Redux Toolkit

A study material for the students of GLS University

Prepared by : Ms. Harshita Maheshwari

Subject : MERN Stack

What is Redux

npm install redux

What is Redux

Redux is a predictable state container for JavaScript apps.

- Redux is not tied to React
- Can be used in React, Angular, Vue or even vanilla JavaScript.
- Store the state of your application
- State of an app is the state shared by all the individual components of that app.

Principles

Single source of truth: The state of your whole application is stored in an object tree within a single store.

State is read-only: The only way to change the state is to emit an action, an object describing what happened.

Changes are made with pure functions: To specify how the state tree is transformed by actions, you write pure reducers.

Why Redux

- If you want to manage the global state of your application in a predictable way, redux can help you.
- The patterns and tools provided by Redux make it easier to understand when, where, why and how the state in your application is being updated, and how your application logic will behave when those changes occur.
- Redux guides you towards writing code that is predictable and testable, which helps give you confidence that your application will work as expected.

Redux Terminologies

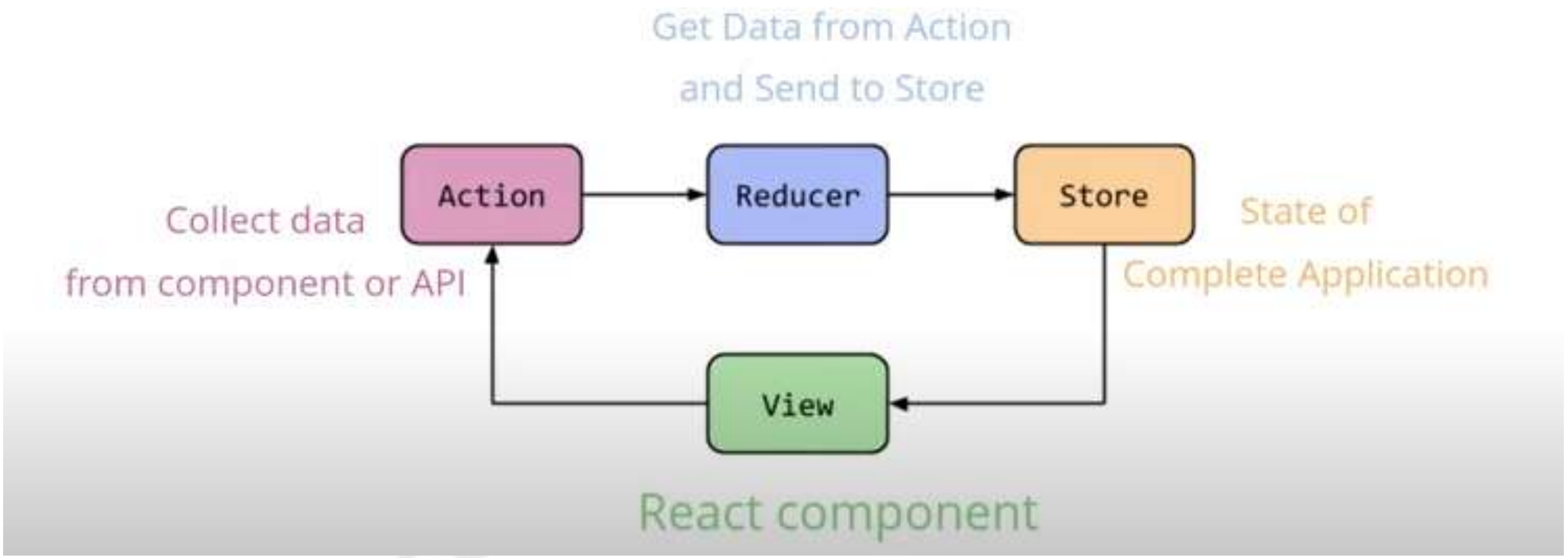
State: an object, probably nested

Actions: describes intended change of state, declarative, think of an action as just a set of instructions used by Redux to update state

