



Framework



Vue.js

Presentated by  
Ana Cristina Henao G  
Juan David Rendon B

# ¿Qué es Vue.js?

Vue.js es un framework progresivo de JavaScript diseñado para construir interfaces de usuario interactivas y dinámicas, facilitando el desarrollo de aplicaciones web modernas y eficientes.

Su arquitectura basada en componentes permite una gestión eficiente del estado y la reactividad, lo que significa que los cambios en los datos se reflejan automáticamente en la interfaz de usuario.

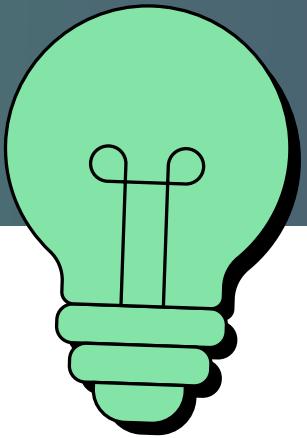
Vue.js se puede integrar fácilmente con otros proyectos y bibliotecas, lo que lo convierte en una opción versátil para desarrolladores que buscan mejorar aplicaciones existentes o crear nuevas desde cero.

Curva de aprendizaje suave

Facilidad de integración

Modularidad y extensibilidad

Reactividad



# Historia y evolución de Vue.js

**2014**  
Vue.js fue creado por Evan You, quien buscaba un framework ligero y flexible para el desarrollo de interfaces de usuario.

**2016**  
La versión 2.0 fue lanzada, consolidando a Vue.js como un framework maduro y estable, con mejoras significativas en rendimiento.

Vue.js

**2018**  
Se lanzó Vue 2.5, que incluyó soporte para TypeScript y mejoras en la documentación, facilitando su adopción.



**2015**

Se lanzó la versión 0.11, introduciendo características clave como la reactividad y el sistema de componentes.



**2017**

Vue.js ganó popularidad en la comunidad de desarrolladores, siendo adoptado por empresas como Alibaba y Xiaomi.



**2020**

Se anunció el desarrollo de Vue 3, que prometía un rendimiento mejorado y una nueva API basada en composición.



# Instalación

importante tener instalada una versión actualizada de [Node.js](#) y de que tu directorio de trabajo actual sea el que deseas utilizar para crear un proyecto.

Ejecuta el siguiente comando en tu línea de comandos:

`npm create vue@latest`

(npm: Node package manager)

Vue.js

Este comando instalará y ejecutará [create-vue](#), la herramienta oficial de desarrollo de proyectos de Vue.

Se le mostrarán indicaciones para varias funciones opcionales, como compatibilidad con TypeScript y pruebas:

```
✓ Project name: ... <your-project-name>
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit testing? ... No / Yes
✓ Add an End-to-End Testing Solution? ... No / Cypress / Nightwatch / Playwright
✓ Add ESLint for code quality? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
✓ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes
```

Scaffolding project in ./<your-project-name>...

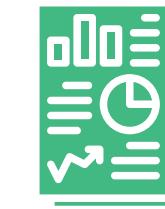
Done.

Si no está seguro de alguna opción, simplemente elija No presionando Enter por ahora. Una vez creado el proyecto, siga las instrucciones para instalar las dependencias e iniciar el servidor de desarrollo:

```
cd <your-project-name>
npm install
npm run dev
```

