UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Diplomatura en programación web full stack con React JS

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 5: React Inicial

Unidad 3: Redux

Presentación:

Redux es un patrón de arquitectura que permite persistir datos en todo el proyecto ReactJS, reduciendo la cantidad de relaciones entre los componentes. Es fundamental conocerlo y manejarlo al trabajar profesionalmente en ReactJS

Objetivos:

Que los participantes:

- Comprendan el concepto de Redux.

- Sean capaces de implementar redux en un proyecto React.

Bloques temáticos:

1. Concepto de redux
2. Ventajas del uso de redux
3. Uso de redux en un proyecto React
4. Implementación de redux en React
5. Ejemplo
6. Trabajo Práctico

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***

Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. Concepto de redux

Redux es un patrón de arquitectura de datos que permite manejar el estado de la aplicación de una manera predecible. Está pensado para reducir el número de relaciones entre componentes de la aplicación y mantener un flujo de datos sencillo.

Redux se encarga en cierta manera de desacoplar el estado global de una aplicación web (en Front-End) de la parte visual, es decir los componentes. Permitiendo manejar un estado global, en vez de un estado por cada componente.

2. Ventajas del uso de redux

Two way binding

El enlace bidireccional o de 2 vías referencia la posibilidad de recuperar un valor desde el redux y también el poder modificarlo, alterando así el contenido del redux. Resumidamente, redux permite recuperar y alterar el valor del estado general desde los componentes de la aplicación.

Un punto **importante** de este enlace bidireccional es que crea un "canal" de comunicación entre el componente y el redux por lo que cada vez que el store sea alterado por algún componente, todos los demás componentes enlazados van a ser notificados de esa modificación a fin de que tengan siempre el estado actualizado.

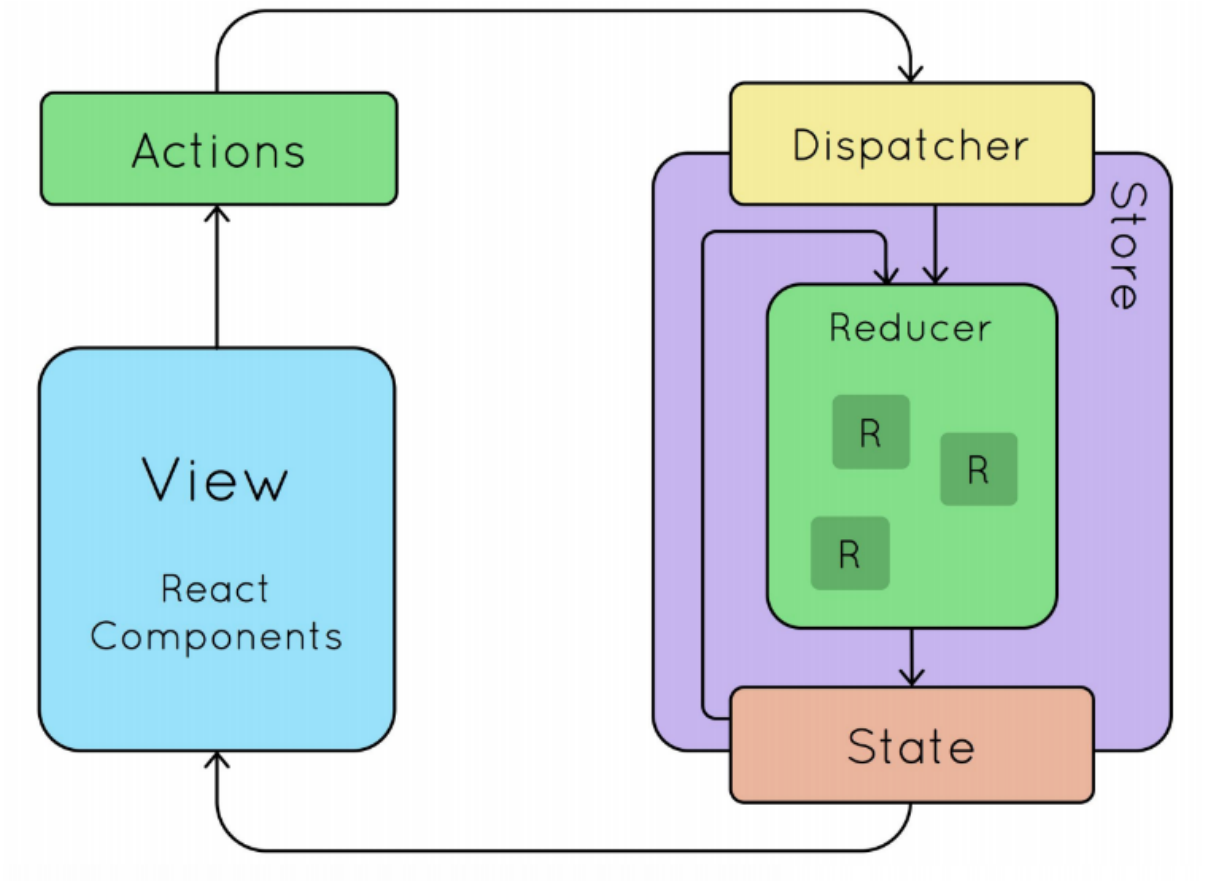
```
import React, { useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';

export default function Contador() {
  const contador = useSelector((state) => state.numero);
  const dispatch = useDispatch();
  return (
    <div>
      <h1>Contador</h1>
      <h3>{contador}</h3>
      <div>
        <button onClick={() =>
                dispatch({ type: 'INCREMENTAR' })}>+</button>
        <button onClick={() =>
                dispatch({ type: 'DECREMENTAR' })}>-</button>
        <button onClick={() =>
                dispatch({ type: 'RESET' })}>Volver a 0</button>
      </div>
    </div>
  );
}
```

