Introducción a React

¿Qué es React?

React es una biblioteca de JavaScript creada por Facebook en 2013 que se utiliza para construir interfaces de usuario. Aunque es común referirse a React como un "framework", técnicamente es una biblioteca enfocada exclusivamente en la **visualización** de datos y el manejo de la **interfaz de usuario**.

A diferencia de otros frameworks como Angular o Vue, React se centra solo en la "V" (vista) de la arquitectura **MVC** (Model-View-Controller). Esto significa que React es responsable únicamente de cómo se **muestra la interfaz de usuario** y de cómo reacciona ante los cambios en los datos.

Principales características de React:

1. **Componentes**: En React, la interfaz se divide en componentes independientes. Cada componente es como un "bloque de construcción" que se puede reutilizar en diferentes partes de la aplicación. Estos componentes pueden ser tan pequeños como un botón o tan grandes como una página entera.

Ventaja: Esta modularidad facilita la **reutilización** del código y la **mantenibilidad** de la aplicación. Un componente puede tener su propio estado y lógica, lo que permite un desarrollo más organizado.

Ejemplo: Si tienes una página con un encabezado, un menú y un pie de página, cada una de estas partes puede ser un componente React separado. Incluso el botón de "Enviar" en un formulario puede ser un componente.

2. **Virtual DOM**: React utiliza un **DOM virtual** (Virtual DOM), lo que significa que mantiene una copia ligera del **DOM real** en la memoria. Cada vez que cambian los datos, React actualiza primero el DOM virtual en lugar del DOM real. Luego, compara las diferencias (un proceso conocido como "diferencia" o **reconciliación**) y realiza solo las actualizaciones necesarias en el DOM real.

Ventaja: Este enfoque mejora significativamente el rendimiento, ya que reduce la cantidad de operaciones costosas en el DOM real.

3. **Unidireccionalidad de los datos**: React sigue un flujo de datos **unidireccional**, lo que significa que los datos siempre fluyen de un componente padre hacia sus componentes hijos mediante "props"

(propiedades). Este flujo es más predecible y fácil de seguir, especialmente en aplicaciones complejas.

Ventaja: Este modelo hace que sea más fácil depurar y comprender cómo los datos cambian a lo largo de la aplicación.

¿Por qué utilizar React?

React es popular por varias razones:

1. **Facilita la creación de aplicaciones interactivas**: Con React, los desarrolladores pueden crear aplicaciones web interactivas que actualicen y rendericen los componentes de manera eficiente, basándose en el cambio de datos sin recargar la página completa.
2. **Escalabilidad**: A medida que las aplicaciones crecen en tamaño y complejidad, la arquitectura basada en componentes de React hace que sea más fácil **mantener** y **escalar** el código. Los componentes reutilizables también reducen la duplicación de código.
3. **Comunidad y Ecosistema**: React tiene una comunidad de desarrolladores muy grande y activa, lo que significa que hay una enorme cantidad de **bibliotecas** y **herramientas** disponibles para resolver casi cualquier problema o necesidad.
4. **React Native**: React no solo es útil para crear aplicaciones web, sino que también se puede usar para crear aplicaciones móviles nativas mediante **React Native**. Esto permite a los desarrolladores usar el mismo conocimiento y las mismas prácticas para crear tanto aplicaciones web como móviles.

Primera actividad práctica:

Instalación del entorno de desarrollo y creación de un proyecto React:

1. **Instalación de Node.js**: Antes de comenzar con React, es necesario tener instalado **Node.js**, ya que React se ejecuta en un entorno de Node para gestionar los paquetes y dependencias. Puedes verificar si Node.js está instalado abriendo la terminal y ejecutando:

```
node -v
```

Si no está instalado, puedes descargarlo desde nodejs.org.

2. **Crear una nueva aplicación React usando create-react-app**: La forma más sencilla de empezar con React es usando create-react-app, una herramienta oficial que genera la estructura básica del proyecto y configura todas las herramientas necesarias para empezar a trabajar con React de inmediato.

   Para crear una aplicación React, abre una terminal y ejecuta:

```
npx create-react-app my-app
```

   - **npx** es una herramienta de Node.js que ejecuta paquetes directamente sin necesidad de instalarlos globalmente.
   - **create-react-app** es el comando que crea una nueva aplicación React con una estructura ya configurada.

3. **Iniciar el servidor de desarrollo**: Una vez creado el proyecto, navega hasta el directorio del proyecto e inicia el servidor de desarrollo:

```
cd my-app
npm start
```

Este comando ejecutará la aplicación en modo de desarrollo y abrirá automáticamente el navegador para mostrar la aplicación en http://localhost:3000.

