Xseed

Curso de React _ Technisys

Xseed

Clase 10 -

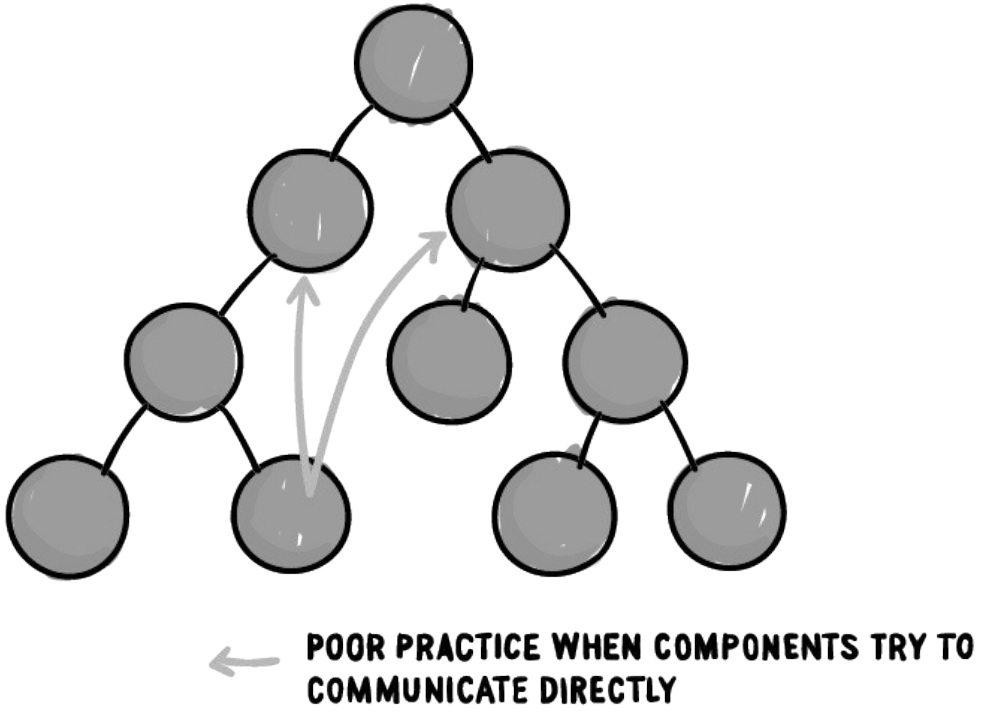
Redux Middleware

---

by Diego Cáceres

Repaso

React – Problema con el flujo de datos

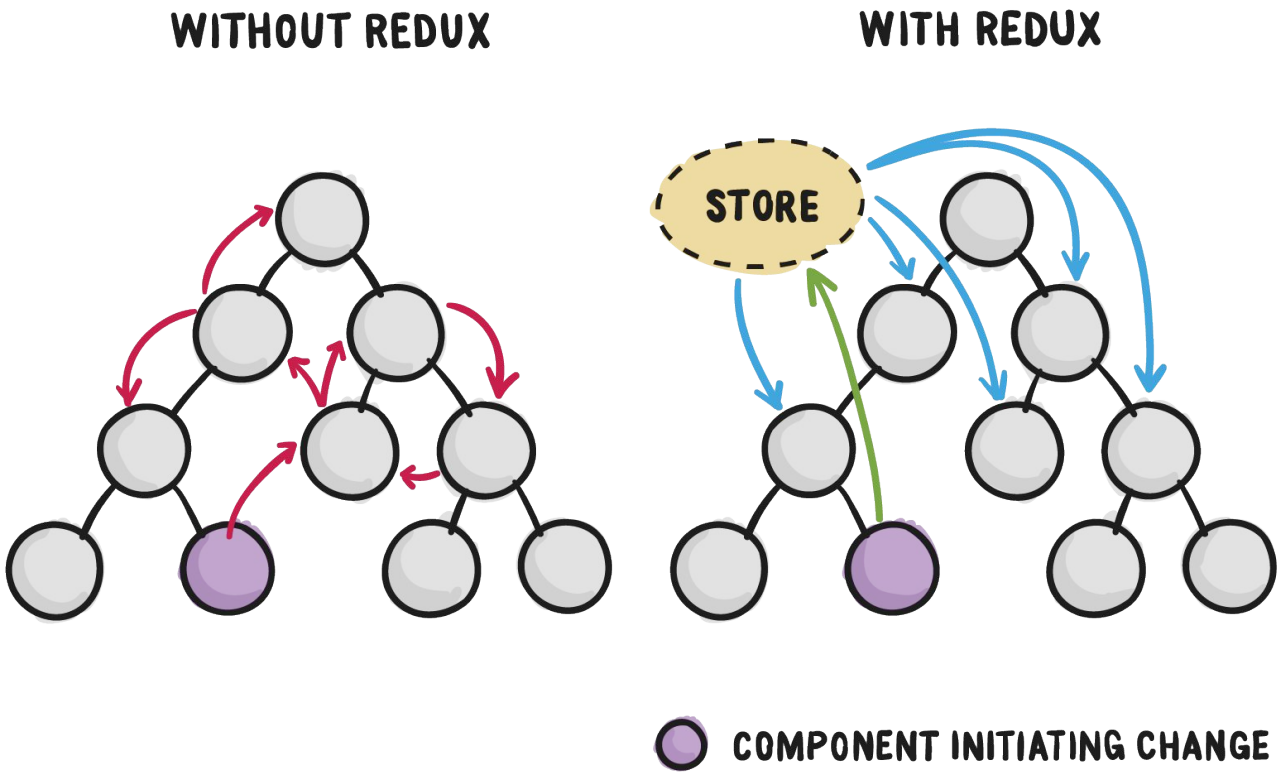


React – Problema con el flujo de datos

La solución que brinda Redux para esto es simple:

Una estructura única que guarda todo el estado de la aplicación en tiempo de ejecución, llamada Store.

React – Redux


WITHOUT REDUX
WITH REDUX
STORE
COMPONENT INITIATING CHANGE

Redux – Store's state

El Store es donde se guarda el `state` de la app entera, en un árbol de objetos.

```
todos: [
   { id: 1, name: 'LearnReact', isComplete: false },
   { id: 2, name: 'Learn Redux', isComplete: true },
