Redux

سید احسان سجادی – کیان جلیلیان

استاد پرهام الوانی

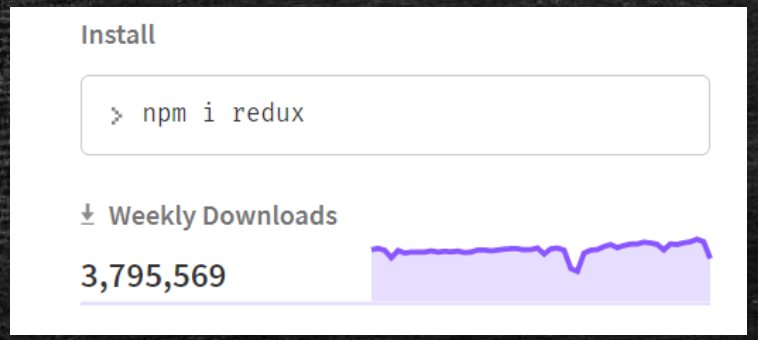
خرداد 1400

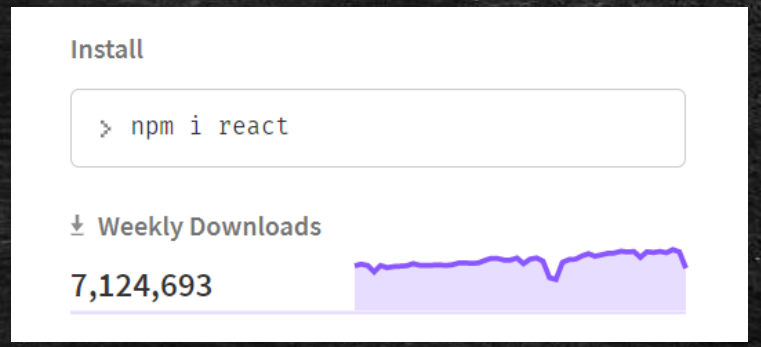
ریداکس (redux) چیست؟

- یک کتابخانه مدیریت اطلاعات (state management) برای برنامه‌های جاوااسکریپتی است. چون فقط state را مدیریت می‌کند و به UI برنامه کاری ندارد، میتوان از آن در react، Vue، Angular و حتی vanilla js استفاده کرد. در برنامه ها، معمولا با ایجاد تغییر در یک قسمت خاص، قسمت های دیگر در UI هم باید تغییر کنند یا بخاطر دریافت اطلاعات جدید در پشت صحنه، باید UI در چندین بخش مختلف تغییر کند. در این برنامه‌ها مخصوصا اگر برنامه پیچیده شود، ردیابی state و دلیل تغییرات رخ داده سخت می‌شود. ریداکس به ما کمک می‌کند که به راحتی روند برنامه را بفهمیم و state را مدیریت کنیم. هر تغییر در state با صدا زدن یکسری action که خودمان تعریفشان می‌کنیم انجام میشود.

کتابخانه مشابه redux

- Flux توسط facebook برای حل مشکلات گفته شده ساخته شد. Redux از flux الهام گرفت و از flux به دلیل سادگی و ظرافتش محبوب تر شد. mobX هم یکی دیگر از کتابخانه های محبوب است.





https://www.npmjs.com/package/redux

https://www.npmjs.com/package/react

Redux چکار میکند؟

- به جای اینکه state ها را در جاهای مختلف برنامه ذخیره کنیم، در یک محل به نام store همه state ها ذخیره میشوند. به نوعی میتوان گفت store، یک دیتابیس برای front end است. ذخیره شدن همه state ها در یک محل باعث میشود که با تغییر state، فقط نیاز باشد یکجا را بروزرسانی کنیم و همه بخش های وابسته هم اطلاعاتشان را از این بخش میگیرند. همچنین به دلیل معماری که redux دارد، میتوان به راحتی فهمید که هر تغییر در state، کجا، چه زمانی و چرا رخ داده است.

مزایای redux:

- ذخیره state در یک محل (باعث میشود به راحتی وضعیت فیلتر، سرچ و ... را ذخیره کنیم.)
- پیشبینی راحت وضعیت بعد هر action
- توانایی استفاده از redux devTools (قابلیت debug همه action ها و time travel debugging)
- با استفاده از کتابخانه های کمکی redux میتوان به راحتی اطلاعات را برای استفاده های بعدی برنامه ذخیره کرد.
- انجام عملیات undo/redo با Redux بسیار راحت است.

