

An abstract graphic on the left side of the slide, consisting of a network of light blue lines and small circles, resembling a circuit board or a neural network, set against a dark blue background.

REDUX

ADVANCED STATE MANAGEMENT

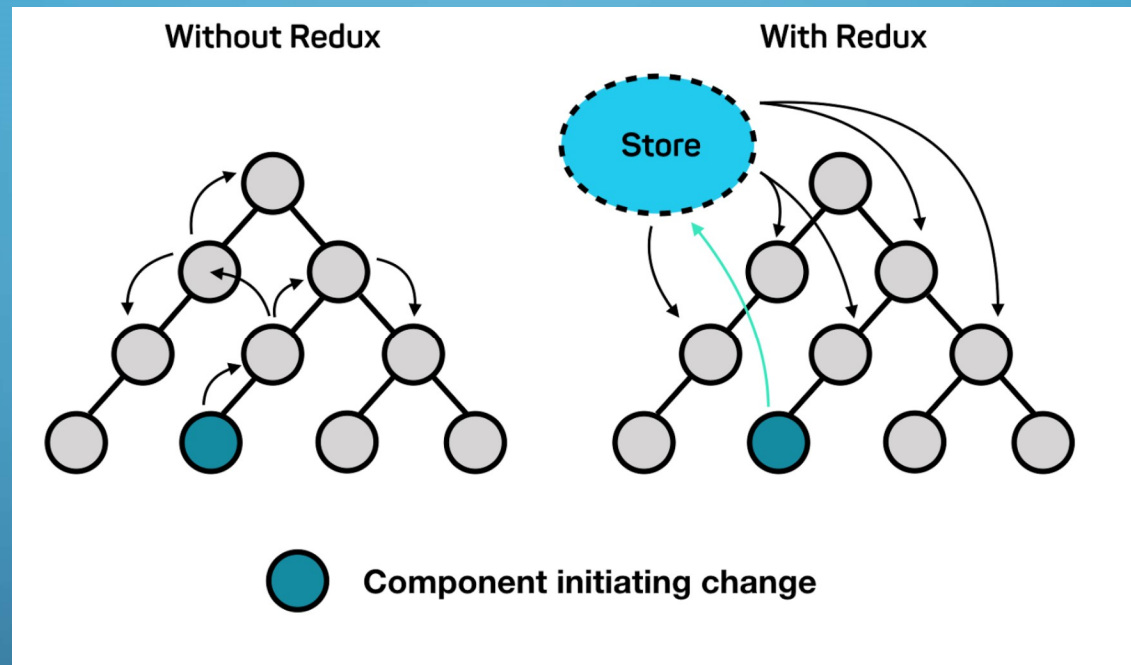
QUÉ ES?

Redux es una librería que nos permite administrar el estado, de manera muy similar a cómo utilizamos context. Con context, utilizamos el provider y el consumer como una especie de portal para omitir pasar props a través de cada componente. Con Redux, estamos eliminando la gestión de estado de React por completo y moviéndola a un almacenamiento separado.

Si bien se usa principalmente con React, se puede usar con cualquier otra librería o framework de JavaScript o sin ellos. Y además es liviano.

POR QUÉ NECESITAMOS UNA HERRAMIENTA DE MANEJO DE ESTADO

- La administración interna del estado por parte de los componentes sin necesidad de librerías externas, funciona bien para aplicaciones con pocos componentes, pero a medida que la aplicación crece, la gestión de estados compartidos entre componentes se convierte en una tarea difícil.
- Idealmente, los datos en un componente deben vivir en un solo componente, por lo que compartir datos entre componentes hermanos se vuelve difícil.
- Por ejemplo, en React, para compartir datos entre hermanos, un estado tiene que vivir en el componente padre, el cual proporciona un método para actualizar su estado y se lo pasa como props a estos componentes hijos.



- Lo mismo sucede cuando un estado tiene que ser compartido entre componentes que están muy separados en el árbol de componentes. El estado tiene que pasar de un componente a otro hasta que llega a donde se necesita.
- Esto hace que el estado sea difícil de mantener y menos predecible. También significa pasar datos a componentes que no los necesitan.
- Redux nos permite facilitar la administración de estos estados.

POR QUÉ USARÍAMOS REDUX?

