CORFO
BecasCapital Humano

adalid
aspasia
LA FORMACIÓN DE TU FUTURO

MÓDULO 4

DESARROLLO DE INTERFACES INTERACTIVAS CON REACT

Manual del Módulo 4: Desarrollo de Interfaces Interactivas con React

Introducción

Este módulo aborda los principios fundamentales y avanzados de **ReactJS**, una biblioteca de JavaScript enfocada en la construcción de **interfaces de usuario dinámicas** y **componentizadas**. A lo largo del módulo, los estudiantes aprenderán a desarrollar aplicaciones interactivas orientadas a componentes utilizando React, explorando sus características clave como el DOM Virtual, la renderización declarativa y el flujo de datos unidireccional. Además, se cubren temas avanzados como el manejo del ciclo de vida de los componentes, hooks, la integración con APIs, la optimización del rendimiento, y el uso de herramientas complementarias como React Developer Tools. Al finalizar el módulo, los estudiantes estarán capacitados para desarrollar **aplicaciones escalables** y **modulares** en React, aprovechando las mejores prácticas y los conceptos avanzados de la biblioteca.

1. Introducción a ReactJs

1.1 ¿Qué es ReactJs y cuál es su origen?

ReactJS es una librería de JavaScript desarrollada por Facebook en 2013, con el objetivo de simplificar el desarrollo de interfaces de usuario. Su creación responde a la necesidad de manejar grandes cantidades de datos dinámicos en interfaces interactivas. React está diseñado para ser modular, eficiente y flexible, permitiendo a los desarrolladores crear componentes reutilizables que gestionan su propio estado.

1.2 ¿Por qué usar ReactJs?

React se ha convertido en una de las librerías más populares para el desarrollo de interfaces por varias razones:

- **Eficiencia**: Utiliza un DOM Virtual que minimiza las operaciones costosas en el DOM real, mejorando el rendimiento.

- **Reutilización de componentes**: Permite crear piezas modulares de código que se pueden reutilizar en toda la aplicación.
- **Comunidad y ecosistema**: React tiene una gran comunidad que proporciona librerías, herramientas y soporte, como Redux para la gestión del estado y Next.js para el rendering del lado del servidor.

1.3 Acerca de las Aplicaciones SPA

SPA (Single Page Application) es un tipo de aplicación web donde todo el contenido se carga en una sola página. React es una herramienta poderosa para crear SPAs, ya que maneja de forma eficiente el cambio de vistas sin recargar la página completa. Esto mejora la experiencia de usuario, haciéndola más rápida y fluida.

1.4 Renderización de Elementos del DOM

En React, el contenido se renderiza utilizando el **DOM Virtual**, una representación en memoria del DOM real que permite que React haga cambios de forma más eficiente.

Ejemplo de renderizado en React:

```
import React from 'react';
import ReactDOM from 'react-dom';

const elemento = <h1>¡Hola, mundo!</h1>;

ReactDOM.render(elemento, document.getElementById('root'));
```

En este ejemplo, ReactDOM actualiza solo el elemento `root` del DOM real cuando hay cambios en el componente `elemento`.

1.5 ReactJs y React Native

React y **React Native** comparten el mismo núcleo de conceptos. React Native permite utilizar React para desarrollar aplicaciones móviles nativas. La diferencia clave es que React Native utiliza componentes nativos para renderizar las interfaces, en lugar del DOM de un navegador.

1.6 ¿Es ReactJs un Framework?

Aunque muchos piensan que React es un framework, en realidad es una **librería** enfocada exclusivamente en la **vista** (V en el patrón MVC). No incluye características comunes en frameworks como el enrutamiento o la gestión del estado global, las cuales deben ser añadidas mediante librerías externas como **React Router** o **Redux**.

1.7 ReactJs versus Otros Frameworks Basados en JavaScript

React se compara frecuentemente con otros frameworks como **Angular** y **Vue**. A diferencia de Angular, que es un framework completo, React es más flexible y permite elegir cómo integrar otras herramientas en la arquitectura de la aplicación. Vue es similar a React, pero ofrece una curva de aprendizaje más sencilla, mientras que React se prefiere en proyectos grandes por su extensibilidad.

1.8 Descripción y uso de ReactJs en aplicaciones web

ReactJS es utilizado ampliamente en el desarrollo de aplicaciones web modernas debido a su capacidad para construir interfaces de usuario dinámicas y altamente responsivas. Su uso principal es en **Single Page Applications (SPA)**, donde React maneja la actualización de vistas sin recargar la página completa, lo que mejora la experiencia del usuario. Además, React facilita la construcción de interfaces basadas en componentes reutilizables, lo que reduce la redundancia y mejora la mantenibilidad del código.

