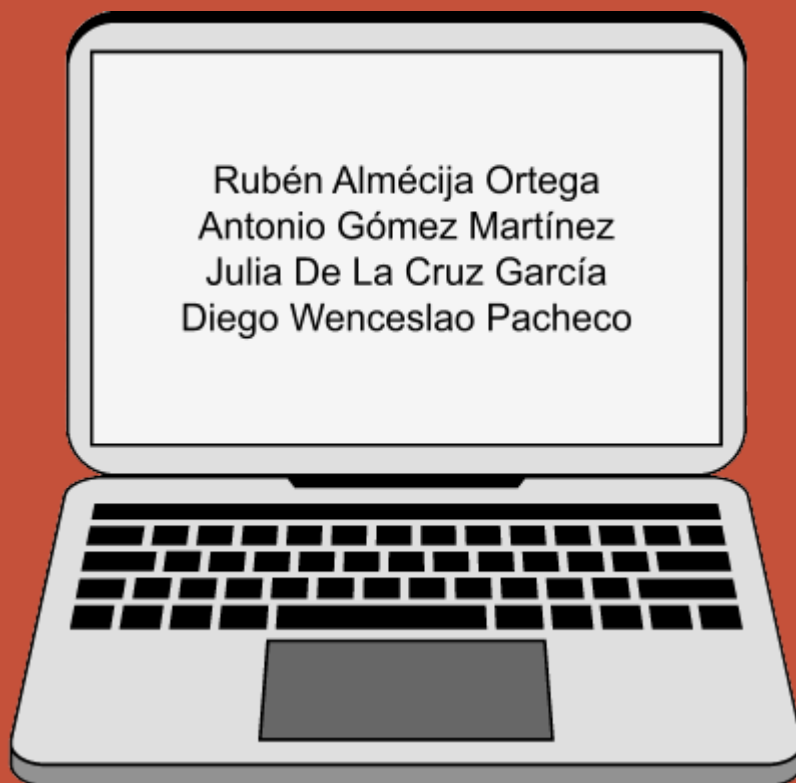


# React vs Vue

## Trabajo Grupal





## HISTORIA:

VUE nació tras la colaboración de Evan You con Google en varios proyectos que usaban AngularJS, llevando así al creador, a darse cuenta de que podía crear un Framework que tuviese lo que le encantaba de AngularJS y además fuese muy ligero. A día de hoy sigue siendo desarrollado y mantenido por un grupo independiente de desarrolladores.

## QUÉ ES:

Es así como nació este framework basado en JavaScript, el cual ofrece herramientas web muy potentes para desarrollar proyectos de FrontEnd. Por las varias fuentes que hemos estado revisando y por experiencia propia, sabemos que VUE es flexible e incluso nos da más de lo que ya conocíamos de JavaScript.

Además, utiliza lo que se conoce como un *DOM Virtual* <sup>1</sup>. VUE se centra en la simplicidad y flexibilidad que permite a los desarrolladores una integración fácil en sus proyectos actuales.

## COMPONENTES QUE LA DESTACAN:

- Virtual DOM.

- Enlace de datos bidireccional:

VUE ofrece un enlace de datos hacia ambas direcciones como parte de su framework MVVN. Con esto podemos editar o aplicar valores a los elementos HTML. También podemos cambiar el estilo y aplicar clases.

- Componentes:

Los *components* son objetos VUE que tienen un HTML personalizado y pueden ser reutilizados. Los elementos HTML y los objetos VUE pueden interactuar uno con el otro mediante eventos y propiedades. El código bloque de un componente VUE es muy importante también para las aplicaciones VUE que son escalables y estables.

- Propiedades computacionales:

Estas nos ayudan a realizar la lógica relevante y a ver las modificaciones introducidas en los elementos de la UI, eliminando la necesidad de una mayor codificación. Cuando tenemos la intención de trabajar en la mutación de una variable que depende de la alteración de otra propiedad, debemos usar una propiedad calculada. Otros atributos de datos influyen en gran medida en las propiedades calculadas.

