Creación de Componentes Avanzados

by LEANDRO HERNAN ZABALA IGLESIAS

Composición y reutilización de componentes

1.1 Componentes padres e hijos

1 Definición:

Los componentes padres pueden pasar datos a sus hijos mediante props.

Ejemplo básico:

```
function Hijo({ mensaje }) {
  return <p>{mensaje}</p>;
}

function Padre() {
  return (
    <div>
      <h1>Componente Padre</h1>
      <Hijo mensaje="¡Hola desde el componente hijo!" />
    </div>
  );
}
```

1.2 Uso de children para renderizado flexible

1 Definición:

La propiedad especial children permite pasar contenido arbitrario de un componente padre a un componente hijo.

Ejemplo práctico:

```
function Tarjeta({ children }) {
  return <div className="tarjeta">{children}</div>;
}

function App() {
  return (
    <Tarjeta>
      <h1>Contenido de la tarjeta</h1>
      <p>Este es un ejemplo de uso de children.</p>
    </Tarjeta>
  );
}
```

1 Ventaja:

Permite crear componentes más reutilizables y flexibles.

Estilización de componentes

2.1 CSS global vs. módulos CSS

CSS global:

- Los estilos aplican a todo el proyecto.
- **Problema:** Puede causar conflictos de nombres.

Ejemplo:

```
/* styles.css */
.boton {
  background-color: blue;
}
```

```
import "./styles.css";
function Boton() {
  return <button className="boton">Haz clic aquí</button>;
}
```

Módulos CSS:

- Los estilos se encapsulan y se aplican solo al componente importado.
- **Ventaja:** Evita conflictos.

Ejemplo:

```
/* Boton.module.css */
.boton {
  background-color: red;
}
```

```
import styles from "./Boton.module.css";
function Boton() {
  return <button className={styles.boton}>Haz clic aquí</button>;
}
```

2.2 Styled-components

1 Definición:

Una librería que permite escribir CSS directamente dentro de componentes.

Ejemplo básico:

```
import styled from "styled-components";

const Boton = styled.button`
  background-color: green;
  color: white;
  padding: 10px;
  border-radius: 5px;
`;

function App() {
  return <Boton>Haz clic aquí</Boton>;
}
```

2.3 Ejercicio: Crear un componente estilizado

1 Objetivo:

Crear un botón que cambie de color al pasar el mouse, usando **styled-components**:

```
import styled from "styled-components";

const Boton = styled.button`
  background-color: blue;
  color: white;
  padding: 10px;
  border: none;
  border-radius: 5px;
  cursor: pointer;

  &:hover {
    background-color: darkblue;
  }
`;

function App() {
  return <Boton>Haz clic aquí</Boton>;
}
```

Listas y renderizado condicional

3.1 Uso de map() para iterar y mostrar datos

1 Definición:

map() permite iterar sobre arrays y renderizar elementos dinámicamente.

Ejemplo práctico:

```
const tareas = ["Aprender React", "Hacer ejercicio", "Leer un libro"];

function ListaTareas() {
  return (
    <ul>
      {tareas.map((tarea, index) => (
        <li key={index}>{tarea}</li>
      ))}
    </ul>
  );
}
```

3.2 Renderizado condicional (ternarios y &&)

1 Definición:

React permite renderizar elementos condicionalmente usando operadores ternarios (condición ? true : false) o &&.

Ejemplo práctico:

```
function Mensaje({ usuario }) {
  return usuario ? <p>Bienvenido, {usuario}.</p> : <p>Inicia sesión.</p>;
}
```

3.3 Ejercicio: Lista de elementos con botones de acción

1 Objetivo:

Crear una lista dinámica donde:

1. Se puedan agregar y eliminar elementos.
2. Los elementos completados se muestren tachados.

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

function Lista() {
  const [tareas, setTareas] = useState([]);
  const [nuevaTarea, setNuevaTarea] = useState("");

  const agregarTarea = () => {
    if (nuevaTarea.trim()) {
      setTareas([...tareas, { texto: nuevaTarea, completada: false }]);
      setNuevaTarea("");
    }
  };

  const toggleCompletada = (index) => {
    const nuevasTareas = [...tareas];
    nuevasTareas[index].completada = !nuevasTareas[index].completada;
    setTareas(nuevasTareas);
  };

  const eliminarTarea = (index) => {
    setTareas(tareas.filter((_, i) => i !== index));
  };

  return (
    <div>
      <input
        type="text"
        value={nuevaTarea}
        onChange={(e) => setNuevaTarea(e.target.value)}
      />
      <button onClick={agregarTarea}>Agregar</button>
      <ul>
        {tareas.map((tarea, index) => (
          <li
            key={index}
            style={{
              textDecoration: tarea.completada ? "line-through" : "none",
            }}
          >
            {tarea.texto}
            <button onClick={() => toggleCompletada(index)}>Completar</button>
            <button onClick={() => eliminarTarea(index)}>Eliminar</button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Comunicación entre componentes

4.1 Props drilling

1 Definición:

Es el proceso de pasar props a través de múltiples niveles de componentes para que lleguen al componente deseado.

Ejemplo práctico:

```
function Nieto({ mensaje }) {
  return <p>{mensaje}</p>;
}

function Hijo({ mensaje }) {
  return <Nieto mensaje={mensaje} />;
}

function Padre() {
  return <Hijo mensaje="¡Hola desde el componente Padre!" />;
}
```

4.2 Ejercicio: Enviar datos y callbacks entre componentes

1 Objetivo:

Crear un componente padre que administre un contador y pase funciones para incrementar, decrementar o reiniciar el contador a un componente hijo.

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

function ContadorHijo({ contador, incrementar, decrementar, reiniciar }) {
  return (
    <div>
      <p>Contador: {contador}</p>
      <button onClick={incrementar}>Incrementar</button>
      <button onClick={decrementar}>Decrementar</button>
      <button onClick={reiniciar}>Reiniciar</button>
    </div>
  );
}

function ContadorPadre() {
  const [contador, setContador] = useState(0);

  const incrementar = () => setContador(contador + 1);
  const decrementar = () => setContador(contador - 1);
  const reiniciar = () => setContador(0);

  return (
    <ContadorHijo
      contador={contador}
      incrementar={incrementar}
      decrementar={decrementar}
      reiniciar={reiniciar}
    />
  );
}
```