1.9 ¿Qué NO es ReactJs?

ReactJS **no es un framework completo**. A diferencia de otros frameworks como Angular o Vue, React se centra exclusivamente en la capa de presentación (vista). Esto significa que React no incluye funcionalidades como la gestión del estado global o el enrutamiento por defecto, las cuales deben ser añadidas mediante librerías externas como **Redux** o **React**

Router. Tampoco es una solución para todas las necesidades del desarrollo web, ya que su enfoque principal es el desarrollo de interfaces de usuario.

2. Características de ReactJs

2.1 Librería Basada en JavaScript

React es una **librería declarativa** que permite construir interfaces de usuario basadas en componentes. Estos componentes se representan utilizando **JavaScript y JSX** (una extensión de JavaScript que permite escribir HTML dentro de JS).

2.2 Lenguaje Declarativo

El enfoque **declarativo** en React significa que los desarrolladores definen cómo debería verse la UI en un momento dado, en lugar de detallar cómo llegar a ese estado. React se encarga de actualizar automáticamente la vista cuando los datos cambian.

2.3 Basado en Componentes

En React, la **componentización** es clave. Cada pieza de la interfaz es un componente independiente que puede tener su propio estado y lógica. Los componentes pueden anidarse y reutilizarse.

Ejemplo de componente en React:

```
function Saludo(props) {
    return <h1>Hola, {props.nombre}</h1>;
}

ReactDOM.render(<Saludo nombre="Carlos" />,
document.getElementById('root'));
```

En este ejemplo, el componente `Saludo` recibe una `prop` (propiedad) llamada `nombre`, lo que permite que el componente sea reutilizable con distintos valores.

2.4 Manejo del DOM Virtual

El **DOM Virtual** de React permite que los cambios en la interfaz se realicen de manera eficiente. En lugar de actualizar el DOM real directamente, React mantiene una representación en memoria del DOM y solo aplica las diferencias entre el DOM virtual y el real, minimizando el trabajo necesario.

2.5 JSX: JavaScript Syntax Extension

JSX es una sintaxis que combina JavaScript y HTML para crear componentes más legibles. Aunque no es obligatorio, JSX simplifica el desarrollo en React.

Ejemplo de JSX:

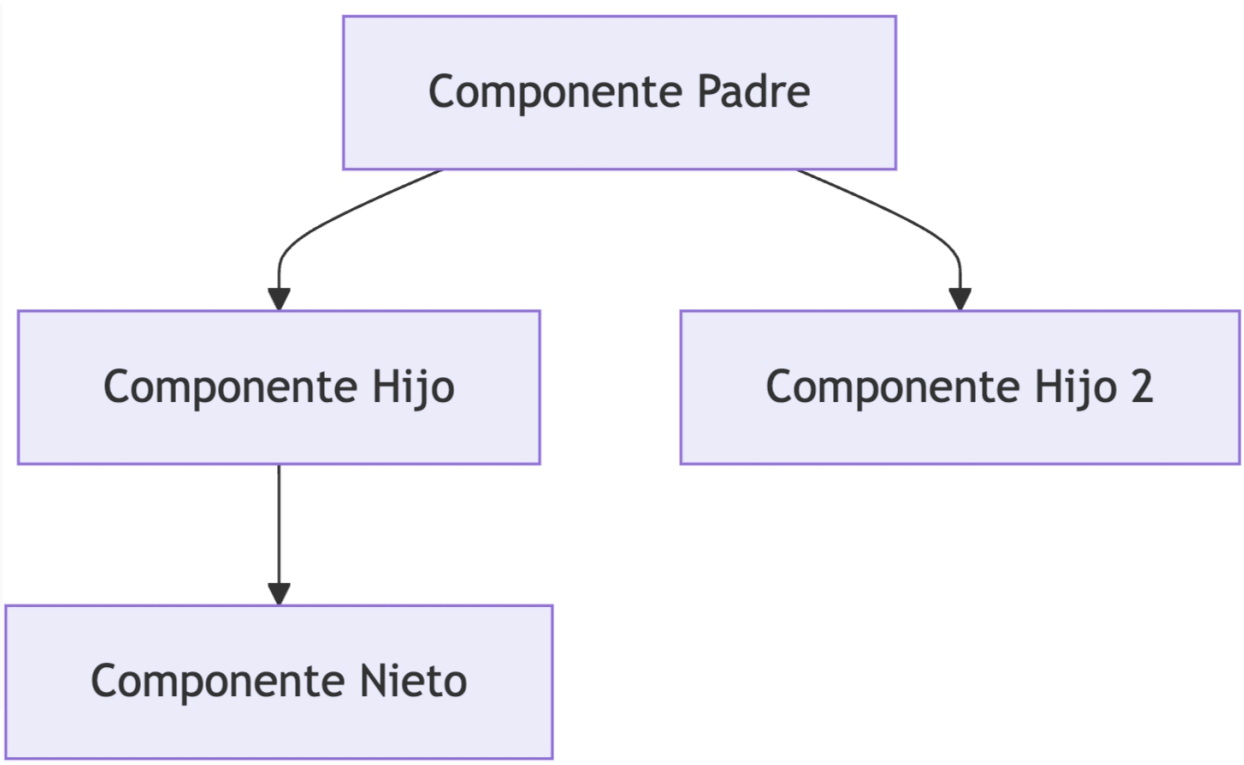
```
const elemento = <h1>¡Hola, mundo!</h1>;
```

