[**https://medium.com/@notrab/getting-started-with-create-react-app-redux-react-router-redux-thunk-d6a19259f71f**](https://medium.com/@notrab/getting-started-with-create-react-app-redux-react-router-redux-thunk-d6a19259f71f)

[**https://egghead.io/lessons/react-redux-react-counter-example**](https://egghead.io/lessons/react-redux-react-counter-example)

**‘yarn start’ to run at your local**

**Redux**

* First principle of Redux—the **single immutable state tree**.
* The second principle of Redux is that the **state tree is read only**. You **cannot modify or write** to it. Instead, anytime you want to change the state, you need to dispatch an action.

An **action** is a plain **JavaScript object** describing the change. Just like the state is the minimal representation of the data in your app, the action is the minimal representation of the change to that data.

second principle of Redux -- the state is read only. The only way to change the state tree is by dispatching an action. An action is a plain JavaScript object, describing in the minimal way what changed in the application. Whether it is initiated by a network request or by user interaction, any data that gets into the Redux application gets there by actions.

* understand the **difference between the pure and impure functions.** The pure functions are the functions whose returned value depends solely on the values of their arguments.

Pure functions do not have any observable side effects, such as network or database calls. The pure functions just calculate the new value. You can be confident that if you call the pure function with the same set of arguments, you're going to get the same returned value. They are predictable.

On the opposite, impure functions may call the database or the network, they may have side effects, they may operate on the DOM, and they may override the values that you pass to them. This is going to be an important distinction because some of the functions that you're going to write in Redux have to be pure, and you need to be mindful of that.

* third and the last principle of Redux. To describe state mutations, you have to write a function that **takes the** **previous state of the app**, the **action being dispatched**, and **returns the next state of the app**. This function has to be pure. This function is called the "**Reducer**."

There is something in common between all Redux applications. They have to implement the reducer: a function that calculates the next state tree based on the previous state tree and the action being dispatched.

**How to bind store with reducer.**

store(reducer)

**Store has 3 methods.**

1. store.**getState()**
2. store.**dispatch()**
3. store.**subscribe()**. It lets you register a callback that the Redux chore will call any time an action has been dispatched, so that you can update the UI of your application. It will reflect the current application state.

**REACT APP creae via create-react-app**

yarn start

Starts the development server.

yarn build

Bundles the app into static files for production.

yarn test

Starts the test runner.

yarn eject

Removes this tool and copies build dependencies, configuration files

and scripts into the app directory. If you do this, you can’t go back!

We suggest that you begin by typing:

cd react-redux-example

yarn start