- Transiciones y Animaciones CSS.

- *Watchers:*

Los observadores se utilizan en datos que es probable que cambien regularmente, como los elementos de entrada de formularios. Un desarrollador no necesita realizar ninguna acción adicional en este caso. El observador maneja cualquier actualización de datos sin dejar de mantener el código simple y rápido.



## HISTORIA:

React es un framework que ya tiene tiempo. Fue creado en 2013 para la creación de interfaces dinámicas en sitios web. En este caso el DOM Virtual (el cual es una representación de los elementos DOM contruidos con React) es en lo que se basa React.

## QUÉ ES:

React es una librería JavaScript que puede ser usada para la creación de *UIs* <sup>2</sup> sofisticadas. Además, permite a los desarrolladores construir componentes que pueden llegar a ser reutilizados, optimizando así la programación. No solo esto, sino que también otorga una velocidad de carga para los proyectos impresionante. React da al desarrollador la simpleza o complejidad que el desarrollador quiera.

## COMPONENTES QUE LA DESTACAN:

- Virtual DOM.

- Enlace de datos unidireccional.

- JSX:

JavaScript XML es un lenguaje de marcado para describir el diseño de la interfaz de una aplicación. Este tiene una sintaxis similar a HTML y se utiliza para desarrollar componentes de React. Uno de los mejores aspectos de React JS es JSX, ya que permite escribir los bloques de construcción de una manera muy sencilla para los desarrolladores.

- Componentes:

Una aplicación basada en React se compone de numerosos componentes, cada uno de los cuales tiene su funcionalidad codificada en JS. Como resultado, los programadores pueden enviar datos a través de la aplicación sin afectar al DOM.

- UI explicativa:

Esto permite que el código de React sea más comprensible y que la corrección de errores sea más sencilla. No solo para aplicaciones en línea, sino también para aplicaciones móviles. React JS es el marco ideal para crear interfaces de usuario dinámicas e interactivas.

## COMPARACIÓN:

Ahora que ya sabemos un poco de React y VUE toca compararlos y ver cual nos es más conveniente y cual deberíamos usar.

<b><i>VUE.JS</i></b>	<b><i>REACT</i></b>
Utiliza SFC <sup>3</sup> para crear componentes.	Utiliza JSX como formato.
Para desarrollo de aplicaciones web.	Desarrollo de aplicaciones web con el uso de <i>React Native</i> <sup>4</sup> .
<i>VueX</i> .	<i>Redux</i> .
Amplia variedad de características complejas.	<i>Soporta el uso de librerías de gestión.</i>

## CUANDO USAR CADA UNA:

La decisión de usar React o Vue depende de las necesidades específicas del proyecto.

React es una buena opción para crear aplicaciones complejas a gran escala, en particular aquellas que necesitan actualizar y representar datos en tiempo real. También es una buena opción si ya usa otras tecnologías de Facebook, como React Native para crear aplicaciones móviles. Es flexible y se puede utilizar con una amplia variedad de lenguajes de programación y tecnologías, lo que lo convierte en una opción versátil para muchos tipos diferentes de proyectos.

Por otro lado, Vue es una buena opción para crear aplicaciones más simples y de menor escala o para integrarse en proyectos existentes. Su simplicidad y flexibilidad lo hacen fácil de aprender y usar, y también tiene una curva de aprendizaje más baja en comparación con React. También es una buena opción si desea crear rápidamente un prototipo de una idea o crear una prueba de concepto.

En resumen, React sería una buena opción para crear aplicaciones versátiles, complejas y de mayor escala, mientras que Vue es adecuado para proyectos más simples, fáciles de aprender, integrar y crear una prueba de concepto.

Por último, es importante evaluar los requisitos específicos de nuestro proyecto y considerar factores como el tamaño y la complejidad del proyecto, las habilidades de nuestro equipo de desarrollo y las tecnologías que ya estamos utilizando.

