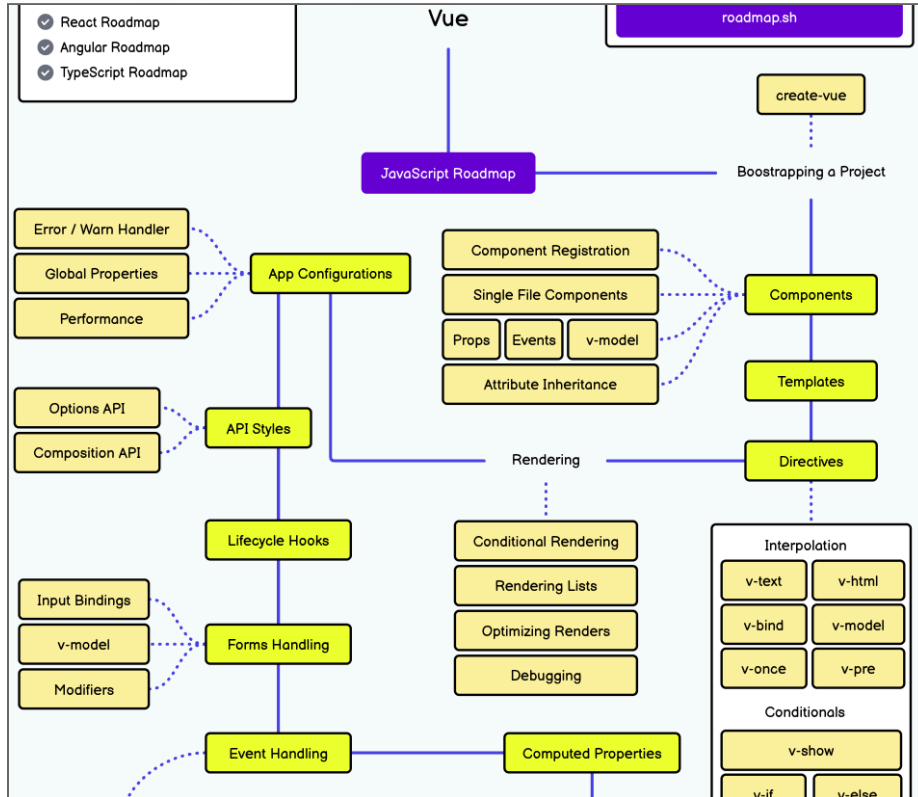


# PROYECTO DE SOFTWARE

# SEGUIMOS CON VUE.JS



<https://roadmap.sh/vue>

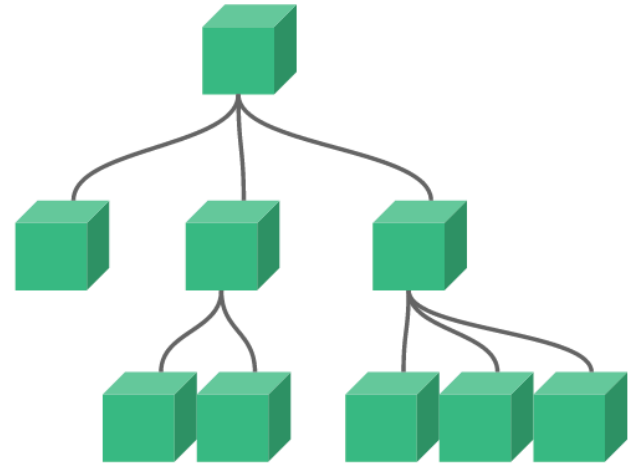
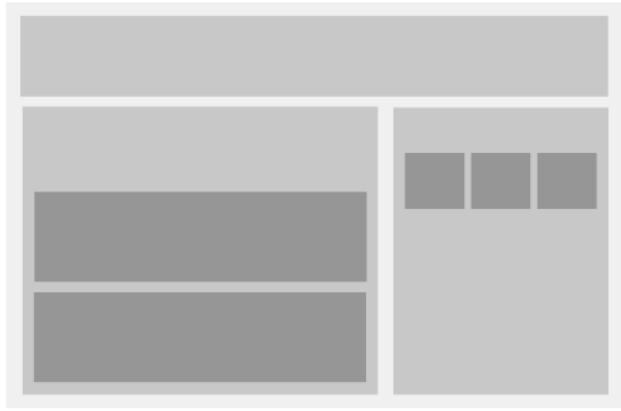
# TEMARIO