- Si bien ahora podemos usar context con useReducer, el context solía ser más difícil de usar y menos útil. Esto hizo que las herramientas de administración de Redux (o similares) fueran la única opción.
- El código de Redux es extremadamente comprobable.

CÓMO FUNCIONA

- La forma en que funciona Redux es simple. Hay un almacenamiento central (store) que contiene todo el estado de la aplicación. Cada componente puede acceder al estado almacenado sin tener que enviar props de un componente a otro.
- Hay tres partes de construcción: actions, store y reducers.

1. STORE IN REDUX

- El store mantiene el estado de la aplicación. Solo hay un store en cualquier aplicación de Redux. Podemos acceder al estado almacenado, actualizar el estado y registrar o cancelar el registro de listeners a través de métodos auxiliares. Las acciones realizadas en el estado siempre devuelven un nuevo estado. Por lo tanto, el estado es muy fácil y predecible.

```
import { createStore } from "redux";  
import reducer from "../reducers";  
  
const store = createStore(reducer);  
  
export default store;
```

1. STORE RESPONSIBILITIES

Mantiene el estado de la aplicación;

Permite el acceso al estado via `getState()`;

Permite la actualización del estado via `dispatch(action)`;

Registra los listeners via `subscribe(listener)`;

Maneja la anulación del registro de listeners via el retorno de la función de `subscribe(listener)`.

2. REDUCERS IN REDUX

- Los Reducers son funciones puras que toman el estado actual de una aplicación, realizan una acción y devuelven un nuevo estado. Estos estados se almacenan como objetos y especifican cómo cambia el estado de una aplicación en respuesta a una acción enviada al store.
- Como funciones puras, no cambian los datos del objeto que se les pasa ni realizan ningún efecto secundario en la aplicación. Dado el mismo objeto, siempre deben producir el mismo resultado.

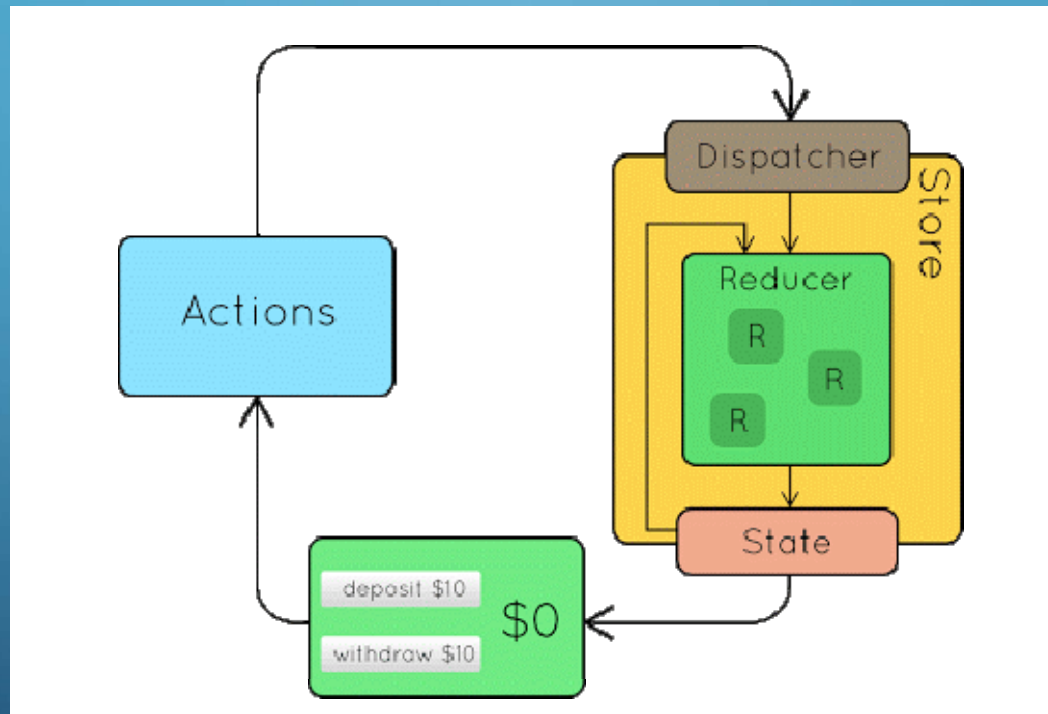
```
export default function theme(state = "darkblue", action) {  
  switch (action.type) {  
    case "CHANGE_THEME":  
      return action.payload;  
    default:  
      return state;  
  }  
}
```

