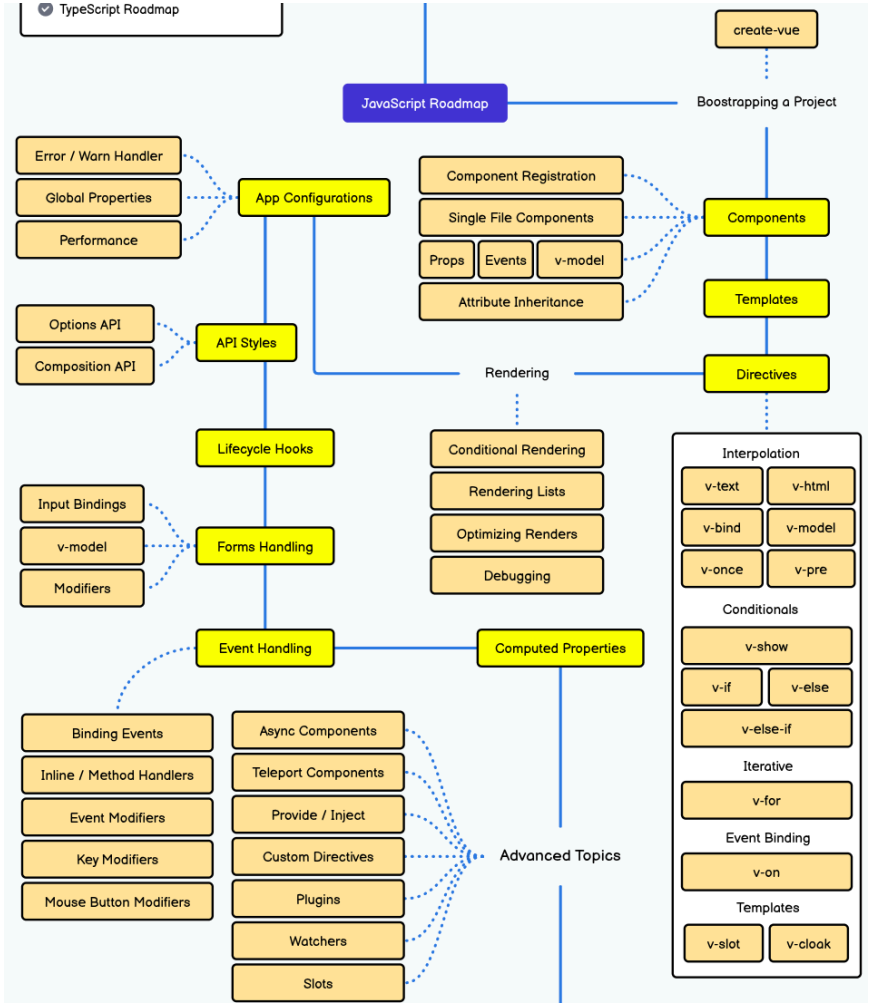
Seguimos con Vue.js



https://roadmap.sh/vue

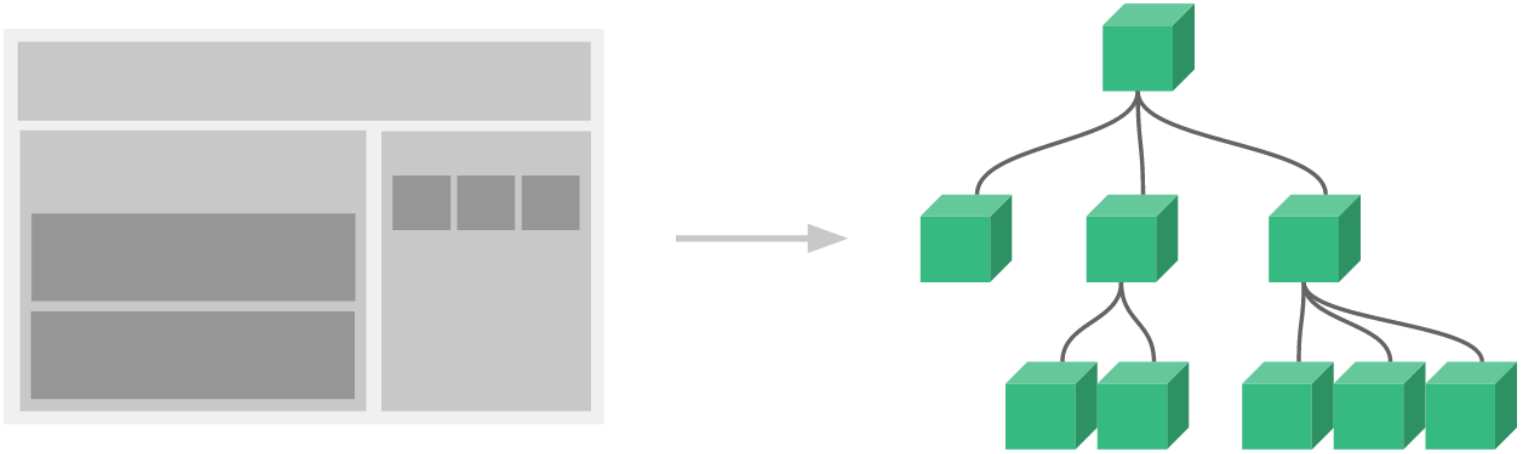
Temario

Vue.js

- Componentes
- Vue CLI -> Vite
- Ruteo: Vue Router
- Gestión de estado: Vuex -> Pinia
- Options Api vs Composition Api

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.

```
// Create Vue application
const app = Vue.createApp(...)
// Define a new component called todo-item
app.component('todo-item', {
  template: `<li>This is a todo</li>`
})
// Mount Vue application
app.mount(...)
```

- 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.

```
<div id="components-demo">
  <button-counter></button-counter>
</div>

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

Pasando datos a una componente con props:

- Veamos componentes-props y componentes-multiple-apps

```
<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>
  const app = Vue.createApp({});
  app.component('blog-post', {
    props: ['title'],
    template: '<h3>{{ title }}</h3>'
  });
  app.mount('#components-demo');
</script>
```

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

Para proyectos más grandes: Vue CLI (antes)

- **Vue CLI** es una herramienta de línea de comandos para generar proyectos Vue.js basada en **webpack**. Se esta reemplazando por **Vite**.
- Instalación:

```
npm install -g @vue/cli
o
yarn global add @vue/cli
```

- Crear un proyecto y seleccionar plugins:

```
vue create my-project
```

- También provee una interfaz web para realizar esto mismo:

```
vue ui
```

Vue CLI (cont.)

- 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
```

Actualmente oficialmente usado: Vite

- Mejor experiencia en desarrollo.
- Mayor eficiencia en build de producción.
- Multiframework.
- Utiliza **esbuild** https://esbuild.github.io/.
- Vite vs webpack: https://www.solucionex.com/blog/el-fin-de-webpack-hola-vite
- Ref: https://vitejs.dev/guide/

Vue + Vite

- Scaffolding oficial para instala vue.js actualmente:

```
$ npm create vue@latest
```

- Migración CLI a Vite: https://vueschool.io/articles/vuejs-tutorials/how-to-migrate-from-vue-cli-to-vite/

Vue + Vite (cont.)