Actividad: Verificar que la aplicación React está funcionando en el navegador.

Explicación de la estructura del proyecto (15 minutos)

Una vez creada la aplicación, veremos cómo está organizada la estructura del proyecto.

- **public/**: Contiene el archivo index.html, que es el único archivo HTML en una aplicación React. Aunque React crea interfaces dinámicas, solo se usa un archivo HTML en toda la aplicación.

- **src/**: Es donde reside el código JavaScript y los componentes de React. Los archivos más importantes son:
    - **App.js**: El componente raíz de la aplicación. Todos los demás componentes se incluyen dentro de este archivo.
    - **index.js**: El archivo que se encarga de renderizar el componente raíz (App.js) dentro del archivo index.html.
- **node_modules/**: Contiene todas las dependencias del proyecto instaladas mediante npm.

Primer vistazo a un archivo React (App.js): El archivo App.js es donde empezaremos a construir nuestra aplicación React.

```
import React from 'react';


function App() {
  return (
    <div className="App">
      <h1>¡Hola, React!</h1>
    </div>
  );
}


export default App;
```

En este código:

- Importamos React.
- Definimos un componente funcional llamado App, que retorna una estructura JSX (similar a HTML).
- Exportamos el componente para que pueda ser utilizado en otros archivos.

Actividad práctica:

1. Los estudiantes deben abrir el archivo App.js, modificar el texto del encabezado <h1> y guardar los cambios para ver cómo React actualiza la página automáticamente.

2. Componentes en React (40 minutos)

¿Qué es un Componente en React?

En React, un **componente** es una pieza reutilizable de la interfaz de usuario. Puedes pensar en un componente como una función de JavaScript que devuelve un fragmento de la interfaz de usuario, generalmente usando **JSX** (una extensión de JavaScript que permite escribir código similar a HTML dentro de JavaScript).

Los componentes son el núcleo de React. Todo en React es un componente o una combinación de componentes. Esto permite que las aplicaciones React sean modulares, lo que facilita la reutilización del código y su mantenibilidad.

Tipos de Componentes en React

En React, hay dos tipos principales de componentes:

1. Componentes Funcionales:

Los componentes funcionales son simples funciones de JavaScript que aceptan **props** (propiedades) como argumento y retornan un elemento de React, normalmente en formato JSX. Desde la introducción de **Hooks** en React, los componentes funcionales han ganado popularidad ya que ahora pueden manejar **estado** y **ciclo de vida**, que anteriormente solo podían manejar los **componentes de clase**.

Ventajas de los componentes funcionales:

- Son más simples y fáciles de entender.
- Mejor rendimiento, ya que no manejan todo el ciclo de vida de un componente de clase.
- Facilitan la reutilización de código.

Ejemplo de un componente funcional:

```
import React from 'react';

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

export default Saludo;
```

En este ejemplo:

- El componente Saludo es una función que devuelve un fragmento JSX con un encabezado <h1>.
- **JSX**: El contenido dentro de <h1> es JSX, que permite mezclar HTML y JavaScript.

2. Componentes de Clase:

Antes de la introducción de los hooks, los **componentes de clase** eran la única forma de manejar el estado y el ciclo de vida en React. Aunque en la actualidad los componentes funcionales son más comunes, los componentes de clase siguen siendo importantes y muchos proyectos antiguos los utilizan.

Ejemplo de un componente de clase:

```
import React, { Component } from 'react';

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

```
export default Saludo;
```

En este ejemplo:

- Saludo es un componente de clase que extiende de Component, lo que le permite tener acceso a más funcionalidades.
- El método render() es obligatorio en los componentes de clase y es el que define qué se mostrará en pantalla.

JSX (JavaScript XML)

React utiliza **JSX** para definir la estructura de los componentes. JSX es una extensión de la sintaxis de JavaScript que permite escribir código que parece HTML, pero en realidad es JavaScript. React convierte este código JSX en llamadas a funciones que manipulan el DOM.

Ejemplo de JSX:

```
const elemento = <h1>¡Bienvenidos a React!</h1>;
```

- **Curiosidad**: JSX no es obligatorio en React, pero es muy recomendado porque simplifica la creación de interfaces. En lugar de usar el método createElement de React para crear elementos, JSX nos permite escribir código más legible y cercano al HTML.

Reglas de JSX:

- Todos los elementos JSX deben estar contenidos en un único elemento padre. Por ejemplo, este código lanzaría un error:

```
return (
  <h1>¡Hola!</h1>
  <p>Bienvenidos a la clase de React</p>
);
```

Para evitar el error, debes envolver los elementos en un solo contenedor, como un o un React.Fragment:

```
return (
  <div>
    <h1>¡Hola!</h1>
    <p>Bienvenidos a la clase de React</p>
  </div>
);
```

- En JSX, las **expresiones de JavaScript** se pueden insertar entre llaves {}:

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

Esto inserta dinámicamente el valor de la variable nombre en el contenido del elemento.

- En lugar de usar atributos tradicionales de HTML como class o for, JSX utiliza className y htmlFor:

```
<div className="contenedor">
  <label htmlFor="nombre">Nombre:</label>
  <input id="nombre" type="text" />
</div>
```

Actividad práctica: Crear componentes funcionales básicos (15 minutos)

1. **Componente Header**:
    - Crea un componente funcional llamado Header que retorne un encabezado con un mensaje de bienvenida.
    - Utiliza JSX para incluir un título y un subtítulo.

Ejemplo:

Componente Header

Archivo: src/Header.jsx

1. **Crear un nuevo archivo** en el directorio src/ llamado Header.jsx.
2. Dentro de este archivo, implementa el siguiente código:

```
function Header() {
  return (
    <header>
      <h1>¡Bienvenidos a mi sitio web!</h1>
      <p>Este es el mejor lugar para aprender React</p>
    </header>
  );
}

export default Header;
```

- Aquí estamos creando un componente funcional llamado Header que retorna un encabezado (<header>) con un título (<h1>) y un subtítulo (<p>).
- Lo exportamos para poder utilizarlo en otros archivos.

2. **Componente Footer**:
    - Crea un componente funcional llamado Footer que retorne un pie de página con el año actual. Utiliza JavaScript dentro de JSX para obtener el año dinámicamente.

Ejemplo:

Archivo: src/Footer.jsx

1. **Crear un nuevo archivo** en el directorio src/ llamado Footer.jsx.
2. Implementa el siguiente código:

```
function Footer() {
  const year = new Date().getFullYear();
  return (
    <footer>
      <p>Todos los derechos reservados &copy; {year}</p>
    </footer>
  );
}

export default Footer;
```

- Este componente muestra el año actual dinámicamente dentro del pie de página usando new Date().getFullYear().

3. **Componente App**:

- Modifica el componente App.js para incluir los componentes Header y Footer que acabamos de crear.

Ejemplo:

Archivo: src/App.js

1. Abre el archivo src/App.js, que es el archivo principal generado por create-react-app.
2. Modifica el contenido de este archivo para incluir los componentes Header y Footer que acabamos de crear.

Modificación del archivo App.js:

```
import React from 'react';
import Header from './Header';
import Footer from './Footer';

function App() {
  return (
    <div>
      <Header />
      <main>
        <p>Esta es la sección principal de la página.</p>
      </main>
      <Footer />
    </div>
  );
}
```

```
export default App;
```

Resultado esperado:

- Al ejecutar este código, deberías ver una página con un encabezado en la parte superior, un texto en el cuerpo y un pie de página que muestra el año actual. Esto demuestra cómo React permite la creación de interfaces modulares con componentes reutilizables.

Ventajas de Componentes Funcionales

- **Simplicidad**: Al ser funciones simples, los componentes funcionales son más fáciles de escribir, leer y depurar.
- **Uso de Hooks**: Desde la introducción de los hooks en React (con React 16.8), los componentes funcionales pueden manejar el estado y otras características que antes eran exclusivas de los componentes de clase.
- **Mejor rendimiento**: Los componentes funcionales son más eficientes, ya que no tienen toda la sobrecarga de los componentes de clase.

4. Probar la aplicación

Después de realizar los cambios en los archivos mencionados:

1. Asegúrate de que el servidor de desarrollo esté ejecutándose. Si no está en ejecución, usa el siguiente comando en la terminal:

```
npm start
```

2. Abre tu navegador y visita http://localhost:3000. Deberías ver la siguiente estructura en la página:

- Un encabezado con el texto **¡Bienvenidos a mi sitio web!**
- Un párrafo que dice **Este es el mejor lugar para aprender React**.
- Un contenido principal con el texto **Esta es la sección principal de la página.**

- Un pie de página con el texto **Todos los derechos reservados** y el año actual.

Estructura del Proyecto después de los Cambios

```
my-app/
│
├── public/
│   └── index.html
│
├── src/
│   ├── App.js
│   ├── Header.jsx
│   ├── Footer.jsx
│   └── index.js
│
└── package.json
```

- **App.js**: El componente principal que incluye los componentes Header y Footer.
- **Header.jsx**: Componente funcional que representa el encabezado de la página.
- **Footer.jsx**: Componente funcional que representa el pie de página.

3. Props (Propiedades) en React

Las **props** (abreviatura de "properties") en React permiten que los datos fluyan de un **componente padre** a un **componente hijo**. Las props son **inmutables**, lo que significa que no pueden cambiarse desde dentro del componente que las recibe. Son unidireccionales: los datos siempre fluyen de padre a hijo.

¿Qué son las props?

Las props en React se pasan a los componentes de la misma manera que los atributos de HTML. Por ejemplo, en HTML, puedes escribir:

