Unidad 3

Gestión de Estado con Redux

Unidad 3.1

Introducción a Redux

Concepto clave de Redux

Redux es una librería de JavaScript para gestionar el estado global de una aplicación, ideal para aplicaciones con grandes volúmenes de datos o que requieren manejar el estado en múltiples componentes

UTE EC | Xtratego ACADEMY

Principios fundamentales de Redux

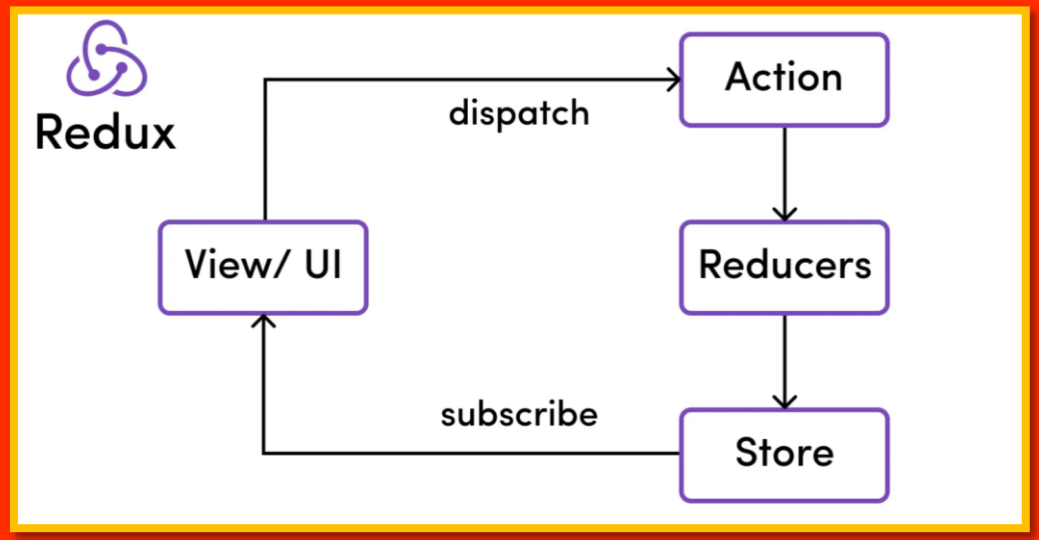
Single Source of Truth (Una sola fuente de la verdad): El estado de toda la aplicación se almacena en un único objeto, denominado **store**.

Estado inmutable: El estado no se modifica directamente, sino que se crean nuevas versiones del estado mediante **acciones** y **reducers**.

Flujo de datos unidireccional: Los datos siempre fluyen en una única dirección dentro de la aplicación.

Beneficios de Redux

Ayuda a gestionar el **estado** de manera centralizada, permitiendo que múltiples componentes compartan datos de forma consistente



Unidad 3.2

Estructura de Redux

Store

Es el único lugar donde reside el estado de la aplicación.
Se crea utilizando **createStore()**, que toma como argumento un **reducer**

```
import { createStore } from 'redux';

const initialState = { count: 0 };

function counterReducer(state = initialState, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    default:
      return state;
  }
}
const store = createStore(counterReducer);
```

Actions (Acciones)

Son objetos simples que describen qué ocurrió en la aplicación.

Tienen al menos una propiedad **type**, que indica el tipo de acción, y pueden incluir datos adicionales

```
const incrementAction = {
  type: 'INCREMENT'
};
```

Reducers

Son funciones puras que reciben el estado actual y una acción, y devuelven un nuevo estado.
El **reducer** es responsable de definir cómo cambia el estado en respuesta a las acciones.

```
function counterReducer(state = initialState, action) {
  switch (action.type) {
    case 'INCREMENT':
      return { count: state.count + 1 };
    case 'DECREMENT':
      return { count: state.count - 1 };
    default:
      return state;
  }
```

UTE EC | Xtratego ACADEMY

Unidad 3.3

Flujo de datos en Redux

Dispatching actions

Las acciones se envían **(dispatch)** para notificar que algo ha ocurrido en la aplicación.
Se utilizan métodos como **store.dispatch()** para enviar acciones.

```
store.dispatch({ type: 'INCREMENT' });
```

Actualización del estado con reducers

Cuando una acción es enviada, el **reducer** procesa esa acción y devuelve un nuevo estado.

Redux garantiza que el flujo de datos sea predecible y que las actualizaciones ocurran de manera controlada

UTE EC | Xtratego ACADEMY

Subscription

Los **componentes** pueden **suscribirse** a la store para ser **notificados** cuando haya un cambio en el estado. Los **suscriptores** se actualizan cuando el **estado** cambia

```
store.subscribe(() => console.log(store.getState()));
```

Unidad 3.4

Integración de Redux con React

Para integrar Redux con React, se utiliza la biblioteca react-redux, que proporciona un conjunto de herramientas para conectar React con Redux de manera sencilla

```
npm install react-redux
```

Provider

El componente **<Provider>** es utilizado para envolver la aplicación **React**, permitiendo que los componentes accedan al store de **Redux**.

```
import { Provider } from 'react-redux';
import { store } from './store';

function App() {
  return (
    <Provider store={store}>
      <MyComponent />
    </Provider>
  );
}
```

Conexión de componentes con Redux (Hooks)

Se utilizan los **hooks** useSelector para leer el estado desde la store y useDispatch para enviar acciones.

