what is Redux ?

State is **Read-only** in Redux. What makes Redux predictable is that to make a change in the state of the application, we need to dispatch an action which describes what changes we want to make in the state.

These actions are then consumed by something known as reducers, whose sole job is to accept two things (the action and the current state of the application) and return a new updated instance of the state.

We'll talk more about actions and reducers in the following sections.

Note that reducers do not change any part of the state. Rather a reducer produces a new instance of the state with all the necessary updates.

According to @[Dan Abramov](https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/@gaearon) (the creator of Redux) himself,

*"Actions can be recorded and replayed later, so this makes state management predictable. With the same actions in the same order, you're going to end up in the same state."*

So continuing with our above example of an e-commerce website, if the initial state of the cart is that it has 0 items, then an action of **adding one item** to the cart will increase the number of items in the cart by 1. And firing the action of **adding one item** to the cart again will increase the number of items in the cart to 2.

Given an initial state, with a specific list of **actions** in a specific order, it'll always provide us with the exact same final state of the entity. This is how Redux makes state management predictable.

In the following section, we will dive deep into the core concepts of Redux – the store, actions and reducers.

**What is the Redux Store?**

*The global state of an application is stored in an object tree within a single store –*[*Redux docs*](https://redux.js.org/understanding/thinking-in-redux/three-principles)

The Redux store is the main, central bucket which stores all the states of an application. It should be considered and maintained as a **single source of truth** for the state of the application.

If the store is provided to the **App.js** (by wrapping the App component within the <Provider> </Provider> tag) as shown in the code snippet below, then all its children (children components of App.js) can also access the state of the application from the store. This makes it act as a global state.

// src/index.js

import React from 'react'

import ReactDOM from 'react-dom'

import { Provider } from 'react-redux'

import { App } from './App'

import createStore from './createReduxStore'

const store = createStore()

// As of React 18

const root = ReactDOM.createRoot(document.getElementById('root'))

root.render(

<Provider store={store}>

<App />

</Provider>

)