*DOM Virtual<sup>1</sup>: Un DOM virtual es una representación ligera de JavaScript del modelo de objeto de documento que se utiliza en marcos web declarativos como React, Vue.js y Elm. Actualizar el DOM virtual es comparativamente más rápido que actualizar el DOM real.*

*UI<sup>2</sup>: User Interface.*

*SFC<sup>3</sup>: Single File Components.*

*React Native<sup>4</sup>: Cross-platform applications.*

## Explicación del Código

Para explicar las diferencias prácticas entre ambos lenguajes vamos a ver un ejemplo sencillo dónde se pueden ver las diferencias de manera clara. Hemos utilizado el código desarrollado en clase de la lista de la compra en Vue.js y hemos implementado el código necesario para realizarlo en React. Para ambos clientes hemos usado el mismo servidor.

### Vue.js

Archivo Adjunto: *./lista.html*

Nuestra aplicación cliente se encuentra encapsulada dentro del html. Para ello, usamos `Vue.createApp()` creando así una aplicación sin montarla directamente en un DOM y poder configurarla antes de que se monte (`app.mount('#app')`).

Una vez creada la aplicación, podemos usar la función `app.component()` para definir los componentes que formarán la aplicación, así como el comportamiento y las propiedades de los mismos.

En nuestro caso la aplicación está formada por 3 componentes 'lista', 'ítem' y 'newItem'.

El componente lista al crearse, crea una instancia que realiza una solicitud a la url indicada `ClienteAPI("http://localhost:3000/items")` y almacena la información que devuelve el servidor en la propiedad 'items'. Este componente está formado por 3 métodos:

- `CambiarEstado(id)`: Dado un id busca el elemento de ítems correspondiente a ese id y cambia el valor de la propiedad 'comprado'. Realiza un llamada a la Api para actualizar el estado del ítem cambiado y finalmente actualiza los datos del listado items.
- `AddItem(nom)`: Dado un nombre (el escrito en el formulario) realiza una solicitud a la Api para agregar ese nuevo componente al listado de ítems, finalmente actualiza los datos del listado items.
- `deleteItem(id)`: Dado un id busca el elemento correspondiente y realiza una petición a la Api para eliminar el elemento, finalmente actualiza los datos del listado items.

También define una propiedad 'template' que define la disposición del componente lista.

```
template: `
  <div>
    <h1>Lista de la compra</h1>
    <ul>
      <newItem @insertar="addItem"/>
      <item v-for="item in items"
        v-bind:nombre="item.nombre"
        v-bind:comprado="item.comprado"
        v-bind:id="item.id"
        @toggle="cambiarEstado"
        @borrar="deleteItem"/>
    </ul>
  </div>`
```

Formado por un título, una lista y dos componentes hijos 'ítem' y 'newItem' pasando los datos necesarios a cada uno de los subcomponentes.

El evento emitido por el componente 'newItem' está vinculado a `@insertar` y

su valor se pasa como argumento al método `AddItem(nom)`.

Los eventos emitidos por el componente 'ítem' están vinculados con `@toggle` y `@borrar` del mismo modo su valor se pasa como argumento a los métodos `CambiarEstado(id)` `deleteItem(id)` respectivamente.

Por otro lado el componente 'ítem' formado por tres propiedades id, nombre y comprado. Los valores de esos atributos se pasan desde el componente padre.

Además de un 'template' que contiene un elemento 'li' en su interior tenemos el nombre del ítem, el botón de borrado que emite un evento 'borrar' con id del ítem.