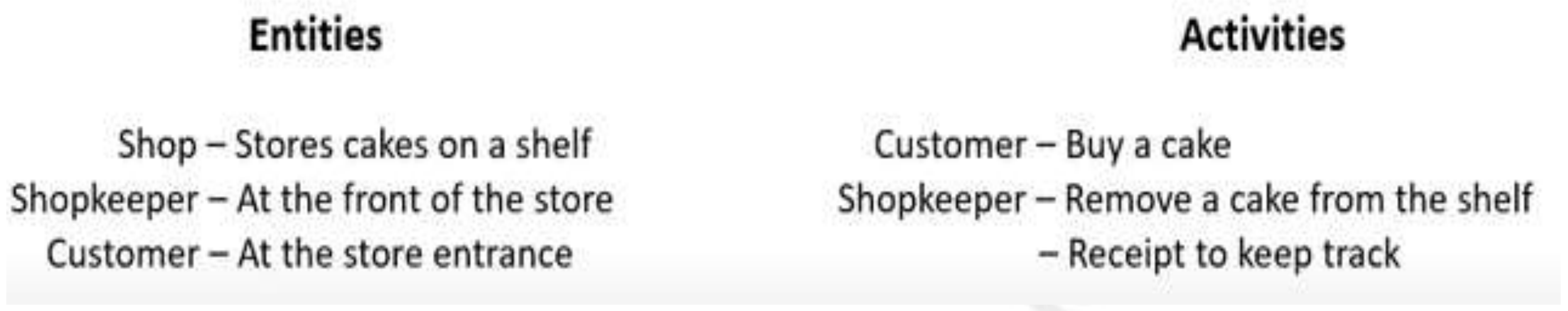
Reducers: pure function, which takes the previous state and action as arguments, returns updated state, should be the only thing updating state

Store: where state lives

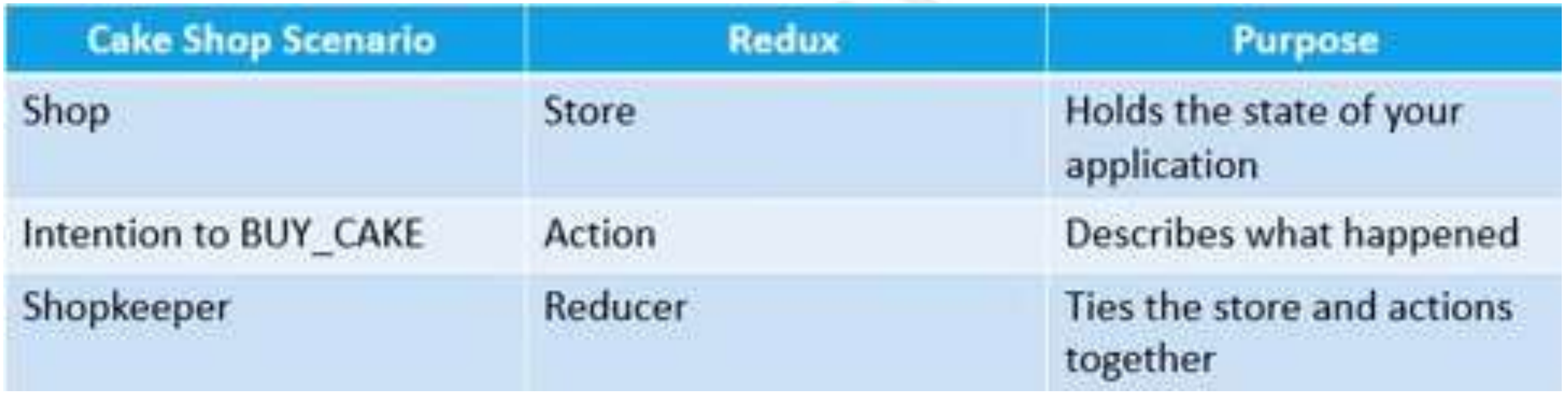
```
Action -> Reducer -> State -> View
```


Get Data from Action
and Send to Store
Action
Reducer
Store
Collect data
from component or API
State of
Complete Application
View
React component

Cake Shop -



Entities	Activities
Shop – Stores cakes on a shelf	Customer – Buy a cake
Shopkeeper – At the front of the store	Shopkeeper – Remove a cake from the shelf
Customer – At the store entrance	– Receipt to keep track



Cake Shop Scenario	Redux	Purpose
Shop	Store	Holds the state of your application
Intention to BUY_CAKE	Action	Describes what happened
Shopkeeper	Reducer	Ties the store and actions together

