Vue

Sistemas y Tecnologías Web (CC3062) - 2026

Vue — Extra: Comparación con React

Semestre 01, 2026

El panorama frontend

Hemos aprendido React a fondo.

React no es el único camino.

Vue es el principal competidor directo — misma categoría, filosofía distinta.

Conocer ambos hace mejores desarrolladores.

Los tres grandes frameworks

| Framework | Creado por | Año | Modelo | | --- | --- | --- | --- | | React | Meta (Facebook) | 2013 | Biblioteca UI | | Vue | Evan You | 2014 | Framework progresivo | | Angular | Google | 2016 | Framework completo |

Esta presentación se enfoca en **Vue vs React** — los más similares en propósito.

¿Por qué aprender Vue si ya sé React?

- Vue domina en Asia y Europa Occidental — muchas empresas lo usan.
- Algunos proyectos existentes están en Vue — hay que poder leerlos.
- Vue tiene ideas propias que complementan el conocimiento de React.
- El mercado laboral premia conocer múltiples herramientas.

Setup

React (Vite)

```
npm create vite@latest mi-app -- --template react
cd mi-app
npm install
npm run dev
```

```
src/
├── App.jsx
├── main.jsx
└── components/
```

Vue (Vite)

```
npm create vite@latest mi-app -- --template vue
cd mi-app
npm install
npm run dev
```

```
src/
├── App.vue
├── main.js
└── components/
```

La configuración inicial es prácticamente idéntica — ambos usan Vite.

La diferencia está en los archivos: `.jsx` vs `.vue`.

Single File Components vs JSX

React: todo en JavaScript

```
// Tarjeta.jsx
import { useState } from 'react';
import './Tarjeta.css';

export function Tarjeta({ titulo, descripcion }) {
  const [expandida, setExpandida] = useState(false);

  return (
    <div className="tarjeta">
      <h2>{titulo}</h2>
      {expandida && <p>{descripcion}</p>}
      <button onClick={() => setExpandida(!expandida)}>
        {expandida ? 'Cerrar' : 'Ver más'}
      </button>
    </div>
  );
}
```

HTML, lógica y estilos en un solo archivo `.jsx` — todo es JavaScript.

Vue: Single File Component

```
<!-- Tarjeta.vue -->
<template>
  <div class="tarjeta">
    <h2>{{ titulo }}</h2>
    <p v-if="expandida">{{ descripcion }}</p>
    <button @click="expandida = !expandida">
      {{ expandida ? 'Cerrar' : 'Ver más' }}
```

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

<script setup>
import { ref } from 'vue';

const props = defineProps({ titulo: String, descripcion: String });
const expandida = ref(false);
</script>

<style scoped>
.tarjeta { border: 1px solid #ccc; padding: 1rem; }
</style>
```

Un `.vue` tiene tres secciones separadas: `<template>`, `<script>`, `<style>`.

Los estilos con `scoped` solo afectan a ese componente — sin colisiones de CSS.

Diferencia filosófica

| | React | Vue | | --- | --- | --- | | Template | JSX — HTML dentro de JS | HTML extendido con directivas | | Estilos | CSS modules, styled-components, Tailwind | `<style scoped>` integrado | | Lógica | JavaScript puro | JavaScript + Composition API | | Curva de aprendizaje | Requiere entender JS bien | HTML más familiar para diseñadores |

Estado reactivo

React: useState

```
import { useState } from 'react';

function Contador() {
```

```
  const [conteo, setConteo] = useState(0);

  return (
    <div>
      <p>Conteo: {conteo}</p>
      <button onClick={() => setConteo(c => c + 1)}>+</button>
      <button onClick={() => setConteo(c => c - 1)}>-</button>
    </div>
  );
}
```

En React: `[valor, setter]` — el estado nunca se muta directamente.

Vue: ref

```
<template>
  <div>
    <p>Conteo: {{ conteo }}</p>
    <button @click="conteo++">+</button>
    <button @click="conteo--">-</button>
  </div>
</template>