بدی های redux:

- کد را پیچیده می‌کند. (چون بر پایه functional programming نوشته شده است)
- پرگویی یا verbosity (با استفاده از کتابخانه هایی مثل redux toolkit این مشکل تا حد زیادی حل میشود.)

توضیحات redux toolkit:

- ساختن store، نوشتن کد های مورد نیاز redux و انجام یکسری کارهای کاربردی با redux سخت و وقت‌گیر است. از redux toolkit استفاده می‌کنیم که wrapper دور redux است و کار با آنرا برای ما آسان می‌کند. این کتابخانه best practice های Redux را دارد، نوشتن کد redux را آسان می‌کند، از اشتباهات افراد تازه‌کار جلوگیری می‌کند.

چه زمانی از redux استفاده کنیم؟

- حالت (state) های برنامه ما زیاد تغییر میکند و این تغییرات در جاهای مختلفی از برنامه تاثیرگذارند.
- منطق بروزرسانی state پیچیده است و میخواهیم فقط یکبار آنرا پیاده‌سازی کنیم.
- مقدار کد بسیار زیاد است و چندین نفر روی پروژه کار میکنند.

چه زمانی از redux استفاده نکنیم؟

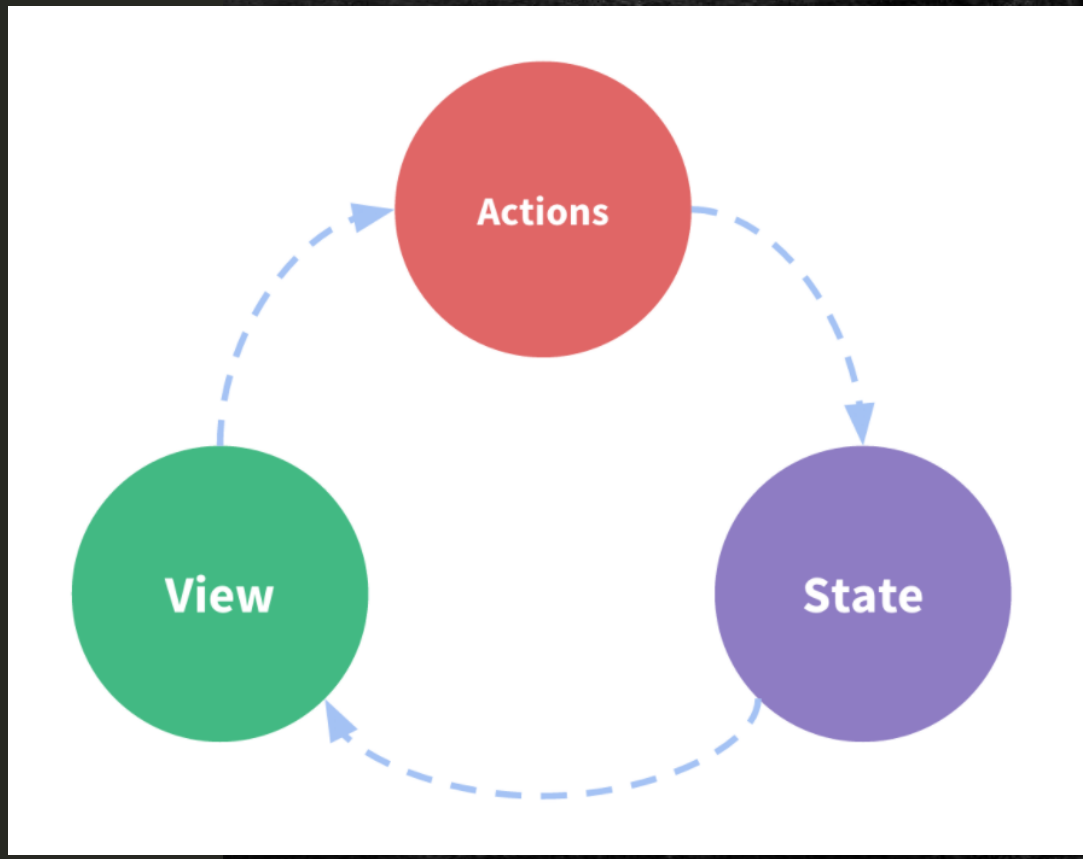
- اگر برنامه ما کوچک است، اطلاعاتش static است و UI ساده‌ای دارد و از بخش های زیادی تشکیل نشده‌است، نیازی به redux نداریم.

