educación IT
CODE YOUR FUTURE

ReactJS Developer

Módulo 7

React Redux

React Redux

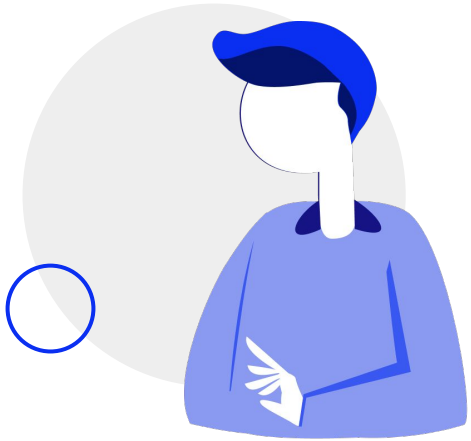
Comencemos enfatizando que *Redux* **no tiene relación alguna con React.** Puedes escribir aplicaciones *Redux* con **React, Angular, Ember, jQuery o Vanilla JavaScript.**

Dicho esto, *Redux* funciona muy bien con librerías como React y Deku porque te permiten **describir la interfaz de usuario como una función de estado** y *Redux* **emite actualizaciones de estado en respuesta a acciones.**

Para asociar React con *Redux* se recurre a la idea de **separación de presentación y componentes contenedores.** Si no estás familiarizado con estos términos, en la próxima slide veremos una tabla para comparar cada uno de los componentes.

	Componentes de presentación	**Componentes contenedores**
Propósito	Cómo se ven las cosas	Cómo funcionan las cosas
Pertinente a Redux	No	Sí
¿Cómo leer datos?	Lee datos de las `props`	Se subscribe al estado en Redux
¿Cómo escriben datos?	*Callback* desde las `props`	Envia acciones a Redux
¿Cómo son generados?	Manualmente	Generados por React Redux usualmente

La mayoría de los componentes que escribiremos serán de presentación, pero necesitaremos generar algunos componentes contenedores para conectarlos al *Store* que maneja *Redux*.



Componentes de *React Redux*

Implementando React Redux

Ahora es el momento de **conectar los componentes de presentación a *Redux* mediante la creación de algunos contenedores.**

Técnicamente, un **componente contenedor** es un componente de React que utiliza **`store.subscribe ()`** para **leer una parte del árbol de estado en *Redux* y suministrar los *props* a un componente de presentación que renderiza.**

Puedes escribir un componente contenedor de forma manual, pero mi sugerencia es generar los componentes contenedores con la función **`connect()`** de la librería ***React Redux***, ya que ofrece muchas optimizaciones útiles para evitar re-renders innecesarios.

Un beneficio de utilizar esta librería es que no tienes que preocuparte por la implementación del método **`shouldComponentUpdate,`** recomendado por React para mejor rendimiento.

Para usar **`connect()`**, es necesario definir una función llamada **`mapStateToProps`** que indica **cómo transformar el estado actual del store Redux en los props que desea pasar a un componente de presentación.**

Por ejemplo

Nuestro componente necesita calcular propiedades para pasar al componente, así que definimos una función que filtra el **`state.propiedades`** de acuerdo con el **`state.visibilityFilter`** y lo usamos en su **`mapStateToProps`**:

```
1  const mapStateToProps = (state) => {
2      return {
3          tasks: state.tasks
4      }
5  }
```

Además de leer el estado, los componentes contenedores pueden **enviar acciones.** De manera similar, puede definir una función llamada **`mapDispatchToProps()`** que recibe el método **`dispatch()`** y devuelve **los callback props que deseas inyectar en el componente de presentación.**

Por ejemplo

Queremos que el componente inyecte un `prop` llamado **`onTodoClick`** en el componente, y queremos que **`onTodoClick`** envíe una acción **`TOGGLE_TODO`**:

```
1  const mapDispatchToProps = (dispatch) => {
2      return {
3          onTodoClick: (id) => {
4              dispatch(toggleTodo(id))
5          }
6      }
7  }
```

Finalmente, creamos un componente llamando **`connect()`** y le pasamos estas dos funciones:

```
1  import { connect } from "react-redux";
2
3  const MiComponente = () => { /**Mi componente React */ }
4
5  const conector = connect(mapStateToProps, mapDispatchToProps);
6
7  export default conector(MiComponente);
8
```

Transferir al *Store*

Todos los componentes contenedores necesitan **acceso al *store* Redux para que puedan suscribirse a ella.**

Una opción sería pasarlo como un `prop` a cada componente contenedor. Sin embargo, se vuelve tedioso, pues hay que enlazar *store* incluso a través del componentes de presentación, ya que puede suceder que tenga que renderizar un contenedor allá en lo profundo del árbol de componentes.

La opción recomendada es usar un componente *React Redux* especial llamado **`<Provider>`** para hacer que el *store* **esté disponible para todos los componentes del contenedor en la aplicación sin pasarlo explícitamente. Solo es necesario utilizarlo una vez al renderizar el componente raíz.** Veremos un ejemplo en la próxima slide.

```
1   import React from 'react'
2   import { render } from 'react-dom'
3   import { Provider } from 'react-redux'
4   import { createStore } from 'redux'
5   import todoApp from './reducers'
6   import App from './components/App'
7
8   let store = createStore(todoApp)
9
10  render(
11    <Provider store={store}>
12      <App />
13    </Provider>,
14    document.getElementById('root')
15  )
16
```

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!