   { id: 3, name: 'Learn ReactNative ', isComplete: false },
   { id: 4, name: 'Learn NodeJS', isComplete: false }
]
```

Un detalle no menor es que el Store es **Read Only**, la forma de modificarlo es emitiendo acciones, llamadas Actions.

Redux – Actions

- Las acciones son simples objetos JavaScript.
- Tienen un **type** (obligatorio).
- Tienen un **payload**, que puede ser cualquier cosa (opcional).
- Su propósito es describir un evento.
- El evento puede provocar un cambio en el state -> Mediante un **Reducer.**

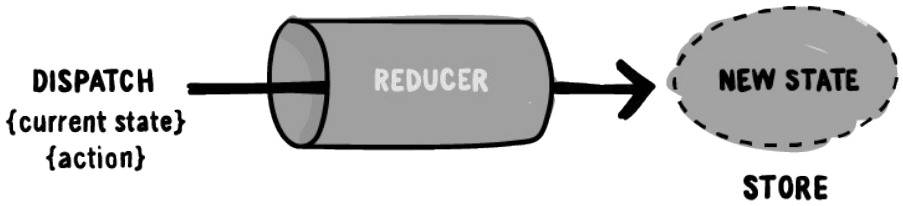
```
const action = {
    type: "Type of the action",
    payload: { data: "Information" }
}
```

```
const addTodo = {
    type: "ADD_TODO",
    payload: "Learn Redux"
}
```

Redux – Reducers (1)