- همچنین Redux نباید برای حل مشکل prop drilling استفاده شود و ابزار های ساده‌تری برای این منظور وجود دارند.

```jsx
function Counter() {
  // State: a counter value
  const [counter, setCounter] = useState(0)

  // Action: code that causes an update to the state when something happens
  const increment = () => {
    setCounter(prevCounter => prevCounter + 1)
  }

  // View: the UI definition
  return (
    <div>
      Value: {counter} <button onClick={increment}>Increment</button>
    </div>
  )
}
```

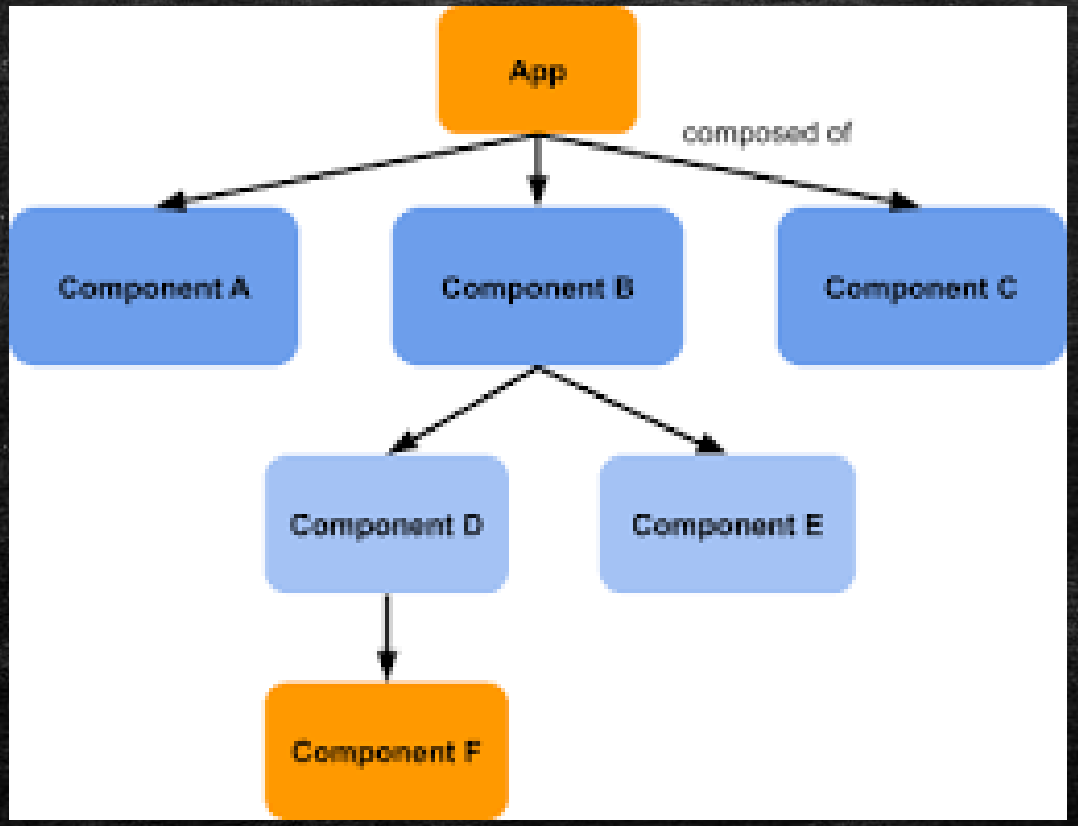


اگر استیت counter در یک کامپوننت نیاز باشد، فقط به state نیاز داریم ولی اگر این شمارنده در چند قسمت مختلف وجود داشته باشد، چکار باید کرد؟ مثلا برنامه مقابل را در نظر بگیرید :

اگر شمارنده ما در هرکدام از بخش های مشخص‌شده مورد نیاز باشد، می‌توانیم دو کار بکنیم:

1. در یک کامپوننت سطح بالاتر این استیت شمارنده را ذخیره کنیم و مقدارش را به کامپونت‌های سطح پایین تر به عنوان prop بفرستیم.

