Introducción a Reducer

En esta sección, se explicará el uso del término "reducer" en React, una comprensión fundamental antes de implementarlo en la aplicación **CartApp**.

¿Qué es Reducer?

Un reducer es una función en JavaScript que recibe dos argumentos: el estado actual de la aplicación y una acción. Una acción es un objeto que contiene dos partes: el tipo de acción (por ejemplo, agregar, editar, actualizar) y los parámetros necesarios (**payload**).

**Ejemplo:**

* **Acción:** agregar un producto.
* **Payload:** el objeto producto que se va a agregar.

Características de un Reducer

* **Síncrono:** No se pueden realizar tareas asíncronas dentro de un reducer; es 100% síncrono.
* **Inmutable:** Un reducer no debe mutar el estado anterior, sino que debe crear un nuevo estado a partir del original.
* **Independiente:** No deben tener dependencias con otras funciones o librerías externas.
* **Acciones:** Debe manejar una o más acciones como agregar, actualizar o eliminar.

Implementación básica

En una función reducer, el nombre debe ser significativo y relacionado con la naturaleza de los datos o el estado que está gestionando. No se debe elegir cualquier nombre; es importante seguir una nomenclatura estándar que facilite la comprensión y el mantenimiento del código.

Por ejemplo, si se está trabajando con productos, la función reducer debe tener un nombre relacionado con los productos, como **productReducer**. El sufijo **"Reducer"** es una convención que indica claramente que la función es un reducer.

Esta función recibe dos argumentos: el estado (**initialState**) y la acción (**action**) a ejecutar:

export const initialState = [

{

id: 1,

name: 'Teclado Mecanico RGB',

description: 'Teclado Mecanico con luces RGB switches cherry red',

price: 1000

},

{

id: 2,

name: 'Samsung Smart TV 55',

description: 'Excelente tv LCD...',

price: 3000

}

];

export const productReducer = (state = initialState, action) => {

switch (action.type) {

case "ADD":

return [...state, action.payload];

case "UPDATE":

return state.map(product =>

product.id === action.payload.id ? { ...product,

...action.payload } : product

);

case "REMOVE":

return state.filter(product =>

product.id !== action.payload.id

);

default:

return state;

}

};

Primero, se define el estado inicial de la aplicación como un arreglo de productos. Cada producto es un objeto con las siguientes propiedades: **id**, **name**, **description**, y **price**. El arreglo **initialState** se usará como el estado inicial de la aplicación. Luego, se define la función **productReducer**, que recibe dos parámetros: **state** y **action**.

* **state:** Representa el estado actual de la aplicación. Si no se proporciona un estado, se usará **initialState** por defecto.
* **action:** Un objeto que describe la acción a realizar. Tiene una propiedad **type** y la propiedad **payload**.

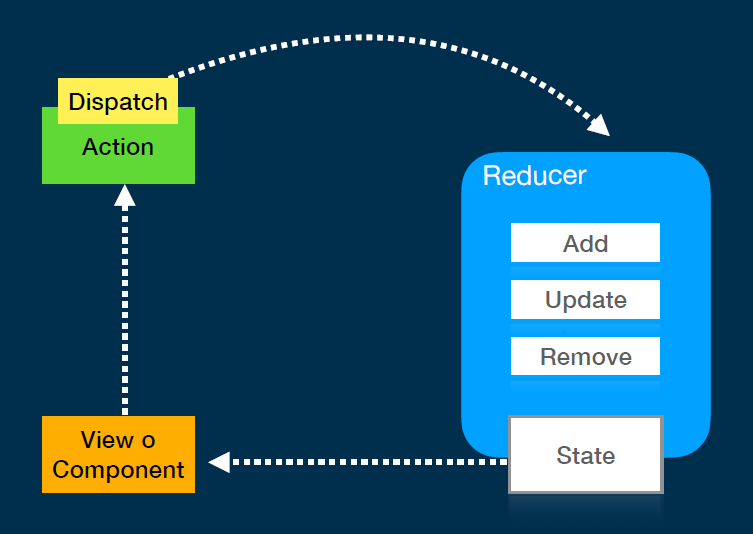
La función utiliza un switch para determinar qué acción ejecutar según el tipo (**type**) de la acción (**action**).

* **case "ADD"**: Este caso se activa cuando la acción tiene el tipo **"ADD"**. Se devuelve un nuevo estado que incluye todos los productos actuales (**state**) más el nuevo producto (**action.payload**). El uso del operador de propagación (**...state**) asegura que el estado original no se muta, manteniendo la inmutabilidad del estado.
* **case "UPDATE":** Este caso se activa cuando la acción tiene el tipo **"UPDATE"**. Se devuelve un nuevo estado donde se actualiza el producto cuyo **id** coincide con **action.payload.id**. El operador **map** se usa para crear un nuevo arreglo de productos, y el producto correspondiente se actualiza con los nuevos valores proporcionados en **action.payload**.
* **case "REMOVE":** Este caso se activa cuando la acción tiene el tipo **"REMOVE"**. Se devuelve un nuevo estado que excluye el producto cuyo **id** coincide con **action.payload.id**. El operador **filter** se usa para crear un nuevo arreglo sin el producto a eliminar.
* **default:** Si el tipo de acción no coincide con ninguno de los casos anteriores, se devuelve el estado actual sin cambios. Esto asegura que el reducer siempre devuelva un estado válido.

Ventajas de usar reducers

* **Centralización:** Centraliza la lógica de manejo de estado en un solo lugar, haciendo la aplicación más ordenada y mantenible.
* **Desacoplamiento:** Los componentes no manejan directamente el estado, sino que envían acciones al reducer.
* **Escalabilidad:** Facilita el manejo de múltiples tipos de acciones y estados complejos.

Ciclo de vida del estado



Se tiene en cuenta lo siguiente:

1. **Estado Inicial:** Se define un estado inicial que contiene los datos necesarios.
2. **Renderizado del Componente:** El estado inicial se pasa al componente, que lo renderiza.
3. **Acciones:** El componente, mediante interacciones del usuario, dispara acciones como agregar, actualizar o eliminar.
4. **Dispatch:** Las acciones se envían al reducer mediante **dispatch**.
5. **Reducer:** El reducer actualiza el estado de forma inmutable según el tipo de acción y retorna un nuevo estado.
6. **Re-renderizado:** El componente se vuelve a renderizar con el nuevo estado.

Reducers en Frameworks y Librerías

El concepto de reducers no solo se aplica en React, sino que también es fundamental en otras librerías y frameworks como Redux y el contexto de React (**useContext**). Estos utilizan reducers para manejar el estado de manera eficiente y escalable.

* **Redux:** Utiliza un store centralizado que maneja el estado de la aplicación con reducers.
* **React Context:** Utiliza reducers para manejar el estado dentro de un contexto compartido entre componentes.

En conclusión, el uso de reducers en React es una práctica poderosa para manejar el estado de la aplicación de manera centralizada, inmutable y eficiente. Facilita el mantenimiento y escalabilidad de la aplicación, haciendo más fácil la colaboración en equipos de desarrollo.

Enlaces de referencias

La siguiente documentación de React que explica el uso de reducer y el hook **useReducer** de manera detallada, el cual se aplicará en la siguiente sección.

* <https://es.react.dev/learn/extracting-state-logic-into-a-reducer>
* <https://es.react.dev/reference/react/useReducer#writing-the-reducer-function>
* <https://es.react.dev/reference/react/useReducer>
* <https://es.react.dev/learn/extracting-state-logic-into-a-reducer#comparing-usestate-and-usereducer>