Finalmente, en función del valor de comprado el nombre del elemento se mostrará tachado o no y a su vez si se pulsa se emite un evento 'toggle' al que se le pasa el id también.

```
template:
  <li>
    <a v-bind:class="comprado ? 'tachado' : '' " @:click="$emit('toggle', id)"> {{nombre}}</a>
    <button @:click="$emit('borrar', id)">Borrar</button>
  </li>
```

Y por último el componente 'newItem', este componente está formado únicamente por una 'template' con un elemento de entrada, con 'v-model' se enlaza el valor de entrada con el dato del componente 'nombre'. Cuando se hace click en el botón insertar se emite un evento 'insertar' al que se le pasa como argumento el valor de nombre.

```
template:
  <input v-model="nombre" placeholder="Nuevo ítem"><button @:click="$emit('insertar', nombre)">Insertar</button></input>
```

El componente permite crear un nuevo ítem y añadirlo al listado de ítems a través del componente padre.

## React

Archivo Adjunto: *./LISTA-COMPRA-REACT-MAIN/cliente*

Siguiendo los conceptos de la práctica en Vue hemos implementado la versión correspondiente en React.

Para que la aplicación se ejecute correctamente tenemos los archivos App.js e index.js, el primero es una función que devuelve el componente lista (el cual explicaremos más adelante), el segundo contiene el punto de entrada para una aplicación de React, con los módulos necesarios para su correcto funcionamiento como son React, ReactDOM, y App (archivo anterior).

Por otro lado, tenemos la clase clienteAPI que es la encargada de realizar las consultas al servidor a partir de la url indicada en el constructor. Los métodos de la clase permiten hacer peticiones asíncronas para obtener los datos (`getItems()`), actualizar el valor del atributo comprado (`toggleItem(id, valor)`), agregar un ítem (`addItem(nom)`) y eliminar un ítem (`delItem(id)`).

El archivo principal y que vamos a explicar en mayor profundidad es Lista.jsx ya que es dónde vamos a poder ver las diferencias más claramente. Tenemos un componente Lista que muestra una lista de ítems y permite realizar acciones sobre la lista (agregar, eliminar o marcar como comprado/no comprado)

`const [items, setItems] = useState([])` establece el estado inicial del componente con un arreglo vacío y una función para actualizar el estado. Es una forma de declarar una variable de estado en React, utilizando el hook `useState`, que permite tener un estado local en el componente. El estado es el valor de 'items' y la función `setItems` es utilizada para actualizar el valor del estado.

Al hacer una actualización mediante la función `setItems`, React detecta el cambio de estado y actualiza automáticamente la interfaz.

`useEffect(() => {}, [])` es otro hook de React que permite ejecutar código cada vez que el componente se monta o se actualiza después de que se renderiza el componente.

En nuestro caso, se ejecuta la función asíncrona `fetchData()` que hace una llamada a la Api para obtener los elementos de la lista. Una vez que se obtienen los datos, se actualiza el estado 'items' con los datos obtenidos mediante `setItems(datos)`.

`NewItem = ({ addItem }) => {...}` es un componente de función de React que se encarga de recibir una función `addItem` y de mostrar un input de texto y un botón para agregar un nuevo elemento a la lista. Se establece el estado inicial del componente con una cadena vacía y una función para actualizar el estado a partir del nombre del valor del input.



En el cuerpo del componente se encuentra un input de texto que tiene un valor asociado a nombre y un evento `onChange={e => setNombre(e.target.value)}` que utiliza la función para actualizar el estado "nombre" cada vez que el usuario escribe algo en el input. El botón tiene un evento `onClick={() => addItem(nombre)}` que llama a la función `addItem` que se recibe como argumento, y le pasa el valor actual del nombre.

Por otro lado tenemos las funciones que se encargan de manejar los eventos como los explicados recientemente.

