educación IT
CODE YOUR FUTURE

ReactJS Developer

Módulo 7

educación IT
CODE YOUR FUTURE

Componentes de *Redux*

Acciones

Las **acciones** son **un bloque de información que envía datos desde tu aplicación a tu *store*.** Son la única fuente de información para el ***store*** y las envías usando el comando **`store.dispatch()`.**

A continuación, tenemos un ejemplo de acciones que representan “agregar nuevas tareas pendientes”:

```
const ADD_ITEM = {
    type: 'ADD_ITEM',
    payload: "Terminar módulo Cobros"
}
```

Las *acciones* son **objetos planos de JavaScript.** Una acción debe tener una propiedad **type** que indique **el tipo de acción a realizar.**

Los tipos normalmente son definidos como *strings* constantes.

Una vez que tu aplicación sea suficientemente grande, quizás quieras moverlos a un módulo separado.

Además del `type`, **el resto de la estructura de los objetos de acciones depende de ti.** Vamos a agregar una acción más para describir a un usuario marcando una tarea como completa.

Nos referimos a **una tarea en particular** como su **índex,** ya que vamos a almacenarlos en un `array`. En una aplicación real, es mejor generar **un ID único cada vez que creamos una nueva:**

```
const SET_COMPLETE = {
    type: 'SET_COMPLETE',
    payload: 5
}
```

Es una buena idea pasar la menor cantidad de información posible. Por ejemplo, es preferible pasar el índex que todo el objeto de tarea.

Por último, vamos a agregar una acción más para cambiar las tareas actualmente visibles:

```
const FILTER_TASK = {
    type: 'FILTER_TASK',
    payload: 'SHOW_COMPLETED'
}
```

Creadores de acciones

Los creadores de acciones son exactamente eso: **funciones que crean acciones.** Es fácil combinar los términos "acción" con "creador de acción", así que haz lo mejor por usar los términos correctos. Esto las hace más portables y fáciles de probar.

En *Redux*, **los creadores de acciones** son funciones que simplemente **regresan una acción.**

```
const FILTER_TASK = (filter) => ({
    type: 'FILTER_TASK',
    payload: filter
});
```

Despachar una acción

Para efectivamente iniciar un despacho,
pasa la acción a la función `dispatch()`:

```
1   const filterTask = (filter) => ({
2       type: 'FILTER_TASK',
3       payload: filter
4   });
5
6   const SHOW_COMPLETE = {
7       type: 'SHOW_COMPLETE'
8   }
9
10  dispatch(filterTask('COMPLETE')); // Creador de acciones
11  dispatch(SHOW_COMPLETE); // Acción normal
12
```

Puedes acceder a la función **`dispatch()`** en el *store* como **`store.dispatch()`**, pero comúnmente usarás utilidades como **`connect()`** de React-Redux. También puedes usar **`bindActionCreators()`** para conectar automáticamente muchos creadores de acciones a **`dispatch()`**.

Además, los creadores de acciones pueden ser **asíncronos** y **tener efectos secundarios.**

En los próximos módulos aprenderemos a manejar respuestas AJAX y combinar creadores de acciones en un flujo de control asincrónico. Pero vamos paso a paso: es muy importante tener bien claras las bases antes de pasar a casos más complejos.

Reducers

El **reducer** es una función pura que **toma el estado anterior junto a una acción y devuelve en nuevo estado.**

```
(prevState, action) => newState
```

Se llama *reducer* porque es el tipo de función que pasarías a `Array.prototype.reduce (reducer, ?initialValue)`.

Es muy importante que los reducer se mantengan puros.

Estas son algunas cosas que **nunca deberías hacer** dentro de un `reducer`:

- Modificar sus argumentos.
- Realizar tareas con efectos secundarios, como llamar a un *API* o transiciones de rutas.
- Llamar una función no pura, por ejemplo `Date.now()` o `Math.random()`.

Entonces, dados los mismos argumentos, debería calcular y devolver el siguiente estado. Sin sorpresas, efectos secundarios, llamadas a *APIs* ni mutaciones, **solo cálculos.**

Dicho esto, empezaremos a escribir nuestro `reducer` gradualmente enseñándole **cómo entender las acciones que definimos antes:**

Empezaremos por **especificar el estado inicial.** *Redux* va a llamar a nuestros *reducers* con **`undefined` como valor del estado la primera vez.** Esta es nuestra oportunidad de devolver **el estado inicial de nuestra aplicación.** Un gran truco es **usar la sintaxis de parámetros por defecto de ES6 para hacer lo anterior de forma más compacta.** Encontrarás un ejemplo en la próxima slide.

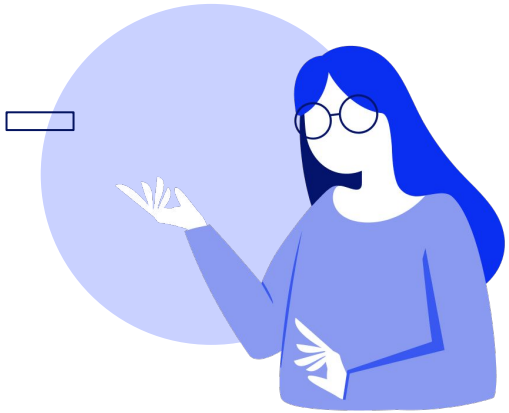
```
const tasksReducer = (prevState, action) => {
    switch(action.type) {
        case 'SHOW_COMPLETE':
            return {
                ...prevState,
                filtered: prevState.tasks.filter(s => s.completed)
            };

        case 'SHOW_ALL':
            return {
                ...prevState,
                filtered: tasks
            };

        default:
            return prevState;
    }
}
```

Nota que:

1. **No modificamos el state:** creamos una copia con el operador `spread { ...state, ...newState }`.
2. **Devolvemos el anterior state en el caso default:** es importante devolver el anterior state por cualquier acción desconocida.



Store

Anteriormente, definimos que las **acciones representan los hechos sobre "lo que pasó"** mientras que los **reductores** son los que **actualizan el estado de acuerdo a esas acciones.**

El ***Store*** es **el objeto que los reúne** y tiene las siguientes responsabilidades:

- Contiene **el estado de la aplicación.**
- Permite **el acceso al estado vía `getState()`.**
- Permite que el estado **sea actualizado a través de `dispatch(action)`.**
- Registra los *listeners* vía **`subscribe(listener)`.**
- Maneja **la anulación del registro de los *listeners*** mediante el retorno de la función de **`subscribe(listener)`.**

Es importante destacar que sólo tendrás un ***store*** en una aplicación Redux. Cuando desees dividir la lógica para el manejo de datos, usarás **composición de reductores en lugar de muchos *stores*.**

Es fácil crear un ***store*** si tienes un reductor. En la sección anterior, usamos **`combineReducers()`** para **combinar varios reductores en uno solo.** Ahora lo vamos a importar y pasarlo a **`createStore()`*:***

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp)
```

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!