Internamente, JSX es convertido a código JavaScript puro mediante herramientas como Babel. Sin JSX, el código se vería así:

```
const elemento = React.createElement('h1', null, '¡Hola, mundo!');
```

2.6 Flujo de Datos Unidireccional

React maneja el flujo de datos de manera **unidireccional**, lo que significa que los datos siempre fluyen desde los componentes padres hacia los componentes hijos a través de las **props**. Esto asegura que el estado de la aplicación sea más predecible.



2.7 Isomorfismo (renderización en cliente y servidor)

El **isomorfismo** en React se refiere a la capacidad de renderizar componentes tanto en el cliente como en el servidor. Esto permite que las aplicaciones puedan aprovechar las ventajas del SEO, ya que el contenido se puede generar en el servidor y luego ser interactivo en el cliente. Un ejemplo de esta práctica es con **Next.js**, un framework basado en React que facilita el **renderizado del lado del servidor (SSR)** y la **generación estática**.

2.8 Sintaxis JSX

JSX es una extensión de la sintaxis de JavaScript que permite escribir HTML dentro de código JavaScript. Aunque no es obligatorio, JSX simplifica el desarrollo al hacer el código más legible y expresivo para los desarrolladores.

2.9 ReactJs sin JSX

Aunque JSX es popular, React también puede ser utilizado sin él. En su lugar, se puede usar la función **React.createElement()** para crear componentes. Esto es útil en algunos entornos donde no se desea o no es posible utilizar JSX.

3. Entornos de Ejecución

3.1 Prerrequisitos para la Utilización de ReactJs

Para comenzar a trabajar con React, se necesitan algunos prerrequisitos:

- Conocimiento de **JavaScript ES6**.
- Familiaridad con conceptos de **componentes** y **estado**.
- Herramientas como **Node.js** y **npm** para instalar paquetes y dependencias.

3.2 Versiones de JavaScript Compatibles con ReactJs

React es compatible con **JavaScript ES5** y versiones más recientes. Sin embargo, se recomienda utilizar las características de ES6 y ES7, como `let`, `const`, `arrow functions`, `destructuring`, y **promesas**.

3.3 Ejecución desde un CDN

Para proyectos pequeños o de prueba, React puede incluirse directamente desde un **CDN** (Content Delivery Network).

Ejemplo de uso de React desde un CDN:

```
<script src="https://unpkg.com/react@17/umd/react.development.js"
crossorigin></script>
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"
crossorigin></script>
```

3.4 Ejecución desde un Ambiente Local

Para proyectos más grandes, se recomienda instalar React en el entorno local utilizando **npm** o **yarn**. Esto permite el uso de herramientas como **Webpack** y **Babel** para compilar el código.

Comando de instalación con npm:

```
npm install react react-dom
```

Esto ha ido cambiando poco a poco, actualmente se recomienda usar **vite** entre otras opciones para iniciar el desarrollo de un proyecto usando **React.**

```
npm create vite@latest nombre-del-proyecto -- --template react
```

Este comando generará un proyecto base de React usando la plantilla por defecto de vite, sien embargo existe otro comando en el cual podrás controlar paso a paso toda su instalación y las opciones que esta ofrece:

```
npm create vite@latest
```

3.5 Librerías Usadas por ReactJs

- **Webpack**: Un empaquetador de módulos que se usa comúnmente para compilar archivos JavaScript y activos como imágenes y hojas de estilo.
- **Babel**: Un transpilador que convierte código ES6+ a versiones de JavaScript compatibles con navegadores más antiguos.
- **Next.js**: Un framework que permite realizar **renderizado del lado del servidor** con React.
- **Redux**: Una librería para la **gestión del estado** global en aplicaciones React.