```
<input type="text" value="Hola" />
```

En React, esto sería similar a:

```
<Componente texto="Hola" />
```

Aquí, texto es una prop que se le pasa al componente Componente.

Cómo usar props en componentes funcionales

Vamos a modificar el proyecto que ya tenemos y agregar props para que los componentes puedan recibir datos dinámicamente.

---

1. Modificar el componente Header para aceptar una prop

Archivo: src/Header.jsx

1. Abre el archivo Header.jsx.
2. Modifícalo para que el encabezado reciba el texto a través de una prop llamada titulo.

```
import React from 'react';

function Header(props) {
  return (
    <header>
      <h1>{props.titulo}</h1>
      <p>Este es el mejor lugar para aprender React</p>
    </header>
  );
}

export default Header;
```

- En este ejemplo, el componente Header recibe una prop llamada titulo. El valor de esta prop se utiliza dentro de {} para mostrar el contenido dinámico.
- Usamos props.titulo para acceder a la prop dentro del componente.

---

2. Modificar el componente Footer para aceptar una prop

Archivo: src/Footer.jsx

1. Abre el archivo Footer.jsx.
2. Modifícalo para que el pie de página reciba un valor dinámico para el año a través de una prop llamada year.

```
import React from 'react';

function Footer(props) {
  return (
```

```
    <footer>
      <p>Todos los derechos reservados &copy; {props.year}</p>
    </footer>
  );
}

export default Footer;
```

- Aquí, props.year es la prop que recibe el valor del año.

---

3. Pasar props desde el componente App

Archivo: src/App.js

1. Abre el archivo App.js.
2. Modifica el código para pasar el texto dinámico como props a Header y Footer.

```
import React from 'react';
import Header from './Header';
import Footer from './Footer';

function App() {
  return (
    <div>
      <Header titulo="¡Bienvenidos a mi aplicación React!" />
      <main>
        <p>Esta es la sección principal de la página.</p>
```

```
      </main>
      <Footer year={new Date().getFullYear()} />
    </div>
  );
}

export default App;
```

- En este caso, estamos pasando una cadena de texto "¡Bienvenidos a mi aplicación React!" como valor de la prop titulo al componente Header.
- Para Footer, estamos pasando el valor dinámico de new Date().getFullYear() como prop year.

Explicación paso a paso

1. **Props en el componente Header**: El texto que se muestra dentro del encabezado (<h1>) ya no está codificado dentro del componente. Ahora se pasa desde el componente App a través de la prop titulo. Esto hace que el componente Header sea más flexible y reutilizable.
2. **Props en el componente Footer**: Similar a Header, Footer ahora recibe el año dinámicamente a través de la prop year.
3. **Componentes reutilizables**: Gracias a las props, ahora podrías reutilizar tanto el componente Header como el Footer en otras partes de tu aplicación simplemente pasando valores diferentes a las props.

---

4. Actividad práctica: Crear un componente Usuario que use props

Vamos a crear un nuevo componente llamado Usuario que acepte dos props: nombre y edad.

Archivo: src/Usuario.jsx

1. Crea un nuevo archivo llamado Usuario.jsx en el directorio src/.

2. Agrega el siguiente código para definir el componente:

```
import React from 'react';

function Usuario(props) {
  return (
    <div>
      <h2>Nombre: {props.nombre}</h2>
      <p>Edad: {props.edad} años</p>
    </div>
  );
}

export default Usuario;
```

- Este componente recibe dos props: nombre y edad, y muestra esta información en una estructura simple.

---

5. Incluir el componente Usuario en App

Archivo: src/App.js

1. Abre nuevamente el archivo App.js.
2. Importa el nuevo componente Usuario y pásale algunos valores como props.