- Instalar dependencias:

```
npm install
```

- Levantar el servicio para desarrollo:

```
npm run dev
npx vite
```

- Preview de producción:

```
npm run preview
npx vite preview
```

- Generar los archivos necesarios para producción:

```
npm run build
npx vite 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** https://webpack.js.org/ y **Vite** https://vitejs.dev/.

Ejemplo Hello.vue

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

```
<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: jade y stylus

```
<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>
```

Axios + Componentes single-file

- Creamos proyecto vacío con **vite**:

```
npm create vue@latest
```

- Agregamos **axios**:

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

Componente ApiClient.vue:

```
<template>
  <div>
    <h1>Provincias:</h1>
    <ul v-if="locations && locations.length">
      <li v-for="(location, index) in locations" :key="index">
        <strong>{{ location.id }}</strong> - {{ location.nombre }}
      </li>
    </ul>
    <ul v-if="errors && errors.length">
      <li v-for="(error, index) in errors" :key="index">
        {{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/>
  </div>
</template>

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

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

Ruteo en Vue

- Para Vue 3 vamos a utilizar la librería vue-router su documentación.

vue-router

- Instalación:

```
$ npm install --save vue-router@4
```

- 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 { createApp } from 'vue'
import App from './App.vue'
import router from './router' // Router being imported

createApp(App).use(router).mount('#app')
```

router.js

- Importamos **createRouter** del paquete **vue-router**.
- Lo exportamos al resto de la aplicación para que lo use.

```
import { createRouter, createWebHistory } from 'vue-router'

const routes = []
const router = createRouter({
  history: createWebHistory(),
  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:

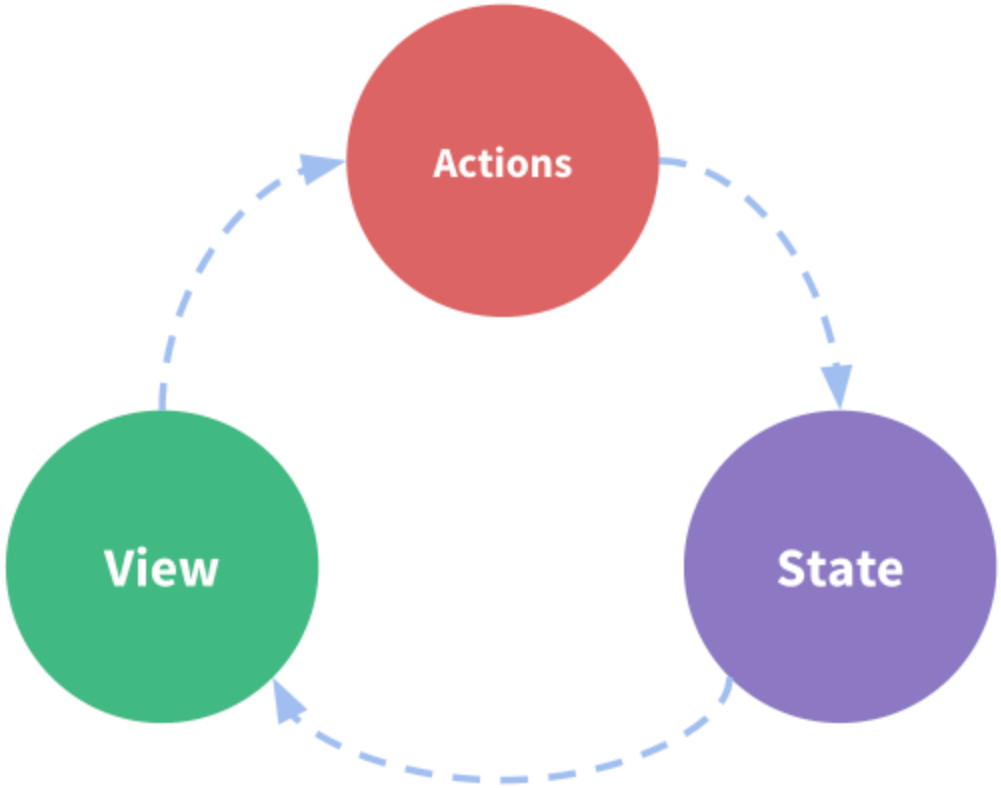
- El componente de la ruta se va a renderizar dentro del tag **router-view**.
- Para acceder a las rutas podemos utilizar un tag **a** que va a recargar la página o utilizar la propiedad **router-link** .