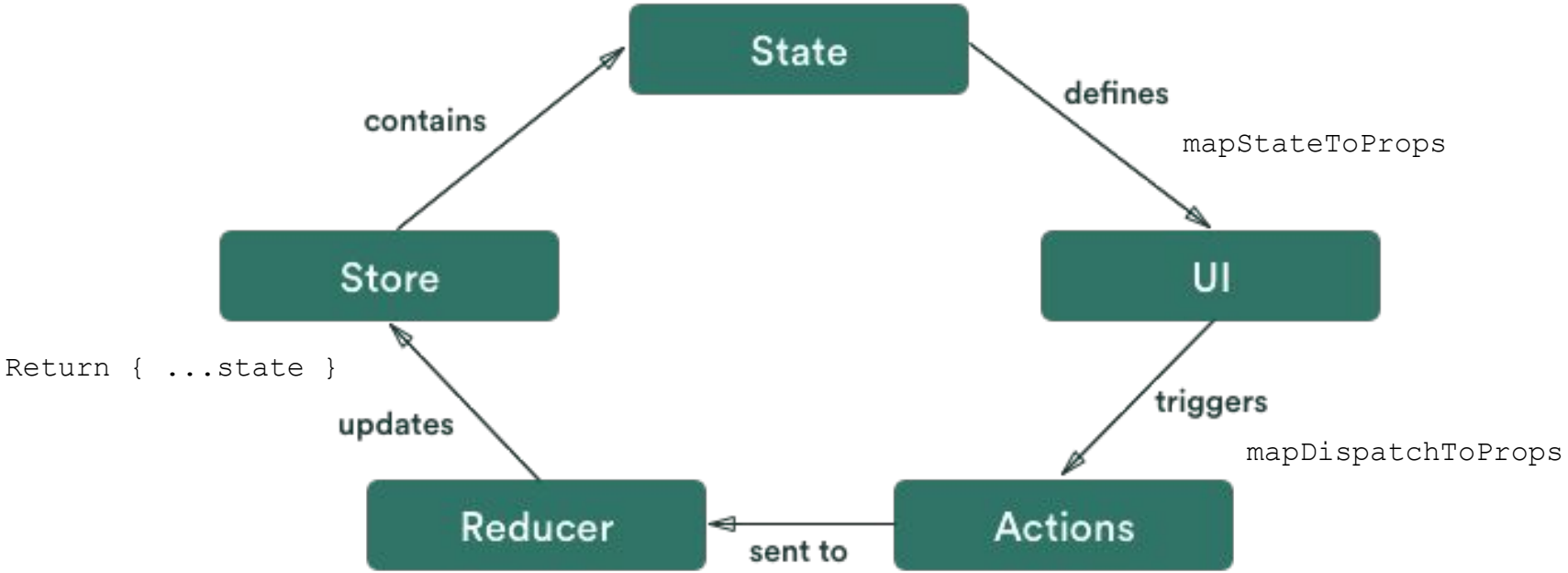
Los **Reducers** son funciones JavaScript con la condición de ser funciones **puras.** No modifican los parámetros que reciben y no dependen de nada externo a la función (dado un mismo *input* siempre producen el mismo *output*).

- Reciben el state actual,
  y el action que ocurrió.
- Siempre devuelven un nuevo state




```
function reducer(state, action) {
   // Nos aseguramos que este definido:
   state = state || initialState;
   if(action.type === "type1"){
       // Calcular nuevo estado.
   }
   else if(action.type === "type2"){
       // Calcular nuevo estado.
   }
   return state;
}
```

Redux Flow


State
defines
mapStateToProps
contains
Store
UI
Return { ...state }
updates
triggers
mapDispatchToProps
Reducer
sent to
Actions

React-Redux

React-redux es una librería que provee bindings para facilitar el uso de Redux en React.

- `Provider` => inyecta el store en React.
- `connect` => es un HOC que permite conectar cualquier componente en la jerarquía al Store de Redux (siempre y cuando esté dentro de Provider).

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

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

React-Redux

`mapStateToProps` es una función que recibe el *state* del Store. Las propiedades del objeto que retorna a partir del *state* llegarán al componente que se conecte como `props` (`TodoList` en el ejemplo). Algo similar sucede con `mapDispatchToProps`, pero aquí se definen las funciones que se quiere que el componente que se conecte reciba por props, de forma de poder "dispatchear" acciones a los Reducers.

```
const mapStateToProps = (state, [ownProps]) => ({
    todos: state.todos
})
const mapDispatchToProps = (dispatch, [ownProps]) => ({
    addTodo: text => dispatch(addTodo(text))
})
export default connect(
    mapStateToProps,
    mapDispatchToProps
)(TodoList);
```