3. ACTIONS IN REDUX

- En pocas palabras, las acciones son eventos. Son la única forma en que podemos enviar datos desde nuestra aplicación al Redux store. Los datos pueden ser de interacciones del usuario, llamadas a la API o incluso envíos de formularios.
- Las actions se envían utilizando el método `store.dispatch()`. Las acciones son objetos JavaScript simples, y deben tener una propiedad `type` para indicar el tipo de acción que se llevará a cabo. También deben tener un `payload` que contenga la información sobre la que debería trabajar la acción. Las actions se crean a través de un `action creator`.

- Aquí creamos una acción que permitirá cambiar el theme en una aplicación:

```
export default function changeTheme(theme) {  
  return {  
    type: "CHANGE_THEME",  
    payload: theme  
  };  
}
```

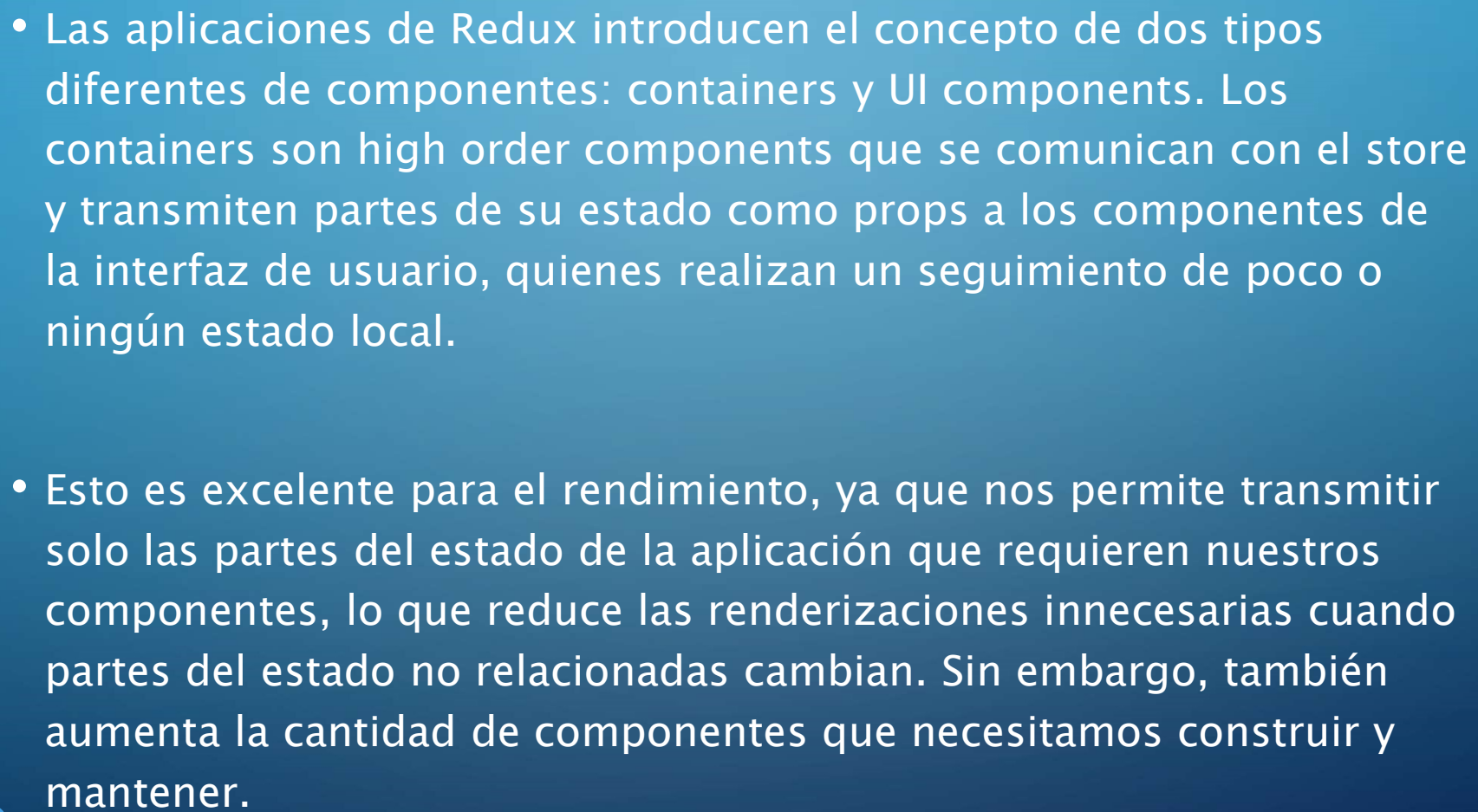


RESUMIENDO

- Al usar Redux con React, los estados ya no necesitarán ser elevados; por lo tanto, le resulta más fácil rastrear qué acción causa algún cambio. Como se vio anteriormente, el componente no necesita proporcionar ningún estado o método para que sus componentes hijos compartan datos entre ellos. Todo lo maneja Redux. Esto simplifica enormemente la aplicación y facilita su mantenimiento.


OTROS BENEFICIOS DE USAR REDUX

- Redux hace que el estado sea predecible.
- Mantenibilidad
- Debugging.
- Facilidad para el testing
- Persistencia del estado.
- Server-side rendering

- 
- The background of the slide is a solid blue color. It is decorated with white, stylized circuit lines that resemble a printed circuit board (PCB) layout. These lines are located along the left and right edges of the slide, with some lines extending slightly into the main content area. The lines consist of straight segments connected by small circles, creating a network-like pattern.
- Las aplicaciones de Redux introducen el concepto de dos tipos diferentes de componentes: containers y UI components. Los containers son high order components que se comunican con el store y transmiten partes de su estado como props a los componentes de la interfaz de usuario, quienes realizan un seguimiento de poco o ningún estado local.
