

React: Nivel Intermedio - Componentes y Props

Introducción

En estos apuntes abordaremos los fundamentos esenciales de los **componentes** y **props** en React, dos conceptos clave que te ayudarán a crear interfaces de usuario de manera modular, reutilizable y adaptable. React se basa en el uso de componentes independientes que pueden personalizarse mediante props, facilitando así la creación de aplicaciones web eficientes y dinámicas.

Temas que Cubriremos


1. **Componentes en React:** ¿Qué son y por qué son esenciales?
2. **Props:** Cómo funcionan y cómo facilitan la comunicación entre componentes.
3. **Estilos Dinámicos con Props:** Uso de props para aplicar estilos inline.
4. **Componentes Reutilizables:** Cómo crear y estructurar componentes reutilizables.
5. **Manejo de Eventos:** Uso de eventos como `onClick` y buenas prácticas para gestionar interacciones.

1. Componentes en React

En React, los componentes son las unidades básicas que permiten construir interfaces de usuario de manera modular y reutilizable. Al dividir una aplicación en componentes, se facilita su desarrollo, mantenimiento y escalabilidad. Cada componente puede manejar su propio estado, recibir información externa mediante propiedades (props) y renderizar contenido de manera autónoma.


Tipos de Componentes en React

React permite crear dos tipos principales de componentes:

1. **Componentes Funcionales** : Son simplemente funciones de JavaScript que retornan JSX (JavaScript XML), un lenguaje de marcado que permite escribir HTML dentro de JavaScript. Los componentes funcionales son la opción preferida en React moderno, debido a su simplicidad y al soporte de hooks, como `useState` y `useEffect`.

Ejemplo básico de un componente funcional:

```
function Saludo() {  
  return <h1>¡Hola, Mundo!</h1>;  
}
```

2. **Componentes de Clase** : Utilizados en versiones anteriores de React, los componentes de clase representan componentes más complejos. Al ser clases de JavaScript, requieren métodos como `render()` para devolver el JSX. Hoy en día se utilizan menos, dado que los hooks permiten manejar el estado y el ciclo de vida en componentes funcionales.

Ejemplo de un componente de clase:

```
class Saludo extends React.Component {  
  render() {  
    return <h1>¡Hola, Mundo!</h1>;  
  }  
}
```

Jerarquía de Componentes 🧠

En una aplicación React, los componentes pueden organizarse jerárquicamente. Existen **componentes padre** que pueden contener **componentes hijo**, y estos pueden transmitir datos a través de props. Esta estructura permite que los datos fluyan de forma descendente, es decir, desde el componente principal hacia los componentes más específicos, manteniendo el flujo de datos unidireccional de React.

2. Props en React: Comunicación entre Componentes 🗨️

Las props (abreviatura de “properties”) permiten que los componentes compartan información. Son **parámetros** que se pasan de un componente padre a un componente hijo y que permiten personalizar la salida de los componentes en función de los datos recibidos. Al igual que los argumentos en una función, las props ayudan a que los componentes sean reutilizables y versátiles.

Características de las Props 📋

3. **Inmutabilidad** 🛑: Las props son de solo lectura. Una vez asignadas, no se pueden modificar desde el componente hijo, asegurando que los datos que vienen del componente padre se mantengan consistentes.
4. **Uso en Componentes Funcionales y de Clase** 📖: Tanto los componentes funcionales como los de clase pueden recibir props, aunque su sintaxis es ligeramente diferente.
5. **Transmisión de Datos** 📡: Las props permiten que los componentes hijos accedan a datos o funciones definidas en sus componentes padres.

Ejemplo básico de uso de props:

```
function Saludo(props) {  
  return <h1>¡Hola, {props.nombre}!</h1>;  
}  
  
// Uso del componente:  
<Saludo nombre="Laura" />
```

En este caso, **nombre** es una prop que el componente **Saludo** recibe y muestra dentro del mensaje de saludo.

Tipos de Props: Children y Props Personalizadas 🧐

Además de las props personalizadas como **nombre**, React tiene una prop especial llamada **children**. Esta prop permite que los componentes puedan incluir contenido variable y dinámico entre sus etiquetas de apertura y cierre.