Historical Complaints About Redux

- Several common complaints about Redux over the years:
- Configuring a Redux store is too complicated
- Have to add a lot of packages to do anything useful (redux-thunk, reselect, etc)
- Too easy to accidentally make a mistake like mutating state
- Amount of "boilerplate" you need to write
  - ➢ Action types
  - ➢ Action creators
  - ➢ Immutable update logic
  - ➢ "Have" to use multiple files
  - ➢ Complex store setup process

Redux Toolkit

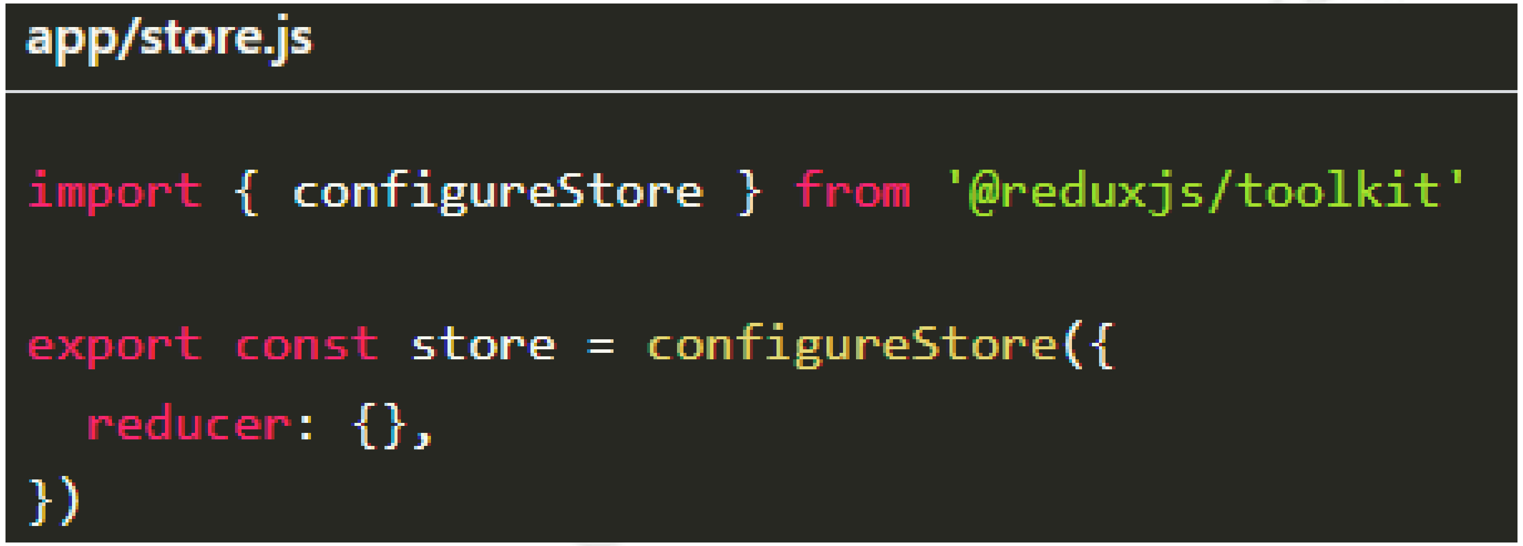
```
npm install @reduxjs/toolkit
```

Redux Toolkit

Created a new official package called Redux Toolkit, inspired by create-react-app
Goals:
Simplify common Redux use cases
Provide good opinionated defaults out of the box
Minimize the amount of code you have to write by hand
Doesn't "hide" that you're using Redux, just makes it easier
Provide a great developer experience for TypeScript users
Originally named "Redux Starter Kit", but renamed after 1.0 release
Renamed to "Redux Toolkit" (package: @reduxjs/toolkit) for 1.0.4 release
Written in TypeScript, designed to simplify TS usage patterns