```
import React from 'react';
import Header from './Header';
import Footer from './Footer';
```

```
import Usuario from './Usuario';

function App() {
  return (
    <div>
      <Header titulo="¡Bienvenidos a mi aplicación React!" />
      <main>
        <p>Esta es la sección principal de la página.</p>
        <Usuario nombre="Juan" edad={30} />
      </main>
      <Footer year={new Date().getFullYear()} />
    </div>
  );
}

export default App;
```

- Aquí hemos añadido el componente Usuario dentro de la sección principal (<main>), pasando "Juan" como prop nombre y 30 como prop edad.

Estructura del Proyecto después de los Cambios

```
my-app/
│
├── public/
│   └── index.html
│
├── src/
```

```
│   ├── App.js
│   ├── Header.jsx
│   ├── Footer.jsx
│   ├── Usuario.jsx
│   └── index.js
│
└── package.json
```

Resultado esperado

Al ejecutar el servidor de desarrollo (npm start), deberías ver lo siguiente en la página:

1. El encabezado mostrando el mensaje: **¡Bienvenidos a mi aplicación React!**
2. Un párrafo que dice: **Esta es la sección principal de la página.**
3. El componente Usuario mostrando:
    - **Nombre: Juan**
    - **Edad: 30 años**
4. El pie de página mostrando el año actual de manera dinámica.

4. Estado (State) en React

En React, el **estado** es un objeto que representa el estado interno de un componente. A diferencia de las props, que son inmutables y se pasan desde un componente padre, el estado es **mutable** y se administra dentro del propio componente.

El estado es especialmente útil para manejar datos que pueden cambiar a lo largo del tiempo, como el valor de un campo de entrada, el conteo de clics en un botón o el resultado de una solicitud a una API.

El Hook useState

Con la llegada de los **Hooks** en React (a partir de la versión 16.8), podemos usar el estado en **componentes funcionales** mediante el hook useState.

useState es una función que nos permite declarar una variable de estado en un componente funcional y manejar su valor a lo largo del tiempo. Este hook nos devuelve dos valores:

1. El **valor actual** del estado.
2. Una función que nos permite **actualizar** ese valor.

Sintaxis básica:

```
const [valorDelEstado, setValorDelEstado] =useState(valorInicial);
```

- **valorDelEstado**: Es el valor actual del estado.
- **setValorDelEstado**: Es la función que se usa para actualizar el estado.
- **useState(valorInicial)**: Inicializa el estado con el valor proporcionado.

Ejemplo básico de uso de useState: Un contador

Vamos a crear un componente funcional llamado Contador que utiliza el hook useState para manejar el valor de un contador que se incrementa cada vez que el usuario hace clic en un botón.

Archivo: src/Contador.jsx

1. **Crear un nuevo archivo** en el directorio src/ llamado Contador.jsx.
2. Implementa el siguiente código:

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


function Contador() {
  // Declarar una variable de estado "contador" e inicializarla en
0
  const [contador, setContador] = useState(0);


  return (
    <div>
```

```
      <p>Has hecho clic {contador} veces</p>
      <button onClick={() => setContador(contador + 1)}>Haz
clic</button>
    </div>
  );
}

export default Contador;
```

Explicación del código:

1. **Importar useState**: Importamos useState de React.
2. **Declaración de estado**: Usamos useState(0) para declarar una variable de estado llamada contador, que se inicializa en 0. También obtenemos la función setContador para actualizar su valor.
3. **Renderizado dinámico**: Cada vez que contador cambia, React vuelve a renderizar el componente mostrando el nuevo valor en el <p>.
4. **Manejo de eventos**: El evento onClick en el botón llama a la función setContador, que incrementa el valor de contador en 1.

Incluir el componente Contador en App

Archivo: src/App.js

1. Abre el archivo App.js.
2. Importa el componente Contador y añádelo a la estructura de la aplicación.

```
import React from 'react';
import Header from './Header';
import Footer from './Footer';
import Usuario from './Usuario';
```

```
import Contador from './Contador';

function App() {
  return (
    <div>
      <Header titulo="¡Bienvenidos a mi aplicación React!" />
      <main>
        <p>Esta es la sección principal de la página.</p>
        <Usuario nombre="Juan" edad={30} />
        <Contador />
      </main>
      <Footer year={new Date().getFullYear()} />
    </div>
  );
}

export default App;
```

- Aquí hemos importado el componente Contador y lo hemos añadido dentro de la sección <main> junto a los otros componentes.

Actividad práctica: Crear un componente Toggle que cambie de estado

Vamos a crear un componente llamado Toggle que cambiará entre dos estados: "Encendido" y "Apagado" cada vez que el usuario haga clic en un botón.