 - Esto es excelente para el rendimiento, ya que nos permite transmitir solo las partes del estado de la aplicación que requieren nuestros componentes, lo que reduce las renderizaciones innecesarias cuando partes del estado no relacionadas cambian. Sin embargo, también aumenta la cantidad de componentes que necesitamos construir y mantener.

LOS TRES PRINCIPIOS DE REDUX:

1. El estado es la única fuente de verdad para la aplicación, representada como un único árbol de objetos en un solo store.
2. El estado es de solo lectura y solo puede modificarse emitiendo una acción que describa el cambio a realizar.
3. Los cambios se realizan utilizando funciones puras que devuelven el siguiente estado. El estado actual es nunca mutado.





Por supuesto, Redux no es la única forma de proporcionar datos a sus componentes. La context API de React hace básicamente lo mismo; le permite pasar el estado global por el árbol de componentes sin pasarlo por props en todos los niveles. Pero eso es todo lo que hace.

Por el contrario, Redux proporciona un conjunto completo de herramientas para administrar el estado:

- Viene con un time traveling debugger
- Proporciona una API de middleware, que le da acceso a herramientas como `redux-sagas`
- Permite evitar muchos renders innecesarios

Como podemos ver, el context no es un reemplazo para Redux. Dependiendo de nuestras necesidades, Redux podría ser una excelente opción.



REDUX VS. CONTEXT

La gestión de estado de React es bastante simple: llamamos a `setState` y React se vuelve a renderizar.

Con Redux hay algunos pasos involucrados:

- El usuario escribe en el input box
- Se llama al creador de acciones para obtener una acción
- Dispatch del action a Redux
- Redux inserta la acción en el root reducer
- El root reducer delega esa acción al reducer correcto
- El reducer devuelve un nuevo estado dado el estado anterior y el action object
- Ese nuevo estado se convierte en el estado del store.
- Redux llama a React y le dice que actualice.