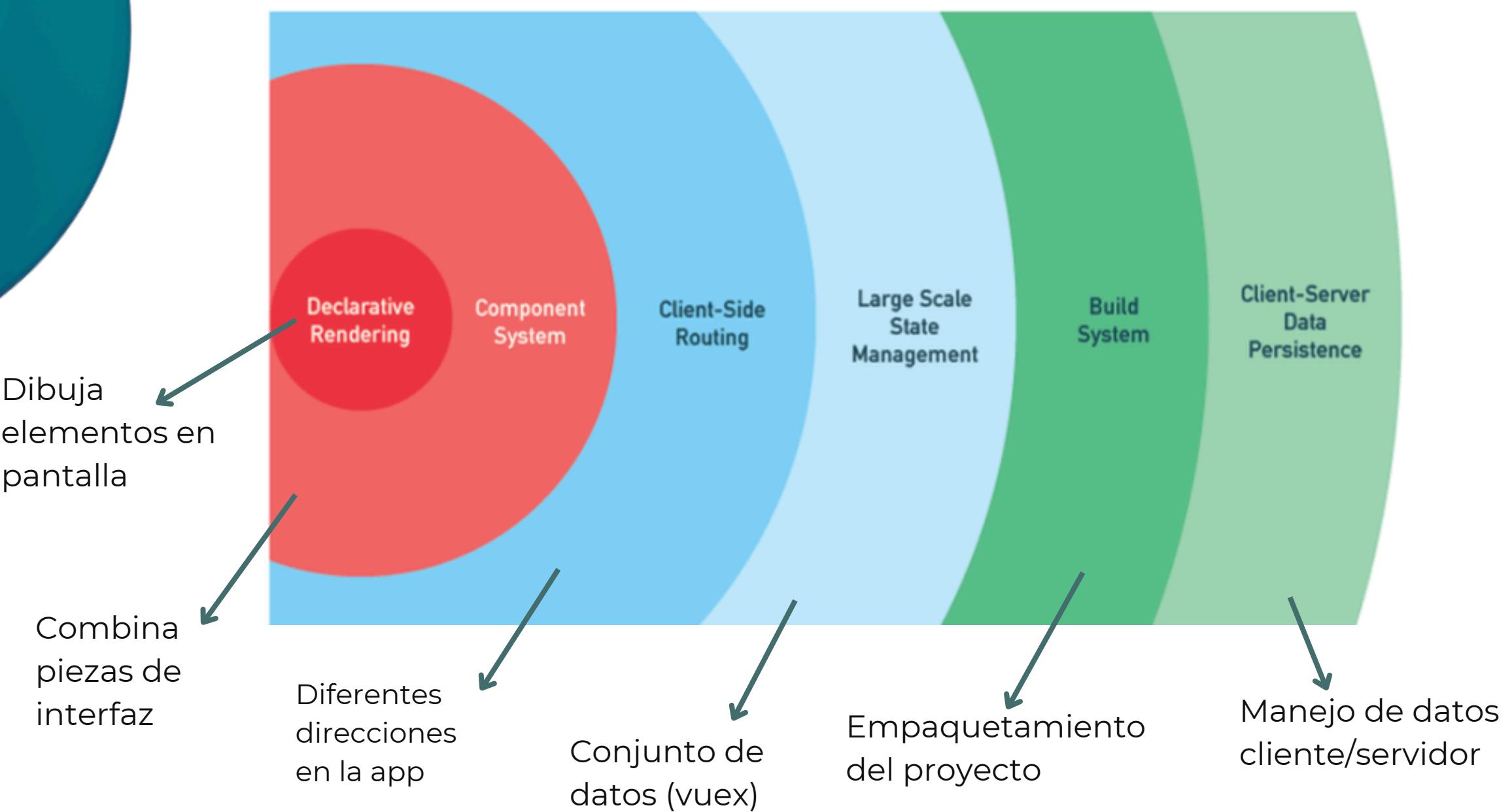


# Características



## Reactivo

Dinamismo del contenido (Actualización del DOM)





# Características



## Modulación de código

- Separar la lógica de la aplicación en módulos/bloques.
- Dividir la parte de la interfaz en componentes independientes para su reutilización.



## Modulación de código

- Tamaño aproximado: 18 KB
- Ecosistema modular: Librerías y herramientas (Vue Router y Vuex)



## Directivas

- Manipulación del DOM de manera declarativa

# Directivas

## (Sistema de datos en Vue)

### V-BIND

enlace: atributo-valor

```
  
!-- También puedes usar simplemente `:src` como atajo para `v-bind:src` -->
```

### V-MODEL

reflejar cambios automáticamente

```
<input v-model="nombre" placeholder="Escribe tu nombre">  
<p>Hola, {{ nombre }}!</p>
```

