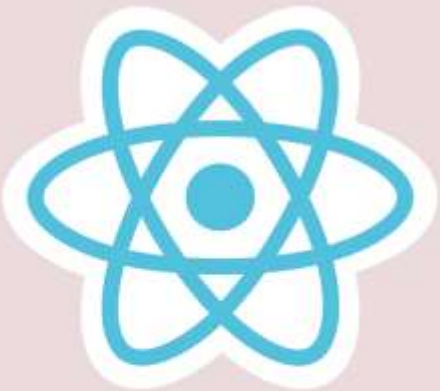
5_1.3

ReactJS

Estado y Ciclo de Vida de los Componentes

En ReactJS, el Estado permite a los componentes gestionar información dinámica que cambia a lo largo del tiempo, provocando actualizaciones automáticas en la interfaz. El Ciclo de Vida del componente abarca fases como montaje, actualización y desmontaje, permitiendo ejecutar acciones en momentos clave de su existencia en el DOM. Estos conceptos son cruciales para el desarrollo eficiente y dinámico de aplicaciones en React.

www.opentowork.es



#01/24@ mihifidem

Tema 2: Estado y Ciclo de Vida de los Componentes

1. **Props**
   1.1. Introducción a las props en React.
   1.2. Pasar datos de un componente padre a un componente hijo mediante props.
   1.3. Acceder a las props dentro del componente hijo.
   1.4. Props como datos inmutables.

2. **Estado en ReactJS:**
   2.1. Introducción al concepto de estado en React.
   2.2. Uso del estado para almacenar y manipular datos en un componente.
   2.3. Inicializar el estado en un componente.
   2.4. Actualizar el estado utilizando el método setState.
   2.5. Renderizar el componente en base a los cambios en el estado.

3. **Ciclo de vida de los componentes de clase:**
   3.1. componentDidMount: método invocado después de que el componente se haya montado en el DOM.
   3.2. componentDidUpdate: método invocado después de que el componente se haya actualizado en elDOM.
   3.3. componentWillUnmount: método invocado antes de que el componente se desmonte y sea eliminadodel DOM.
   3.4. Utilidad de los métodos del ciclo de vida para realizar tareas específicas en diferentes etapas del ciclode vida de un componente.

@mihifidem
If you want to master something, teach it.

Tema 2: Estado y Ciclo de Vida de los Componentes

1. Props

1.1. Introducción a las props en React.

En React, las props (abreviatura de "properties") son utilizadas para pasar datos de un componente padre a un componente hijo. Las props permiten que los componentes se comuniquen entre sí y compartan información de manera estructurada.

Las props son objetos que contienen valores o funciones, y se pasan como atributos a los componentesal momento de su creación. Estos atributos pueden ser cualquier tipo de dato, como cadenas de texto,números, arreglos u objetos.

Cuando un componente recibe props, puede acceder a ellas a través de su argumento de función. Las props son de solo lectura, lo que significa que no se pueden modificar directamente en el componentehijo. Solo el componente padre puede actualizar las props y enviar los nuevos valores.

Ejemplo de uso de props:

```
// Componente padre
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const message = 'Hola desde el componente padre';

  return <ChildComponent message={message} />;
}
```

```
// Componente hijo
import React from 'react';

const ChildComponent = (props) => {
  return <p>{props.message}</p>;
}
```

En este ejemplo, el componente padre ParentComponent pasa la prop message al componente hijo ChildComponent a través del atributo message={message}. Luego, el componente hijo puede acceder ala prop message utilizando props.message y mostrarlo en el navegador.

Las props en React permiten una comunicación eficiente y estructurada entre componentes, lo quefacilita la reutilización de código y la construcción de aplicaciones más modulares.

1.2. Pasar datos de un componente padre a un componente hijomediante props.

En ReactJS, puedes pasar datos de un componente padre a un componente hijo utilizando props (propiedades).Los props son objetos que contienen valores y se utilizan para comunicar datos entre componentes.