2. اطلاعاتی که در چندین مکان مختلف استفاده شده‌است را از کامپونت‌ها خارج کنیم و در یکجای دیگر (مثلا store در redux) ذخیره کنیم.



مشکل روش اول این است که
اگر تعداد کامپوننت‌ها زیاد شود،
prop drilling رخ می‌دهد.

context

- با استفاده از context می‌توان این مشکل را حل کرد. با استفاده از context، اطلاعات را در بالاترین کامپوننت تعریف می‌کنیم و در همه کامپوننت‌های زیرمجموعه آن می‌توانیم به آن دسترسی داشته باشیم.

Redux

- راه حل دوم این است که یک محل خاص (store) برای ذخیره همه اطلاعات مشترک داشته باشیم که خارج از درخت کامپونت‌ها است و از همه کامپونت ها می‌توان آنرا صدا کرد و به مقادیرش دسترسی داشت. همچنین از همه کامپونت ها می‌توان action هایش را صدا زد.

Actions

- اکشن یا عملیات یک شی جاوااسکریپتی است که یک فیلد type دارد. این فیلد نوع عملیات را نشان می‌دهد. این type معمولا یک استرینگ مانند "domain/eventName" است. کلمه اول دسته‌بندی عملیات را نشان می‌دهد. کلمه دوم عمل دقیقی که صدا شده‌است را نشان می‌دهد.
- یک اکشن معمولا این شکلی است:

```
const addTodoAction = {
  type: 'todos/todoAdded',
  payload: 'Buy milk'
}
```

Action Creators

- سازنده اکشن تابعی است که اکشن میسازد و آنرا بر میگرداند. معمولا برای صدا زدن یک اکشن از Action Creators استفاده میشود تا اکشن هارا دستی نسازیم.

```
const addTodo = text => {
  return {
    type: 'todos/todoAdded',
    payload: text
  }
}
```

Reducers

- Reducer یک تابع است که state فعلی را به همراه اکشن می‌گیرد و بر اساس اکشن، در صورت نیاز state را تغییر می‌دهد و state جدید را برمی‌گرداند. می‌توان گفت reducer همان event listener است. Reducer ها pure function هستند. یعنی باید قوانین زیر را رعایت کنند:
- State جدید باید فقط بر حسب state قبلی و اکشنی که دریافت شده‌است تولید شود. یعنی مثلا نباید به api درخواستی بدهیم و براساس اطلاعات آن state جدید را بسازیم یا نباید از اعداد رندم استفاده کرد. به طور خلاصه، state و اکشن یکسان باید همیشه خروجی یکسان به ما بدهند.
- نباید state فعلی را تغییر دهند. State در ریداکس immutable هست و نباید به هیچ وجه تغییر کند. باید ابتدا از state فعلی کپی بگیریم، بعد شی جدید را تغییر دهیم و آنرا به عنوان state جدید برگردانیم.

Copy

```
const initialState = { value: 0 }

function counterReducer(state = initialState, action) {
  // Check to see if the reducer cares about this action
  if (action.type === 'counter/increment') {
    // If so, make a copy of `state`
    return {
      ...state,
      // and update the copy with the new value
      value: state.value + 1
    }
  }
  // otherwise return the existing state unchanged
  return state
}
```

Store

- State در store وجود دارد و ذخیره می‌شود. Store متد های کاربردی زیادی دارد. برای مثال با متد getState می‌توان state فعلی برنامه را دید.

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

const store = configureStore({ reducer: counterReducer })

console.log(store.getState())
// {value: 0}
```

Copy

Dispatch

- انبار (store) یک متد به نام dispatch دارد. تنها راه تغییر state در یک انبار، صدا زدن متد dispatch است که به آن به عنوان آرگومان، شی action را می‌دهیم. پس از صدا زده شدن dispatch، در داخل انبار، reducer صدا زده می‌شود و تغییرات لازم در state صورت می‌گیرد و می‌توان با صدا زدن getState، state جدید را دریافت کرد. مثال:

```
store.dispatch({ type: 'counter/increment' })

console.log(store.getState())
// {value: 1}
```

Copy

```
const increment = () => {
  return {
    type: 'counter/increment'
  }
}

store.dispatch(increment())

console.log(store.getState())
// {value: 2}
```

Copy

Selectors

- سلکتورها توابعی هستند که تکه خاصی از state را به ما می‌دهند.
- سلکتور ها زمانی که انبار ما بزرگ می‌شود و از بخش های مختلفی مانند Authentication، shopping cart، theme و ... تشکیل شده است،کمک زیادی به ما می‌کنند.