## 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="Bloggng 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

- **Vue CLI** es una herramienta de línea de comandos para generar proyectos Vue.js basada en **webpack**. Se está 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 (<https://cli.vuejs.org/>).

## 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.
- 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 construcción: 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** y **Vite**.

## 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:

```
<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 + 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>

<style>
</style>
```

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

## RUTEO EN VUE

- Para Vue 3 vamos a utilizar la librería vue-router con 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**.

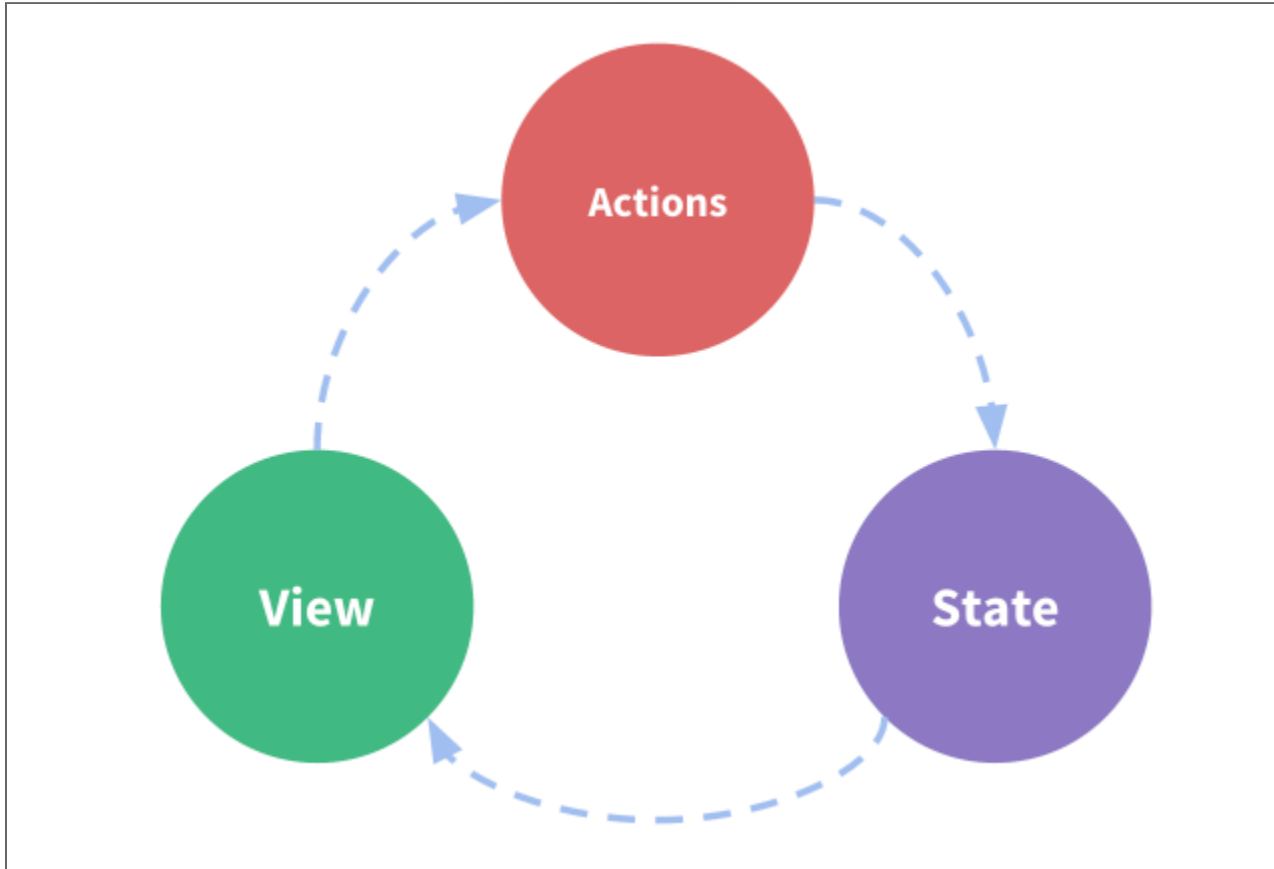
```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/ruta1">Ruta 1</router-link> |