```
<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/>
  </div>
</template>
```

- Veamos el ejemplo en ejemplo-router.

Gestión de estado: Vuex -> Pinia

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

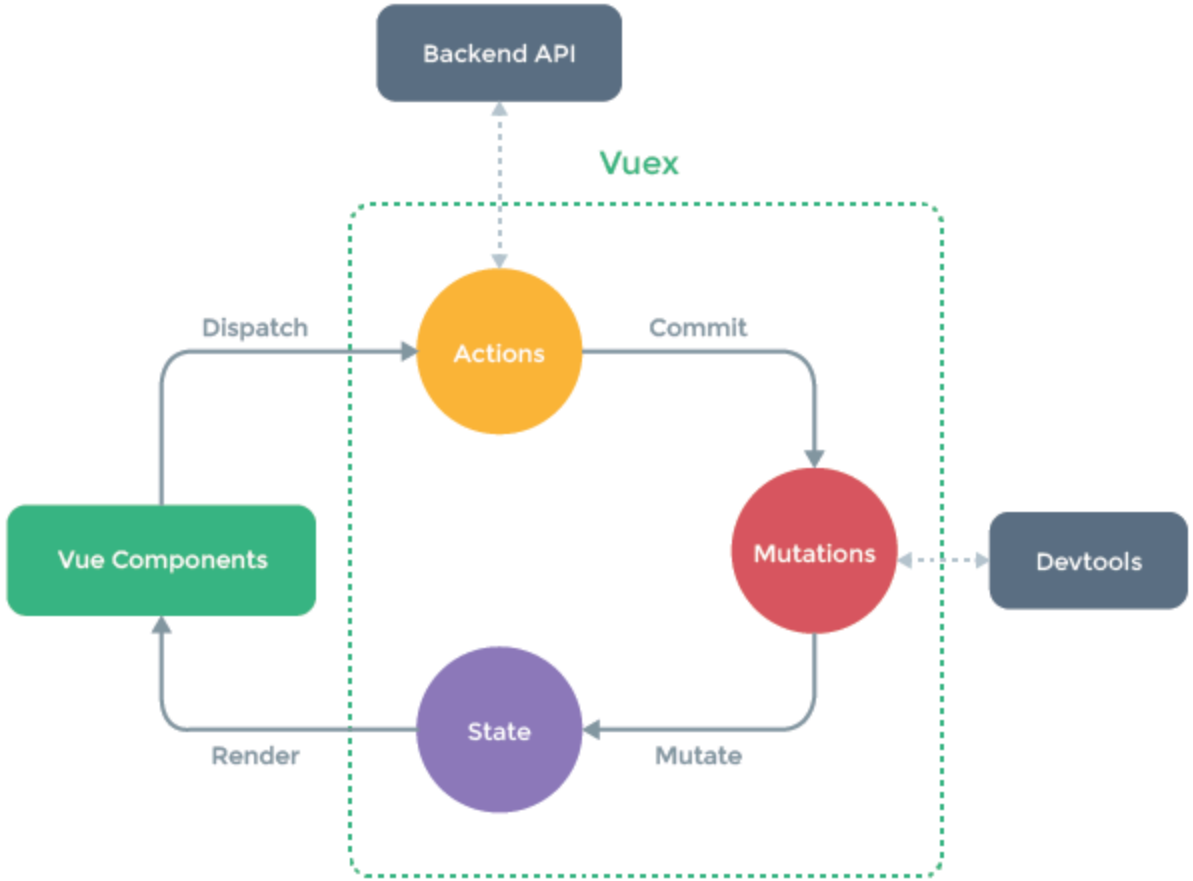
- Una **"store"** es básicamente un contenedor del estado de la aplicación.
- Hay 2 cosas en la que las stores de Vuex se diferencian de un objeto global plano:
    - Las stores Vuex **son reactivas**: cuando un componente saca sus valores de una store, este va a actualizarse reactivamente ante un cambio 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@next --save
```

