.component** poseen varias **desventajas**:
    - Definiciones globales: cada componente debe tener un nombre único.
    - Templates como strings: no poseemos syntax highlight en el desarrollo.
    - No tenemos soporte para CSS.
    - No hay etapa de contrucción: nos restringe a utilizar puramente HTML y JavaScript.
- Todo esto se soluciona utilizando componentes single-file (con extensión **.vue**) y gracias a herramientas como **Webpack** o **Browserify**.

EJEMPLO HELLO.VUE

- Veamos el fuente **Hello.vue**
- En **package.json** se encuentran las dependencias, instalemos con **npm install** y corramos **npm run serve**.

```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped="">
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

UTILIZANDO OTROS PREPROCESADORES:

```
<template lang="jade">
div
  p {{ greeting }} World!
  OtherComponent
</template>

<script>
import OtherComponent from
'./OtherComponent.vue'
export default {
  components: {
    OtherComponent
  },
  data () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style lang="stylus" scoped="">
p
  font-size 2em
  text-align center
</style>
```

- En este caso un manejador de templates: **jade** y un preprocesador css: **stylus**.

AXIOS UTILIZANDO VUE CLI + COMPONENTES SINGLE-FILE

- Creamos proyecto vacío con **vue cli**:

```
vue create api_client
cd api_client
npm run serve
```

- Agregamos **axios**:

```
npm install --save axios
```

COMPONENTE APICLIENT.VUE:

```
<template>
  <div>
    <h1>Provincias:</h1>
    <ul v-if="locations && locations.length">
      <li v-for="location of locations">
        <strong>{{ location.id }}</strong> - {{
location.nombre }}
      </li>
    </ul>

    <ul v-if="errors && errors.length">
      <li v-for="error of errors">
        {{error.message}}
      </li>
    </ul>
  </div>
</template>

<script>
import axios from 'axios';

export default {
  data() {
    return {
      locations: [],
      errors: []
    }
  },

  // Fetches posts when the component is
created.
  created() {

axios.get('https://apis.datos.gob.ar/georef/api
/provincias')
          .then(response => {
          // JSON responses are automatically
parsed.
          this.locations =
response.data.provincias;
          })
          .catch(e => {
            this.errors.push(e)
          })
  }
}
</script>
```

MODIFICAMOS APP.VUE PARA INCORPORAR EL COMPONENTE ANTERIOR:

```
<template>
  <div id="app">
    <apiclient>
  </apiclient></div>
</template>

<script>
import ApiClient from
'./components/ApiClient.vue'

export default {
  name: 'app',
  components: {
    ApiClient
  }
}
</script>

<style>
</style>
```

- Levantemos el servicio con **npm run serve** y veamos lo contruido con **npm run build** **acá**.

RUTEO EN VUE

- Vamos a utilizar la librería **vue-router**.

VUE-ROUTER

- Instalación:

```
$ npm install vue-router
```

- Esto va a agregar **vue-router** a nuestro archivo **package.json**.

MAIN.JS

- Es recomendable escribir el código de ruteo en un archivo separado **router.js** y luego agregarla a la aplicación Vue dentro del **main.js**:

```
import Vue from 'vue'
import App from './App.vue'
import router from './router' // Router being
imported

Vue.config.productionTip = false

new Vue({
  router,  // router added to the Vue instance
  render: function (h) { return h(App) }
}).$mount('#app')
```

ROUTER.JS

- Importamos **Router** del paquete **vue-router** y la agregamos a Vue con el método **use**.
- Lo exportamos al resto de la aplicación.