Create a Redux Store

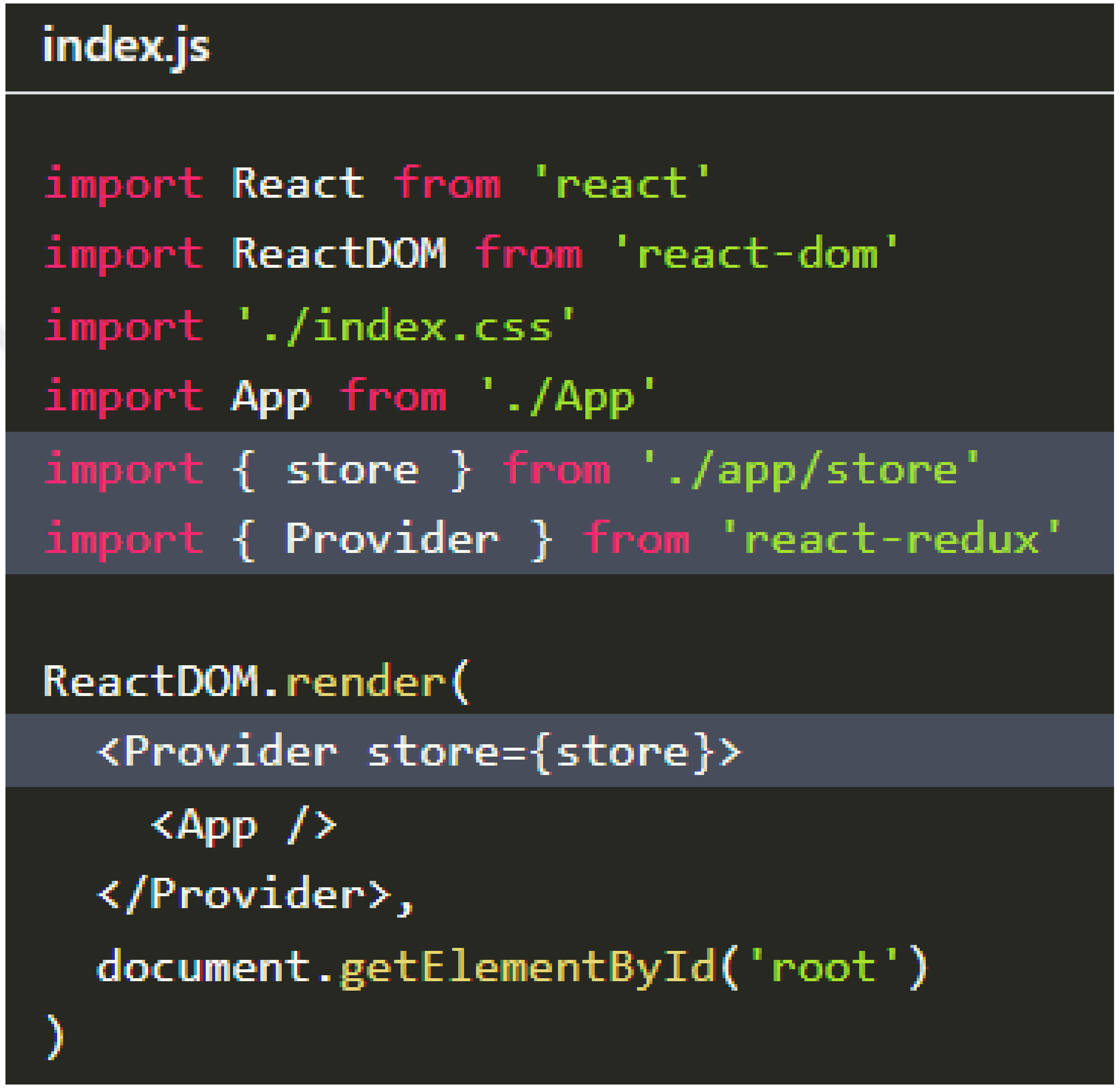
Create a file named src/app/store.js. Import the configureStore API from Redux Toolkit.


app/store.js

```
import { configureStore } from '@reduxjs/toolkit'

export const store = configureStore({
  reducer: {},
})
```

Provide the Redux Store to React

Once the store is created, we can make it available to our React components by putting a React-Redux <Provider> around our application in src/index.js. Import the Redux store we just created, put a <Provider> around your <App>, and pass the store as a prop.