1. **Prop children** 🧐: Se utiliza cuando un componente necesita mostrar contenido que varía en función del contexto o la aplicación.

Ejemplo de uso de children:

```
function Contenedor(props) {  
  return <div className="contenedor">{props.children}</div>;  
}  
  
// Uso del componente con children  
<Contenedor>  
<p>Este es un contenido dentro del contenedor.</p>  
</Contenedor>
```

2. **Props Personalizadas** 🛠️: Podemos definir props personalizadas en cada componente según la información que necesitemos transmitir, como textos, estilos, clases CSS, eventos, etc.

Ejemplo con props personalizadas:

```
function Boton({ texto, color }) {  
  return <button style={{ backgroundColor: color }}>{texto}</button>;  
}  
  
// Uso del componente Boton:  
<Boton texto="Enviar" color="blue" />
```

3. Estilos en Componentes usando Props 🧠

React permite utilizar props para aplicar estilos dinámicos a los componentes. Esta técnica es útil para crear interfaces adaptables sin necesidad de clases CSS adicionales, especialmente en casos donde los estilos deben ajustarse en tiempo de ejecución.

Estilos Inline 🖋️

Los estilos inline en React se definen como objetos JavaScript. Podemos utilizar props para cambiar propiedades como el color, el tamaño de la fuente o el fondo, proporcionando así una forma eficiente de manejar los estilos en componentes reutilizables.

Ejemplo de estilo inline usando props:

```
function BotonEstilizado({ texto, color }) {  
  return (  
    <button style={{ backgroundColor: color, color: 'white', padding: '10px' }}>  
      {texto}  
    </button>  
  );  
}  
  
// Uso del componente  
<BotonEstilizado texto="Enviar" color="green" />
```

4. Creación de Componentes Reutilizables 🔄

Un componente reutilizable es aquel que puede ser utilizado múltiples veces en diferentes partes de la aplicación sin necesidad de duplicar código. La reutilización de componentes se facilita con el uso de

props y children, permitiendo que cada instancia del componente tenga un comportamiento personalizado sin necesidad de crear componentes adicionales.

Ejemplo de un componente reutilizable:

```
function Producto({ nombre, precio, descripcion }) {
  return (
    <div className="producto">
      <h2>{nombre}</h2>
      <p>{descripcion}</p>
      <span>Precio: ${precio}</span>
    </div>
  );
}

// Uso del componente
<Producto nombre="Camiseta" precio="19.99" descripcion="Camiseta de algodón" />
```

5. Manejo de Eventos en React 🕒

El manejo de eventos en React es similar al de JavaScript, pero con una sintaxis propia que simplifica la asociación de eventos a funciones y garantiza la compatibilidad entre navegadores.

onClick y Otros Eventos 🚩

React permite manejar eventos de manera declarativa. Los eventos en React se nombran con camelCase (ej. **onClick**, **onChange**) y se asocian a funciones JavaScript. Uno de los eventos más utilizados es **onClick**, que se activa al hacer clic en un elemento y permite realizar acciones como actualizar el estado, mostrar mensajes o modificar el DOM virtual.

Ejemplo de manejo de evento onClick:

```
function BotonAlerta() {
  const mostrarAlerta = () => {
    console.log("¡Botón clickeado!");
  };

  return (
    <button onClick={mostrarAlerta}>Haz clic aquí</button>
  );
}
```

Mejores Prácticas en el Manejo de Eventos 🔍

3. **Evitar funciones anónimas en onClick** 🕒: Definir la función fuera del JSX mejora la legibilidad.
 4. **Deestructuración de Props** 🕒: En lugar de usar **props.propiedad**, la destructuración mejora la claridad en el uso de props.
 5. **Pasar funciones por props** 📦: Las funciones se pueden pasar como props para crear componentes más flexibles y acoplados a eventos.
-

Con estos apuntes, tienes una base sólida para entender React y trabajar en tus ejercicios. A medida que avances, puedes expandir estos temas con hooks más complejos, manejo del estado global y el uso de librerías como React Router.

¡Espero que te sirvan! 