Richard Sebastian Guaitarilla Erazo

Jhon Fredy Rodriguez Portillo

Julian David Rosales Portilla

Universidad Mariana

Facultad de Ingeniería

Programa Ingeniería de Sistemas

San Juan de Pasto

2024

¿Qué es un componente en React y cómo se crea uno?

En React, un componente es una pieza reutilizable de interfaz de usuario que puede contener tanto lógica como estructura de presentación. Pueden ser componentes funcionales (funciones de JavaScript) o componentes de clase (clases ES6 que extienden React.Component). Para crear un componente, puedes definir una función o una clase que devuelva JSX, que es una sintaxis similar a HTML pero que se puede utilizar en JavaScript.

Para crearlo es así:

```
import React, { Component } from 'react';
import ReactDOM from 'react-dom';

// Componente de función
function MiComponenteFuncional(props) {
  return <div>{props.mensaje}</div>;
}

// Componente de clase
class MiComponenteClase extends Component {
  render() {
    return <div>{this.props.mensaje}</div>;
  }
}

// Renderizar ambos componentes
ReactDOM.render(
  <div>
    <MiComponenteFuncional mensaje="Hola, desde el componente funcional" />
    <MiComponenteClase mensaje="Hola, desde el componente de clase" />
  </div>,
  document.getElementById('root')
);

```

¿Cuál es la diferencia entre ReactDOM.render() y ReactDOM.hydrate()?

La diferencia principal entre ReactDOM.render() y ReactDOM.hydrate() radica en el contexto en el que se utilizan y cómo interactúan con el DOM existente:

1. **ReactDOM.render():**
    a. Se utiliza cuando la aplicación se renderiza completamente en el cliente (navegador).

b. Reemplaza completamente el contenido del contenedor especificado en el DOM con el nuevo árbol de elementos virtual generado por React.
c. Es útil cuando la aplicación se carga inicialmente en el cliente y no hay contenido preexistente en el contenedor de renderizado.

2. ReactDOM.hydrate():

a. Se utiliza en el contexto de aplicaciones de renderizado en el servidor (SSR), donde la aplicación se renderiza inicialmente en el servidor y luego se envía al cliente.
b. Interactúa con el HTML preexistente en el contenedor de renderizado en el cliente.
c. No reemplaza el contenido del contenedor, sino que "hidrata" el DOM preexistente con la lógica de React, lo que significa que solo agrega interactividad a los elementos del DOM que coinciden con los elementos del árbol de elementos virtual de React.
d. Es útil para mejorar el tiempo de carga inicial y la accesibilidad de la aplicación al utilizar SSR, ya que aprovecha el HTML pregenerado del servidor y solo agrega la lógica interactiva necesaria en el cliente.

¿Por qué es importante usar keys en las listas de elementos en React?

Usar keys en las listas de elementos en React es importante por varias razones:

- Identificación de elementos: Las keys ayudan a React a identificar qué elementos han cambiado, se han agregado o se han eliminado de una lista. Cuando React recorre una lista y encuentra una key, la utiliza como un identificador único para ese elemento. Esto mejora el rendimiento y la eficiencia en la actualización del DOM al manipular listas.
- Optimización de rendimiento: Al usar keys únicas para cada elemento en una lista, React puede realizar actualizaciones más eficientes del DOM. Sin keys, React necesita recorrer todos los elementos en la lista para determinar cuáles han cambiado. Con keys, React puede comparar las keys antiguas y nuevas para identificar los elementos que han cambiado, agregado o eliminado, lo que reduce el tiempo de procesamiento y mejora el rendimiento de la aplicación.
- Mantenimiento del estado del componente: Las keys ayudan a mantener el estado de los componentes hijos cuando se actualiza una lista. Si los elementos en una lista no tienen keys únicas, React puede eliminar y recrear los componentes hijos al reordenar la lista, lo que puede llevar a la pérdida del estado y a un comportamiento inesperado. Con keys adecuadas, React puede asociar correctamente el estado de los

componentes hijos con sus elementos correspondientes en la lista, lo que garantiza un comportamiento más predecible y consistente.