<script setup>
import { ref } from 'vue';
const conteo = ref(0);
</script>
```

En Vue: `ref(valor)` devuelve un objeto reactivo. Se accede con `.value` en el script, sin `.value` en el template.

Estado con objetos

React — siempre crear un nuevo objeto:

```
const [usuario, setUsuario] = useState({ nombre: '', email: '' });

// □ Crear copia con spread
setUsuario({ ...usuario, nombre: 'Ana' });
```

Vue — se puede mutar directamente:

```
<script setup>
import { reactive } from 'vue';

const usuario = reactive({ nombre: '', email: '' });

// □ Mutación directa — Vue detecta el cambio
function cambiarNombre() {
  usuario.nombre = 'Ana';
}
</script>
```

`reactive` en Vue trackea los cambios internos del objeto automáticamente.

Valores computados vs useMemo

React: useMemo

```
const productosFiltrados = useMemo(() =>
  productos.filter(p => p.precio < filtro),
  [productos, filtro]
);
```

Vue: computed

```
<script setup>
import { ref, computed } from 'vue';
```

```
const productos = ref([...]);
const filtro = ref(1000);

const productosFiltrados = computed(() =>
  productos.value.filter(p => p.precio < filtro.value)
);
</script>
```

Vue: `computed` se recalcula automáticamente cuando sus dependencias cambian.

No hay que declarar el array de dependencias — Vue lo detecta solo.

Efectos secundarios

React: useEffect

```
useEffect(() => {
  document.title = `Conteo: ${conteo}`;
}, [conteo]);

useEffect(() => {
  const sub = suscribirse(id);
  return () => sub.cancelar(); // cleanup
}, [id]);
```

Vue: watch y watchEffect

```
<script setup>
import { ref, watch, watchEffect } from 'vue';

const conteo = ref(0);
const id = ref(1);

// watch: reaccionar a un valor específico
```

```
watch(conteo, (nuevo, anterior) => {
  document.title = `Conteo: ${nuevo}`;
});

// watchEffect: se ejecuta y detecta dependencias automáticamente
watchEffect((onCleanup) => {
  const sub = suscribirse(id.value);
  onCleanup(() => sub.cancelar()); // cleanup
});
</script>
```

`watch` — reaccionar a cambios de un valor específico (como `useEffect` con dependencias).

`watchEffect` — detecta dependencias automáticamente (como `useEffect` sin array).

Props y comunicación

Pasar datos hacia abajo

React:

```
function Padre() {
  return <Hijo nombre="Ana" edad={22} />;
}

function Hijo({ nombre, edad }) {
  return <p>{nombre}, {edad} años</p>;
}
```

Vue:

```
<!-- Padre.vue -->
<template>
```

```
  <Hijo nombre="Ana" :edad="22" />
</template>

<!-- Hijo.vue -->
<script setup>
const props = defineProps({ nombre: String, edad: Number });
</script>

<template>
  <p>{{ props.nombre }}, {{ props.edad }} años</p>
</template>
```

`:edad="22"` — los dos puntos indican que el valor es JavaScript, no string.

Sin dos puntos: `edad="22"` sería el string `"22"`.

Comunicación hacia arriba

React — callbacks como props:

```
function Padre() {
  function handleClick(valor) {
    console.log('El hijo envió:', valor);
  }
  return <Hijo onSeleccionar={handleClick} />;
}

function Hijo({ onSeleccionar }) {
  return <button onClick={() => onSeleccionar('opción A')}>Seleccionar</button>;
}
```

Vue — eventos con emit:

```
<!-- Hijo.vue -->
<script setup>
const emit = defineEmits(['seleccionar']);
```

```
</script>

<template>
  <button @click="emit('seleccionar', 'opción A')">Seleccionar</button>
</template>

<!-- Padre.vue -->
<template>
  <Hijo @seleccionar="handleSeleccion" />
</template>

<script setup>
function handleSeleccion(valor) {
  console.log('El hijo emitió:', valor);
}
</script>
```

Directivas de Vue vs JSX

Esta es la diferencia más visible. Vue usa directivas en el HTML, React usa JavaScript puro.

Renderizado condicional

React:

```
{activo && <Contenido />}
{activo ? <Activo /> : <Inactivo />}
```

Vue:

```
<Contenido v-if="activo" />
<Activo v-if="activo" />
<Inactivo v-else />
```

Listas

React:

```
{items.map(item => (
  <li key={item.id}>{item.nombre}</li>
))}
```

Vue:

```
<li v-for="item in items" :key="item.id">
  {{ item.nombre }}
</li>
```

Binding de atributos

React:

```
<input
  value={texto}
  onChange={e => setTexto(e.target.value)}
  className="campo"
  disabled={cargando}
/>
```

Vue:

```
<input
  v-model="texto"
  class="campo"
  :disabled="cargando"
/>
```

`v-model` es two-way binding — equivalente a `value` + `onChange` juntos en React.

Eventos

React:

```
<button onClick={handleClick}>Clic</button>
<input onKeyDown={e => console.log(e.key)} />
```

Vue:

```
<button @click="handleClick">Clic</button>
<input @keydown="e => console.log(e.key)" />
```

`@click` es el shorthand de `v-on:click`.

Tabla de equivalencias React → Vue

| React | Vue | Descripción | | --- | --- | --- | | `{condicion && <El />}` | `v-if="condicion"` | Renderizado condicional | | `{a ? <A/> : <B/>}` | `v-if` + `v-else` | Condicional con alternativa | | `{items.map(...)}` | `v-for` | Listas | | `className="clase"` | `class="clase"` | Clase CSS | | `value` + `onChange` | `v-model` | Input controlado | | `onClick` | `@click` | Evento de clic | | `onChange` | `@change` | Evento de cambio | | `{variable}` | `{{ variable }}` | Interpolación | | `style={{ color: 'red' }}` | `:style="{ color: 'red' }"` | Estilos inline |

Composables vs Custom Hooks

Ambos son la forma de reutilizar lógica entre componentes.

React: Custom Hook

```
// useFetch.js
import { useState, useEffect } from 'react';
```

```
export function useFetch(url) {
  const [data, setData] = useState(null);
  const [cargando, setCargando] = useState(true);

  useEffect(() => {
    fetch(url)
      .then(r => r.json())
      .then(data => { setData(data); setCargando(false); });
  }, [url]);

  return { data, cargando };
}

// Uso en un componente
const { data, cargando } = useFetch('/api/productos');
```

Vue: Composable

```
// useFetch.js
import { ref, watchEffect } from 'vue';

export function useFetch(url) {
  const data = ref(null);
  const cargando = ref(true);

  watchEffect(async () => {
    cargando.value = true;
    const res = await fetch(url.value ?? url);
    data.value = await res.json();
    cargando.value = false;
  });

  return { data, cargando };
}

// Uso en un componente
const { data, cargando } = useFetch('/api/productos');
```

La convención de nombre es idéntica: `use` + nombre descriptivo.

Estado global

React: Context + Zustand

Context para estado simple, Zustand para apps medianas/grandes.

```
// Con Zustand
import { create } from 'zustand';

const useCarritoStore = create(set => ({
  items: [],
  agregar: item => set(state => ({ items: [...state.items, item] })),
}));

// En cualquier componente
const { items, agregar } = useCarritoStore();
```

Vue: Pinia (el estándar actual)

```
// stores/carrito.js
import { defineStore } from 'pinia';

export const useCarritoStore = defineStore('carrito', {
  state: () => ({ items: [] }),
  actions: {
    agregar(item) { this.items.push(item); }
  },
});

// En cualquier componente
const carrito = useCarritoStore();
carrito.agregar(producto);
```

Pinia es al estado global de Vue lo que Zustand es al de React — ambos son la solución moderna y recomendada.

Router

React Router

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Inicio</Link>
        <Link to="/productos">Productos</Link>
      </nav>
      <Routes>
        <Route path="/"          element={<Inicio />} />
        <Route path="/productos" element={<Productos />} />
        <Route path="/productos/:id" element={<Producto />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Vue Router

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

const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: '/',              component: Inicio },
    { path: '/productos',     component: Productos },
```

```
    { path: '/productos/:id', component: Producto },
  ],
});
```

```
<!-- App.vue -->
<template>
  <nav>
    <RouterLink to="/">Inicio</RouterLink>
    <RouterLink to="/productos">Productos</RouterLink>
  </nav>
  <RouterView />
</template>
```

`<RouterView />` es el equivalente de `<Routes>` con todos sus `<Route>`.