4. Elementos Fundamentales de ReactJS

4.1 Introducción a JSX

JSX es una extensión de la sintaxis de JavaScript que permite mezclar HTML con JavaScript en una sintaxis fácil de entender.

4.2 Pensando en ReactJs

React se centra en la creación de **componentes reutilizables** que gestionan su propio estado y luego se ensamblan para formar una interfaz más compleja.

4.3 Componentes en ReactJs

Un **componente** en React es una función o clase que devuelve una parte de la interfaz. Estos componentes pueden tener **props** y **estado**, y se pueden anidar para crear interfaces complejas.

Ejemplo de Componente Funcional:

```
function Boton(props) {
    return <button>{props.texto}</button>;
}
```

4.4 Composición versus Herencia

En React, se prefiere la **composición** de componentes en lugar de la **herencia** para reutilizar código. La composición permite ensamblar componentes más pequeños en estructuras más complejas.

Ejemplo de Composición:

```
function
```

```
 Cuadro(props) {
    return <div>{props.children}</div>;
}
```

4.5 Paso de Datos con Props

Las **props** permiten pasar datos de los componentes padres a los componentes hijos.

Ejemplo de Uso de Props:

```
function Titulo(props) {
    return <h1>{props.texto}</h1>;
}

ReactDOM.render(<Titulo texto="Bienvenido a React" />,
document.getElementById('root'));
```

4.6 Listas y Keys

En React, cuando renderizas una lista de elementos, es fundamental proporcionar una **key** única para cada elemento. Las **keys** ayudan a React a identificar qué elementos han cambiado, agregado o eliminado, lo que optimiza el proceso de renderizado.

Ejemplo de renderizado de lista con keys:

```
const numeros = [1, 2, 3, 4, 5];

const listaNumeros = numeros.map((numero) =>
    <li key={numero.toString()}>
        {numero}
    </li>
);

ReactDOM.render(<ul>{listaNumeros}</ul>, document.getElementById('root'));
```

Las keys deben ser únicas entre los elementos hermanos, y ayudan a mejorar el rendimiento al identificar cambios precisos en la lista.

4.7 Formularios

En React, los formularios funcionan de manera diferente a los formularios tradicionales de HTML. React controla el valor de los campos de formulario a través del **estado**, lo que se conoce como **componentes controlados**.

Ejemplo de formulario controlado:

```
class Formulario extends React.Component {

    constructor(props) {
        super(props);

        this.state = { valor: '' };
        this.handleChange = this.handleChange.bind(this);
        this.handleSubmit = this.handleSubmit.bind(this);
    }

    handleChange(event) {
        this.setState({ valor: event.target.value });
    }

    handleSubmit(event) {
        alert('Un nombre fue enviado: ' + this.state.valor);
        event.preventDefault();
    }

    render() {
        return (
            <form onSubmit={this.handleSubmit}>
                <label>
                    Nombre:
```

```
                    <input type="text" value={this.state.valor}
onChange={this.handleChange} />
                </label>
                <input type="submit" value="Enviar" />
            </form>
        );
    }
}

ReactDOM.render(<Formulario />, document.getElementById('root'));
```

Aquí, el valor del input es controlado por el estado del componente, y se actualiza en tiempo real conforme el usuario escribe.

4.8 El Ciclo de Vida de ReactJS

El **ciclo de vida de un componente** en React define las fases por las que pasa un componente desde su creación hasta su destrucción. Estas fases son:

1. **Montaje**: Cuando el componente se inserta en el DOM. Se utiliza el método `componentDidMount`.
2. **Actualización**: Cuando se actualizan las props o el estado. Se utilizan métodos como `componentDidUpdate`.
3. **Desmontaje**: Cuando el componente se elimina del DOM. Se utiliza el método `componentWillUnmount`.

4.9 Desplegando Datos en la Interfaz (Renderización)

La **renderización** en React se refiere al proceso de mostrar elementos de interfaz en el DOM. El método `render()` en los componentes de clase o la función de retorno en componentes funcionales es donde se define qué debe aparecer en la UI.

Ejemplo básico de renderización:

```
function Mensaje(props) {
    return <h1>Hola, {props.nombre}!</h1>;
}

ReactDOM.render(<Mensaje nombre="María" />,
document.getElementById('root'));
```

Aquí, React actualiza automáticamente la interfaz cuando cambian los datos que se muestran.