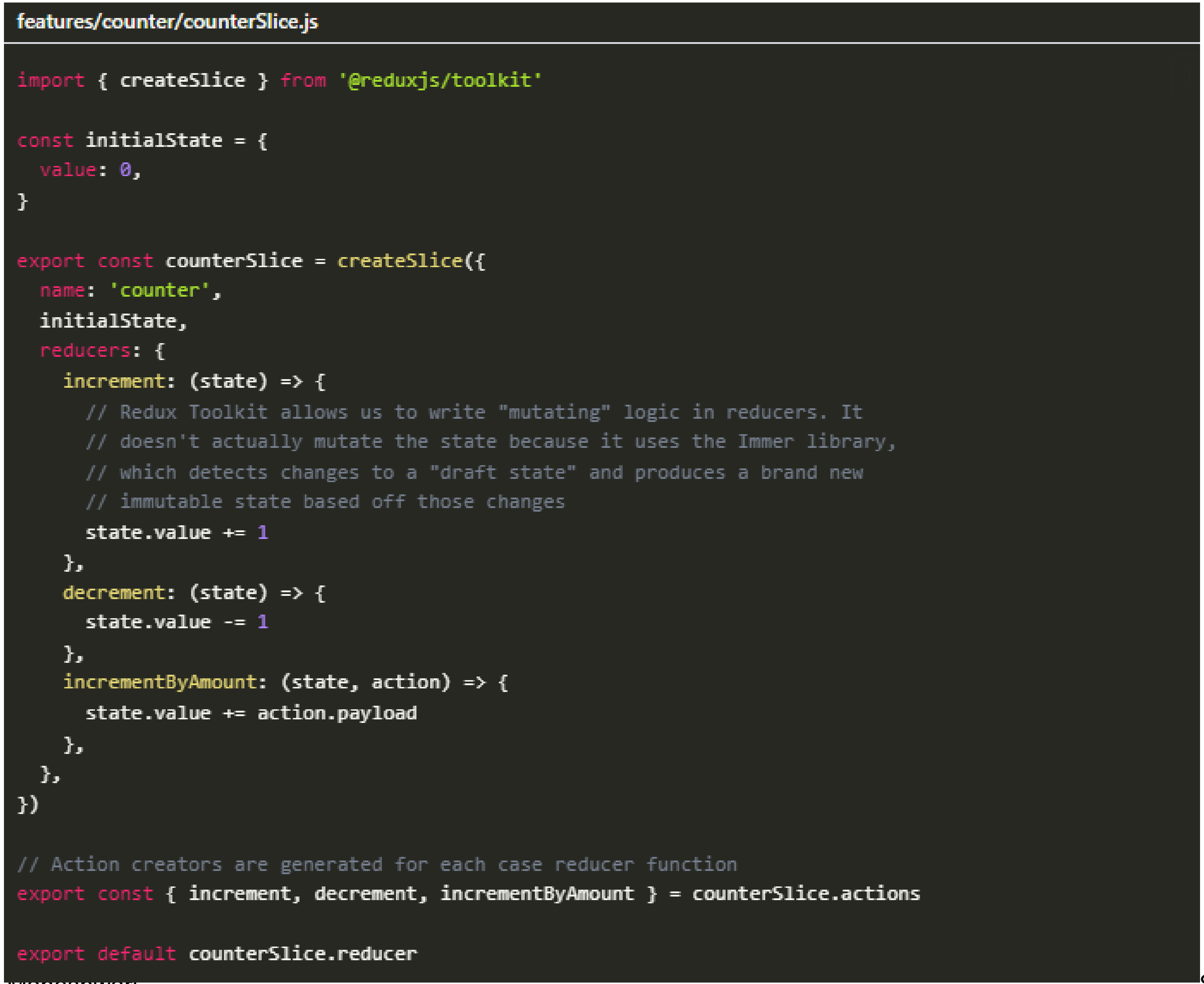
index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'
import { store } from './app/store'
import { Provider } from 'react-redux'

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

Create a Redux State Slice

Import the createSlice API from Redux Toolkit.

Creating a slice requires a string name to identify the slice, an initial state value, and one or more reducer functions to define how the state can be updated. Once a slice is created, we can export the generated Redux action creators and the reducer function for the whole slice.