Ciclo de vida

React: todo con useEffect

```
useEffect(() => { /* onMounted */ }, []);
useEffect(() => { return () => { /* onUnmounted */ }; }, []);
useEffect(() => { /* onUpdated */ }, [dependencia]);
```

Vue: hooks de ciclo de vida explícitos

```
<script setup>
import { onMounted, onUnmounted, onUpdated } from 'vue';

onMounted(() => {
  console.log('Componente montado');
});

onUnmounted(() => {
  console.log('Componente desmontado');
```

```
});

onUpdated(() => {
  console.log('Componente actualizado');
});
</script>
```

Vue tiene hooks específicos para cada momento — más explícito que `useEffect`.

Comparación global

| Concepto | React | Vue | | --- | --- | --- | | Componente | `.jsx` con función | `.vue` con `<template>` + `<script setup>` | | Estado | `useState` | `ref` / `reactive` | | Efecto | `useEffect` | `watch` / `watchEffect` | | Computed | `useMemo` | `computed` | | Listas | `array.map()` en JSX | `v-for` en template | | Condicional | `&&` / ternario | `v-if` / `v-else` | | Input controlado | `value` + `onChange` | `v-model` | | Props | Desestructuración | `defineProps` | | Emit eventos | Callback como prop | `defineEmits` + `emit` | | Lógica reutilizable | Custom Hook | Composable | | Estado global | Context / Zustand | Pinia | | Router | React Router | Vue Router | | SSR / Full-stack | Next.js | Nuxt.js |

¿Cuál elegir?

Elegir React si…

- Prefieren un enfoque 100% JavaScript (JSX, lógica en funciones puras).
- El proyecto usa TypeScript intensivamente — el ecosistema React está muy maduro.
- Quieren Next.js para SSR/SSG — el ecosistema es más amplio.
- El equipo ya conoce React.

- El mercado laboral objetivo es principalmente Estados Unidos o startups.

Elegir Vue si…

- Prefieren separación clara de template, script y estilos.
- Vienen de un background HTML/CSS — la sintaxis de Vue es más familiar.
- El proyecto es pequeño a mediano y quieren productividad rápida.
- Trabajarán en empresas europeas o asiáticas, o en proyectos con Laravel (el ecosistema PHP lo prefiere).
- Quieren two-way binding sin boilerplate (`v-model`).

La verdad objetiva

| Criterio | Ventaja | | --- | --- | | Popularidad y empleo global | React | | Curva de aprendizaje inicial | Vue | | Flexibilidad | React | | Opinión incluida (convenciones claras) | Vue | | Ecosistema de librerías | React | | Rendimiento | Similares | | Documentación | Ambos excelentes |

No hay respuesta incorrecta. Los conceptos son transferibles — quien sabe uno, aprende el otro en días.

Principios clave

- Vue y React son herramientas distintas para el mismo problema
- Los conceptos son los mismos: estado, props, efectos, routing, estado global
- Vue usa directivas en el template; React usa JavaScript puro en JSX
- `ref` en Vue ≈ `useState` en React; `computed` ≈ `useMemo` ; `watch` ≈ `useEffect`
- Los estilos `scoped` de Vue son una ventaja real para proyectos grandes
- `v-model` es más cómodo que el patrón `value + onChange` de React

- React tiene mayor ecosistema; Vue tiene curva de aprendizaje más suave
- Los Composables de Vue y los Custom Hooks de React son la misma idea con distinta sintaxis