educación IT
CODE YOUR FUTURE

ReactJS Developer

Módulo 7

educación IT
CODE YOUR FUTURE

Introducción a *Redux*

Redux

Redux es **un contenedor predecible del estado de aplicaciones JavaScript.**

Te ayudará a **escribir aplicaciones que se comportan de manera consistente, corren en distintos ambientes** (cliente, servidor y nativo) y **son fáciles de probar.** Además de eso, provee una gran experiencia de desarrollo, gracias a la edición en vivo combinada con un depurador sobre una línea de tiempo.

Puedes usar *Redux* junto con React o cualquier otra librería de vistas. Es muy **pequeño** (2 kB) y **no tiene dependencias.**

Instalación

Para instalar *Redux*, vamos a necesitar **una librería de base para su propio código** y **una librería extra en caso que queramos los bindings con una aplicación React:**

```
npm install redux react-redux
```

Empezando

Todo el estado de tu aplicación está almacenado en **un solo árbol dentro de un *store*.** La única forma para cambiar el árbol de estado es **emitiendo una acción, un objeto que describa qué ocurrió.**

Para especificar cómo las acciones transforman el árbol de estado, **usas *reducers* puros que son funciones.**

```
import { createStore } from 'redux';
function counter(state = 0, action) {
  switch (action.type) {
  case 'INCREMENT':
    return state + 1;
  case 'DECREMENT':
    return state - 1;
  default:
    return state;
  }
}

// Creamos un store de Redux almacenando el estado de la aplicación.
// Su API es { subscribe, dispatch, getState }.
let store = createStore(counter);

// Puedes suscribirte manualmente a los cambios, o conectar tu vista
// directamente
store.subscribe(() => {
  console.log(store.getState())
});

// La única forma de modificar el estado interno es despachando acciones.
// Las acciones pueden ser serializadas, registradas o almacenadas luego para
// volver a ejecutarlas.
store.dispatch({ type: 'INCREMENT' });
// 1
store.dispatch({ type: 'INCREMENT' });
// 2
store.dispatch({ type: 'DECREMENT' });
//1
```

Cuando tu aplicación crezca, en vez de agregar más *stores*, **divides tu *reducer* principal en varios *reducers* pequeños que operan de forma independiente en distintas partes del árbol de estado.** Esto es como si sólo hubiese un componente principal en una aplicación de React pero esta estuviese compuesta de muchos componentes pequeños.

Esta arquitectura puede parecer una exageración para una aplicación de contador, pero lo útil de este patrón es **lo bien que escala en aplicaciones grandes y complejas.**

También permite diseñar herramientas de desarrollo muy poderosas, ya que es posible registrar cada modificación que causan las acciones: podrías registrar la sesión de un usuario y reproducirlas ejecutando las mismas acciones.

Tres principios

Redux puede ser descrito en **tres principios fundamentales:**

- **Centralización del estado**
- **Estado de solo lectura**
- **Funciones puras para modificar el estado**

Centralización del estado

El estado de toda tu aplicación está almacenado **en un árbol guardado en un único *store*.**

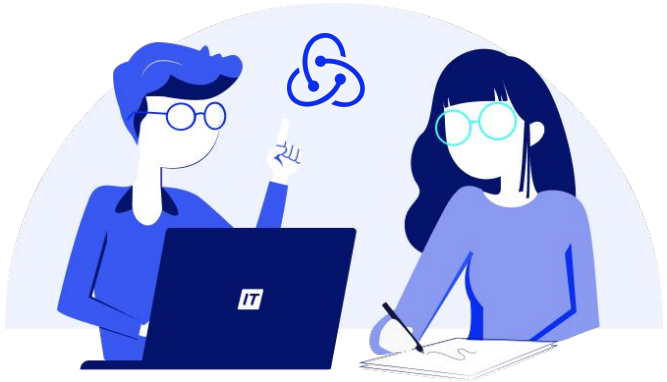
Esto simplifica la tarea de crear **aplicaciones universales,** ya que el estado en tu servidor puede ser serializado y enviado al cliente sin ningún esfuerzo extra. Además, contar con un único árbol de estado hace más fácil **depurar una aplicación** y facilita el mantenimiento del **estado de la aplicación en desarrollo,** para un ciclo de desarrollo más veloz.

Algunas funcionalidades que han sido difíciles de implementar —como *deshacer/rehacer*, por ejemplo— se vuelven triviales si todo tu estado se guarda en un único árbol.

Estado de *solo lectura*

La única forma de modificar el estado es **emitiendo una acción, un objeto que describa qué ocurrió.** Esto te asegura que ni tu *vista* ni *callbacks* de red vayan a modificar el estado directamente. En vez de eso, expresarán **un intento de modificar el estado.**

Ya que todas las modificaciones están centralizadas y suceden en un orden estricto, no hay que preocuparse por una carrera entre las acciones, y como las acciones son objetos planos, pueden ser **registrados, serializados y almacenados** para volver a ejecutarlos por cuestiones de depuración y pruebas.



Funciones puras para modificar el estado

Para especificar cómo el árbol de estado es transformado por las acciones, se utilizan *reducers* puros.

Los reducers son **funciones puras que toman el estado anterior y una acción** y devuelven un **nuevo estado.** Recuerda devolver ***un nuevo objeto de estado en vez de modificar el anterior.***

Puedes comenzar con **un único *reducer*** y, mientras tu aplicación crece, **dividirlo en varios *reducers* más pequeños que manejan partes específicas del árbol de estado.**

Ya que los *reducers* son funciones puras, puedes controlar **el orden en que se ejecutan, pasarle datos adicionales o incluso hacer *reducers* reusables para tareas comunes como, por ejemplo, paginación.**

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!