```
const selectCounterValue = state => state.value

const currentValue = selectCounterValue(store.getState())
console.log(currentValue)
// 2
```

Slice

- یک تکه (slice) مجموعه‌ای از منطق reducer و action ها است که برای یک feature خاص از انبار تعریف می‌شود (مثلا تکه مربوط به authentication یا تکه مربوط به سبد خرید)
- نامش slice یا تکه است زیرا state مارا به چند تکه تقسیم می‌کند. مثلا اگر وبلاگی داشته‌باشیم، مثال:

```
import { configureStore } from '@reduxjs/toolkit'
import usersReducer from '../features/users/usersSlice'
import postsReducer from '../features/posts/postsSlice'
import commentsReducer from '../features/comments/commentsSlice'

export default configureStore({
  reducer: {
    users: usersReducer,
    posts: postsReducer,
    comments: commentsReducer
  }
})
```

Copy

جریان داده‌ها در redux

- Redux جریان داده یک طرفه دارد. یعنی به طور دقیق می‌توان گفت :

- State وضعیت برنامه را در یک زمان خاص نشان می‌دهد.
- ظاهر یا UI برنامه بر اساس این state، رندر می‌شود.
- وقتی که اتفاقی می‌افتد، مثلا دکمه‌ای زده میشود، state براساس آن عمل آپدیت می‌شود.
- ظاهر یا UI برنامه براساس state جدید آپدیت می‌شود.

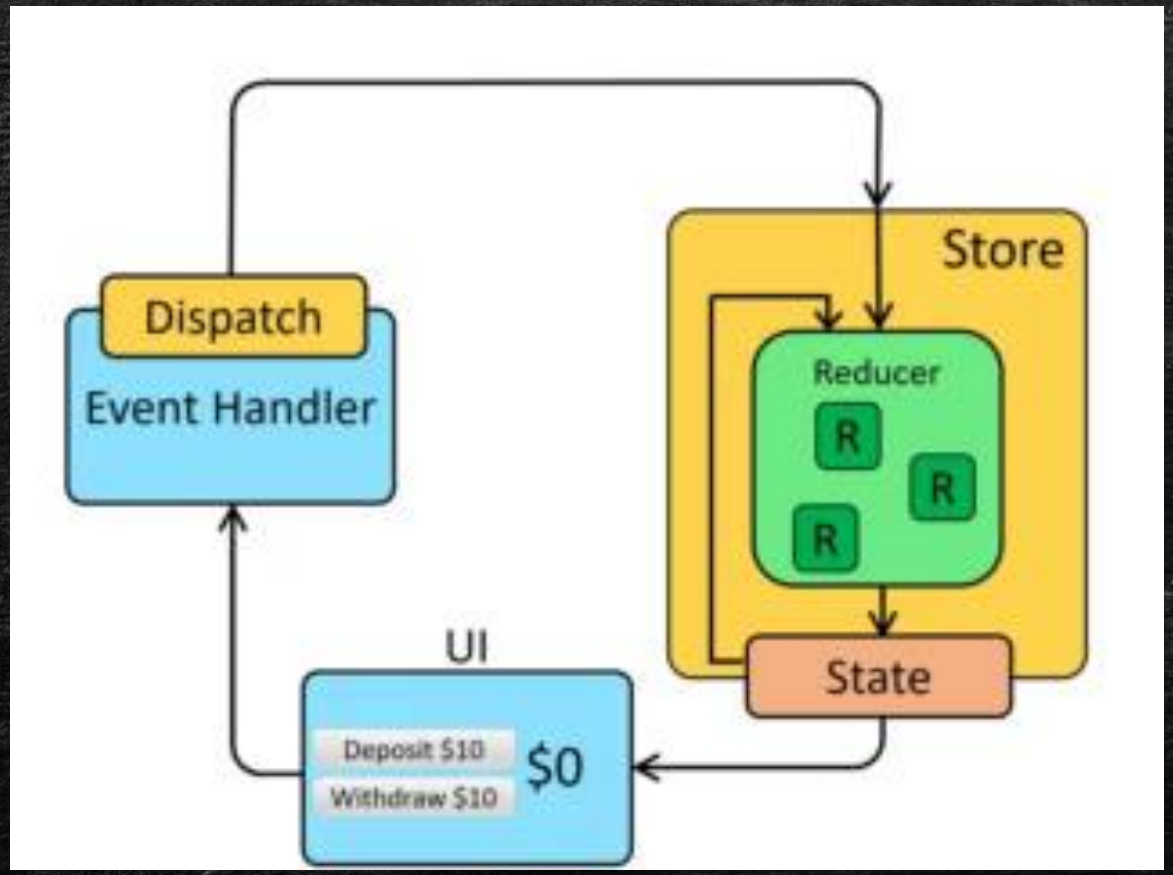
راه‌اندازی اولیه

- با استفاده از reducer ریشه، انبار را می‌سازیم.
- انبار به طور اتوماتیک، reducer را صدا می‌زند و initial state را مساوی مقداری که خودمان مشخص کرده‌ایم، می‌گذارد.
- زمانی که UI می‌خواهد برای بار اول رندر شود، به store دسترسی پیدا می‌کند و از مقادیر آن استفاده می‌کند. سپس به مقادیری که بهشان وابسته است، subscribe (مشترک شدن) می‌کند تا از تغییرات state، باخبر شود.

بروزرسانی

- اتفاقی در برنامه می‌افتد (مثلا بر روی دکمه‌ای زده می‌شود، یا نوتیفیکیشن از طرف backend فرستاده می‌شود.)
- کدی در برنامه، برحسب اتفاقی که افتاده‌است، یک عملیات (action) را dispatch می‌کند.
- انبار، Reducer را صدا می‌زند و به آن action و state فعلی را می‌دهد.
  Reducer با توجه به دو ورودیش، state جدید را می‌سازد و برمی‌گرداند و state انبار، تغییر می‌کند.
- انبار به همه subscriber ها اطلاع می‌دهد که انبار تغییر کرده‌است.