Aquí tienes un ejemplo de cómo pasar datos a través de

props:Componente Padre:

```
import React from 'react';
import MiComponenteHijo from './MiComponenteHijo';

const MiComponentePadre = () => {
 const nombre = 'Juan';
 const edad = 25;

 return (
  <div>
   <h1>Componente Padre</h1>
   <MiComponenteHijo nombre={nombre} edad={edad} />
  </div>
 );
};

export default MiComponentePadre;
```

En este ejemplo, tenemos un componente padre llamado MiComponentePadre que contiene dos variables:nombre y edad. Luego, pasamos estos valores como props al componente hijo MiComponenteHijo.

Componente Hijo:

```
import React from 'react';

const MiComponenteHijo = (props) => {
 return (
  <div>
   <h2>Componente Hijo</h2>
   <p>Nombre: {props.nombre}</p>
   <p>Edad: {props.edad}</p>
  </div>
 );
};

export default MiComponenteHijo;
```

En el componente hijo MiComponenteHijo, recibimos los props como argumento en la función y luego los utilizamos dentro del código JSX. En este caso, mostramos los valores de nombre y edad pasados desde elcomponente padre.

Al renderizar el componente padre, el componente hijo recibirá los valores de nombre y edad a través de lasprops y los utilizará para mostrarlos en su propia estructura de elementos JSX.

Este es solo un ejemplo básico de cómo pasar datos a través de props. Puedes pasar cualquier tipo de dato, como números, strings, arrays o incluso funciones, como props a tus componentes hijos. Los props son una forma eficiente y flexible de compartir datos entre componentes y permiten una comunicación unidireccional dearriba hacia abajo en la jerarquía de componentes.

1.3.- Acceder a las props

Recepción y uso de props en el componente hijo:

En el componente hijo, recibimos los props como argumento en la función del componente y luego losutilizamos dentro del código JSX.

```
import React from 'react';

const MiComponenteHijo = (props) => {
  return (
    <div>
      <h2>Componente Hijo</h2>
      <p>Nombre: {props.nombre}</p>
      <p>Edad: {props.edad}</p>
    </div>
  );
};

export default MiComponenteHijo;
```

En este ejemplo, el componente hijo MiComponenteHijo recibe los props (nombre y edad) como argumento enla función y los utiliza dentro del código JSX para mostrar los valores correspondientes.

Los props se pueden utilizar en cualquier parte del componente hijo donde necesites acceder a los datos pasados desde el componente padre. Puedes utilizar los props en expresiones de JavaScript, condicionales,bucles y más para personalizar la lógica y el contenido de tu componente hijo.

En resumen, los props son una forma de pasar datos de un componente padre a un componente hijo en ReactJS.Se pasan como atributos en la sintaxis JSX desde el componente padre y se reciben y utilizan en el componente hijo. Los props permiten una comunicación unidireccional de arriba hacia abajo en la jerarquía de componentes y son una forma eficiente de compartir datos entre componentes en una aplicación de React.

Tema 2: Estado y Ciclo de Vida de los Componentes

Propiedades por defecto y validación de props.

En ReactJS, puedes establecer propiedades por defecto y realizar validaciones en los props que se pasan a un componente. Esto te permite definir valores predeterminados para los props y garantizar que los props cumplancon ciertas condiciones antes de ser utilizados en el componente.

Aquí tienes información sobre cómo establecer propiedades por defecto y realizar validaciones de props enReactJS:

Propiedades por defecto (Default Props):
Puedes definir propiedades por defecto para un componente utilizando la propiedad defaultProps. Estaspropiedades se utilizarán cuando no se pase un valor correspondiente para un prop determinado.

```
import React from 'react';

const MiComponente = (props) => {
  return (
    <div>
      <h1>Título: {props.titulo}</h1>
    </div>
  );
};

MiComponente.defaultProps = {
  titulo: 'Título por defecto'
};

export default MiComponente;
```

En este ejemplo, si no se pasa un valor para el prop titulo al utilizar el componente MiComponente, se utilizaráel valor por defecto 'Título por defecto'.

Tema 2: Estado y Ciclo de Vida de los Componentes

Validación de props (Prop Types):
Puedes realizar validaciones en los props utilizando la biblioteca prop-types. Esta biblioteca te permite definirreglas y tipos de datos para los props de un componente. Si los props no cumplen con las reglas definidas, se mostrará un mensaje de advertencia en la consola.