      <router-link to="/ruta2">Ruta 2</router-link> |
      <router-link to="/about">About</router-link>
    </div>
    <router-view/>
  </div>
</template>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

- Veamos el ejemplo en [ejemplo-router](#).

**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

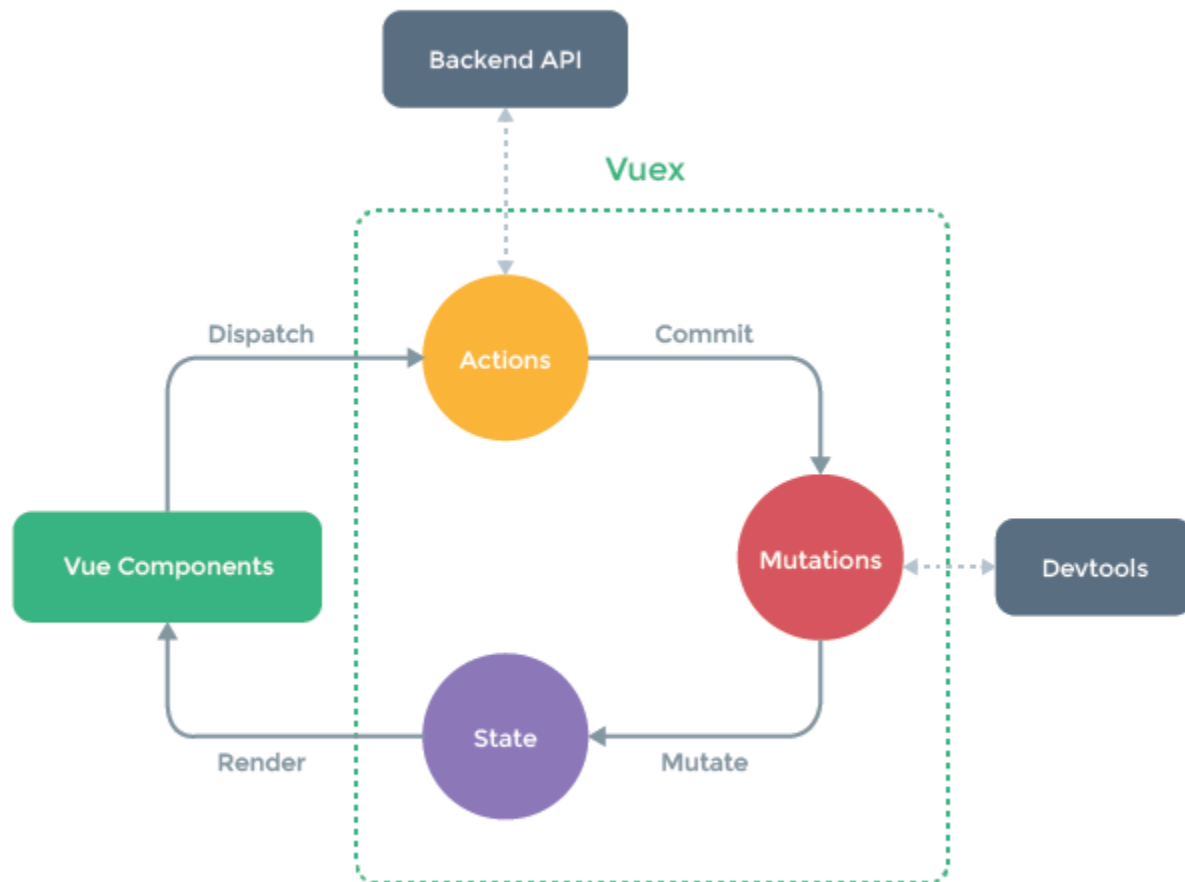
- Una "**store**" es básicamente un contenedor del estado de la aplicación.
- Hay 2 cosas en la que las stores de Vuex se diferencian de un objeto global plano:
  - Las stores Vuex **son reactivas**: cuando un componente saca sus valores de una store, este va a actualizarse reactivamente ante un cambio 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
    }
  },
  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 { 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.$store.commit('increment', { amount: 1 })  
console.log(this.$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 ejemplo con **Múltiples Contadores**.