- `handleToggle = async (id)`: Se activa al hacer click en un elemento de la lista y se encarga de buscar el elemento en el estado 'items' mediante una función `find` y una comparación de ids. Luego, se actualiza el valor de la propiedad 'comprado'. Se hace una petición PATCH al servidor utilizando la función `toggleItem(id, value)` y se vuelve a hacer una petición `getItems()` para obtener los datos actualizados y se actualiza el estado 'items' con los datos obtenidos.
- `handleAdd = async (nom)`: Se activa al hacer clic en el botón 'Insertar' y recibe el nombre del nuevo elemento como argumento. Utiliza la función `addItem(nom)` para hacer una petición POST al servidor y agregar el nuevo ítem. Igual que la función anterior actualiza items.
- `handleDelete = async (id)`: Se activa al hacer clic en el botón 'Borrar' y recibe el id del ítem a eliminar. Se hace una petición DELETE al servidor utilizando la función `delItem(id)` para eliminar el elemento del servidor. Igual que la función anterior actualiza items.

Y por último el cuerpo del componente que renderiza la interfaz de usuario. Devuelve una estructura de elementos HTML que contiene un título, una lista y un componente `NewItem` (llama a `addItem={handleAdd}` para agregar un nuevo ítem a la lista).

La lista es generada utilizando la función `map` para recorrer `items` y generar un elemento por cada ítem de la lista. Cada elemento contiene el nombre del ítem, un botón 'Borrar' y un evento `onClick` para llamar a la función `handleToggle(item.id)` que se encarga de marcar el ítem como comprado/no comprado.

Finalmente, hemos añadido las funcionalidades de registro, inicio sesión y logout. Otra forma de comparar ambos lenguajes ya que en las prácticas hemos desarrollado estos módulos en un cliente en Vue.js

Hemos añadido los métodos necesarios en `./ClienteApi.js`

```
loginRegistro(username, password)
```

y la referencia en `./App.js`

```
<RegistroLogin />
```

Al cambiar el componente que se muestra y ya no ser la lista al ejecutar la aplicación iremos a `./RegistroLogin.jsx` y no a `./Lista.jsx`. En ese nuevo archivo reside todo lo necesario para que estas nuevas acciones funcionen correctamente.

Ya hemos explicado el significado de la sintaxis anteriormente, nuestro componente posee dos atributos `const [username, setUsername] = useState("")` y `const [password, setPassword] = useState("")` establece el estado inicial de los componentes con una cadena vacío y una función para actualizar el valor. También está compuesto por las siguientes funciones agrupadas por funcionalidad.

### Login

`handleLogin = async (username, password)` esta función manda una petición al servidor con el valor del 'username' y del 'password' para iniciar sesión y muestra una alerta, si no se han producido errores muestra el mensaje Ok y si se han producido errores mostrará el mensaje de error correspondiente. Una vez se acepta la alerta iremos al listado, `document.getElementById('lista').style.display = 'block'`, explicado antes.



The image shows a login form on a light yellow background. At the top, the word "Login" is written in a large, bold, teal font. Below it, there are two input fields. The first is labeled "Username" and has a placeholder text "Escriba el nombre del usuari". The second is labeled "Password" and has a placeholder text "Escriba la contraseña". At the bottom of the form, there are two buttons: "Login" and "Ir a Registro".

### Registro

`handleRegistro = async (username, password)` esta función manda una petición al servidor con el valor del 'username' y del 'password' para registrar un nuevo usuario y muestra una alerta, si no se han producido errores muestra el mensaje Ok y si se han producido errores mostrará el mensaje de error correspondiente.