Para utilizar prop-types, primero debes instalarlo como una dependencia:

```
npm install prop-types
```

```
import React from 'react';
import PropTypes from 'prop-types';
const MiComponente = (props) => {
return (
   <div>
     <h1>Título: {props.titulo}</h1>
   </div>
 );
};
MiComponente.propTypes = {
 titulo: PropTypes.string.isRequired
};

export default MiComponente;
```

En este ejemplo, estamos utilizando PropTypes para definir que el prop titulo debe ser de tipo string y se requiere su presencia. Si el prop titulo no es de tipo string o no se proporciona, se mostrará un mensaje deadvertencia en la consola.

La biblioteca prop-types proporciona una variedad de validadores de propiedades que puedes utilizar para realizar validaciones más complejas, como verificar tipos de datos, requerir propiedades, validar valores mínimos o máximos, etc. Puedes consultar la documentación de prop-types para obtener más información sobrelos diferentes validadores disponibles.

Es importante destacar que las propiedades por defecto y la validación de props son herramientas útiles paragarantizar que los componentes se utilicen correctamente y se les pasen los datos adecuados. Esto ayuda a prevenir errores y mejora la mantenibilidad de tu aplicación ReactJS.

Tema 2: Estado y Ciclo de Vida de los Componentes

Acceso a los props dentro de un componente.

En ReactJS, puedes acceder a los props dentro de un componente utilizando el objeto props que se pasa comoargumento a la función del componente. Los props contienen los valores y las propiedades que se pasan al componente desde su componente padre.

Aquí tienes ejemplos de cómo acceder a los props dentro de un

componente:Componente Funcional:

```
import React from 'react';

const MiComponente = (props) => {
  return (
    <div>
      <h1>Título: {props.titulo}</h1>
      <p>Contenido: {props.contenido}</p>
    </div>
  );
};

export default MiComponente;
```

En este ejemplo, los props se pasan como argumento (props) a la función del componente MiComponente.Luego, dentro del código JSX, puedes acceder a los props utilizando la sintaxis props.nombreDePropiedad.

Componente de Clase:

```
import React from 'react';

class MiComponente extends React.Component {
  render() {
    return (
      <div>
        <h1>Título: {this.props.titulo}</h1>
        <p>Contenido: {this.props.contenido}</p>
      </div>
    );
  }
}

export default MiComponente;
```

En este ejemplo, en un componente de clase, los props se acceden utilizando this.props dentro del método render() del componente.

@mihifidem
If you want to master something, teach it.

En ambos casos, puedes acceder a los props y utilizar sus valores en cualquier parte del componente donde los necesites. Puedes utilizar los props en expresiones de JavaScript, condicionales, bucles, interpolaciones de textoy más para personalizar la lógica y el contenido de tu componente.

Es importante tener en cuenta que los props son de solo lectura y no se deben modificar directamente dentrodel componente. Si necesitas mantener un estado interno que pueda cambiar, puedes utilizar el estado de componente (mediante el uso de state) o utilizar algún tipo de gestión de estado adicional, como Redux.

En resumen, puedes acceder a los props dentro de un componente en ReactJS utilizando el objeto props que sepasa como argumento a la función del componente o mediante this.props en un componente de clase. Los props contienen los valores y las propiedades que se pasan al componente desde su componente padre y se pueden utilizar en cualquier parte del componente donde los necesites para personalizar su lógica y contenido.

1.4. Props como datos inmutables.

En React, las props se consideran datos inmutables, lo que significa que no se pueden modificar directamente dentro del componente hijo. Esto se debe a que React utiliza un modelo de flujo de datosunidireccional, donde los datos fluyen desde el componente padre al componente hijo a través de las props.

Al tratar las props como datos inmutables, se promueve una programación más segura y predecible. Cada vez que las props cambian en el componente padre, React detecta los cambios y vuelve a renderizar el componente hijo con las nuevas props.

Si se necesita modificar o actualizar los datos de una prop en un componente hijo, se debe realizar a través de la modificación de las props en el componente padre y volver a pasar las nuevas props actualizadas al componente hijo.