- هر بخش از UI که به اطلاعات انبار وابسته است، بررسی می‌کند که بخشی که برایش مهم است تغییر کرده‌است یا خیر.
- هر بخشی از UI که اطلاعاتش تغییر کرده‌است، rerender می‌شود.

Redux Data Flow



مرور Reducer

1. باید state جدید فقط به state قبلی و action وابسته باشد.
2. نباید state فعلی را تغییر دهند و باید حتما state جدیدی بسازند.
3. نباید منطق async داشته باشیم و همچنین side effect ها هم مجاز نیستند.

Immer

- یکی از مزایای اصلی redux toolkit این است که می‌توان در reducer ها state را تغییر داد یا mutate کرد. این کتابخانه از Immer استفاده می‌کند. Immer از نوعی proxy استفاده می‌کند و از state اصلی محافظت می‌کند. تغییراتی که در کد به state داده می‌شود را ذخیره می‌کند و در نهایت، کپی از state می‌سازد و تغییراتی که ذخیره کرده‌است را می‌دهد و به ما برمی‌گرداند. پس می‌توانیم در reducer هایی که در createSlice یا createReducer می‌سازیم، کد mutable بنویسیم.
- برای اینکه کاربرد Immer را بهتر متوجه شوید، این مثال را ببینید.

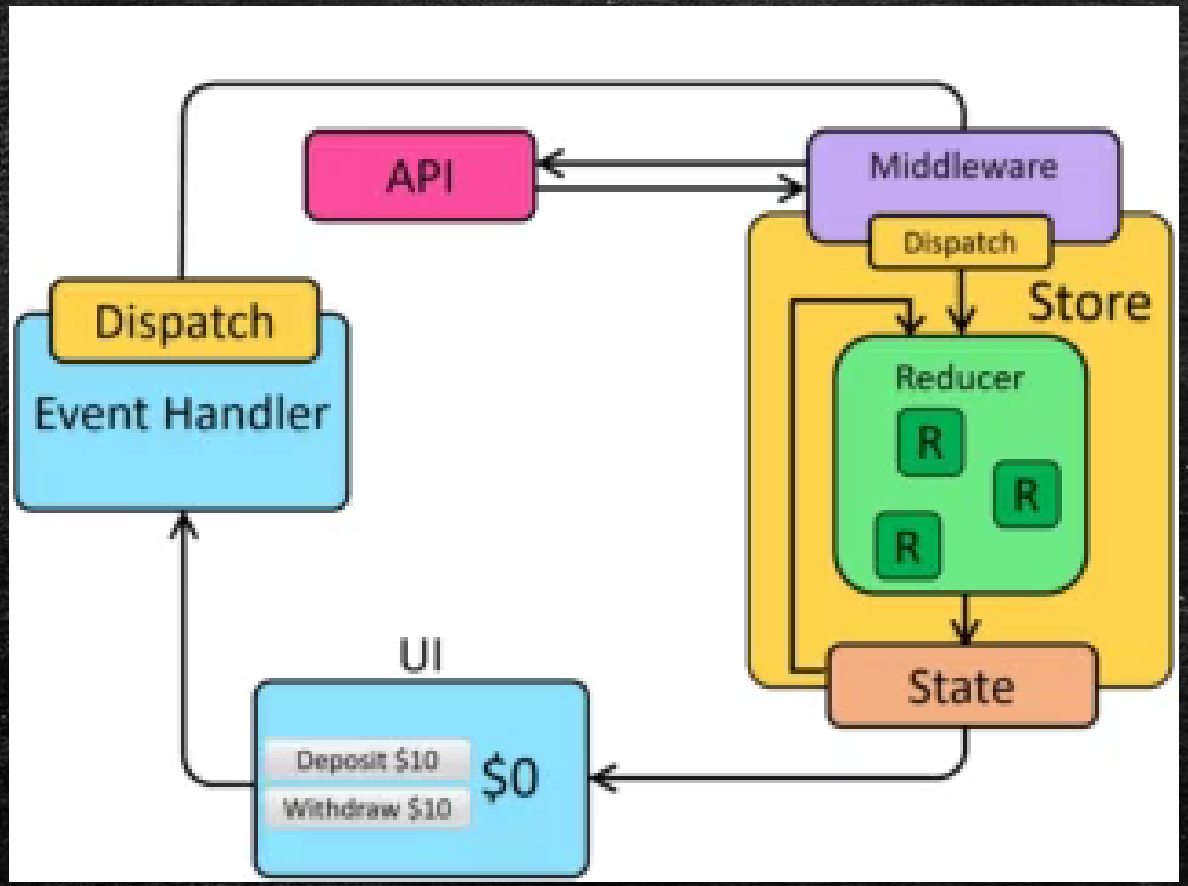
بدون Immer