4.10 Uso de la Extensión del Navegador "React Developer Tools"

La extensión **React Developer Tools** es una herramienta poderosa para depurar aplicaciones React. Permite inspeccionar la jerarquía de componentes, monitorear el estado y las props, y visualizar el DOM Virtual.

Pasos para usar React Developer Tools:

1. Instalar la extensión desde la tienda de extensiones del navegador.
2. Abrir la pestaña "React" en las herramientas de desarrollador.
3. Inspeccionar los componentes y sus estados en tiempo real.

4.11 Hojas de Estilo en ReactJS

React permite múltiples formas de manejar los estilos CSS en las aplicaciones:

1. **CSS en archivos separados**: La forma más tradicional, donde los estilos se importan en los componentes.
2. **Estilos en línea**: React permite aplicar estilos en línea mediante objetos de JavaScript.
3. **Styled-components**: Librerías como `styled-components` permiten escribir CSS directamente dentro de los componentes de React.

Ejemplo de estilos en línea:

```
const estilo = {
    color: 'blue',
    fontSize: '20px'
};

function Titulo() {
    return <h1 style={estilo}>Título con estilo en línea</h1>;
}
```

4.12 Introducción a Hooks

Los **Hooks** son una adición reciente a React (a partir de la versión 16.8) que permiten usar estado y otras características de React en componentes funcionales.

- **useState**: Para manejar el estado en un componente funcional.
- **useEffect**: Para manejar efectos secundarios como llamadas a APIs o suscripción a eventos.

Ejemplo de uso de `useState` y `useEffect`:

```
import React, { useState, useEffect } from 'react';

function Contador() {

    const [cuenta, setCuenta] = useState(0);

    useEffect(() => {
        document.title = `Has hecho clic ${cuenta} veces`;
    });

    return (
        <div>
            <p>Has hecho clic {cuenta} veces</p>
            <button onClick={() => setCuenta(cuenta + 1)}>
                Haz clic aquí
```

```
            </button>
        </div>
    );
}
```

4.13 Introducción a Uso de APIs

React facilita el consumo de **APIs** utilizando métodos asincrónicos como **fetch** o librerías como **Axios**.

Ejemplo de consumo de una API con fetch:

```
import React, { useState, useEffect } from 'react';

function DatosAPI() {

    const [datos, setDatos] = useState([]);

    useEffect(() => {
        fetch('https://jsonplaceholder.typicode.com/posts')
            .then(response => response.json())
            .then(data => setDatos(data));
    }, []);

    return (
        <ul>
            {datos.map(item => (
                <li key={item.id}>{item.title}</li>
            ))}
        </ul>
    );
}

export default DatosAPI;
```

5. JSX, la Simplificación de ReactJS

5.1 ¿Por qué usar JSX?

JSX es una extensión de JavaScript que permite escribir HTML dentro de archivos JS. Simplifica la creación de interfaces de usuario y mejora la legibilidad del código.

Ventajas de usar JSX:

- Sintaxis clara y legible.
- Permite expresar la UI de forma más natural y cercana a HTML.
- Compatible con todas las características de JavaScript.

5.2 Insertando Expresiones en JSX

En JSX, puedes insertar cualquier expresión JavaScript dentro de llaves `{}`.

Ejemplo:

```
const nombre = "Carlos";

const elemento = <h1>Hola, {nombre}!</h1>;
```

5.3 Notación y Palabras Reservadas en JSX

JSX usa convenciones ligeramente diferentes a HTML. Por ejemplo, en lugar de `class`, se utiliza `className`.

5.4 JSX También es una Expresión

JSX es una **expresión** y se puede asignar a variables, retornar desde funciones e incluso pasarse como argumentos a funciones.

Ejemplo:

```
function crearElemento() {
    return <h1>Elemento creado</h1>;
}
```

5.5 Especificando Atributos con JSX

Puedes pasar atributos a los elementos JSX como lo harías en HTML, pero con la sintaxis JavaScript.

Ejemplo:

```
const elemento = <img src="logo.png" alt="Logo" />;
```

5.6 Previniendo Ataques de Inyección con JSX

React **escapa** automáticamente cualquier valor embebido dentro de JSX para prevenir ataques de inyección XSS (Cross-Site Scripting).

5.7 Representación de Objetos en JSX

Puedes insertar objetos JavaScript directamente en JSX usando llaves `{}`.

Ejemplo:

```
const estilo = {
    backgroundColor: 'blue',
    color: 'white'
};

const elemento = <div style={estilo}>Texto con estilo en línea</div>;
```

5.8 El Rol de Babel en la Conversión de JSX

JSX no es entendido directamente por los navegadores. **Babel** transcompila el JSX a JavaScript puro para que pueda ser ejecutado en cualquier entorno.