En este ejemplo, la línea `<h3>{contador}</h3>` muestra el contenido de contador que es una variable que se configuró ligada al state del redux mediante el código `const contador = useSelector((state) => state.numero);` (una de las vías indicadas anteriormente, la posibilidad obtener el valor del estado). Luego dentro del return en cada botón (button) se enlaza el evento onClick con el dispatch que se encargará de gestionar las solicitudes de modificación del estado del redux (la segunda vía, la modificación del estado del redux)

Manejo de estado

Cuando la aplicación comienza a ser más grande y compleja, cada vez es más difícil poder manejar el estado

Redux es un paquete que nos facilita el manejo del estado en React



Principios

- Todo el estado de la aplicación está guardado es un STORE
- El estado es de solo lectura (solo se puede actualizar por medio de acciones)
- Los cambios de estados se realizar en funciones (REDUCERS)

Estado guardado en un STORE

Permite mantener todo el estado en un lugar
Permite persistir el estado en el servidor (opcional)
Permite restaurar un estado de la aplicación desde el servidor (opcional)

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

```
const estadoInicial = {
  numero: 0,
};

export default createStore(reducer);
```

El store de nuestro ejemplo de contador solo tiene una propiedad (número) pero un store puede almacenar muchas propiedades que pueden ser simples con el caso de número y también objetos y/o arrays.

export default createStore(reducer) crea efectivamente el store brindándole la función reducers como medio de modificación de su estado.

Actualización del estado

Para actualizar el estado debemos emitir (DISPATCH) una acción
Generalmente indicamos que acción deseamos realizar y los datos necesarios para realizar dicha acción

```
  import { useDispatch, useSelector } from 'react-redux';
.
.
.
  const dispatch = useDispatch();
.
.
.
     <div>
        <button onClick={() =>
                dispatch({ type: 'INCREMENTAR' })}>+</button>
        <button onClick={() =>
                dispatch({ type: 'DECREMENTAR' })}>-</button>
        <button onClick={() =>
                dispatch({ type: 'RESET' })}>Volver a 0</button>
      </div>
```

Los dispatch van en los componentes donde se desee modificar el estado del redux.
Se debe importar useDispatch del paquete react-redux y crearse un hooks para tal fin (const dispatch = useDispatch())

Reducers

Son las funciones que reciben:

- El estado anterior
- La acción a realizar
- Datos necesarios para realizar la acción

Y retornan el nuevo estado **completo**.

```
function reducer(state = estadoInicial, action) {
  switch (action.type) {
    case 'INCREMENTAR':
      return {
        numero: state.numero + 1,
      };
    case 'DECREMENTAR':
      return {
        numero: state.numero - 1,
      };
    case 'RESET':
          return {
              numero: 0
          }
    default:
      return state;
  }
}
```

Como se puede apreciar, el dispatch va a indicar una de las opciones que ofrece el