Archivo: src/Toggle.jsx

1. **Crear un nuevo archivo** en el directorio src/ llamado Toggle.jsx.
2. Implementa el siguiente código:

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


function Toggle() {
  // Declarar una variable de estado "encendido" e inicializarla
en "false"
  const [encendido, setEncendido] = useState(false);


  return (
    <div>
      <p>El interruptor está {encendido ? 'Encendido' :
'Apagado'}</p>
      <button onClick={() => setEncendido(!encendido)}>
        Cambiar
      </button>
    </div>
  );
}


export default Toggle;
```

Explicación del código:

1. **useState(false)**: Inicializamos el estado encendido como false (apagado).
2. **Renderizado condicional**: En el párrafo, utilizamos una expresión condicional ({encendido ? 'Encendido' : 'Apagado'}) para mostrar el texto adecuado en función del estado actual.
3. **Actualizar el estado**: Cuando el usuario hace clic en el botón, usamos setEncendido(!encendido) para alternar el estado entre true (encendido) y false (apagado).

Incluir el componente Toggle en App

Archivo: src/App.js

1. Abre el archivo App.js.
2. Importa el componente Toggle y añádelo a la estructura de la aplicación.

```
import React from 'react';
import Header from './Header';
import Footer from './Footer';
import Usuario from './Usuario';
import Contador from './Contador';
import Toggle from './Toggle';

function App() {
  return (
    <div>
      <Header titulo="¡Bienvenidos a mi aplicación React!" />
      <main>
        <p>Esta es la sección principal de la página.</p>
        <Usuario nombre="Juan" edad={30} />
        <Contador />
        <Toggle />
      </main>
      <Footer year={new Date().getFullYear()} />
    </div>
  );
}
```

```
export default App;
```

- Ahora hemos añadido el componente Toggle dentro de <main>, justo debajo del contador.

Resultado esperado

Al ejecutar el servidor de desarrollo (npm start), deberías ver lo siguiente en la página:

1. Un contador que incrementa cada vez que haces clic en el botón "Haz clic".
2. Un interruptor que alterna entre "Encendido" y "Apagado" cada vez que haces clic en el botón "Cambiar".

Estructura del Proyecto después de los Cambios

```
my-app/
│
├── public/
│   └── index.html
│
├── src/
│   ├── App.js
│   ├── Header.jsx
│   ├── Footer.jsx
│   ├── Usuario.jsx
│   ├── Contador.jsx
│   ├── Toggle.jsx
│   └── index.js
│
```

```
└── package.json
```

5. Renderizado Condicional en React

El **renderizado condicional** en React nos permite mostrar o no mostrar ciertos elementos de la interfaz en función de condiciones específicas. Esta funcionalidad es crucial para crear interfaces dinámicas, donde los elementos pueden cambiar o aparecer según el estado de la aplicación.

¿Qué es el renderizado condicional?

El renderizado condicional en React se refiere a la capacidad de hacer que los componentes o elementos se muestren solo si se cumple una condición. Es muy similar a cómo se usan las declaraciones if o las expresiones ternarias en JavaScript.

Por ejemplo, en HTML tradicional, podrías ocultar o mostrar un elemento utilizando CSS o manipulando el DOM directamente. En React, puedes hacer esto directamente dentro de la función que retorna el JSX, lo que permite mayor control sobre cómo se renderiza la UI.

Formas de realizar renderizado condicional en React

1. **Usar if en el retorno del JSX**: La manera más básica de hacer renderizado condicional es utilizando una sentencia if.
2. **Operador ternario (condición ? expr1 : expr2)**: El operador ternario es una forma más concisa de realizar un renderizado condicional y es ampliamente utilizado en React debido a su simplicidad.
3. **Operador lógico &&**: Otra forma común es usar el operador &&, que solo renderiza un elemento si la condición es verdadera.

Ejemplos de renderizado condicional

1. Usando if en el JSX

Vamos a crear un componente llamado Mensaje que muestra un mensaje diferente dependiendo de si el usuario está registrado o no.

Archivo: src/Mensaje.jsx

1. **Crear un archivo** llamado Mensaje.jsx en el directorio src/.
2. Implementa el siguiente código:

```
import React from 'react';

function Mensaje(props) {
  const esUsuarioRegistrado = props.esUsuarioRegistrado;

  if (esUsuarioRegistrado) {
    return <h1>Bienvenido de nuevo</h1>;
  } else {
    return <h1>Por favor, regístrate</h1>;
  }
}

export default Mensaje;
```

Explicación del código:

- props.esUsuarioRegistrado: Recibimos esta prop desde el componente padre para determinar si el usuario está registrado.
- Usamos una declaración if para decidir qué mensaje mostrar. Si esUsuarioRegistrado es true, se muestra "Bienvenido de nuevo". Si es false, se muestra "Por favor, regístrate".

---

2. Usando el Operador Ternario

El operador ternario permite hacer un renderizado condicional de manera más compacta. Vamos a modificar el componente Mensaje para utilizar un operador ternario.

Archivo: src/Mensaje.jsx