Tema 2: Estado y Ciclo de Vida de los Componentes

Ejemplo de uso de props inmutables:

```
// Componente padre
import React, { useState } from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  const [count, setCount] = useState(0);

  const incrementCount = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Contador: {count}</p>
      <ChildComponent count={count} incrementCount={incrementCount} />
    </div>
  );
}
```

```
// Componente hijo
import React from 'react';

const ChildComponent = (props) => {
  return <p>Valor recibido: {props.counter}</p>;
}
```

@mihifidem
If you want to master something, teach it.

2. Estado en ReactJS:

2.1. Introducción al concepto de estado en React.

El estado es un concepto fundamental en React que representa la información dinámica de un componente. Es una forma de almacenar y gestionar datos que pueden cambiar a lo largo del tiempo.

El estado en React **es una de las maneras en las que se procesan datos en esta librería de JavaScript.**Desde el punto de vista de los datos que React puede manejar, tenemos el objeto props, con el que podemos insertar todo tipo de propiedades a un elemento, y tenemos los estados.

El objeto props nos permite insertar datos estáticos. Con esto nos referimos a que pasamos propiedades a un componente o elemento que luego se renderiza o visualiza de una determinada manera, con base a los datos que le pasamos. Entonces, si necesitamos darle dinamismo a un elemento, es decir, datos que se modifiquen a lo largo del tiempo, **necesitaremos utilizar el estado enReact**.

El estado en React, también conocido como state, **es el segundo tipo de dato que maneja esta librería de JavaScript**. Mientras que las props son los datos que podemos pasarle a un componente o elementoReact desde afuera, **un estado se conforma por los datos internos que un componente puede manejar**. A medida que estos datos son modificados, ya sea por una interacción del usuario o por una recepción de datos de la API, el estado será modificado. Entonces, cada cambio de ese estado **provocará que el elemento o componente se renderice de nuevo con una nueva representación en pantalla.**

Ten presente que podemos utilizar un estado en cualquier tipo de componente o elemento, sea delnivel más bajo o el nivel más alto. Allá donde necesitemos datos que cambian a lo largo del tiempo,podemos utilizar un estado.

Tema 2: Estado y Ciclo de Vida de los Componentes

¿Cómo utilizar el estado en React?

Para ejemplificar cómo podemos utilizar el estado en React:

```
<!DOCTYPE html>
<html>
<head>
 <title>Ejemplo de React con useState en HTML</title>
 <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
 <script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
</head>
<body>
 <div id="root"></div>

 <script>
  // Definir el componente usando useState
  const Example = () => {
   const [count, setCount] = React.useState(0);

   const increment = ()
    => {setCount(count
    + 1);
   };

   const decrement = ()
    => {setCount(count -
    1);
   };

   return (
    React.createElement('div',
    null,
     React.createElement('p', null, 'Contador: ' + count),
     React.createElement('button', { onClick: increment }, 'Incrementar'),
     React.createElement('button', { onClick: decrement }, 'Decrementar')
    )
   );
  };
  // Renderizar el componente en el elemento
  'root'ReactDOM.render(
  React.createElement(Example),
  document.getElementById('root')
  );
 </script>
</body>
</html>
```

@mihifidem
If you want to master something, teach it.

- Se inicia el documento HTML.

```
<!DOCTYPE html>
```

- Se especifica el título de la página y se agregan los scripts de React y ReactDOM desde una CDN. Estos scripts proporcionan las funcionalidades necesarias para ejecutar el código de React en el navegador.

```
<head>
 <title>Ejemplo de React con useState en HTML</title>
 <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
 <script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
</head>
```

- Se crea un contenedor con el id "root". Este es el lugar donde se renderizará nuestrocomponente React.

```
<div id="root"></div>
```

- Se define el componente Example. Utilizamos la función React.useState para declarar una variable de estado llamada count con un valor inicial de 0, y la función setCount para actualizardicho estado.

```
const [count, setCount] = React.useState(0);
```

- Se definen dos funciones, increment y decrement, que utilizan la función setCount paraincrementar y decrementar el valor de count respectivamente.

```
const increment = ()
      => {
      setCount(count
      + 1);
    };
const decrement = ()
      => {
      setCount(count -
      1);
    };
```

- Se utiliza React.createElement para construir la estructura del componente. Se crea un principal que contiene un <p> para mostrar el valor del contador (count), y dos botones
  <button> que llaman a las funciones increment y decrement respectivamente.

```
React.createElement('div', null,
        React.createElement('p', null, 'Contador: ' + count),
        React.createElement('button', { onClick: increment }, 'Incrementar'),
        React.createElement('button', { onClick: decrement }, 'Decrementar')
      )
```