REDUX VS. CONTEXT

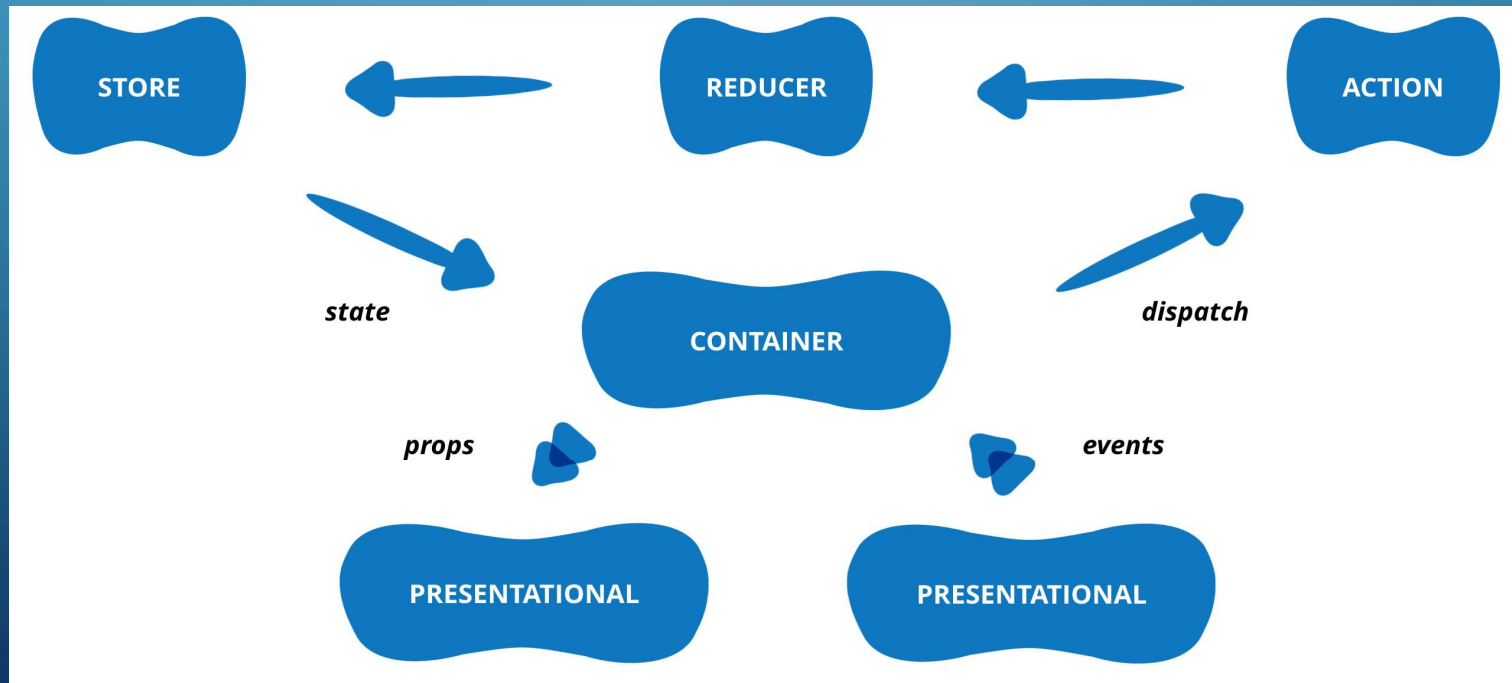
Entonces, lo que era un paso se convirtió en varios. Pero cada paso de estos es comprobable, y eso es genial.

Dado que ahora tenemos la nueva context API, ¿con qué frecuencia usaremos Redux?

- Para actualizaciones de baja frecuencia como cambios de plantilla o autenticación de usuarios, React Context estaría bien.
- Cuando tenemos estados complejos con actualizaciones de alta frecuencia, React Context no será una buena solución, porque activará una nueva renderización en cada actualización, y optimizarlo manualmente puede ser realmente difícil.
Si tenemos orquestaciones complejas de datos asíncronos, necesitamos hacer debugging en aplicaciones complejas o necesitamos optimizar la performance, Redux puede ser inmensamente útil.

Un componente container es un componente responsable de recuperar datos, y para obtener esos datos, necesita usar las funciones `connect` y `mapStateToProps` de Redux. Este componente tomará datos del estado a través de `mapStateToProps`. Y luego pasará las porciones necesarias de esos datos a sus hijos como props. También es responsable de hacer `dispatch` de actions que realizan cambios en el estado de la aplicación.

Un componente de presentación simplemente recibe esos datos de su contenedor padre y los muestra.



INSTALACIÓN DE REDUX:

`npm i redux react-redux`