# EVOLUCIÓN DE VUEX: **PINIA**

- Según la documentación oficial, es una nueva versión de Vuex
- <https://pinia.vuejs.org/>
- Pinia vs Vuex: <https://medium.com/@haidizakelek/vuex-or-pinia-8bfbeda11339>
- Veamos ejemplos:
- Pinia + Options API
- Pinia + Composition API => <https://github.com/piniajs/example-vue-3-vite>

# OPTIONS API VS COMPOSITION API

## OPTIONS API

VUE 2 / VUE 3



## COMPOSITION API

VUE 3



# OPTIONS API

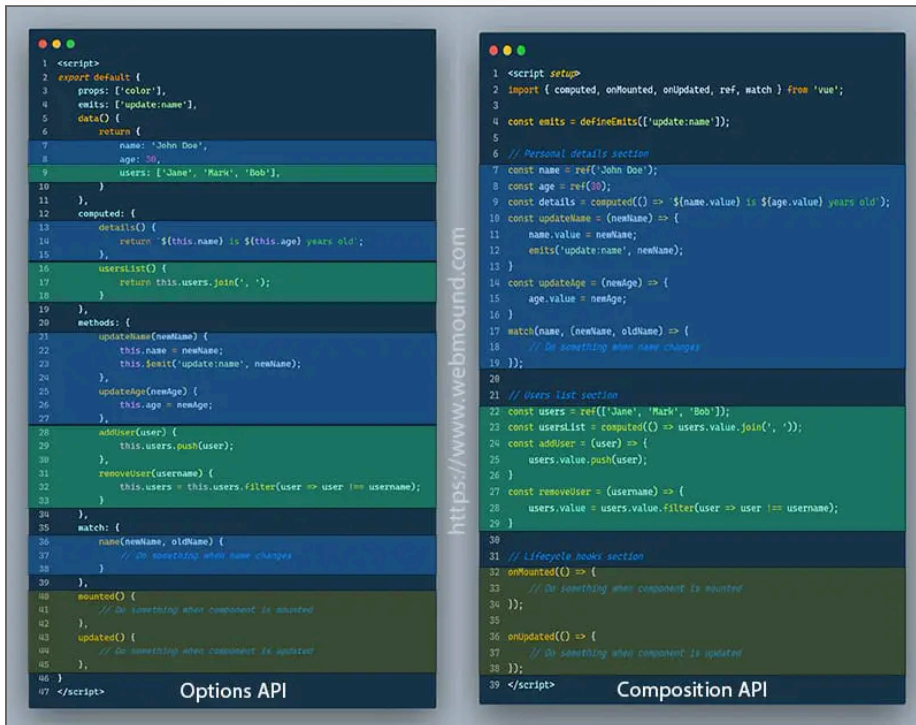
- Forma tradicional de escribir componentes en Vue 2 y anteriores.
- Se basa en un objeto que contiene propiedades como **data**, **methods**, **computed**, **watch**, etc.
- Los datos y métodos se definen en el objeto data y se acceden mediante **this** dentro del componente.
- Ideal para **componentes 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, efectos secundarios 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://www.webmound.com/composition-api-vs-options-api-in-vue-3/>
- <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**

Speaker notes