Ejemplo de JSX compilado por Babel:

```
const elemento = <h1>Hola, mundo!</h1>;

// Se convierte en:

const elemento = React.createElement('h1', null, 'Hola, mundo!');
```

5.9 Comentarios en JSX

Para añadir comentarios dentro de JSX, se deben utilizar llaves `{}`.

Ejemplo:

```
const elemento = (
  <div>

    {/* Esto es un comentario en JSX */}
    <h1>Hola, mundo!</h1>
  </div>

);
```

6. Implementar Componentes Reutilizables

La **componentización** es uno de los pilares fundamentales de React. Un componente reutilizable es una unidad independiente y modular de la UI que se puede usar en diferentes partes de una aplicación sin duplicar código. Esto aumenta la mantenibilidad y escalabilidad del proyecto.

6.1 Describir los Aspectos Fundamentales y Beneficios de la Componentización

Los componentes reutilizables son clave para organizar una aplicación de manera modular. Los beneficios incluyen:

- **Reutilización**: Un componente se puede utilizar varias veces con diferentes datos o configuraciones.
- **Mantenibilidad**: El código está más organizado y centralizado, lo que facilita su mantenimiento.
- **Consistencia**: Usar el mismo componente en varias partes de la aplicación asegura coherencia en la interfaz de usuario.

Ejemplo de componente reutilizable:

```
function Boton(props) {
    return <button>{props.texto}</button>;
}

function Aplicacion() {
    return (
        <div>
            <Boton texto="Aceptar" />
            <Boton texto="Cancelar" />
        </div>
    );
}
```

En este ejemplo, el componente `Boton` se reutiliza con distintos valores para `props.texto`.

6.2 Reconocer los Elementos y Sintaxis Requerida para la Especificación de un Componente

Los componentes en React pueden ser **funcionales** o **de clase**:

- **Funcionales**: Son funciones de JavaScript que retornan JSX.
- **De clase**: Son clases que extienden de `React.Component` y tienen un método `render()`.

Ejemplo de Componente Funcional:

```
function Titulo(props) {
    return <h1>{props.texto}</h1>;
}
```

Ejemplo de Componente de Clase:

```
class Titulo extends React.Component {

    render() {
        return <h1>{this.props.texto}</h1>;
    }
}
```

6.3 Utilizar Componentes Previamente Construidos Usando sus Propiedades para su Personalización

En React, los **props** permiten personalizar componentes al pasarles datos desde los componentes padres.

Ejemplo de personalización usando props:

```
function Boton(props) {
    return <button style={{color: props.color}}>{props.texto}</button>;
```

```
}

function Aplicacion() {
    return (
        <div>
            <Boton texto="Aceptar" color="green" />
            <Boton texto="Cancelar" color="red" />
        </div>
    );
}
```

Aquí, los botones cambian de color dependiendo del valor de `props.color`.

7. Manejo del DOM

7.1 Elementos DOM en ReactJS

React crea su propio modelo en memoria del DOM real, conocido como el **DOM Virtual**, que optimiza el proceso de actualización de la interfaz de usuario. Los elementos React son representaciones ligeras de los elementos DOM reales.

7.2 Acerca de React.Component

En React, los componentes basados en clases deben extender de `React.Component` para manejar estado y ciclo de vida.

Ejemplo de componente de clase:

```
class MiComponente extends React.Component {

    render() {
        return <h1>Hola desde un componente de clase</h1>;
    }
```

```
}
```

7.3 El DOM Virtual en ReactJS

El **DOM Virtual** es un concepto clave en React. Cuando se actualiza el estado o las props de un componente, React actualiza el DOM Virtual primero, comparando la nueva representación con la anterior. Solo las diferencias se aplican al DOM real, lo que hace que las actualizaciones sean más rápidas y eficientes.

7.3.1 ¿Qué es el Shadow DOM?

El **Shadow DOM** es una tecnología diferente al DOM Virtual y se utiliza principalmente en los **Web Components**. El Shadow DOM encapsula el árbol DOM para un componente, evitando que los estilos o scripts externos lo afecten.

7.3.2 ¿Qué es React Fiber?

React Fiber es la nueva arquitectura de React introducida a partir de la versión 16, que permite priorizar tareas como el renderizado y el manejo de la interfaz, mejorando la experiencia del usuario en aplicaciones grandes y complejas.

7.4 Uso de Referencias

Las **referencias** permiten acceder directamente a elementos del DOM en React. Esto puede ser útil para manipular elementos del DOM directamente, algo que no es recomendable hacer frecuentemente, pero puede ser necesario en algunos casos.