Interacción con Vuex



Creamos una store

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

```
import { createStore } from 'vuex'

// Create a new store instance.
const store = createStore({
  state () {
    return {
      count: 0
    }
  },
  actions: {
    increaseCount({ commit }, payload ) {
      commit("increment", { amount: payload.amount });
    },
    decreaseCount({ commit }, payload) {
      commit("decrement", { amount: payload.amount });
    }
  },
  mutations: {
    increment (state, payload) {
        state.count += payload.amount
    },
    decrement (state, payload) {
        state.count -= payload.amount
    }
  }
```

Incorporamos el store a la app Vue

- En el **main.js**.

```
import { createApp } from 'vue'
import App from './App.vue'
import store from './store'

createApp(App).use(store).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**:

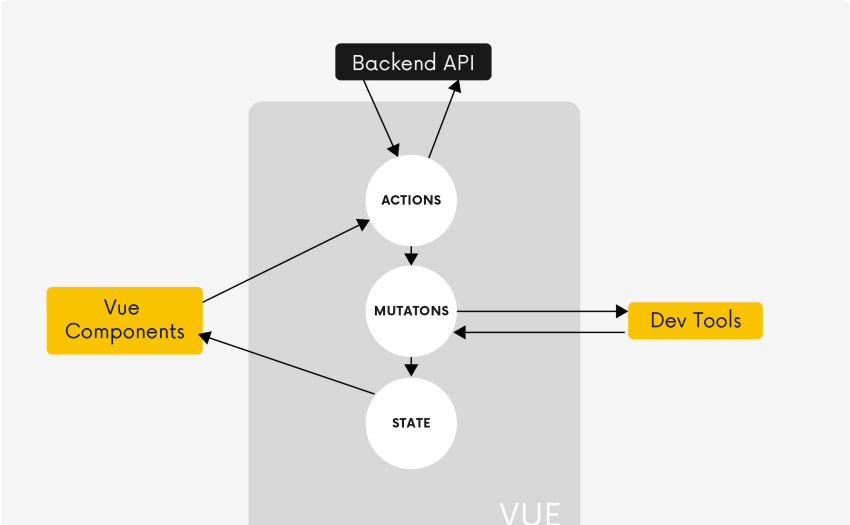
```
this.$store.commit('increment', { amount: 1 })
```

- O utilizando una **action** del store que nos permite agregar lógica de negocio interno de la store:

```
this.$store.dispatch('increaseCount', {amount: this.num})
```

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.
- O utilizar **actions** que hagan el commit de las mutaciones es una buena práctica.



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.dispatch('increaseCount',{amount: this.num})
    },
    decrement () {
      this.$store.commit('decrement',{amount: this.num})
    }
```