- Se utiliza ReactDOM.render para renderizar el componente Example dentro del elemento con elid "root". Esto hace que el componente se muestre en el navegador.

```
ReactDOM.render(
    React.createElement(Example),
    document.getElementById('root')
  );
```

2.2. Uso del estado para almacenar y manipular datos en un componente.

En React, el estado es utilizado para almacenar y manipular datos en un componente. El estado nos permite mantener información dinámica que puede cambiar durante la interacción del usuario con laaplicación.

Cuando se utiliza el estado en un componente, se utiliza la función useState proporcionada por React para declarar una variable de estado y su función de actualización asociada. La función useState recibeun valor inicial para el estado y devuelve un array con dos elementos: el valor actual del estado y la función para actualizar ese estado.

```
<!DOCTYPE html>
<html>
<head>
 <title>Ejemplo de React con Estado en HTML</title>
</head>
<body>
 <div id="root"></div>

 <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
 <script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
 <script>
  // Componente de React
  const Example = () => {
   const [name, setName] = React.useState('');

   const handleChange = (event) => {
    setName(event.target.value);
   };

   return (
    React.createElement('div', null,
     React.createElement('p', null, 'Nombre: ' + name),
     React.createElement('input', { type: 'text', value: name, onChange: handleChange })
    )
   );
  };
```

```
  // Renderizar el componente en el elemento 'root'
  ReactDOM.render(
  React.createElement(Example),
  document.getElementById('root')
  );
 </script>
</body>
</html>
```

Explicación del código

- Se definió un componente de React llamado Example utilizando una función flecha (() => { ... }). Este componente utiliza React.useState para declarar una variable de estado llamada name y sufunción de actualización asociada setName. El valor inicial del estado es una cadena vacía ('').

```
const Example = () => {
const [name, setName] = React.useState('');
```

- Se definió una función handleChange que se ejecutará cada vez que se produzca un cambio enel input. Dentro de esta función, se utiliza setName para actualizar el estado name con el valoringresado por el usuario (event.target.value).

```
const handleChange = (event) => {
      setName(event.target.value);
};
```

- Dentro del retorno del componente Example, se utiliza React.createElement para construir laestructura del componente. Se crea un principal que contiene un <p> para mostrar el valor del estado name, y un <input> vinculado al estado name a través del atributo value.
  Además, se configura el evento onChange del input para llamar a la función handleChange y actualizar el estado en respuesta a los cambios del usuario.

```
return (
      React.createElement('div', null,
        React.createElement('p', null, 'Nombre: ' + name),
        React.createElement('input', { type: 'text', value: name, onChange: handleChange })
      )
```

- Utilizando ReactDOM.render, se renderiza el componente Example dentro del elemento HTMLcon el id "root". Esto hace que el componente se muestre en el navegador.

```
-       ReactDOM.render(
```

```
-         React.createElement(Example),
-         document.getElementById('root')
-       );
```

Al abrir el archivo HTML en el navegador, verás un input donde puedes ingresar un nombre. A medidaque ingreses el nombre, el valor se mostrará en tiempo real debajo del input gracias al uso del estado(useState) en React. Cada vez que escribas en el input, se llamará a la función handleChange, actualizando el estado name y provocando una nueva renderización del componente para reflejar el nuevo valor en la interfaz de usuario.

2.3. Inicializar el estado en un componente.

En React, puedes inicializar el estado de un componente proporcionando un valor inicial al utilizar useState. Este valor inicial se utiliza para establecer el estado por primera vez cuando se renderiza el componente.

