React JSComponentes

Parte 2

Presentación

En esta unidad nos adentramos más en el mundo de React, incorporando los conocimientos necesarios para hacer componentes más avanzados y aplicaciones o sitios con funcionalidades más complejas..

Objetivos

Que los participantes logren...

- Armar componentes más avanzados mediante el uso de props y estado.
- Repetir elementos de forma dinámica y responder a eventos generados por estos..
- Separar la aplicación o sitio en componentes que respondan a ciertas rutas.

Bloques temáticos

1. Props.
2. Estado.
3. Hooks.
4. Eventos.
5. Listar elementos.
6. Ruteo.

1. Props

Las propiedades (o props de aquí en adelante) son valores que se pasan a los componentes para modificar su comportamiento o valor de retorno. Se especifican el elemento de la misma forma que un atributo HTML.

Como vimos previamente los valores de las props pueden ser valores literales como strings pasados entre comillas, expresiones de JavaScript, referencias a funciones u otros componentes.

Ya sea que declares un componente como una función o como una clase, este nunca debe modificar sus props. Considera esta función sum:

```
function suma(a, b) {

   return a + b;
}
```

Tales funciones son llamadas “puras” porque no tratan de cambiar sus entradas, y siempre devuelven el mismo resultado para las mismas entradas.

En contraste, esta función es impura porque cambia su propia entrada:

```
function retiro(cuenta, monto) {
   cuenta.total -= monto;
}
```

React es bastante flexible pero tiene una sola regla estricta: **Todos los componentes de React deben actuar como funciones puras con respecto a sus props.**

2. Estado

El estado es el “corazón” de los componentes de React. Es la característica que te permitirá desarrollar aplicaciones mucho más interesantes. Un estado en React es, entonces, un almacén de datos mutable de componentes y que además son autónomos.

Diferencia entre estado y props

Los estados actúan en el contexto del componente y por otro, las propiedades crean una instancia del componente cuando le pasas un nuevo valor desde un componente padre.
Los valores de las propiedades los pasas de padres a hijos y los valores de los estados los defines en el componente, no se inician en el componente padre.

Estado en clases

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }
  render() {
    return (
      <div>
        Hoy es{this.state.date.toLocaleTimeString()}
      </div>
    );
  }
}
```

En los componentes de clase el estado se define como una propiedad de la clase llamada state representada por un objeto literal de Javascript, la cual podemos modificar mediante el uso del método setState que todos los componentes que extienden a React.Component tienen.

Estado en componentes funcionales

```
import React, { useState } from "react";
const App = (props) => {
   const [hoy, setHoy] = useState(new Date());
   return (
      <div>
         Hoy es {hoy.toLocaleDateString()}
      </div>
   );
}
export default App;
```

En el caso de componentes funcionales podemos hacer uso del hook `useState` para definir una variable local a nuestro componente. Dicha variable podría ser modificada con la función asociada a la misma (en este caso `setHoy`).

3. Hooks

Hooks son una nueva característica en React 16.8. Estos te permiten usar el estado y otras características de React sin escribir una clase.

Los hooks son básicamente **funciones especiales** que permiten "conectarnos" a distintas características de React y permiten su reutilización entre componentes. Por ejemplo usando componentes de clase, la lógica de manejo de estado sería muy difícil (sino imposible) de reutilizar en otros componentes. Creando nuestros propios hooks podríamos compartir esta funcionalidad con todos los componentes que la necesitaran.

Igualmente vamos a centrarnos en 2 de los hooks que incluye React que utilizaremos más a menudo.

Ejemplo useState

En una clase, inicializamos el estado `count` a 0 estableciendo `this.state` a `{ count: 0 }` en el constructor:

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

En un componente funcional no existe `this` por lo que no podemos asignar o leer `this.state`. En su lugar, usamos el Hook `useState` directamente dentro de nuestro componente

```
import React, { useState } from 'react';
function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);
```

¿Qué hace la llamada a useState? Declara una “variable de estado”. Nuestra variable se llama `count`, pero podemos llamarla como queramos, por ejemplo banana. Esta es una forma de “preservar” algunos valores entre las llamadas de la función - **`useState`** es una nueva forma de usar exactamente las mismas funciones que this.state nos da en una clase. Normalmente, las variables “desaparecen” cuando se sale de la función, pero las variables de estado son conservadas por React

¿Qué pasamos a useState como argumento? El único argumento para el Hook `useState()` es el estado inicial. Al contrario que en las clases, el estado no tiene por qué ser un objeto. Podemos usar números o strings si es todo lo que necesitamos. En nuestro ejemplo, solamente queremos un número para contar el número de clicks del

usuario, por eso pasamos 0 como estado inicial a nuestra variable. (Si queremos guardar dos valores distintos en el estado, llamaríamos a `useState()` dos veces)

¿Qué devuelve useState? Devuelve una pareja de valores: el estado actual y una función que lo actualiza. Por eso escribimos `const [count, setCount] = useState()`. Esto es similar a `this.state.count` y `this.setState` en una clase, excepto que se obtienen juntos. Si no conoces la sintaxis que hemos usado volveremos a ella al final de esta página.

Ahora que sabemos que hace el Hook `useState`, nuestro ejemplo debería tener más sentido:

```
import React, { useState } from 'react';
function Example() {
  // Declaración de una variable de estado que llamaremos "count"
  const [count, setCount] = useState(0);
```

Declaramos una variable de estado llamada `count` y le asignamos a 0. React recordará su valor actual entre re-renderizados, y devolverá el valor más reciente a nuestra función. Si se quiere actualizar el valor de count actual, podemos llamar a `setCount`

useEffect

El Hook de efecto te permite llevar a cabo efectos secundarios en componentes funcionales:

```
import React, { useState, useEffect } from 'react';
function Example() {
  const [count, setCount] = useState(0);
  // De forma similar a componentDidMount y componentDidUpdate
  useEffect(() => {
    // Actualiza el título del documento usando la API del navegador
    document.title = 'You clicked ${count} times';
```

```
  });
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Este fragmento está basado en el ejemplo del contador anterior, pero le hemos añadido una funcionalidad nueva: actualizamos el título del documento con un mensaje personalizado que incluye el número de clicks.

Peticiones de datos, establecimiento de suscripciones y actualizaciones manuales del DOM en componentes de React serían ejemplos de efectos secundarios. Tanto si estás acostumbrado a llamar a estas operaciones “efectos secundarios” (o simplemente “efectos”) como si no, probablemente los has llevado a cabo en tus componentes con anterioridad.

Por defecto se ejecuta después del primer renderizado y después de cada actualización.

Si queremos controlar cuando se ejecuta el hook useEffect podemos pasarle como 2do parámetro un array con los valores que deseamos monitorear.

```
const Ejemplo = (props) => {
  const [count, setCount] = useState(0)
  useEffect(() => {
    // código del hook
  }, [count])
};
```

Si pasamos un array vacío, solo se ejecutará el código del hook cuando el componente se incluya en el documento por primera vez. Este modo es muy útil para cargar datos desde el servidor cuando se incluye un componente.

```
const Ejemplo = (props) => {
    useEffect(() => {
        console.log('Se agregó Ejemplo');
    }, [])
}
```

Si necesitamos ejecutar una función al eliminar un componente podemos incluir ese código dentro de una función que sea devuelta por el hook

```
const Ejemplo = (props) => {
    useEffect(() => {
        console.log('Se agregó Ejemplo');
        return () => console.log('Se quitó Ejemplo')
    }, [])
}
```

4. Eventos

Los eventos en React se manejan de forma similar a los eventos de HTML con algunas diferencias:

- En React los eventos se nombre con **camelCase** en vez de todo minuscula
- JSX permite pasar una función como manejador del evento en vez de un string como HTML.
- En React no podemos devolver false para prevenir el comportamiento por defecto del evento. Debemos llamar explícitamente al método preventDefault().

Ejemplo de manejo de click

```
const MiBoton = (props) => {
   const [prendido, setPrendido] = useState(false);
   return (
      <button onClick={() => setPrendido(!prendido)}>
         {prendido ? 'Apagar' : 'Prender'}
      </button>
   );
}
```

5. Listar elementos

La forma más común de listar elementos en React es utilizando el método map() de los arrays. De esta forma podemos iterar la lista completa obteniendo cada uno de los ítems de la lista en cada iteración y utilizarlo para devolver un componente u objeto.

```
const ListaPaises = (props) => {
  const paises = ['Argentina', 'Uruguay', 'Brasil', 'Chile'];
  return (
    <ul>
      {paises.map(pais => <li key={pais.toLowerCase()}>
        {pais}</li>)}
    </ul>
  )
}
```

En el ejemplo nuestro componente recorre el array países y por cada elemento (llamado país en el método map) devolveremos un nuevo elemento li cuyo contenido es el país que estamos recorriendo.

Cabe aclarar que cada vez que listemos elementos de forma dinámica React nos va a pedir que incluyamos una propiedad llamada key (sería el id de cada elemento). Esto facilita a React el trabajo de modificar el DOM. En caso de omitirlo veremos una advertencia en la consola.