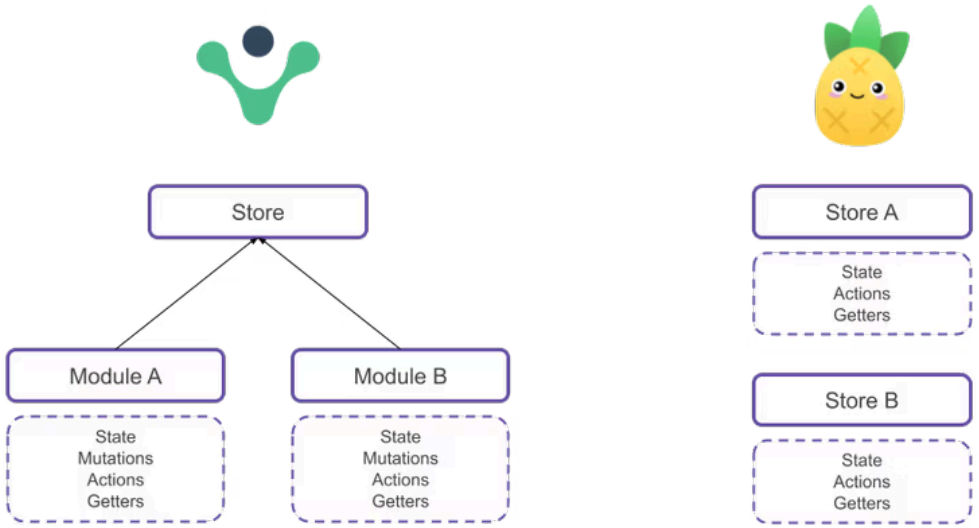
Evolución de Vuex: Pinia

Según la documentación oficial, es una nueva versión de VueX

Sitio de Pinia: https://pinia.vuejs.org/

Características Pinia: Múltiples stores

- Pinia genera un Store por módulo, a diferencia de VueX donde se tiene un único store con varios módulos.



https://rubenr.dev/es/pinia-vuex

Características Pinia: No más mutaciones

- El estado se actualiza directamente en nuestras **actions**, reduciendo así la verbosidad y complejidad. Tampoco es necesario **context** en la acciones.

```
import { defineStore } from "pinia";
export const useCounterStore = defineStore("counter", {
  state: () => {
    return { count: 0 };
  },
  actions: {
    increment(value = 1) {
      this.count += value;
    },
  },
  getters: {
    doubleCount: (state) => {
      return state.count * 2;
    },
    doublePlusOne() {
      return this.doubleCount + 1
```

Otras características Pinia

- Completa integración con **TypeScript**.
- Muy ligero, pesa sobre **1kb**.