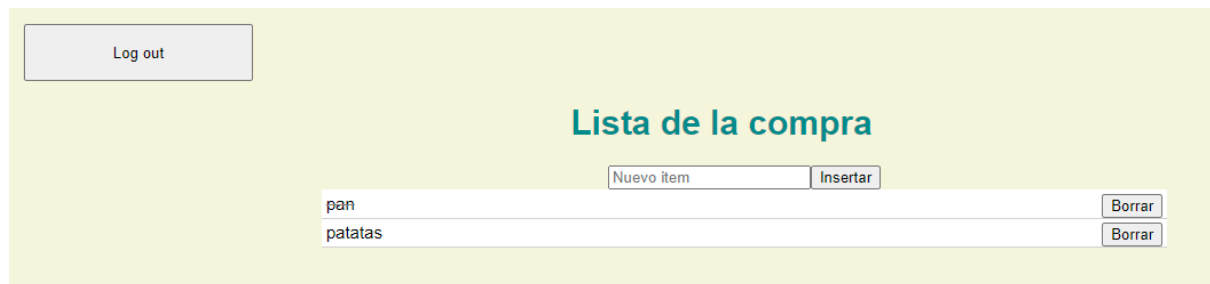


The image shows a registration form on a light yellow background. At the top, the word "Registro" is written in a large, bold, teal font. Below it, there are two input fields. The first is labeled "Username" and has a placeholder text "Escriba el nombre del usuari". The second is labeled "Password" and has a placeholder text "Escriba la contraseña". At the bottom of the form, there are two buttons: "Registrarse" and "Ir a Login".

## Logout

Se ha implementado un botón para cerrar sesión y después se añade el listado.

```
<div id="lista" style={{display: 'none'}}>
  <button onClick={() => volverAlLogin()} style={{height: '50px', width: '200px',
marginLeft: '30px', marginTop: '30px'}}> Log out </button>
  <Lista></Lista>
</div>
```



The screenshot shows a web application interface with a light yellow background. In the top left corner, there is a button labeled "Log out". Below it, centered on the page, is the title "Lista de la compra" in a teal color. Under the title, there is a form for adding items to the list. It includes a text input field labeled "Nuevo ítem", an "Insertar" button, and a table with two rows of items: "pan" and "patatas", each with a "Borrar" button next to it.

Para cambiar el contenido que se muestra al usuario utilizamos las funciones auxiliares que actúan directamente sobre los componentes visuales `cambiarVista = ()` y `volverAlLogin = ()`

Estas funciones las usamos en los botones para cambiar entre Registro y Login y para realizar las funciones `handleLogin` y `handleRegistro` ambos div están compuesto por un pequeño formulario con los mismos datos. '

```
onClick={() => handleLogin(username, password)}
onClick={() => handleRegistro(username, password)}
onClick={() => cambiarVista () }
```

## Diferencias generales

La principal diferencia aparente es que el archivo con el que se encuentra ejecutándose Vue es simplemente un HTML mientras que en React hemos trabajado montando una aplicación basada en archivos jsx.

Por otro lado, React maneja estados de una forma diferente a lo que hemos visto a lo largo del curso, lo realiza mediante la función predefinida **useState**, se puede ver su uso a la hora de actualizar la constante Items, me ha resultado curioso buscar en internet una comparativa visual de useState en React vs Vue.

```
import { useState } from 'react'

function Example() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}

<script>
import { ref } from 'vue'

export default {
  setup () {
    return {
      count: ref(0)
    }
  }
}
</script>

<template>
<p>You clicked {{count}} times</p>
<button @click="count += 1">
  Click me
</button>
</template>
```

Otra cosa que cabe mencionar de Vue es el cómo declaramos los componentes.

### En Vue:

```
app.component('item',{
  props: ["nombre", "comprado", "id"],
  template: `
    <li>
      <a v-bind:class="comprado ? 'tachado' : ' ' "
@:click="$emit('toggle', id)"> {{nombre}}
    </a>
      <button @:click="$emit('borrar', id)">Borrar</button>
    </li>
```

## En React:

```
const Item = (nombre, comprado, id) => {
  return (
    <div>
      <li>
        <a className={comprado ? 'tachado' : ''}>{nombre}</a>
        <button onClick={() => handleDelete(id)}>Borrar</button>
      </li>
    </div>
  );
}
```

Cabe considerar también que en React se hace uso de **useEffect**, este nos permite ejecutar un trozo de código según el momento en el que se encuentre el ciclo de vida de nuestro componente, el equivalente en Vue sería añadir un flag a modo de escucha, mientras que en React queda de la siguiente manera:

```
useEffect(() => {
  async function fetchData() {
    const datos = await cliente.getItems();
    setItems(datos);
  }
  fetchData();
}, []);
```