```
import Vue from 'vue'
import Router from 'vue-router'

Vue.use(Router)

const routes = []

const router = new Router({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})

export default router
```

LAS RUTAS

```
const routes = [
 {
   path: '/',
   name: 'Home',
   component: Home
 }
]
```

- **path**: El path relativo a la base de la aplicación.
- **name**: El nombre de la ruta para referenciarla en los componentes.
- **component**: El componente que va a estar en esa ruta.
- **redirect**: Una redirección.
- **alias**: Alias.
- **children**: Un arreglo con mas rutas que se concatenan a la ruta padre.
- **params**: Parámetros del componente.

UTILIZANDO EL ROUTER EN UNA COMPONENTE:

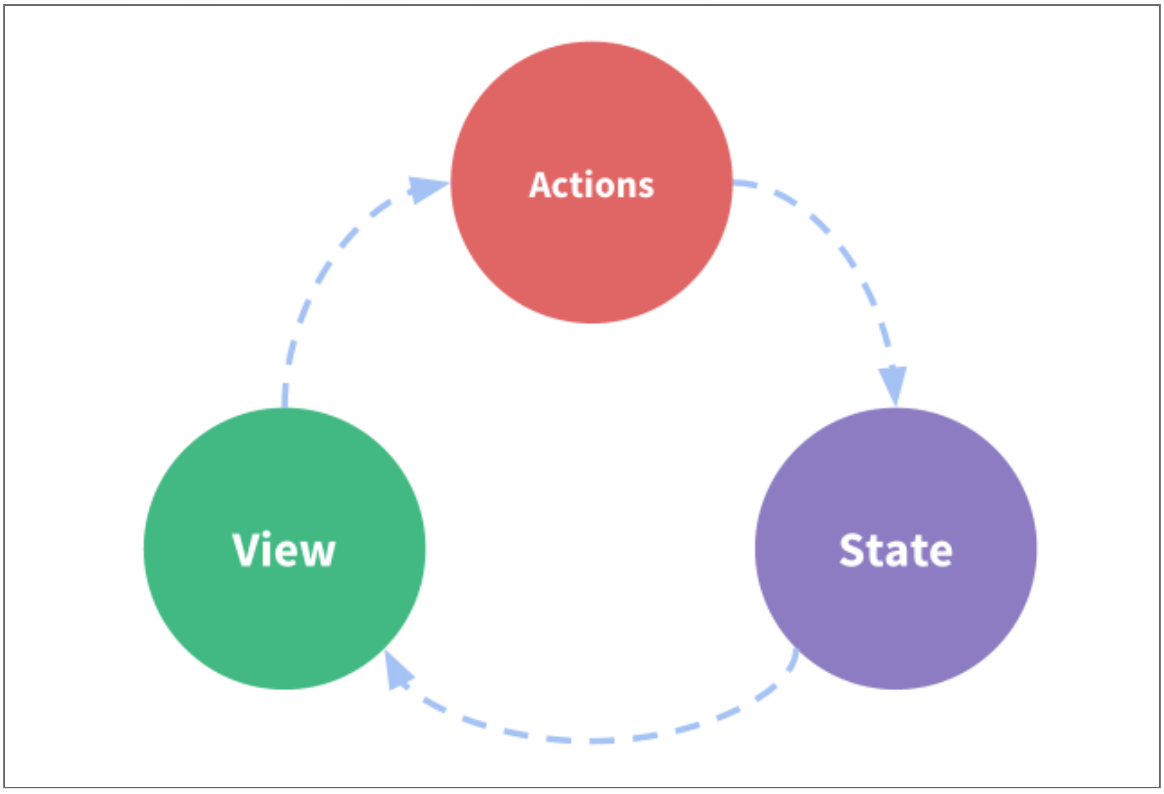
- El componente de la ruta se va a renderizar dentro del tag **router-view**.
- Para acceder a las rutas podemos utilizar un tag **a** que va a recargar la página o utilizar la propiedad **router-link** .