A continuación, se muestra un ejemplo de cómo inicializar el estado en un componente de React:

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

const Example = () => {
  // Inicialización del estado con un valor inicial de 0
  const [count, setCount] = useState(0);

  // Resto del código del componente
  // ...
};

export default Example;
```

En este ejemplo, se utiliza la función useState para inicializar el estado del componente. Se declara unavariable de estado llamada count y su función de actualización setCount. El valor inicial del estado se establece en 0 al proporcionar 0 como argumento a useState.

Puedes inicializar el estado con cualquier tipo de valor: números, cadenas de texto, arreglos, objetos,etc. Al proporcionar el valor inicial, el estado se establecerá con ese valor cuando el componente se renderice por primera vez.

Es importante destacar que el valor inicial solo se utilizará al renderizar el componente por primeravez. Si posteriormente actualizas el estado utilizando la función de actualización (setCount en el ejemplo), el nuevo valor reemplazará el valor inicial proporcionado.

Por lo tanto, inicializar el estado en un componente te permite establecer el valor inicial del estado demanera controlada. Esto es útil cuando deseas tener un valor específico al comenzar y luego modificarlo según las interacciones del usuario o eventos dentro del componente.

2.4. Renderizar el componente en base a los cambios en el estado.

En React, cuando el estado de un componente cambia, React se encarga automáticamente de volver arenderizar el componente para reflejar esos cambios en la interfaz de usuario. Esto se conoce como renderizado condicional, ya que el componente se renderiza de manera diferente según el estado actual.

A continuación, se muestra un ejemplo de cómo renderizar el componente en base a los cambios en elestado en un componente funcional de React utilizando el hook useState:

```
<!DOCTYPE html>
<html>
<head>
 <title>Ejemplo de Renderizado Condicional en HTML</title>
</head>
<body>
 <div id="root"></div>

 <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
 <script src="https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js"></script>
 <script>
  // Componente de React
  const Example = () => {
   const [isLoggedIn, setIsLoggedIn] = React.useState(false);

   const handleLogin = () => {
    setIsLoggedIn(true);
   };

   const handleLogout = () => {
    setIsLoggedIn(false);
   };

   return (
    React.createElement('div', null,
    isLoggedIn ? (
      React.createElement('p', null, 'Bienvenido, usuario. ',
       React.createElement('button', { onClick: handleLogout }, 'Cerrar sesión')
```

```
        )
      ) : (
        React.createElement('p', null, 'Inicia sesión para continuar. ',
          React.createElement('button', { onClick: handleLogin }, 'Iniciar sesión')
        )
      )
    )
  );
};

// Renderizar el componente en el elemento 'root'
ReactDOM.render(
React.createElement(Example),
document.getElementById('root')
);
</script>
</body>
</html>
```

En este ejemplo, hemos agregado los scripts de React y ReactDOM utilizando una CDN. Luego, definimos el componente Example que utiliza React.useState para controlar si el usuario ha iniciado sesión o no. El estado se inicializa en false.

Dentro del retorno del componente, utilizamos React.createElement para construir la estructura del componente. Utilizamos una expresión condicional para renderizar diferentes elementos según el valorde isLoggedIn. Si isLoggedIn es true, se muestra un mensaje de bienvenida y un botón para cerrar sesión. Si isLoggedIn es false, se muestra un mensaje para iniciar sesión y un botón para iniciar sesión.

Al hacer clic en los botones, se actualiza el estado isLoggedIn utilizando la función setIsLoggedIn, lo queprovoca una nueva renderización del componente y muestra los elementos correspondientes en función del nuevo valor del estado.

3. Ciclo de vida de los componentes de clase:

3.1. componentDidMount: método invocado después de que elcomponente se haya montado en el DOM.

En React, el método componentDidMount es uno de los métodos del ciclo de vida de los componentesde clase. Este método se invoca automáticamente después de que el componente se haya montado enel DOM, es decir, cuando el componente ha sido creado y se ha adjuntado al árbol de elementos del DOM.

El método componentDidMount se utiliza comúnmente para realizar tareas de inicialización, como la obtención de datos desde una API, establecer eventos o cualquier otra acción que deba realizarse unavez que el componente esté listo y visible en el DOM.

A continuación, se muestra un ejemplo de cómo utilizar el método componentDidMount en un componente de clase de React:

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

class Example extends Component {
  componentDidMount() {
    // Lógica o tareas de inicialización aquí
    console.log('El componente se ha montado en el DOM');
  }

  render() {
    return (
      <div>
        <h1>Componente de Ejemplo</h1>
        {/* Contenido del componente */}
      </div>
    );
  }
}

export default Example;
```

En este ejemplo, el componente Example se extiende de Component de React y se define el métodocomponentDidMount. Dentro de este método, puedes escribir la lógica o realizar tareas de inicialización necesarias una vez que el componente se haya montado en el DOM.