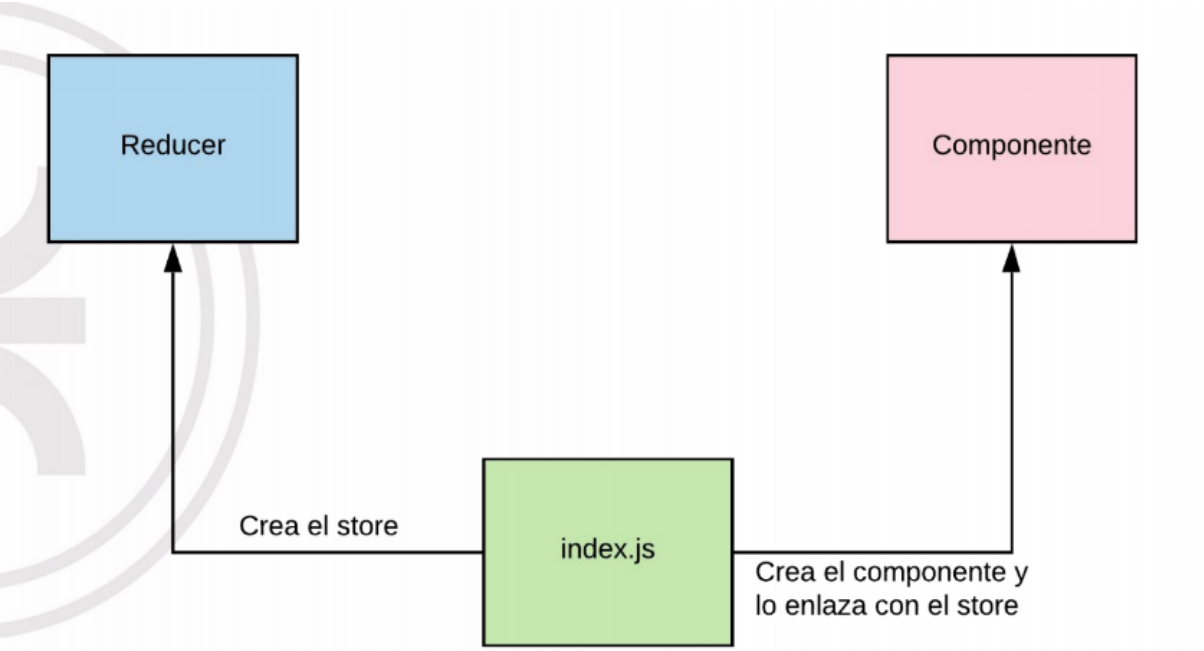
reducer para la modificación del estado (INCREMENTAR, DECREMENTAR, RESET).

Generalmente los reducers incluyen una estructura switch con un case por cada opcion de modificacion que ofrece el reducer.

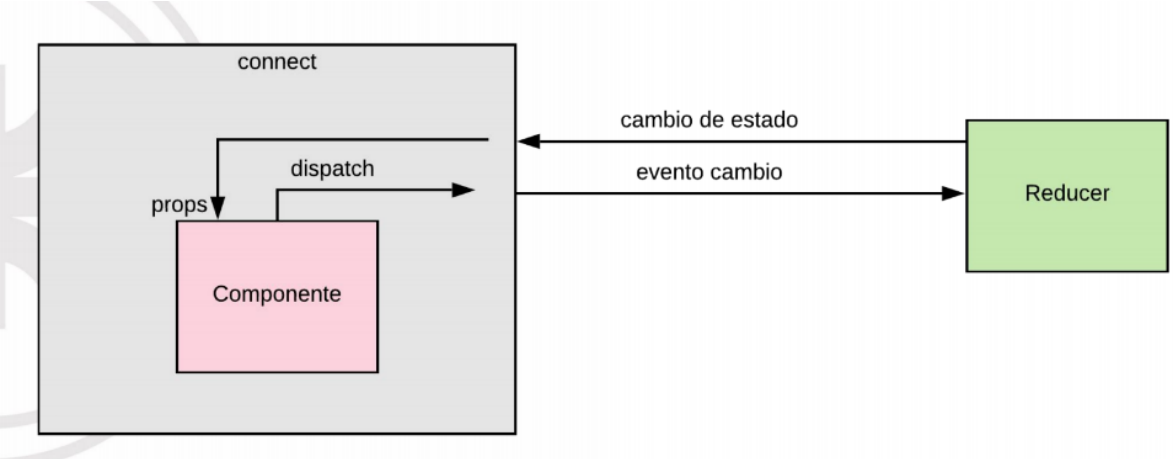
Como se mencionó anteriormente, no es posible modificar el valor del state de forma directa, es decir, no se debe hacer state.numero = state.numero + 1. Para modificar el state se debe “pisar” el contenido anterior, en este ejemplo cada case retorna hacia el redux un nuevo objeto {numero: 0} o {numero: state.numero + 1} o {numero: state.numero - 1}

3. Uso de redux en un proyecto React

Pasos en React



Comunicación componente - store



1. Se crea el store y los reducers
2. Se enlazan los componentes al redux
3. Mediante los dispatch se modifica el estado general

4. Implementación de redux en React

Paso 1: Instalación

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

Nota: No es necesario instalar redux ya que se instalará automaticamente al instalar el react-redux.

