Redux involves usage of pure functions which is achieved by using …spread operator, slice(), concat() methods to change items in arrays without mutating them.

You may use deepfreeze library from npm to keep a check on mutation. It can be used like deepfreeze(<variable/state\_name>);

We need a single function from Redux called createStore. CreateStore returns a store object.

const { createStore } = Redux;

// var createStore = Redux.createStore;

// import { createStore } from ‘redux’ // (if you use NPM and something like Babel to transpile your ES6)

const store = createStore ( reducer );

This store binds together the three principles of Redux (or 3 methods provided by store object). It holds the current application's **state** object returned by createStore. It lets you dispatch actions.

1. getState(): It retrieves the current state of the Redux store.
2. Dispatch(): It lets you dispatch actions to change the state of your application.
3. subscribe : It lets you register a callback which is called whenever any action is dispatched, so that you can update your UI.

When you create a store, you need to specify the **reducer** that tells how state is updated with actions.

What a reducer is, it's a pure function you write to implement the update logic of your application -- that is, how the next state is calculated given the current state and the action being dispatched.

Re-Implementing createStore function (shipped with redux) from Scratch to see what happens inside of it.

const createStore = (reducer) => {

let store;

let listerners = [];

const getState = () => state;

const dispatch = (action) => {

state = reducer(state, action);

listerners.forEach(listener => listener());

}

const subscribe = (listener) => {

Listeners.push(listener);

Return () => {

Listeners = listeners.filter( l => l !== listener))

}

}

return ( getState, dispatch, subscribe);

}

Store: We know that the store holds the current state. We keep it in this variable.

listerners: Subscribe function can be called multiple times with respective callback functions. We need to store the list of all the callbacks which are to be executed/called after state is updated. So, this listerners array will store all the callbacks.

This array is actually loaded in subscribe method, whenever subscribe method is called with a callback method, then that particular callback is pushed to this array.

In dispatch method - In order to calculate the new state we call the reducer with the current state and the action being dispatched, this reducer returns us the updates value of state which is then assigned to state variable in createStore function. After the state was updated, we need to notify every changed listener, by calling it.

To unsubscribe a listener we'll just return a function from the Subscribe method that removes this listener from the listeners' array.

This implementation of the Redux store apart from a few minor details in that case is, is the createStore was shipped with Redux.