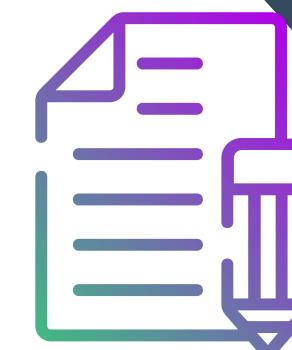
```
export default {  
  data() {  
    return {  
      mensaje: 'Hola, Vue.js'  
    };  
  }  
};
```



### V-IF ; V-ELSE

incluir/ excluir partes interfaz

```
<p v-if="usuarioAutenticado">Bienvenido de nuevo, usuario.</p>  
<p v-else>Por favor, inicia sesión.</p>
```



### V-FOR

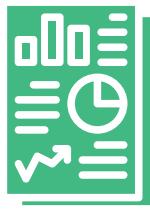
Iterar sobre lista de elementos

```
<ul>  
  <li v-for="(item, index) in items" :key="index">{{ item }}</li>  
</ul>
```



# Estructura básica de un componente Vue

Cada aplicación Vue comienza creando una nueva instancia de aplicación con la `createApp` función:



<Instancia>

Núcleo de la Aplicación. Donde se inicializa

```
import { createApp } from 'vue'

const app = createApp({
  /* root component options */
})
```

El componente raíz

```
import { createApp } from 'vue'
// import the root component App from a single-file component.
import App from './App.vue'

const app = createApp(App)
```



# Componentes en Vue

Son bloques reutilizables que permiten dividir la aplicación en partes pequeñas y manejables, cada una con su propia lógica, estilos y estructura.



## <template>

Aquí defines el HTML de tu componente. Este es el lugar donde colocas la estructura visual del componente.

```
<template>
  <div>
    <h1>{{ mensaje }}</h1>
    <button @click="saludar">Haz clic aquí</button>
  </div>
</template>
```



# Componentes en Vue



<script>

Aquí defines la lógica de tu componente. En este bloque, usas export default para exportar un objeto de configuración que incluye las propiedades data, methods, computed, etc.

```
<script>
export default {
  name: 'MiComponente', // Opcional pero recomendado para depuración
  data() {
    return {
      mensaje: '¡Hola, Vue!'
    }
  },
  methods: {
    saludar() {
      alert('¡Hola desde Vue!');
    }
  }
}
</script>
```



# Componentes en Vue



<style>

Aquí defines el CSS que afectará solo a este componente  
(con scoped para evitar conflictos de estilos con otros  
componentes)

```
<style scoped>
  h1 {
    color: blue;
  }
</style>
```



# Métodos

## manejo de eventos

los métodos y el manejo de eventos permiten crear interactividad en los componentes.

### Definir Métodos

```
export default {
  data() {
    return {
      contador: 0
    };
  },
  methods: {
    incrementarContador() {
      this.contador++;
    }
  }
};
```

### Manejo de Eventos con v-on

```
<button v-on:click="incrementarContador">Incrementar</button>
<p>Contador: {{ contador }}</p>
```

### Pasar Parámetros a Métodos

```
<button @click="incrementarContador(5)">Incrementar en 5</button>
```

```
methods: {
  incrementarContador(valor) {
    this.contador += valor;
  }
}
```

# Propiedades computadas y watchers



- Propiedades Computadas (computed): las propiedades computadas derivan de otros datos y se actualizan automáticamente cuando estos cambian.
- Watchers: son observadores que responden a cambios específicos en los datos.

## Computed

```
export default {
  data() {
    return {
      nombre: 'Juan',
      apellido: 'Pérez'
    },
    computed: {
      nombreCompleto() {
        return `${this.nombre} ${this.apellido}`;
      }
    }
};
```

## Watch

```
export default {
  data() {
    return {
      mensaje: 'Hola'
    },
    watch: {
      mensaje(nuevoValor, valorAntiguo) {
        console.log('mensaje cambió de', valorAntiguo, 'a', nuevoValor);
      }
    }
};
```

# Enrutamiento con Vue Router

Es una herramienta para manejar rutas dentro de una aplicación Vue, permitiendo navegación sin recargar la página.