```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/ruta1">Ruta 1</router-
link> |
      <router-link to="/ruta2">Ruta 2</router-
link> |
      <router-link to="/about">About</router-
link>
    </div>
    <router-view>
  </router-view></div>
</template>
```

- Veamos el ejemplo en **ejemplo-router**.

VUEX

ESTADO CON UN ÚNICO COMPONENTE



- Únicamente se modifica el estado de la componente actual.

MANEJANDO EL ESTADO: VUEX

- En una aplicación grande es inevitable tener que compartir datos entre los distintos componentes.
- Ir pasando las variables de componente en componente a través del árbol de componentes es engorroso.
- La solución es **Vuex**, una librería para manejar un estado global para aplicaciones Vue.js.

STORES EN VUEX

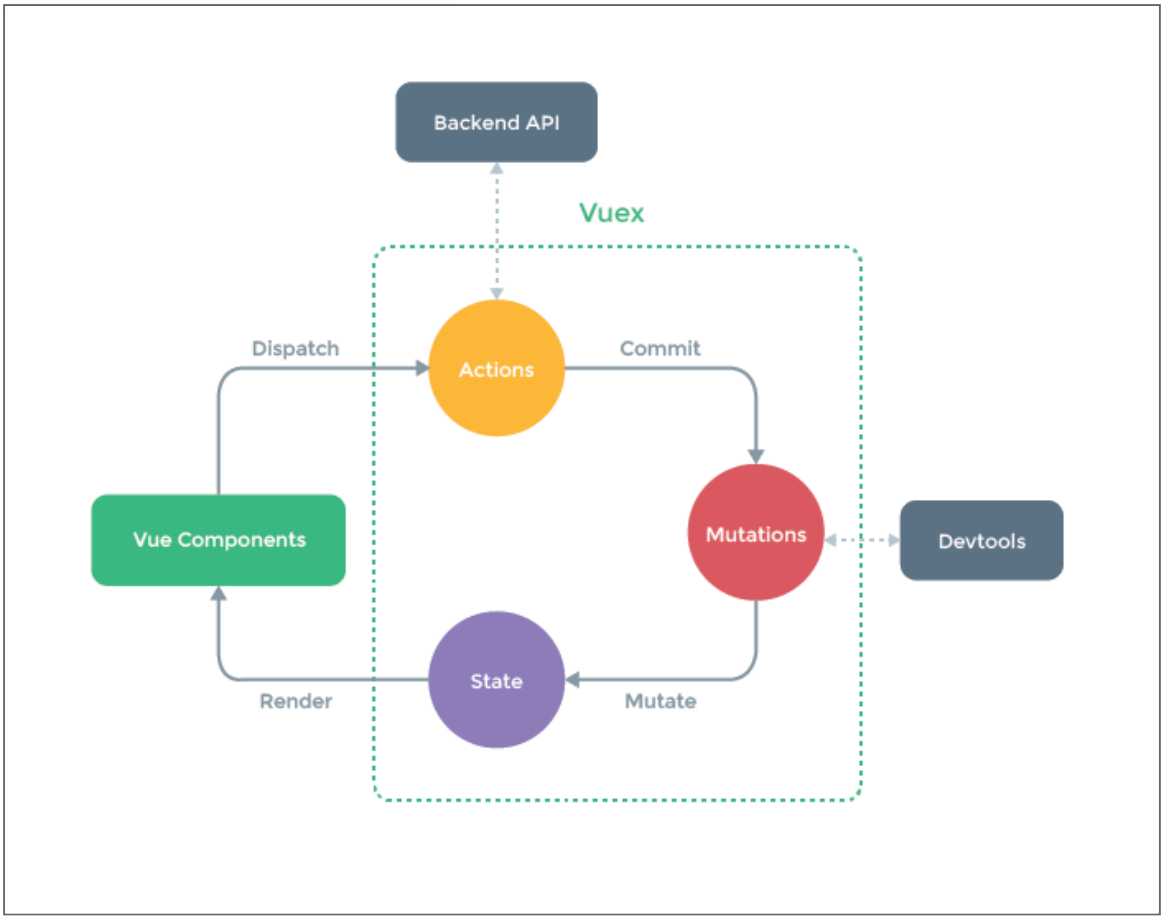
- Una **"store"** es básicamente un contenedor que contiene el estado de la aplicación.
- Hay 2 cosas en la que las stores de Vuex se diferencian de un objeto global plano:
    - Las stores Vuex **son reactivas**: cuando un componente saca sus valores de una store, este va a actualizarse reactivamente ante un cambio de estado.
    - **No es posible cambiar directamente el estado** de una store. La única forma es si explícitamente se realizan **mutaciones**. Cada cambio deja un registro del mismo.

AGREGANDO VUEX

- Instalación:

```
$ npm install vuex
```

INTERACCIÓN CON VUEX



CREAMOS UNA STORE

- En un **store.js** por ejemplo:

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment (state, payload) {
      state.count += payload.amount
    },
    decrement (state, payload) {
      state.count -= payload.amount
    }
  }
})

export default store
```

INCORPORAMOS EL STORE A LA APP VUE

- En el **main.js**.

```
import Vue from 'vue'
import App from './App.vue'
import store from './store'

Vue.config.productionTip = false

new Vue({
  store,
  render: function (h) { return h(App) }
}).$mount('#app')
```

ACCEDIENDO AL STORE

- Se puede acceder al estado con **store.state**, y disparar un cambio en el estado utilizando el método **store.commit**:

```
this.$store.commit('increment', { amount: 1 })
console.log(this.$store.state.count) // -> 1
```

UTILIZANDO EL ESTADO DE UNA STORE

- Podemos simplemente retornar el estado utilizando una **propiedad computada**, ya que el estado de la store es reactivo.
- Disparar cambios significa simplemente **commitear mutaciones** en métodos de la componente hacia la store.

COMPONENTE ACCEDIENDO AL ESTADO GLOBAL (EJEMPLO COUNTER.VUE)

```
<template>
  <div class="counter">
    <h1>{{ msg }}</h1>
    <p>count = {{ count }}</p>
    <p>
      <button v-on:click="increment">+{{ num }}
</button>
      <button v-on:click="decrement">-{{ num }}
</button>
    </p>
  </div>
</template>

<script>
export default {
  name: 'Counter',
  props: {
    msg: String,
    num: Number
  },
  computed: {
    count () {
      return this.$store.state.count
    }
  },
  methods: {
    increment () {
      this.$store.commit('increment', {
        amount: this.num
      })
    },
    decrement () {
      this.$store.commit('decrement', {
        amount: this.num
      })
    }
  }
}
</script>
```

- Veamos el ejemplos con **Múltiples Contadores**.

EJEMPLO TUTORIAL

- Ruteo con **vue-router**: **https://router.vuejs.org/**.
- Estado global con **vuex**: **https://vuex.vuejs.org/**.
- Validación con **VeeValidate**: **https://vee-validate.logaretm.com/v2/**.
- Veamos el ejemplo de **Tutorial+Router+Vuex**.

PARA SEGUIR LEYENDO: VUEJS

- Vue Routing: **https://es.vuejs.org/v2/guide/routing.html**
- Vue Vuex : **https://vuex.vuejs.org/**
- Artículo Vuex : **https://nexlab.dev/tech/2018/12/08/vuex-manejo-centralizado-de-datos-en-vuejs.html**
- Webpack: **https://www.youtube.com/watch?v=2UBKjshUwM8**
- Componentes y Plugins: **https://madewithvuejs.com/**
- Vue 2 -> Vue 3: **https://v3.vuejs.org/guide/migration/introduction.html**
- Vue 2 vs Vue 3: **https://medium.com/javascript-in-plain-english/differences-between-vue-2-and-vue-3-ee627e2c83a8**

PARA SEGUIR LEYENDO: JS FRAMEWORKS 2

- Gitlab.com usa vue: **https://about.gitlab.com/2016/10/20/why-we-chose-vue/**
- **https://hackernoon.com/the-status-of-javascript-libraries-frameworks-2018-beyond-3a5a7cae7513**
- **https://bestofjs.org/**
- Promesas JS: **https://slides.com/juanramb/promesas-en-javascript#/**
- Axios vs fetch: **https://medium.com/@thejasonfile/fetch-vs-axios-js-for-making-http-requests-2b261cdd3af5**
- **https://alligator.io/vuejs/rest-api-axios/**

¿DUDAS?

FIN