```
function handwrittenReducer(state, action) {
  return {
    ...state,
    first: {
      ...state.first,
      second: {
        ...state.first.second,
        [action.someId]: {
          ...state.first.second[action.someId],
          fourth: action.someValue
        }
      }
    }
  }
}
```

با Immer

```
function reducerWithImmer(state, action) {
  state.first.second[action.someId].fourth = action.someValue
}
```

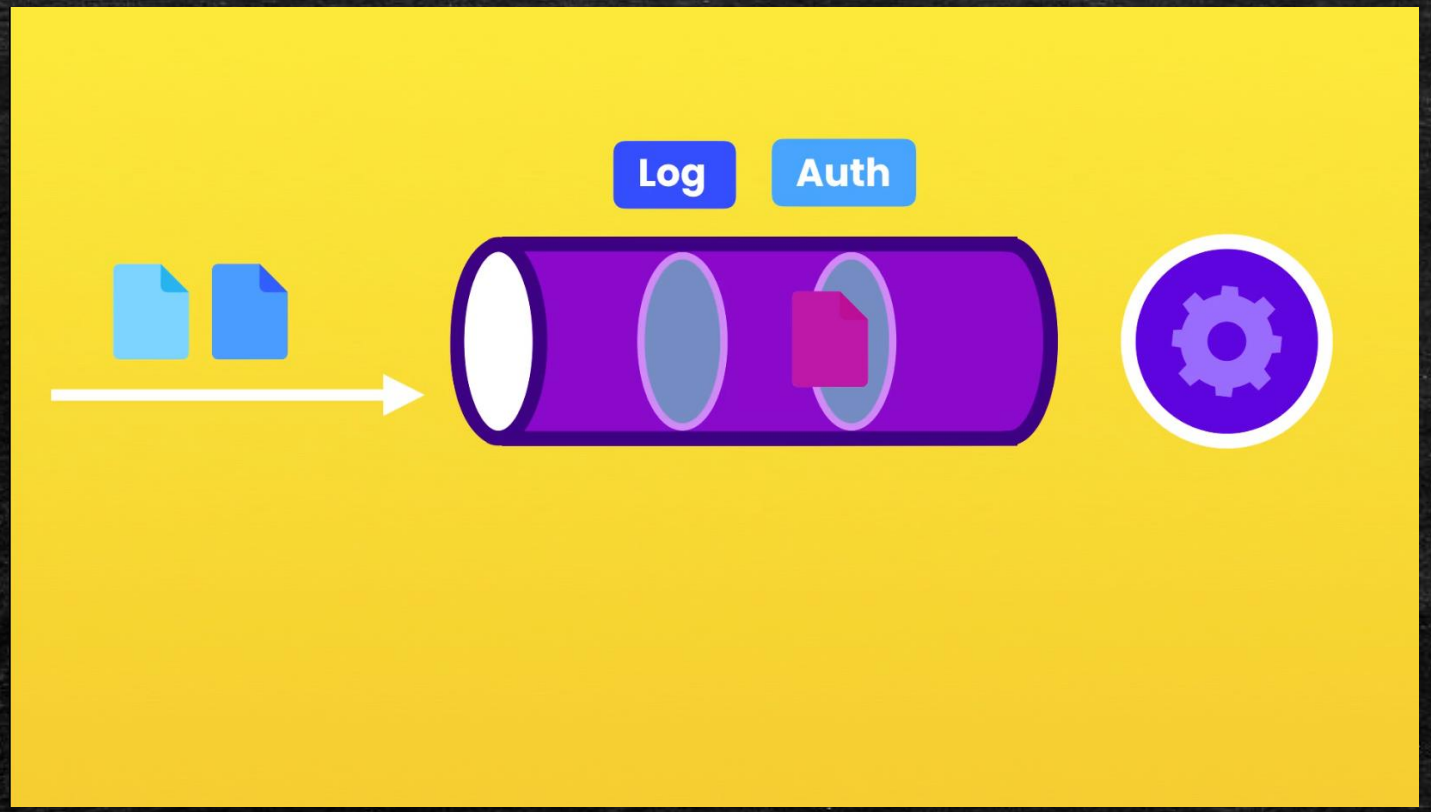
Middleware



قبلا گفتیم که پس از ارسال action، این شی به reducer مربوطه داده می‌شود و در صورت نیاز، state جدیدی تولید می‌شود. می‌توان بر سر راه action، middleware قرار داد.

با middleware ها می‌توان کارهای زیادی کرد. مثلا عملیات async انجام داد یا از همه action ها لاگ گرفت و ... . ظاهر برنامه اگر middleware به آن اضافه‌شود:

مثالی از middle ware



اضافه‌کردن middleware به Store

```
import { createStore, combineReducers, applyMiddleware } from 'redux'

const todoApp = combineReducers(reducers)
const store = createStore(
  todoApp,
  // applyMiddleware() tells createStore() how to handle middleware
  applyMiddleware(logger, crashReporter)
)
```

Copy

کتابخانه‌های کاربردی در استفاده از Redux

- Redux-thunk: با استفاده از redux-thunk می‌توانید action creator هایی بسازید که بجای action، function برمی‌گردانند. کاربرد های زیادی دارد مثلا می‌توان از این کتابخانه برای پیاده‌سازی منطق async استفاده کرد.
- Redux-saga: با استفاده از این کتابخانه میتوان منطق Async پیچیده و side effect هارا مدیریت کرد.
- Redux-persist: اطلاعات store به طور دائم ذخیره نمی‌شوند. یعنی مثلا در وبسایتی که از redux استفاده شده‌است، اگر صفحه را refresh کنیم یا ببندیم و دوباره باز کنیم، اطلاعات Store پاک می‌شود. با استفاده از این کتابخانه می‌توان اطلاعات را در Storage مربوطه به پلتفرم ذخیره کرد. مثلا در وبسایت ها می‌توان در local storage و در برنامه react native در Async Storage اطلاعات را ذخیره کرد. استفاده از این کتابخانه بسیار راحت است. این کتابخانه بر reducer ما تاثیر می‌گذارد و به middleware ها وابسته نیست. توجه کنید که نیاز نیست همه اطلاعات store حتما در storage ذخیره شوند و می‌توان مشخص کرد که اطلاعات هر reducer چگونه مدیریت شود.

- برای مشاهده موارد بیشتر می‌توانید به https://redux.js.org/introduction/ecosystem مراجعه کنید.