## Configuración básica

```
// router/index.js
import { createRouter, createWebHistory } from 'vue-router';
import Home from '../views/Home.vue';
import About from '../views/About.vue';

const routes = [
  { path: '/', name: 'Home', component: Home },
  { path: '/about', name: 'About', component: About }
];

const router = createRouter({
  history: createWebHistory(),
  routes
});

export default router;
```

## Importación

```
// main.js
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

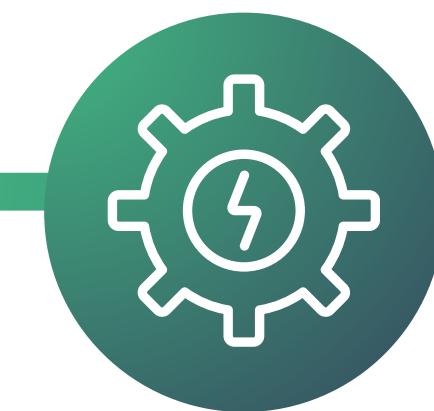
const app = createApp(App);
app.use(router);
app.mount('#app');
```

## Uso de rutas

```
<template>
  <div>
    <nav>
      <router-link to="/">Inicio</router-link>
      <router-link to="/about">Acerca de</router-link>
    </nav>
    <router-view></router-view>
  </div>
</template>
```

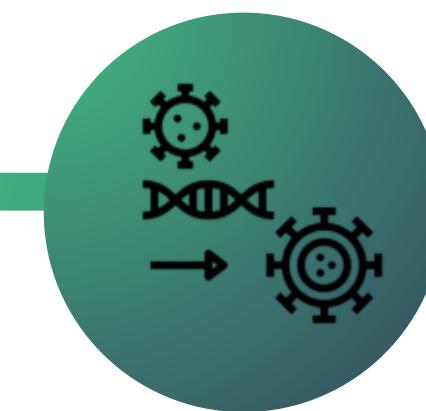


# Gestión de estado con Vuex



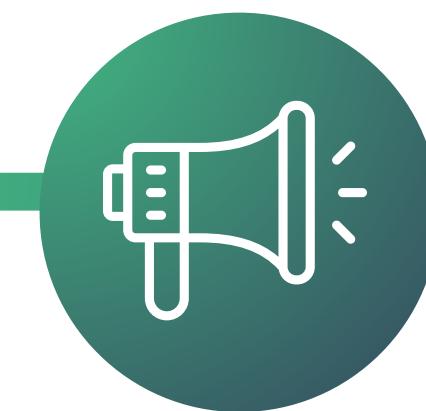
## state

Define el estado o los datos de la aplicación.



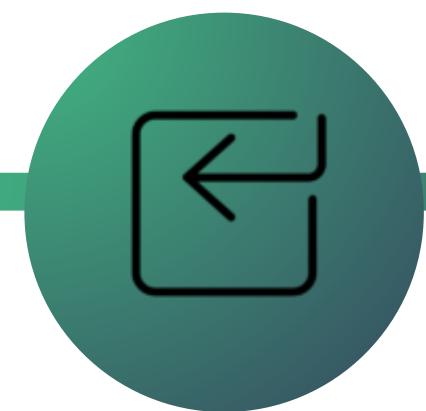
## mutations

Son funciones sincrónicas que modifican el state directamente.  
Son la única forma de cambiar el estado en Vuex.



## actions

Son funciones asincrónicas (como llamadas a API) que, una vez completadas, llaman a las mutaciones para modificar el state.



## getters:

Son funciones que derivan o calculan valores a partir del state.