- Soporte para Vue **devtools**.
- Soporte para **SSR**.

Ejemplos: Pinia

- Veamos ejemplos:
  - Pinia + Options API
  - Pinia + Composition API => https://github.com/piniajs/example-vue-3-vite

Pinia vs VueX - ref

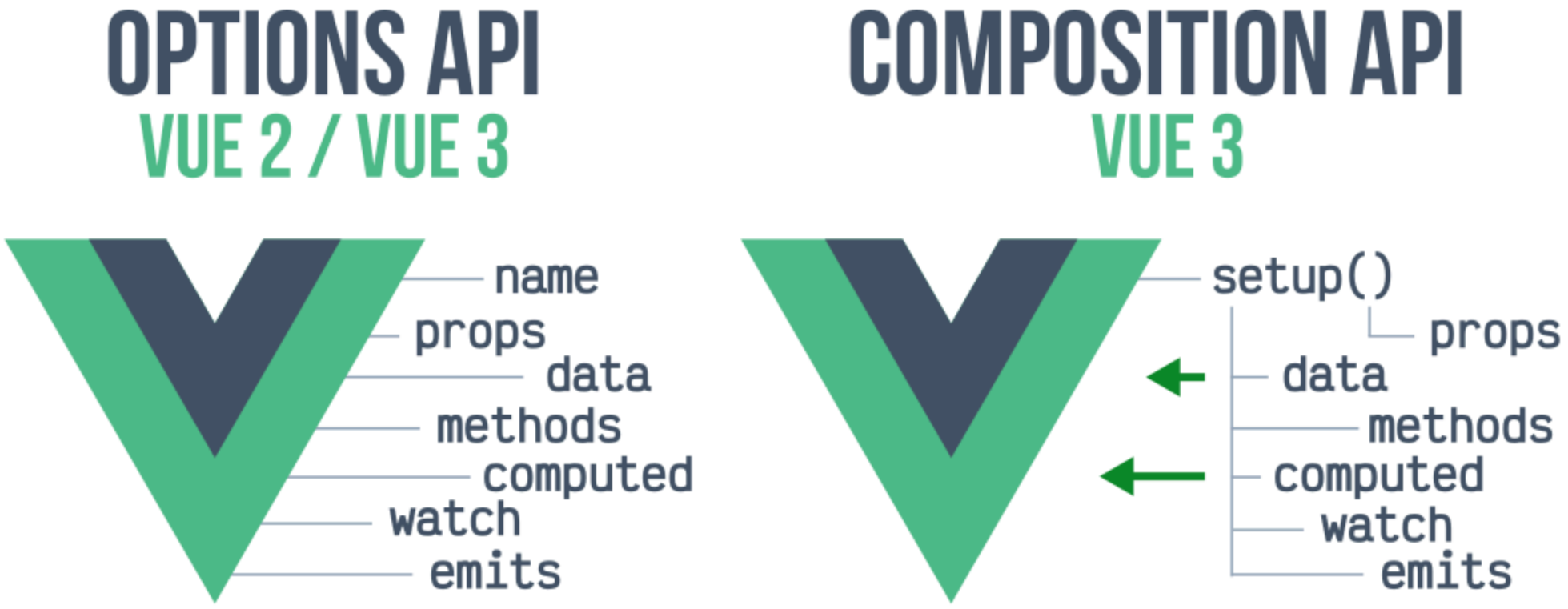
	Pinia	Vuex
Integration	Built specifically for Vue 3 and Composition API	Built for Vue 2
TypeScript	Strong TypeScript support and type inference	Limited TypeScript support
Performance	Efficient reactivity system with reduced overhead	Uses Vue 2's reactivity system
Modularity	Encourages modular architecture with separate stores	Single global store by default
DevTools	Seamlessly integrates with Vue DevTools	Seamlessly integrates with Vue DevTools
API Simplicity	Minimalistic and straightforward API	Comprehensive API with multiple concepts (state, mutations, actions)
Composition	Works seamlessly with the Composition API	Not specifically designed for Composition API
Adoption	Gaining traction in the Vue ecosystem	Widely adopted and established in the Vue ecosystem