Redux Middleware

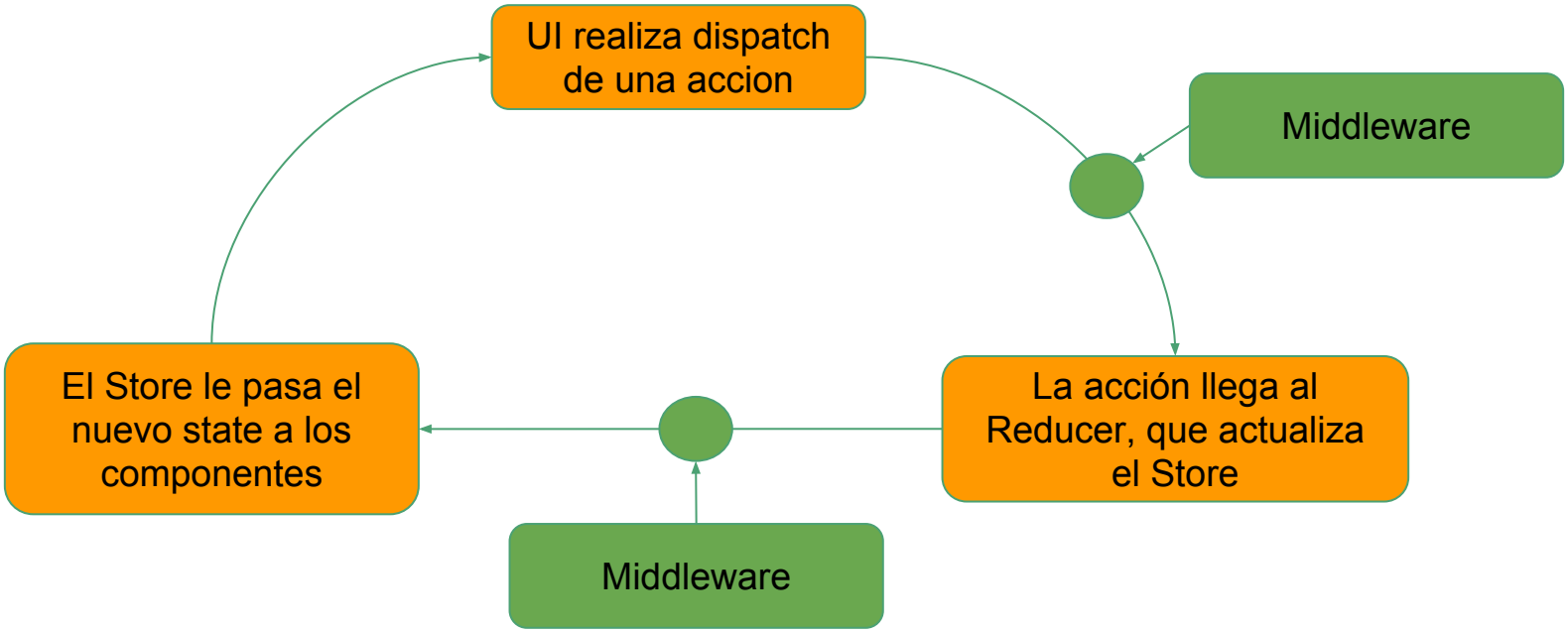
Redux Middleware

Un Middleware es, por definición, un software utilizado para ayudar a la conexión entre dos aplicaciones, algo que se ejecuta "en el medio" de dos cosas.

En Redux se pueden utilizar varios tipos de Middlewares distintos y hay dos lugares donde estos middlewares pueden realizar acciones:

- Entre que se "dispatchea" una acción y llega al reducer.
- Inmediatamente luego de que se ejecuta el reducer, con el state actualizado.