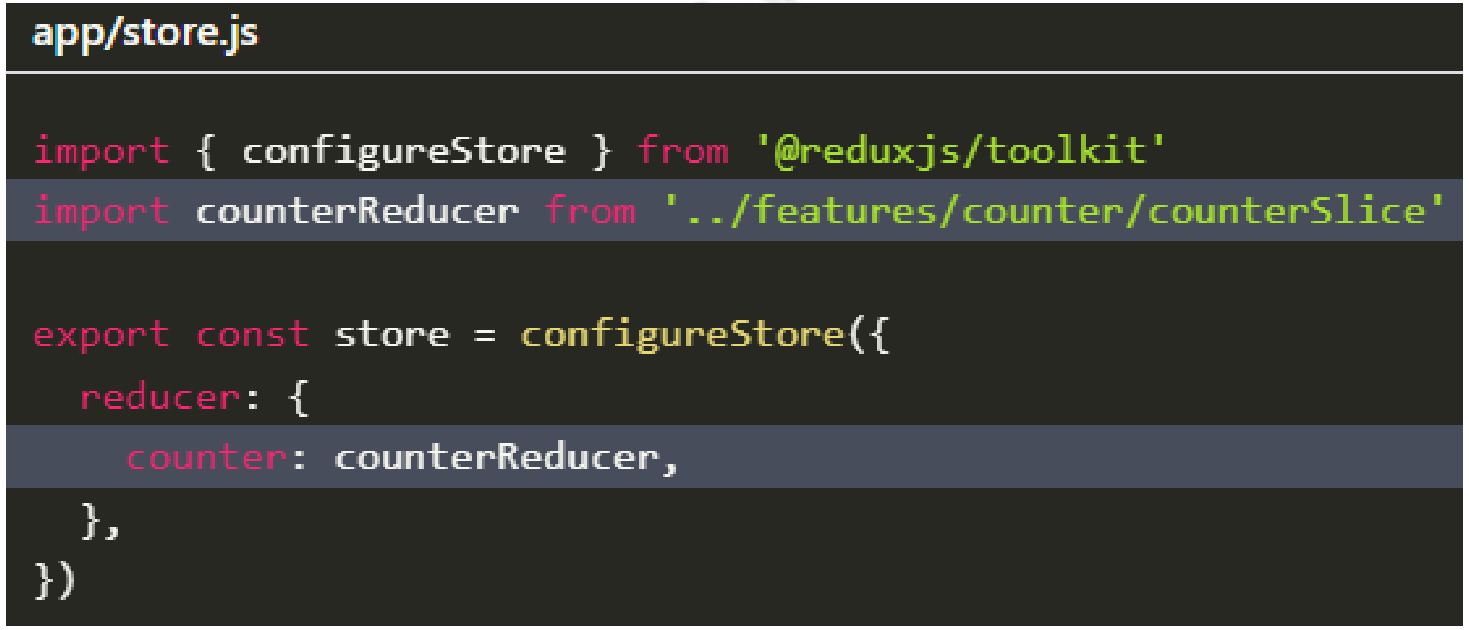


features/counter/counterSlice.js

```
import { createSlice } from '@reduxjs/toolkit'

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      // Redux Toolkit allows us to write "mutating" logic in reducers. It
      // doesn't actually mutate the state because it uses the Immer library,
      // which detects changes to a "draft state" and produces a brand new
      // immutable state based off those changes
      state.value += 1
    },
    decrement: (state) => {
      state.value -= 1
    },
    incrementByAmount: (state, action) => {
      state.value += action.payload
    },
  },
})

// Action creators are generated for each case reducer function
export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

Add Slice Reducers to the Store

we need to import the reducer function from the counter slice and add it to our store. By defining a field inside the reducer parameter, we tell the store to use this slice reducer function to handle all updates to that state.



app/store.js

```
import { configureStore } from '@reduxjs/toolkit'
import counterReducer from '../features/counter/counterSlice'