- Renderizado eficiente en múltiples dispositivos: Al renderizar listas en dispositivos con capacidades de pantalla limitadas, como dispositivos móviles, el uso de keys puede mejorar significativamente el rendimiento al minimizar la cantidad de trabajo necesario para actualizar la interfaz de usuario. Esto es especialmente importante en aplicaciones con grandes conjuntos de datos o en aplicaciones con actualizaciones frecuentes de la interfaz de usuario.

¿Qué es la prop children y cómo se usa en React?

La prop children en React es una propiedad especial que permite pasar elementos hijos directamente a un componente padre. Es una prop predefinida que contiene cualquier contenido que se coloque entre las etiquetas de apertura y cierre de un componente en JSX.

Por ejemplo, en el siguiente componente PadreMotos, cualquier cosa colocada entre las etiquetas <PadreMotos> y </PadreMotos> se pasará automáticamente como la prop 'children':

Se usa de esta manera:

```
import React from 'react';

function PadreMotos(props) {
  return (
    <div>
      {props.children}
    </div>
  );
}

function App() {
  return (
    <PadreMotos>
      <h1>¡Bienvenido a nuestra tienda de motos!</h1>
      <p>Estas son algunas de las motos disponibles:</p>
      <ul>
        <li>Moto deportiva</li>
        <li>Moto de turismo</li>
        <li>Moto de aventura</li>
      </ul>
    </PadreMotos>
  );
}

export default App;

```

¿Cómo se pasa información entre componentes en React?

Así pasamos información en los React cumpliendo estos pasos;

1. **Props (Propiedades):**
   - Los props son la forma más común de pasar datos de un componente padre a un componente hijo en React.
   - Los componentes hijos reciben datos como propiedades y pueden utilizar esa información para renderizar su propio contenido.
   - Las propiedades son de solo lectura y no se pueden modificar directamente por el componente hijo.
   - Los cambios en las props pueden provocar una actualización en el componente hijo.

2. **State (Estado):**
   - El estado es utilizado para manejar datos internos de un componente.
   - Los componentes de clase tienen estado, mientras que los componentes funcionales pueden usar el estado mediante el hook useState.
   - Los cambios en el estado de un componente provocan una re-renderización del componente y de sus hijos.
   - El estado debe ser actualizado utilizando la función setState en los componentes de clase y el setter devuelto por el hook useState en los componentes funcionales.

Contexto:

El contexto es una característica de React que permite pasar datos a través del árbol de componentes sin tener que pasar props manualmente en cada nivel.

Es útil para compartir datos que son considerados "globales" para un árbol de componentes.

El contexto consta de dos partes principales: el Provider que proporciona los datos y el Consumer que consume los datos.

El hook useContext proporciona una forma más sencilla de consumir el contexto en componentes funcionales.

Este es un ejemplo de como se implementa.

```jsx
import React, { useState, createContext, useContext } from 'react';

// Creamos un contexto para los datos de las motos
const MotosContext = createContext();

// Componente principal que contiene el contexto y el estado de las motos
function App() {
  const [motos, setMotos] = useState([
    { id: 1, marca: "Honda", modelo: "CBR1000RR" },
    { id: 2, marca: "Yamaha", modelo: "YZF-R6" },
    { id: 3, marca: "Kawasaki", modelo: "Ninja ZX-6R" }
  ]);

  return (
    <MotosContext.Provider value={motos}>
      <div>
        <h1>Mis motos favoritas</h1>
        <ListaMotos />
      </div>
    </MotosContext.Provider>
  );
}

// Componente que consume el contexto para mostrar una lista de motos
function ListaMotos() {
  const motos = useContext(MotosContext);

  return (
    <div>
      <h2>Lista de motos:</h2>
      <ul>
        {motos.map(moto => (
          <li key={moto.id}>{moto.marca} - {moto.modelo}</li>
        ))}
      </ul>
    </div>
  );
}

export default App;

```