Redux Middleware



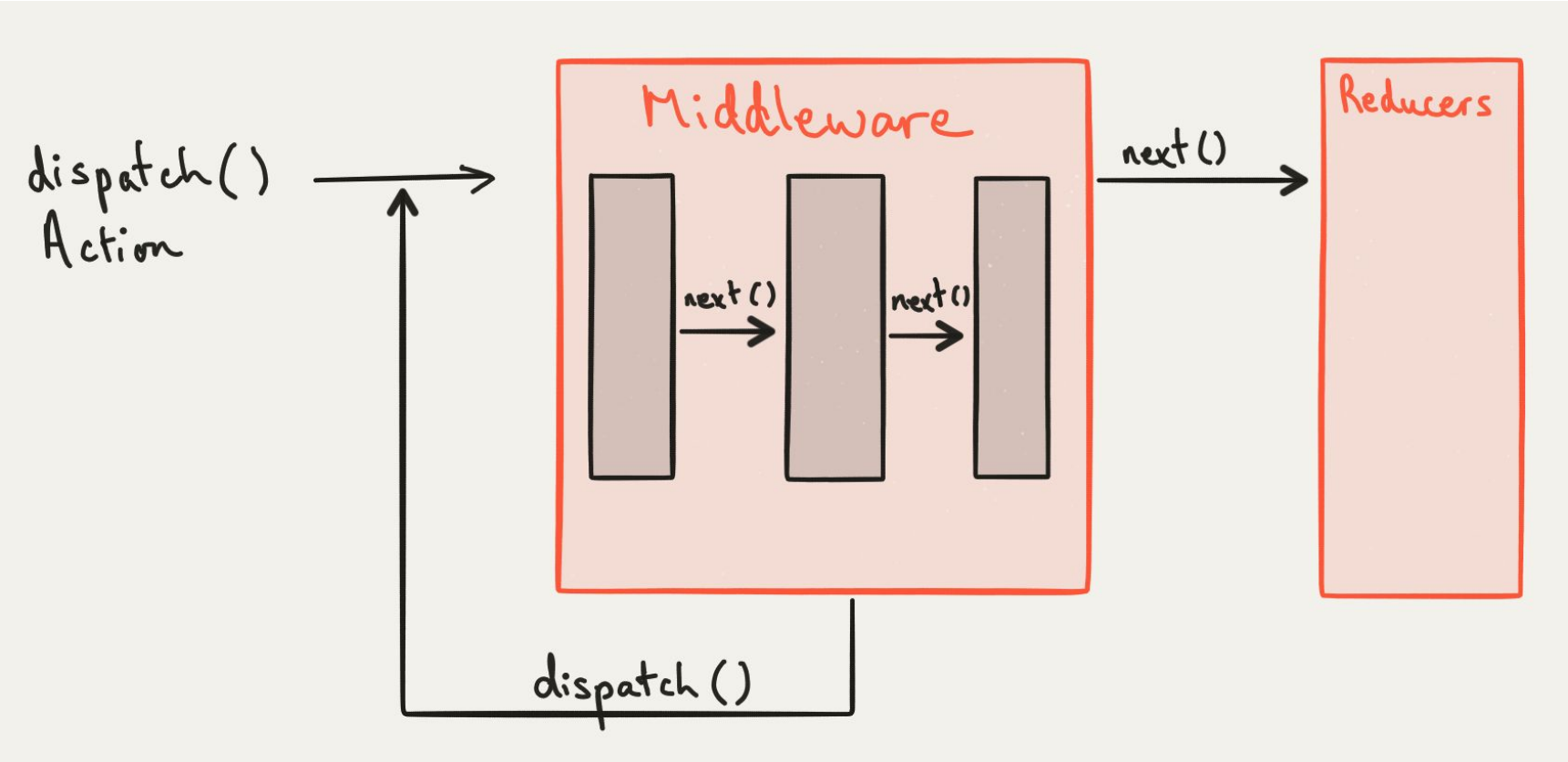
Redux Middleware

Los **Middlewares** son un punto de extensión en Redux, ya que nos proveen una forma de interactuar con todas las acciones que son emitidas al Reducer.

Hay varios ejemplos de uso, como loguear las acciones, reportar errores, hacer llamadas asíncronas, o dispatchear (emitir) nuevas acciones.

Los **Middlewares** se encadenan uno detrás de otro, y son llamados en orden para todas las acciones que se emitan. Tienen la posibilidad de modificar la Acción, o incluso de detener su propagación, osea, que no se envíe al siguiente **Middleware** ni al Reducer.

Redux Middleware



Redux Middleware - Implementación

Básicamente, es una función que recibe una acción e interactúa de alguna forma con ella:

```
function middleware(action) { /*...*/ }
```

Cómo funcionan en cadena, cada Middleware le debe enviar la acción al siguiente en la cadena, y si fuera el ultimo, seria al Reducer:

```
function middlewareWrapper(nextDispatch) {
 return function(action) {
     /*...*/
     nextDispatch(action);
 }
}
```

Redux Middleware - Implementación

Pero además, cada Middleware tiene acceso a una copia del Store (osea, tiene acceso a la función Dispatch y al getState para ver el State). Esto se logra siendo que la función final de un Middleware tiene otro wrapper:

```
function storeWrapper(store) {
 function middlewareWrapper(nextDispatch) {
   return function(action) {
     nextDispatch(action);
   }
 }
}
```

=

```
function middleware(store) {
 return function(nextDispatch) {
   return function(action) {
     nextDispatch(action);
   }
 }
}
```

=

```
const middleware = store => nextDispatch => action {
 nextDispatch(action);
}
```