export const store = configureStore({
  reducer: {
    counter: counterReducer,
  },
})
```

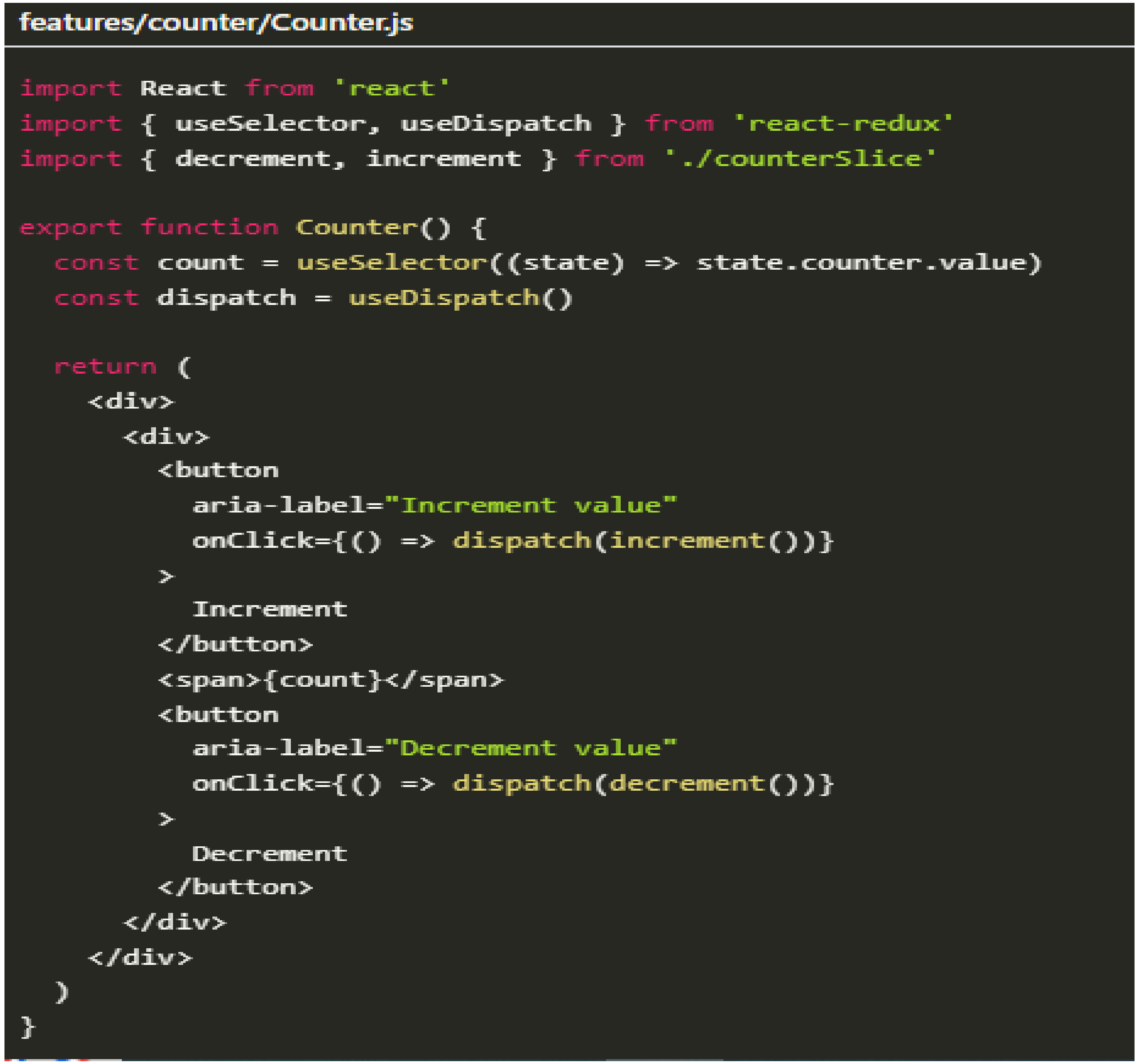
React Redux Hooks

```
npm  install react-redux
```

useDispatch and useSelector Hooks

Read data from the store with the useSelector hook

Get the dispatch function with the useDispatch hook, and dispatch actions as needed



features/counter/Counter.js

```
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import { decrement, increment } from './counterSlice'

export function Counter() {
  const count = useSelector((state) => state.counter.value)
  const dispatch = useDispatch()

  return (
    <div>
      <div>
        <button
          aria-label="Increment value"
          onClick={() => dispatch(increment())}
        >
          Increment
        </button>
        <span>{count}</span>
        <button
          aria-label="Decrement value"
          onClick={() => dispatch(decrement())}
        >
          Decrement
        </button>
      </div>
    </div>
  )
}
```

Now, any time you click the "Increment" and "Decrement" buttons:

- The corresponding Redux action will be dispatched to the store
- The counter slice reducer will see the actions and update its state
- The <Counter> component will see the new state value from the store and re-render itself with the new data

Thank you