## Creación del Store

```
// store/index.js
import { createStore } from 'vuex';

export default createStore({
  state: {
    contador: 0
  },
  mutations: {
    incrementar(state) {
      state.contador++;
    },
    setContador(state, valor) {
      state.contador = valor;
    }
  },
  actions: {
    incrementarAsync({ commit }) {
      setTimeout(() => {
        commit('incrementar');
      }, 1000);
    }
  },
  getters: {
    contadorCuadrado(state) {
      return state.contador * state.contador;
    }
  }
});
```

## Configurar

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

const app = createApp(App);
app.use(store);
app.mount('#app');
```

## Acceder a Datos

```
<template>
  <div>
    <p>Contador: {{ contador }}</p>
  </div>
</template>

<script>
export default {
  computed: {
    contador() {
      return this.$store.state.contador;
    }
  }
};
</script>
```



## Llamar a Mutaciones

```
<template>
<div>
  <button @click="incrementarContador">Incrementar</button>
</div>
</template>

<script>
export default {
  methods: {
    incrementarContador() {
      this.$store.commit('incrementar');
    }
  }
};
</script>
```

## Llamar a Acciones

```
export default {
  methods: {
    incrementarContadorAsync() {
      this.$store.dispatch('incrementarAsync');
    }
  }
};
</script>
```

## Usar Getters

```
export default {
  computed: {
    contadorCuadrado() {
      return this.$store.getters.contadorCuadrado;
    }
  }
};
```



# Uso y Comunicación de Componentes

Se registra en la instancia principal (por ejemplo, en main.js), y luego puede usarse en cualquier lugar de la aplicación.

## Componente Global

```
// main.js
import { createApp } from 'vue';
import App from './App.vue';
import ComponenteHijo from './components/ComponenteHijo.vue';

const app = createApp(App);
app.component('ComponenteHijo', ComponenteHijo); // Registro global
app.mount('#app');
```

# Uso y Comunicación de Componentes



## ComponenteHijo.vue

```
<template>
  <div>
    <p>{{ mensaje }}</p>
    <button @click="enviarMensajeAlPadre">Enviar al padre</button>
  </div>
</template>

<script>
export default {
  name: 'ComponenteHijo',
  props: ['mensaje'],
  methods: {
    enviarMensajeAlPadre() {
      this.$emit('mensajeEnviado', 'Mensaje desde el hijo');
    }
  }
};
</script>

<style scoped>
p {
  color: blue;
}
</style>
```

## App.vue

```
<template>
  <div>
    <ComponenteHijo :mensaje="mensajePadre" @mensajeEnviado="manejarMensajeDesdeHijo" />
  </div>
</template>

<script>
import ComponenteHijo from './ComponenteHijo.vue';

export default {
  components: {
    ComponenteHijo
  },
  data() {
    return {
      mensajePadre: 'Hola desde el padre'
    };
  },
  methods: {
    manejarMensajeDesdeHijo(mensaje) {
      console.log('Mensaje recibido del hijo:', mensaje);
    }
  }
};
</script>
```



# Lista de Tareas



04

## Tarealtem.vue

Cada tarea es un componente individual que muestra su texto y un botón para eliminarla. Cuando el usuario hace clic en "Eliminar", se emite un evento que permite al componente padre ListaTareas eliminar la tarea.

01

03

02

## App.vue

Este componente será el "padre" que maneje el estado de la lista de tareas y pasará los datos necesarios a los componentes AgregarTarea y ListaTareas.

## AgregarTarea.vue

Este componente es un formulario que envía la nueva tarea al componente App.vue usando un evento personalizado.

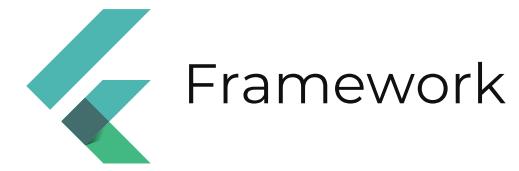
## ListaTareas.vue

Este componente recibe la lista de tareas del componente App.vue y las muestra usando el componente Tarealtem.vue. Además, emite un evento para eliminar tareas.

# Documentación

- Introducción a VUE:  
<https://vuejs.org/guide/introduction>
- Vue 3.0:  
<https://cloudappi.net/vue-3-0/>
- ¿Que es vue?:  
<https://www.youtube.com/watch?v=AqesL138vMA>
- Por qué elegir vue:  
<https://www.genbeta.com/desarrollo/por-que-elegir-vuejs-5-razones-para-considerarlo-nuestro-proximo-framework-de-referencia>





# Muchas Gracias