En el retorno del componente (render), se muestra el contenido del componente, que puede incluir cualquier otro elemento o componente.

Cuando el componente Example se renderice por primera vez y se monte en el DOM, se llamará automáticamente al método componentDidMount. En este ejemplo, simplemente se imprime un mensaje en la consola, pero en una aplicación real, puedes realizar cualquier tarea requerida en este punto, como la configuración de un temporizador, la suscripción a eventos, la llamada a una API, entreotras acciones.

El método componentDidMount se ejecutará solo una vez, después del primer renderizado del componente. Si necesitas realizar acciones adicionales en respuesta a cambios posteriores en el estadoo las propiedades, puedes utilizar otros métodos del ciclo de vida, como componentDidUpdate o componentWillUnmount.

Es importante tener en cuenta que el método componentDidMount solo está disponible para componentes de clase en React. Si estás utilizando componentes funcionales con hooks, puedes lograrel mismo comportamiento utilizando el hook useEffect con una dependencia vacía ([]).

3.2. componentDidUpdate: método invocado después de que elcomponente se haya actualizado en el DOM.

En React, el método componentDidUpdate es uno de los métodos del ciclo de vida de los componentesde clase. Este método se invoca automáticamente después de que el componente se haya actualizado en el DOM, es decir, cuando se han realizado cambios en el componente y se han reflejado en la interfaz de usuario.

El método componentDidUpdate se utiliza comúnmente para realizar acciones adicionales después de que el componente se haya actualizado, como realizar llamadas a API, actualizar el estado o interactuarcon el DOM.

A continuación, se muestra un ejemplo de cómo utilizar el método componentDidUpdate en un componente de clase de React:

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

class Example extends Component {
 constructor(props) {
  super(props);
  this.state = {
  count: 0
  };
 }

 componentDidMount() {
  console.log('El componente se ha montado en el DOM');
 }

 componentDidUpdate(prevProps, prevState) {
  if (prevState.count !== this.state.count) {
   console.log('El estado count se ha actualizado:', this.state.count);
  }
 }

 increment() {
  this.setState((prevState) => ({ count: prevState.count + 1 }));
 }

 render() {
  return (
   <div>
```

```
      <p>Contador: {this.state.count}</p>
      <button onClick={() => this.increment()}>Incrementar</button>
    </div>
   );
  }
}

export default Example;
```

En este ejemplo, además del método componentDidMount, se ha añadido el método componentDidUpdate. Dentro de componentDidUpdate, se verifica si el estado anterior (prevState.count) es diferente al estado actual (this.state.count). Si hay una diferencia, se imprime unmensaje en la consola con el nuevo valor del estado count.

En el método increment, se actualiza el estado count utilizando la función de actualización del estado,pasando una función que toma el estado anterior (prevState) y devuelve un nuevo estado actualizado.

Cada vez que se hace clic en el botón "Incrementar", el estado count se actualiza y se llama a componentDidUpdate. En este ejemplo, el método componentDidUpdate verifica si el estado count hacambiado y, en caso afirmativo, imprime un mensaje en la consola con el nuevo valor del estado.

Es importante tener en cuenta que, al utilizar componentDidUpdate, debes asegurarte de incluir unacondición para evitar actualizaciones infinitas, ya que cualquier cambio en el estado dentro de este método también provocará una nueva llamada a componentDidUpdate.

Recuerda que, si estás utilizando componentes funcionales con hooks, puedes lograr el mismo comportamiento utilizando el hook useEffect con una dependencia específica (useEffect(callback, [dependency])) y comprobando los cambios en la dependencia dentro de la función de efecto.

3.3. componentWillUnmount: método invocado antes de que elcomponente se desmonte y sea eliminado del DOM.

En React, el método componentWillUnmount es uno de los métodos del ciclo de vida de los componentes de clase. Este método se invoca automáticamente antes de que el componente sedesmonte y sea eliminado del DOM.

El método componentWillUnmount se utiliza para realizar tareas de limpieza y liberación de recursosantes de que el componente se elimine, como cancelar suscripciones a eventos, detener temporizadores, eliminar escuchadores, entre otros.

A continuación, se muestra un ejemplo de cómo utilizar el método componentWillUnmount en un componente de clase de React:

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

class Example extends Component {
 timerID = null;

 componentDidMount() {
  this.timerID = setInterval(() => {
   console.log('Ejecutando tarea periódica...');
  }, 1000);
 }

 componentWillUnmount() {
  clearInterval(this.timerID);
 }

 render() {
  return (
   <div>
    <h1>Componente de Ejemplo</h1>
    {/* Contenido del componente */}
   </div>
  );
 }
}

export default Example;
```