Redux Middleware

Para usar Middlewares se componen las distintas funciones utilizando un método de Redux llamado ‘applyMiddleware’, y dándole el resultado al createStore de Redux. En este caso, `middlewareOne` será el que reciba las acciones primero, y `middlewareTwo`, como es el último, es el que dispatcheara las acciones al Reducer.

```
import { applyMiddleware, createStore } from "redux";

const store = createStore(
 reducers,
 initialState,
 applyMiddleware(
   middlewareOne,
   middlewareTwo
 )
);
```

```
import { createLogger } from 'redux-logger'
import { createStore, applyMiddleware } from 'redux'
import rootReducer from './reducers'

const loggerMiddleware = createLogger();
const store = createStore(
 rootReducer,
 applyMiddleware(
     loggerMiddleware // middleware que loguea toda las acciones
 )
);

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

Redux Middleware - Otros usos

Podemos crear uno para confirmaciones de usuario:

```
const confirmationMiddleware = store => next => action => {
 if (action.shouldConfirm) {
   if (confirm('Are you sure?')) {
     next(action);
   }
 } else {
   next(action);
 }
};
```

Redux Middleware - Otros usos

Podemos crear uno para mostrar errores, utilizando alguna librería de mensajes tipo Toast u otros (vanillatoasts):

```
const toastMiddleware = store => next => action => {
 if (action.error) {
   vanillatoasts.create({ text: action.error.toString(), timeout: 5000 });
 }
 next(action);
};
```

Redux Middleware

Algunas de las librerías más comunes de Redux Middlewares:

- redux-thunk
- redux-saga
- redux-logger

Redux Middleware - Redux Dev Tools

Para poder aplicar middlewares, a la función `createStore` de `Redux` le debemos pasar como segundo o tercer parámetro la llamada a `applyMiddleware` (el segundo parámetro es opcional, y es el estado inicial), pero esto genera un conflicto con el código que agregamos de la extensión de Redux Dev Tools. Lo resolvemos de la siguiente forma:

1.2 Advanced store setup

If you setup your store with middleware and enhancers, change:

```
  import { createStore, applyMiddleware, compose } from 'redux';

+ const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
+ const store = createStore(reducer, /* preloadedState, */ composeEnhancers(
- const store = createStore(reducer, /* preloadedState, */ compose(
    applyMiddleware(...middleware)
  ));
```

> Note that when the extension is not installed, we're using Redux compose here.

Ejercicios

Ejercicios

Descargar el zip `EjerciciosClase10`. Cada carpeta dentro corresponde a un ejercicio y es un proyecto creado con Create React App, por lo que para ejecutarlo es necesario:

- Si usan yarn:
  - `$ yarn install`
  - `$ yarn start`
- Si usan npm:
  - `$ npm install`
  - `$ npm start`

Se debería abrir automáticamente el explorador con el ejercicio corriendo en http://localhost:3000/

Ejercicio 1

El ejercicio ya contiene redux, y la configuración inicial. La idea es crear un `Timer`, que cada un segundo actualice el tiempo desde que comenzó.

- Es necesario conectar el componente `Timer` con Redux, de forma que pueda leer el tiempo actual, y emitir las acciones `StartTimer` y `StopTimer` con los botones correspondientes.
- Luego, se debe completar el Middleware `timerMiddleware` que está en el archivo `configureStore.js`.
    - Cuando se emita una acción de tipo '`START_TIMER`', debe crear un intervalo que cada 1 segundo emita una acción '`TICK`' con el tiempo actual. Debe agregar el intervalo creado a la acción para que se guarde en el `State` de el `Store`.
    - Cuando se emita una acción de tipo '`STOP_TIMER`', debe llamar al `clearInterval`, obteniendo el intervalo guardado en el `State` de el `Store`.

Ejercicio 1 - Parte 2

- Agregar el Middleware redux-logger, que ya está instalado en el ejercicio, para que se logueen todas las acciones emitidas (brinda en estado anterior, la acción, y el estado resultante)

- Crear un nuevo middleware llamado ‘deaf’. Este middleware será un poco ‘sordo’, por lo que solo dejara que la acción se emita luego de la tercera vez que llegue una acción. (se preguntan ¿porque? Por que si...)

Ejercicio 1 - Parte 3

- Crear un nuevo middleware llamado ‘persist’. Este middleware persistirá el state del `Store` en el **local storage** del navegador, luego de cada acción emitida. Luego agregar en la creación del store inicial el código necesario para que se lea esta información, y se pase como segundo parámetro a `‘createStore’` el estado inicial.
    - Leer y guardar en local storage:

      ```
      window.localStorage.setItem("SAVESTATE", str);
      window.localStorage.getItem("SAVESTATE") || "{}"
      ```

    - Para transformar el state a string, y viceversa:

      `JSON.stringify()` y `JSON.parse()`