```
import React from 'react';


function Mensaje(props) {
  const esUsuarioRegistrado = props.esUsuarioRegistrado;


  return (
    <div>
      {esUsuarioRegistrado ? <h1>Bienvenido de nuevo</h1> :
<h1>Por favor, regístrate</h1>}
    </div>
  );
}


export default Mensaje;
```

Explicación del código:

- El operador ternario {esUsuarioRegistrado ? <h1>Bienvenido de nuevo</h1> : <h1>Por favor, regístrate</h1>} simplifica el código comparado con el if, pero hace exactamente lo mismo.
- Si esUsuarioRegistrado es true, se muestra el mensaje "Bienvenido de nuevo". De lo contrario, se muestra "Por favor, regístrate".

---

3. Usando el Operador Lógico &&

El operador && solo renderiza el segundo valor si la condición es verdadera. Esto es útil cuando no necesitas una opción "else".

Archivo: src/MensajeSimple.jsx

1. **Crear un archivo** llamado MensajeSimple.jsx en el directorio src/.
2. Implementa el siguiente código:

```
import React from 'react';

function MensajeSimple(props) {
  const esVisible = props.esVisible;

  return (
    <div>
      {esVisible && <h1>Este mensaje es visible</h1>}
    </div>
  );
}

export default MensajeSimple;
```

Explicación del código:

- En este caso, si esVisible es true, el mensaje "Este mensaje es visible" será renderizado. Si esVisible es false, no se renderizará nada.
- El operador && solo ejecuta la parte derecha de la expresión si la parte izquierda es verdadera.

Actividad práctica: Mostrar un mensaje de login basado en el estado

Vamos a crear un componente llamado LoginControl que maneje el estado de un usuario registrado o no registrado. Dependiendo del estado del usuario, se mostrará un mensaje diferente.

Archivo: src/LoginControl.jsx

1. **Crear un archivo** llamado LoginControl.jsx en el directorio src/.
2. Implementa el siguiente código:

```
import React, { useState } from 'react';
import Mensaje from './Mensaje';


function LoginControl() {
  const [esUsuarioRegistrado, setEsUsuarioRegistrado] =
useState(false);


  return (
    <div>
      <Mensaje esUsuarioRegistrado={esUsuarioRegistrado} />
      <button onClick={() =>
setEsUsuarioRegistrado(!esUsuarioRegistrado)}>
        {esUsuarioRegistrado ? 'Cerrar sesión' : 'Iniciar sesión'}
      </button>
    </div>
  );
}


export default LoginControl;
```

Explicación del código:

1. **useState(false)**: Inicializamos el estado esUsuarioRegistrado como false, lo que significa que el usuario no está registrado al cargar la página.
2. **Renderizado condicional**: Pasamos el valor de esUsuarioRegistrado al componente Mensaje para que muestre el mensaje correcto basado en el estado.
3. **Botón dinámico**: El botón cambia su texto entre "Iniciar sesión" y "Cerrar sesión" dependiendo de si el usuario está registrado o no. Además, cuando el usuario hace clic en el botón, el estado cambia entre true y false, lo que provoca que React vuelva a renderizar el componente con la nueva información.

Incluir el componente LoginControl en App

Archivo: src/App.js

1. Abre el archivo App.js.
2. Importa el componente LoginControl y añádelo a la estructura de la aplicación.

```
import React from 'react';
import Header from './Header';
import Footer from './Footer';
import Usuario from './Usuario';
import Contador from './Contador';
import Toggle from './Toggle';
import LoginControl from './LoginControl';

function App() {
  return (
    <div>
      <Header titulo="¡Bienvenidos a mi aplicación React!" />
```

```
      <main>
        <p>Esta es la sección principal de la página.</p>
        <Usuario nombre="Juan" edad={30} />
        <Contador />
        <Toggle />
        <LoginControl />
      </main>
      <Footer year={new Date().getFullYear()} />
    </div>
  );
}

export default App;
```

Resultado esperado

Al ejecutar el servidor de desarrollo (npm start), deberías ver lo siguiente en la página:

1. Un mensaje que dice "Por favor, regístrate" y un botón "Iniciar sesión".
2. Al hacer clic en "Iniciar sesión", el mensaje cambiará a "Bienvenido de nuevo" y el botón cambiará a "Cerrar sesión".
3. Al hacer clic en "Cerrar sesión", el mensaje volverá a ser "Por favor, regístrate".

Estructura del Proyecto después de los Cambios