En este ejemplo, el componente Example utiliza componentDidMount para iniciar una tarea periódica utilizando setInterval. Se almacena el ID del temporizador en la propiedad timerID del componente.

En componentWillUnmount, se utiliza clearInterval para detener y limpiar la tarea periódica antes deque el componente se desmonte.

Es importante destacar que, en este ejemplo, se ha declarado la propiedad timerID directamente en lainstancia de la clase (this.timerID) para evitar problemas de referencia y asegurar que el intervalo se limpie correctamente.

Cuando el componente Example se monta en el DOM, se inicia la tarea periódica. Y cuando el componente se desmonta, se llama automáticamente a componentWillUnmount, lo que detiene y limpia la tarea periódica utilizando clearInterval.

El uso de componentWillUnmount es especialmente útil cuando necesitas realizar tareas de limpieza yliberación de recursos al finalizar la vida útil de un componente. Esto garantiza que no haya fugas de memoria o tareas en segundo plano activas después de que el componente se haya eliminado del DOM.

Recuerda que, si estás utilizando componentes funcionales con hooks, puedes lograr el mismo comportamiento utilizando el hook useEffect con una función de limpieza (useEffect(() => { return cleanup }, [])). La función de limpieza se ejecutará antes de que el componente se desmonte.

3.4. Utilidad de los métodos del ciclo de vida para realizar tareas específicas en diferentes etapas del ciclo de vida de un componente.

Los métodos del ciclo de vida en React son funciones predefinidas que se ejecutan automáticamenteen diferentes etapas del ciclo de vida de un componente. Estos métodos proporcionan puntos de enganche para realizar tareas específicas en momentos clave del ciclo de vida del componente.

A continuación, se detallan algunos ejemplos de tareas comunes que se pueden realizar utilizando losmétodos del ciclo de vida:

componentDidMount: Este método se invoca después de que el componente se haya montado en elDOM. Se utiliza para realizar tareas de inicialización, como la obtención de datos iniciales de una API,configurar suscripciones a eventos o iniciar tareas periódicas.

componentDidUpdate: Este método se invoca después de que el componente se haya actualizado enel DOM. Se utiliza para realizar acciones adicionales después de que el componente se haya actualizado, como realizar llamadas a API en respuesta a cambios en las propiedades o el estado, actualizar el estado o interactuar con el DOM.

componentWillUnmount: Este método se invoca antes de que el componente se desmonte y sea eliminado del DOM. Se utiliza para realizar tareas de limpieza y liberación de recursos, como cancelarsuscripciones a eventos, detener temporizadores o eliminar escuchadores, para evitar fugas de memoria y mantener el rendimiento óptimo de la aplicación.

Estos son solo algunos ejemplos de las tareas que se pueden realizar utilizando los métodos del ciclo devida. La utilidad de cada método depende de las necesidades específicas de tu aplicación.

Es importante tener en cuenta que, a partir de React 16.3, algunos de los métodos del ciclo de vida, como componentWillMount, componentWillReceiveProps y componentWillUpdate, se consideran obsoletos y están en desuso. Se recomienda utilizar los nuevos métodos como componentDidMount,componentDidUpdate y componentWillUnmount, o utilizar los hooks de React en componentes funcionales para lograr un código más conciso y fácil de mantener.

Al comprender y utilizar correctamente los métodos del ciclo de vida, puedes controlar el flujo y el comportamiento de tus componentes en diferentes etapas y realizar tareas específicas en cada momento adecuado. Esto te permite crear componentes más sofisticados y responder de manera efectiva a los cambios en el estado, las propiedades y la interacción del usuario en tu aplicación React.

Tema 2: Estado y Ciclo de Vida de los Componentes