Es importante aclarar que el método map() también nos permite usar el índice (número actual de la "vuelta") dentro de la función que utilizamos. Mucha gente usa este índice como valor de la propiedad key, lo cual es incorrecto debido a que los elementos del array pueden cambiar de posición, agregarse nuevos o eliminarse algunos y esto haría que ese índice cambie, confundiendo a React. Siempre es importante usar un valor lo más único posible. En el ejemplo de recién usamos el nombre del país pasado a minúsculas. En el caso de que estuviéramos trabajando con datos que vienen de una base de datos, seguramente tendríamos disponible el id del registro que estemos mostrando.

6. Ruteo

Mediante el ruteo podremos hacer que nuestra aplicación o sitio hecho en React responda a diferentes urls y pueda navegar entre ellas. Como esta no es una funcionalidad incluida en React utilizaremos una librería llamada React Router DOM y sus distintos componentes.

React Router habilita el "enrutamiento del lado del cliente".

En los sitios web tradicionales, el navegador solicita un documento de un servidor web, descarga y evalúa los activos de CSS y JavaScript, y presenta el HTML enviado desde el servidor. Cuando el usuario hace clic en un enlace, comienza el proceso nuevamente para una nueva página.

El enrutamiento del lado del cliente permite que su aplicación actualice la URL desde un clic en un enlace sin realizar otra solicitud de otro documento desde el servidor. En su lugar, su aplicación puede mostrar inmediatamente una nueva interfaz de usuario y realizar solicitudes de datos fetch para actualizar la página con nueva información.

Esto permite experiencias de usuario más rápidas porque el navegador no necesita solicitar un documento completamente nuevo o volver a evaluar los activos de CSS y JavaScript para la página siguiente. También permite experiencias de usuario más dinámicas con cosas como la animación.

Requisitos previos.

Debe usar **BrowserRouter** de `react-router-dom` para que el desplazamiento suave de `react-router-hash-link` funcione. Para instalar y usar `react-router`, ingrese el siguiente comando:

```
npm i react-router-dom
```

Los principales componentes de la librería son:

BrowserRouter
Es una envoltura para nuestra aplicación. Esta envoltura nos da acceso al API de historia de HTML5 para mantener nuestra interfaz gráfica en sincronía con la locación actual o URL. Debemos tener en cuenta que esta envoltura sólo puede tener un hijo. Por lo general es Switch.

Switch
Este componente, causa que solo se renderice el primer hijo Route o Redirect que coincida con la locación o URL actual. En el caso que no usemos Switch todas las rutas que cumplan con la condición se renderizarán.

Route
Para definir las diferentes rutas de nuestra aplicación, podemos usar el componente Route. La función de este componente es elegir que renderizar según la locación actual.

Este componente recibe las siguientes propiedades:

- **path:** la ruta en la que se renderizará el componente en forma de cadena de texto.
- **exact:** un booleano para definir si queremos que la ruta tiene o no que ser exacta para renderizar un componente. Ej: /index !== /index/all.
- **component:** recibe un componente a renderizar. Crea un nuevo elemento de React cada vez. Esto causa que el componente se monte y desmonte cada vez (no actualiza).

```
import React, { Component } from 'react';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
export default class App extends Component {
  render() {
    return (
      <BrowserRouter>
        <Switch>
          <Route path=' /' exact component={HomePage} />
```

```
            <Route path=' /nosotros' exact component={NosotrosPage} />
          </Switch>
        </BrowserRouter>
      );
    }
}
```

Link

Crea un hipervínculo que nos permite navegar por nuestra aplicación, agrega una nueva locación a la historia. Su principal propiedad es:

- **to**: una locación en forma de cadena de texto. Esta será la locación a la que navegaremos cuando le damos click a un hipervínculo.

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';
import ' ./NavBar.css';
class NavBar extends Component {
  render() {
    return (
      <div className="NavBar">
        <div className="link-container">
          <Link to="/page1" className="link">Página 1</Link>
        </div>
        <div className="link-container">
          <Link to="/page2" className="link">Página 2</Link>
        </div>
      </div>
    );
  }
}
export default NavBar;
```

Bibliografía utilizada y sugerida

Artículos de revista en formato electrónico:

Enrutando-en-react. Disponible desde la URL: https://medium.com/@simonhoyos/enrutando-en-react-cd9e4ad6e3d3

MDN Web Docs. Disponible desde la URL: https://developer.mozilla.org/

React.Disponible desde la URL: https://es.reactjs.org/

React.**Router.** Disponible desde la URL: https://reactrouter.com/web/api/BrowserRouter

Libros y otros manuscritos:

Alex Banks &Eve Porcello. Learning React: desarrollo web funcional con React y Redux. 2017.