React.JS

Introducción

Módulo 04

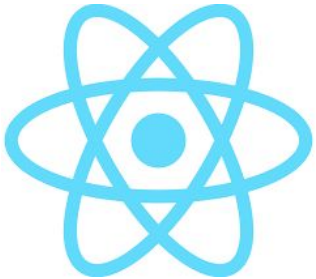
React Redux Connect

Componentes de orden superior (HOC)

Connect es un componente React que modifica un componente preexistente. ¿Cómo es esto?

En React.Js todo es un componente. Por otro lado, muchas veces necesitamos reutilizar lógica dentro de componentes ¿Cómo podemos reutilizar procesos de componente en componente? A través de los componentes de orden superior.

Así como una función de orden superior es una función que retorna otra función, así también un componente de orden superior es una función que retorna un componente React. Los componentes de orden superior, en general, modifican las propiedades del componente que toman.



```
import React from "react";

function conFecha(Component) {
    const fecha = Date.now();

    return function(props) {
        return (
            <Component fecha={fecha} />
        )
    }
}

function ComponenteConFecha(props) {
    return(
        <div>
            La fecha actual es {props.fecha}
        </div>
    )
}

export default conFecha(ComponenteConFecha);
```

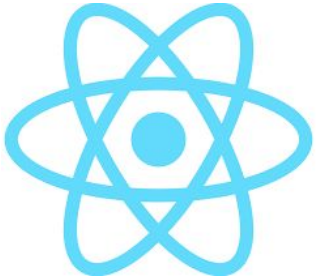
Acá podemos ver un componente de orden superior: una función que retorna un componente React.Js. El componente retornado será el mismo componente que se pasó como argumento, pero con la fecha actual anidada en las props.

Otro ejemplo de un componente de orden superior es la función *withRouter()* de React Router, que inyecta el acceso a los objetos del Router (como location o match) a un componente.

El componente Connect

Connect es un componente de orden superior que permite ofrecer a otro componente acceso al estado de Redux y al método dispatch, a través de las Props.

Para usar *connect,* además de importarlo, es necesario definir cómo el estado actual de Redux es mapeado a las props del componente destino y, si es necesario, como el método dispatch es mapeado a las props de un componente.



```
import React from "react";

import {connect} from 'react-redux';

function Nav(props) {

    let userField = <div>INGRESAR</div>

    if(props.user.isLogged) {
        userField = <div>Bienvenido {props.user.name}</div>
    }

    return (
        <div>
            <nav></nav>
            {userField}
        </div>
    )
}

const mapStateToProps = state => ({
    user: state.users.actualUser
});

export default connect(mapStateToProps)(Nav);
```

El componente ***connect()*** tiene dos instancias.

Primero, la función *connect()* recibe los “mapeadores”, aquellas funciones que conviertan los elementos de Redux (el state y el dispatch si es necesario) en props del futuro componente.

Luego que la función connect está configurada con sus “mapeadores”, se puede implementar en un componente definido. En ese componente, se inyectarán las props definidas.

```
import React from "react";
import { connect } from "react-redux";
import { logInAction } from "./ruta/a/acciones/usuario";

function LoginButton(props) {
    let toRender = (
        <button onClick={() => props.logIn(props.user)}>Iniciar sesion</button>
    );
    if (props.user.isLogged) {
        toRender = <div>Bienvenido! {props.user.name}</div>;
    }
    return <div>{toRender}</div>;
}

const mapStateToProps = (state) => ({
    user: state.users.actualUser,
});
const mapDispatchToProps = (dispatch) => ({
    logIn: (user) => dispatch(logInAction(user)),
});

export default connect(mapStateToProps, mapDispatchToProps)(LoginButton);
```

Además del estado, se puede mapear el método dispatch, encargado de ejecutar acciones en el Store. Con estos dos métodos tenemos las operaciones de lectura (state) y escritura (dispatch) del Store en el componente.

En vez de ir pasando el estado mediante las props, comunicamos cada componente individualmente con el Store global suministrado por Provider.

¡Muchas gracias!

¡Sigamos trabajando!