7.4.1 ¿Qué son las Referencias?

Una referencia es una manera de almacenar una referencia a un nodo del DOM en un componente React. Esto se logra mediante `React.createRef()`.

7.4.2 Creación de Referencias

Las referencias se crean y se asocian a un componente mediante `ref`.

Ejemplo de referencia:

```
class MiComponente extends React.Component {

    constructor(props) {
        super(props);
        this.miReferencia = React.createRef();
    }

    componentDidMount() {
        this.miReferencia.current.focus();
    }

    render() {
        return <input ref={this.miReferencia} />;
    }
}
```

Aquí, la referencia creada con `React.createRef()` se usa para enfocar un input cuando el componente se monta.

7.5 El Paquete React-DOM

El paquete **React-DOM** proporciona métodos para interactuar con el DOM real. React-DOM incluye métodos como `render()` para montar componentes en el DOM.

7.5.1 React DOM en el Cliente

En aplicaciones cliente, `ReactDOM.render()` se usa para renderizar componentes en el navegador.

Ejemplo:

```
ReactDOM.render(<App />, document.getElementById('root'));
```

7.5.2 React DOM en el Servidor

React también soporta la **renderización del lado del servidor** con `ReactDOMServer`, útil para mejorar el rendimiento de las aplicaciones y SEO.

7.6 Diferencias en Atributos HTML que Funcionan Diferente en ReactJS

React tiene algunas diferencias clave en el manejo de atributos HTML:

- `class` en HTML se convierte en `className`.
- `for` en etiquetas `<label>` se convierte en `htmlFor`.
- Eventos como `onChange`, `onClick` se manejan de forma camelCase.

Ejemplo:

```
<input className="input-clase" htmlFor="inputId"
onChange={this.handleChange} />
```

7.7 Uso de referencias

7.7.1 ¿Cuándo usar referencias?

Las referencias son útiles cuando necesitas interactuar directamente con un elemento del DOM o un componente. Esto puede ser necesario en casos donde las bibliotecas de terceros requieren un nodo DOM o cuando necesitas gestionar el enfoque, seleccionar texto o realizar animaciones directamente en el DOM.

7.7.2 Accediendo a las referencias

Las referencias se crean utilizando **React.createRef()** y se acceden a través de la propiedad **current**. Cuando el componente se monta, la referencia apunta al nodo del DOM.

```
class MiComponente extends React.Component {

    constructor(props) {
        super(props);
        this.miRef = React.createRef();
    }

    componentDidMount() {
        this.miRef.current.focus();
    }

    render() {
        return <input ref={this.miRef} />;
    }
}
```

7.7.3 Agregando referencias al DOM

Una referencia se asocia a un elemento DOM a través del atributo **ref**. Este enfoque permite acceder directamente al DOM cuando es necesario, pero se debe usar con precaución para no romper el modelo declarativo de React.

7.6.4 Referencias mediante Callback

También se pueden crear referencias mediante funciones de callback. Este método permite un control más preciso sobre cuándo la referencia se asocia al elemento DOM.

```
class MiComponente extends React.Component {

    setRef = (elemento) => {
        this.miElemento = elemento;
    };

    render() {
        return <input ref={this.setRef} />;
    }
}
```

8. El paquete React-DOM

8.1 Soporte en navegadores

React-DOM es el paquete responsable de conectar React con el navegador. El soporte para React-DOM incluye todos los navegadores modernos, y React se asegura de ofrecer una experiencia coherente a través de ellos. Es importante señalar que React también es compatible con versiones anteriores de navegadores mediante polyfills y transpiladores como Babel.

8.2 Utilidades para pruebas

React proporciona una serie de utilidades para pruebas, entre ellas, el **renderizador de prueba**. Este renderizador permite simular la estructura de componentes sin necesidad de un navegador real, lo que facilita la creación de tests unitarios.

8.3 Renderizador de prueba

El **renderizador de prueba** en React es una herramienta útil para generar salidas de componentes React y compararlas con las esperadas en un entorno de prueba.

8.4 Requerimientos del entorno de JS

React es compatible con las últimas versiones de **JavaScript (ES6+)**, por lo que se recomienda utilizar versiones modernas del lenguaje para aprovechar todas sus características, como **arrow functions**, **destructuring**, y **async/await**.

9. Elementos avanzados de ReactJS

9.1 Manejo de Socket.io

React se puede integrar con **Socket.IO** para habilitar comunicación en tiempo real entre el cliente y el servidor, lo que resulta útil en aplicaciones como chats o notificaciones.