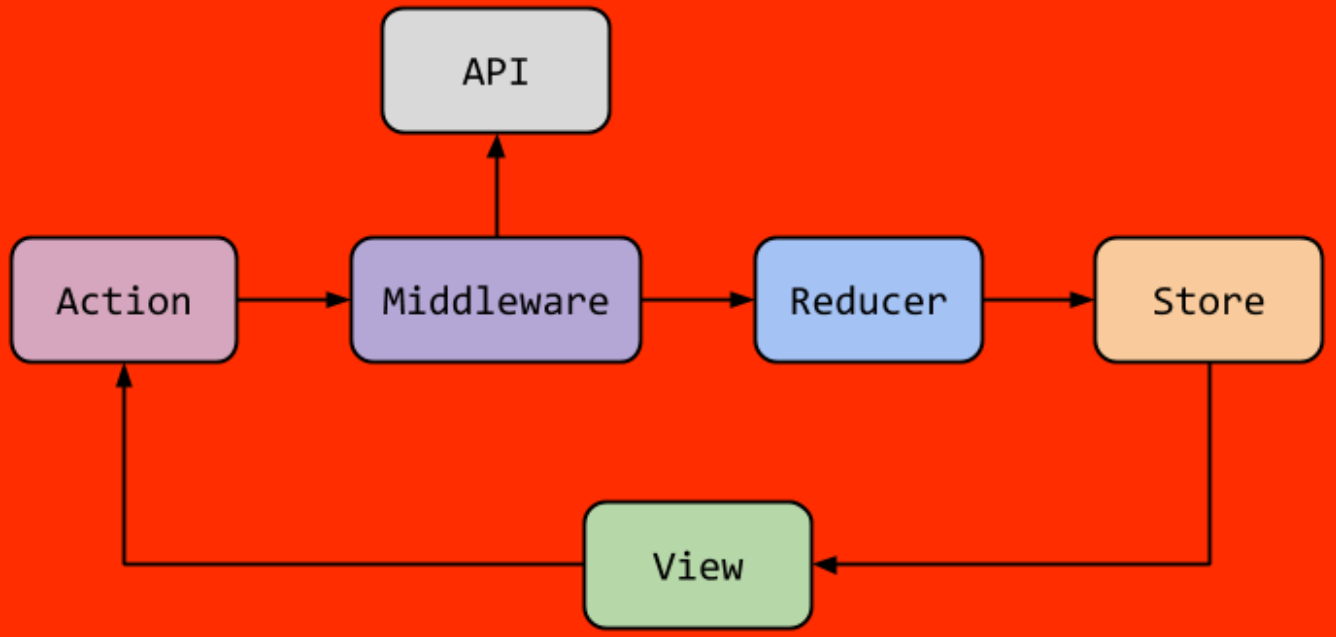
```jsx
import { useSelector, useDispatch } from 'react-redux';

function Counter() {
  const count = useSelector((state) => state.count);
  const dispatch = useDispatch();

  return (
    <div>
      <p>Contador: {count}</p>
      <button onClick={() => dispatch({ type: 'INCREMENT' })}>Incrementar</button>
      <button onClick={() => dispatch({ type: 'DECREMENT' })}>Decrementar</button>
    </div>
  );
}
```

Unidad 3.5

Middleware en Redux

Middleware

Los **middleware en Redux** interceptan las acciones antes de que lleguen al **reducer**, lo que permite realizar tareas adicionales como llamadas a **APIs** o manejar acciones **asíncronas**.



Redux Thunk

redux-thunk es un **middleware** que permite crear acciones **asíncronas**. Se utiliza para manejar **operaciones** que requieren esperar datos, como consultas a una API

```
npm install redux-thunk
```

UTE EC | Xtratego ACADEMY

Ejemplo de acción asíncrona con Thunk

```
function fetchData() {
  return function(dispatch) {
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => {
        dispatch({ type: 'FETCH_SUCCESS', payload: data });
      });
  };
}
```

Unidad 3.6

Herramientas para Depuración

Redux DevTools

Redux DevTools es una extensión de navegador que permite monitorear las acciones y el estado de la aplicación en tiempo real.
Proporciona una visualización clara del flujo de datos y facilita la depuración de errores.

```
const store = createStore(
  rootReducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__()
);
```

Unidad 3.7

Ejercicio Práctico

Ejercicio: Contador global con Redux

Crear una aplicación que maneje el estado de un contador global utilizando **Redux**.

Utilizar **react-redux** para conectar la aplicación **React** con **Redux**.

Añadir botones para incrementar y decrementar el contador y mostrar el valor actualizado en todos los componentes que lo necesiten.

Extensión: Utilizar **Redux Thunk** para simular una operación asíncrona que actualice el **contador**.