```
my-app/
│
├── public/
│   └── index.html
│
```

```
├── src/
│   ├── App.js
│   ├── Header.jsx
│   ├── Footer.jsx
│   ├── Usuario.jsx
│   ├── Contador.jsx
│   ├── Toggle.jsx
│   ├── LoginControl.jsx
│   ├── Mensaje.jsx
│   ├── MensajeSimple.jsx
│   └── index.js
│
└── package.json
```

Actividad Final y Resumen

Objetivo: Crear una aplicación completa que combine los conceptos aprendidos en la clase, incluyendo manejo de estado y renderizado condicional.

Descripción del proyecto final

Vamos a crear una aplicación React que permita a los usuarios registrarse y desloguearse. Dependiendo de si el usuario está registrado o no, la aplicación mostrará un mensaje de bienvenida o de solicitud de registro. Además, incluiremos las funcionalidades de contador y un interruptor que cambie entre encendido y apagado.

Características de la aplicación:

1. **Manejo de estado** para el registro de usuarios, el contador y el interruptor.
2. **Renderizado condicional** para mostrar mensajes diferentes si el usuario está registrado o no.

3. **Componentes reutilizables** y modulares, como Header, Footer, Usuario, Contador, Toggle, y LoginControl.

Estructura de la aplicación

Archivo: src/App.js

El archivo principal App.js integrará todos los componentes que hemos creado a lo largo de la clase.

```
import React from 'react';
import Header from './Header';
import Footer from './Footer';
import Usuario from './Usuario';
import Contador from './Contador';
import Toggle from './Toggle';
import LoginControl from './LoginControl';

function App() {
  return (
    <div>
      <Header titulo="¡Bienvenidos a mi aplicación React!" />
      <main>
        <p>Esta es la sección principal de la página.</p>
        <Usuario nombre="Juan" edad={30} />
        <Contador />
        <Toggle />
        <LoginControl />
      </main>
      <Footer year={new Date().getFullYear()} />
```

```
    </div>
  );
}

export default App;
```

Componentes utilizados:

1. **Header.jsx**: Muestra el título de la página. Recibe la prop titulo para mostrar un mensaje dinámico.
2. **Footer.jsx**: Muestra el pie de página con el año actual. Recibe la prop year para mostrar el año de forma dinámica.
3. **Usuario.jsx**: Muestra el nombre y la edad del usuario. Recibe las props nombre y edad.
4. **Contador.jsx**: Maneja un contador que se incrementa cada vez que el usuario hace clic en un botón. Utiliza el hook useState para manejar el valor del contador.
5. **Toggle.jsx**: Un interruptor que alterna entre "Encendido" y "Apagado". Utiliza el hook useState para manejar el estado del interruptor.
6. **LoginControl.jsx**: Gestiona el inicio de sesión y el cierre de sesión del usuario. Usa renderizado condicional para mostrar mensajes diferentes dependiendo de si el usuario está registrado o no.

Prueba y Ejecución de la Aplicación

1. Asegúrate de que el servidor de desarrollo está ejecutándose. Si no es así, usa el siguiente comando en la terminal:

```
npm start
```

1. Abre tu navegador y visita http://localhost:3000. La aplicación debería cargar con los siguientes elementos:

- Un encabezado con el mensaje **¡Bienvenidos a mi aplicación React!**
- Un componente Usuario que muestra el nombre y la edad.
- Un contador que incrementa al hacer clic en el botón "Haz clic".
- Un interruptor que cambia entre "Encendido" y "Apagado".
- Un botón para iniciar o cerrar sesión, que cambia el mensaje de "Por favor, regístrate" a "Bienvenido de nuevo" dependiendo del estado.

Resumen final de la clase

En esta clase hemos cubierto los siguientes conceptos fundamentales de React:

1. **Componentes en React**:
   - Cómo crear componentes funcionales.
   - Cómo estructurar y modularizar una aplicación usando componentes reutilizables.
2. **JSX**:
   - Uso de JSX para mezclar HTML y JavaScript en componentes React.
3. **Props**:
   - Cómo pasar datos desde un componente padre a un componente hijo usando props.
   - Cómo hacer que los componentes sean dinámicos y reutilizables mediante props.
4. **Estado (State) en React**:
   - Uso del hook useState para manejar datos internos de los componentes.
   - Cómo actualizar el estado en respuesta a eventos del usuario.
5. **Renderizado Condicional**:
   - Uso de if, el operador ternario y el operador && para mostrar o no mostrar elementos en función del estado.

Actividad recomendada para casa

Para reforzar los conceptos aprendidos, se recomienda que los estudiantes practiquen:

1. **Extender la aplicación** para incluir más funcionalidades dinámicas.
    - Por ejemplo, agregar un formulario para que el usuario pueda ingresar su propio nombre y edad, y mostrar esta información en el componente Usuario.
2. **Documentación oficial**: Explorar más sobre los hooks y el ciclo de vida de los componentes en la Documentación oficial de React.