9.2 División de código (Code Splitting)

React facilita la división del código utilizando **React.lazy()** y **Suspense**. Esto permite cargar componentes de forma asíncrona, mejorando el rendimiento de la aplicación.

```
const ComponenteCargado = React.lazy(() => import('./ComponenteCargado'));

function App() {
    return (
        <React.Suspense fallback={<div>Cargando...</div>}>
            <ComponenteCargado />
        </React.Suspense>
    );
}
```

9.3 Fragmentos

Los **Fragmentos** en React permiten agrupar elementos JSX sin añadir nodos adicionales al DOM.

```
return (
  <React.Fragment>
    <h1>Título</h1>
    <p>Descripción</p>
  </React.Fragment>
);
```

9.4 Transitions

Las **transiciones** permiten gestionar cambios en el estado de la interfaz de usuario de manera fluida. Librerías como **React Transition Group** son populares para este propósito.

9.5 Componentes de orden superior

Un **Componente de Orden Superior (HOC)** es una función que toma un componente como argumento y devuelve un nuevo componente con funcionalidades adicionales.

9.6 Portales

Portales permiten renderizar elementos fuera del nodo padre del componente actual. Esto es útil para casos como **modales** que necesitan ser renderizados fuera de la jerarquía principal.

```
ReactDOM.createPortal(
  <Modal />,
  document.getElementById('modal-root')
);
```

Material de Referencia

Libros:

- **"Fullstack React: The Complete Guide to ReactJS and Friends"** de Anthony Accomazzo, Nathaniel Murray, y Ari Lerner
  Un libro ideal para desarrolladores que quieran aprender React desde los fundamentos hasta la construcción de aplicaciones completas. Cubre conceptos clave como JSX, el DOM Virtual, el ciclo de vida de los componentes, y también se adentra en temas como Redux y la integración con API.

- **"Learning React"** de Alex Banks y Eve Porcello
  Este libro introduce a los desarrolladores al ecosistema React, mostrando cómo crear aplicaciones dinámicas mediante componentes y Hooks. Es excelente para quienes buscan un enfoque práctico y actualizado de React.

- **"React Up & Running"** de Stoyan Stefanov
  Proporciona una visión completa del desarrollo con React, comenzando con lo básico de los componentes y avanzando a características como la manipulación del estado y la gestión de eventos.

- **"React Design Patterns and Best Practices"** de Michele Bertoli
  Este libro es perfecto para quienes ya tienen conocimientos básicos de React y quieren aprender cómo estructurar sus aplicaciones de manera eficiente utilizando patrones de diseño y mejores prácticas.

Enlaces a Recursos Online:

- **React**: La documentación oficial de React es el mejor recurso para mantenerse al día con las últimas características de la librería. Cubre desde conceptos básicos como JSX y componentes, hasta temas avanzados como Hooks y la gestión de estados con Context API.

- **Getting started with React - Learn web development | MDN (mozilla.org)**: La sección de MDN sobre React proporciona una excelente introducción a la librería y ejemplos prácticos de cómo usarla en proyectos de front-end.

- **enaqx/awesome-react: A collection of awesome things regarding React ecosystem (github.com)**: Un repositorio en GitHub que recopila una extensa lista de herramientas, tutoriales, y artículos sobre React. Es un recurso valioso para quienes desean profundizar en el ecosistema React.

- **React Patterns**: Un recurso que detalla varios patrones de diseño comúnmente utilizados en React. Proporciona soluciones a problemas habituales en el desarrollo de aplicaciones React.

Videos Recomendados:

- **React JS Crash Course - YouTube**: Un video introductorio a React donde se cubren los conceptos básicos como componentes, props, estado, y Hooks, todo ello explicado con ejemplos prácticos.

- **React Hooks useState Tutorial - YouTube**: Este video profundiza en el uso de Hooks en React, mostrando cómo usar `useState`, `useEffect`, y otros Hooks para construir componentes funcionales.

- **Redux For Beginners | React Redux Tutorial - YouTube:** Un tutorial detallado sobre cómo gestionar el estado en aplicaciones React utilizando Redux, una de las librerías más populares para este propósito.

- **Learn useContext In 13 Minutes - YouTube**: Un video que cubre cómo usar el Context API en React para manejar el estado global sin necesidad de usar Redux, ideal para aplicaciones más pequeñas.

CORFO
BecasCapital Humano

adalid
aspasia
LA FORMACIÓN DE TU FUTURO

MÓDULO 4

DESARROLLO DE INTERFACES INTERACTIVAS CON REACT