Paso 2: Crear El store

Es conveniente crear una carpeta específica para redux para mantener el código ordenado y predecible su ubicación. Ejemplo reducer

Dentro de esa carpeta se puede optar por mantener en archivos separados el store (store.js) con su correspondiente valor inicial y los reducers (action.js) pero tambien pueden quedar ambos dentro de un mismo archivo. Algunos desarrolladores eligen crear el store dentro del index.js

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

```
const estadoInicial = {
  numero: 0,
};
.
.
.
export default createStore(reducer);
```

Los 3 puntos uno debajo del otro indican que hay codigo allí que se está obviando.

Para la creación del store se debe importar createStore de redux (recuerde que no es necesario instalar específicamente este paquete porque instala automáticamente al instalar el react-redux)

Se suele configurar un valor inicial para el store (const estadoInicial…)

Paso 3: Crear reducer

La función reducer puede ser parte del mismo archivo del store. Es la función que indica los procedimientos permitidos para modificación del store. Siempre hay que recordar que no se debe modificar el estado directamente, se debe retornar otro nuevo estado que incluya los nuevos valores para todo el store.

```
function reducer(state = estadoInicial, action) {
  switch (action.type) {
    case 'INCREMENTAR':
      return {
        numero: state.numero + 1,
      };
    case 'DECREMENTAR':
      return {
```

```
        numero: state.numero - 1,
      };
    case 'RESET':
          return {
              numero: 0
          }
    default:
      return state;
  }
}
```

Como se mencionó anteriormente, el reducer suele contener una estructura condicional switch-case para cada una de las operaciones que se permita realizar sobre el store.

Ejemplo

Tomando que el estado actual es

```
{ numero: 10 }
```

Y se desea incrementar el número, se realizará un `dispatch({type: 'INCREMENTAR'})` lo que va a provocar que se llame a la función reducer con los siguientes parámetros

```
state = {numero: 10}
action = {type: 'INCREMENTAR'}
```

El switch entrará a la opción 'INCREMENTAR' la cual retorna un nuevo estado

```
{ numero: state.numero + 1}
```

Que, para el ejemplo, es lo mismo que

```
{numero: 11}
```

Paso 4: Componente inicial

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import store from './reducers/store';

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById('root'),
);
```

Como se aprecia, se debe “envolver” toda la aplicación dentro de la etiqueta <Provider …> a la cual se le pasa como parámetro el store. De esta manera redux quedará disponible para todos los componentes de la aplicación (luego cada componente que desee utilizarlo deberá incorporarlo pero si el Provider no envuelve a toda la aplicación esto no sería posible).

También es necesario importar el store que se pasara al Provider (import store from ‘./reducers/store’)

Paso 5: Utilización dentro de un componente

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

export default function Contador() {
  const contador = useSelector((state) => state.numero);
  const dispatch = useDispatch();
  return (
    <div>
      <h1>Contador</h1>
      <h3>{contador}</h3>
      <div>
        <button onClick={() =>
                dispatch({ type: 'INCREMENTAR' })}>+</button>
        <button onClick={() =>
                dispatch({ type: 'DECREMENTAR' })}>-</button>
        <button onClick={() =>
                dispatch({ type: 'RESET' })}>Volver a 0</button>
      </div>
    </div>
  );
}
```

Como hemos visto a lo largo de todo el material, se requiere la importación de los hooks de redux: import { useDispatch, useSelector} cabe aclarar que no es obligatoria la importación de ambos hooks, solo de los que sean necesarios. En nuestro ejemplo, estamos accediendo al valor del store (useSelector) y también modificandolo (useDispatch).

Merece destacarse la creación del contador:

```
const contador = useSelector((state) => state.numero);
```

En este caso, “contador” se enlaza a la propiedad numero del state (store). Si hubiera otras propiedades el procedimiento sería similar, pudiendo asociar diferentes variables a cada una de las propiedades del state que se requieran. Nuevamente, no es obligación enlazar con todas las propiedades, solo con las que van a ser necesarias para el componente en cuestión.

5. Ejemplo

Contador en React con Redux y Hooks

https://github.com/cursos-utn/react-redux-hooks

6. Trabajo Práctico

Implementar el contador del ejemplo como proyecto nuevo.

Bibliografía utilizada y sugerida

Azaustre, Carlos:

https://carlosazaustre.es/como-funciona-redux-conceptos-basicos

Desarrollo Web:

https://desarrolloweb.com/articulos/que-es-redux.html#:~:text=Redux%20es%20un%20patr%C3%B3n%20de,un%20flujo%20de%20datos%20sencillo.

Node JS (n.d.) Recuperado de. https://nodejs.org/es/

React JS (n.d.) Recuperado de. https://es.reactjs.org/

Wikipedia (n.d.) Recuperado de. https://es.wikipedia.org/wiki/React

Lo que vimos:

Redux.

Lo que viene:

Ciclo de vida de los componentes + conexión con el servidor