Pinia vs VueX

Para seguir leyendo:

- https://www.vuemastery.com/blog/advantages-of-pinia-vs-vuex/#a-pinia-example
- https://medium.com/@haidizakelek/vuex-or-pinia-8bfbeda11339
- https://imaginaformacion.com/tutoriales/pinia-vs-vuex-cual-es-mejor
- https://vuejsdevelopers.com/2023/04/11/pinia-vs-vuex---why-pinia-wins/

Options Api vs Composition Api



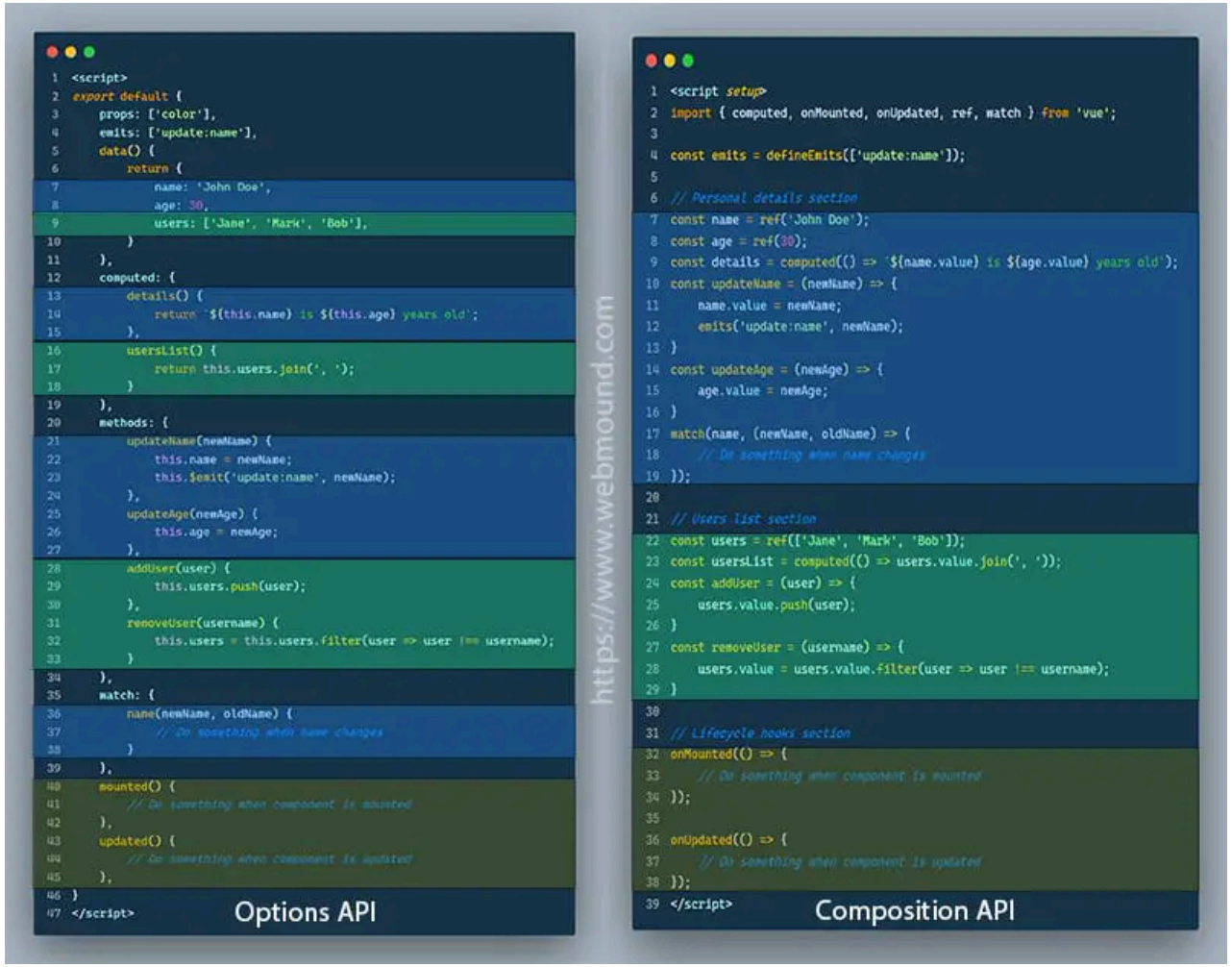
Options Api

- Forma tradicional de escribir componentes en Vue 2 y anteriores.
- Se basa en un objeto que contiene propiedades como **data**, **methods**, **computed**, **watch**, etc.
- Los datos y métodos se definen en el objeto data y se acceden mediante **this** dentro del componente.
- Ideal para **componentes simples** y fáciles de entender.

Composition Api

- Introducida en Vue 3 como una forma moderna y flexible de definir componentes.
- Se basa en funciones que pueden ser reutilizadas y agrupadas en **composiciones**.
- Permite una organización **modular** y **reutilizable** de la lógica del componente.
- Facilita la agrupación de datos, métodos y reactividad en composiciones.
- Ideal para **componentes más complejos y grandes**, mejora la mantenibilidad y la comprensión del código.

Options Api vs Composition Api



- https://medium.com/codex/options-api-vs-composition-api-4a745fb8610

Para seguir leyendo: Vuejs

- Vue Router: https://router.vuejs.org/
- Vue Vuex : https://vuex.vuejs.org/ y Pinia: https://pinia.vuejs.org/
- Webpack: https://www.youtube.com/watch?v=2UBKjshUwM8
- Componentes y Plugins: https://madewithvuejs.com/
- Gitlab.com usa vue: https://about.gitlab.com/2016/10/20/why-we-chose-vue/
- Podcast Pinia vs Vuex - Vite vs Webpack para Vue: https://www.youtube.com/watch?v=FAmdgaYpaOc

¿Dudas?

Fin