


Reducer is interface & React-Reducer is implementation
index.js

```
import './index.css'
import App from './App'
import React from 'react'
import { ReactDOM } from 'react'

import { store } from './app/store'
import { Provider } from 'react-redux'

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

root.render(
  <React.StrictMode>
    <Provider store={store}> Making components aware of the store
      <App />
    </Provider>
  </React.StrictMode>
)
```

store.js

```
JS store.js [0] store
import { configureStore } from "@reduxjs/toolkit";
↳ store can be created with this store
export const store = configureStore({
  reducer: {} → no need to define the type and then
  map it. Can directly call
  a method
})
```

& Slice means Functionality

* Payload means value / parameters

```
EXPLORER
REACT
-> node_modules
-> App.js M
-> .gitignore M
-> package-lock.json M
-> README.md M
-> tailwind.config.js M

JS counterSlice.js U JS store.js U
src > features > counter > JS counterSlice.js [0] initialState
1  const { createSlice } = require("@reduxjs/toolkit");
2
3  const initialState = {
4    value: 0
5  }
6
7  export const counterSlice = createSlice({
8    name: 'counter',
9    initialState,
10   reducers: {
11     increment: (state) => {
12       state.value += 1
13     },
14     decrement: (state) => {
15       state.value -= 1
16     },
17     // action => action.TYPE, action.payload
18     incrementByValue: (state, action) => {
19       state.value += action.payload
20     }
21   }
22
23  export const {increment, decrement, incrementByValue} =
24  counterSlice.actions
25
26  export default counterSlice.reducer
```

- ① Install redux & bindings of react & redux
- ② Create a store
- ③ Make all components aware of the store

* state is a built-in React Object that is used to contain Data or information about the component whenever it changes the component re-renders

```
import { configureStore } from "@reduxjs/toolkit";
import { counterSlice } from "../features/counter/counterSlice";
↳ make the store aware of methods
export const store = configureStore({
  reducer: {
    counter: counterSlice
  }
  // videoPlayerSlice
})
```

any random name

} These are the only ways / methods by which my state can be manipulated

- * **useSelector** : Select the current value from the store
- * **useDispatch** : after I imported the methods for my state manipulation in App.js and I want to use them. I access them via useDispatch

```

JS App.js M JS store.js U JS counterSlice.js U SouravChauhan
src > JS App.js > App
1 import React, { useState } from 'react'
2
3 import { increment, decrement, incrementByValue } from './features/counter/counterSlice'
4
5 import { useSelector, useDispatch } from 'react-redux'
6
7 function App() {
8   const [input, setInput] = useState('')
9   const count = useSelector((state) => state.counter.value)
10  const dispatch = useDispatch()
11
12
13  return (
14    <div
15      className='bg-zinc-900 text-zinc-100 flex justify-center
16      items-center flex-col min-h-screen gap-6'
17    >
18    ...

```

here not state.initialValue (from store)
because we need the real time value and not
the initialValue as it will always be 0.

- ① installed redux.js/toolkit which is independent
 - used react-redux library (binding) to attach with react

Step 1 : Create a store

↓
configureStore which creates the store
↓
create object and inside it create reducer

Step 2 : Making every component aware that there exists a store

↓
import store
import Provider

Wrapped the entire App with Provider & pass on a prop store

* New our store is ready we have to write some rules how are going to manipulate the values in our store and in term of redux we call them Slice (functionalities)

Step 3: We create createSlice, which provides a lot of abstraction to us. First we had to match the actions and payloads (values) so in order to avoid these. So we use createSlice to avoid this matching

now we create functionalities:

* Inner Library helps us to prevent mutation of the state

```
const initialState = {  
    value: 0;  
}
```

```
export const counterSlice = createSlice ({
```

name: 'counter',

initialState,

reducers: {

increment: (state) => {

state.value += 1

},

decrement: (state) => {

state.value -= 1 ↑

I don't have to use it because of counterSlice

}, // action: action.type, action.payload

incrementByValue: (state, action) => {

state.value += action.payload

},

},

```
export const { increment, decrement, incrementByValue } =  
    counterSlice.actions
```

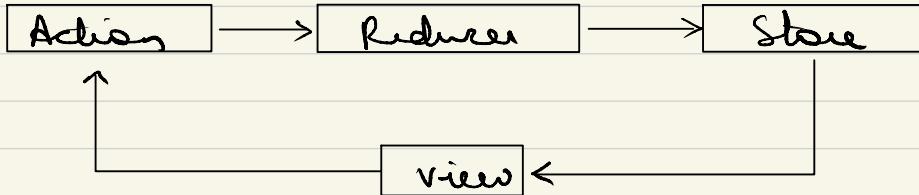
```
export default counterSlice.reducer
```

```
# in App.js
import { increment, decrement, incrementByValue } from './features/
counter/counterSlice'
import { useSelector, useDispatch } from 'react-redux'

function App () {
  const [input, setInput] = useState('')local scope(from store.js)
  const count = useSelector ((state) => state.counter.value)
  const dispatch = useDispatch()

  return (
    <button onclick={() => dispatch(increment())}>
      </button>
    <br/>
    <button onclick={() => dispatch(decrement())}>
      </button>
  )
}
```

- * An Action is a plain Object that describes the intention to cause change.
- * A reducer is a function that determines changes to an application's state: returns the new state and tell the store how to do.
- * it uses the action it receives to determine this change.



- * Action
 - Type : 'BRING'
 - Payload: 1 ball + 1 bat