15.02.2023

Redux 1

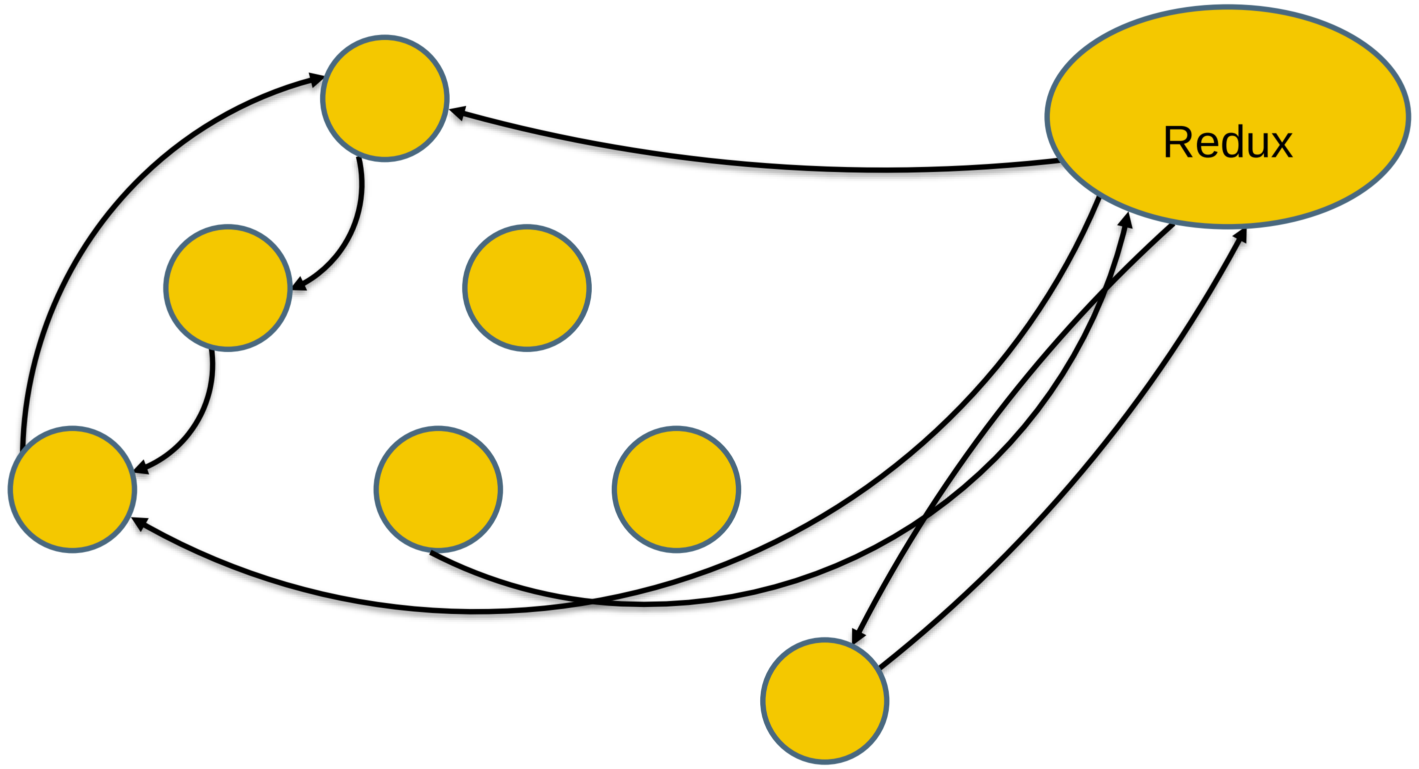
Valentins Jegorovs

Topics for today:

- General overview
- Store
- Actions
- Reducers
- Selectors
- Devtools

Tired of having to do this?

Try Redux!

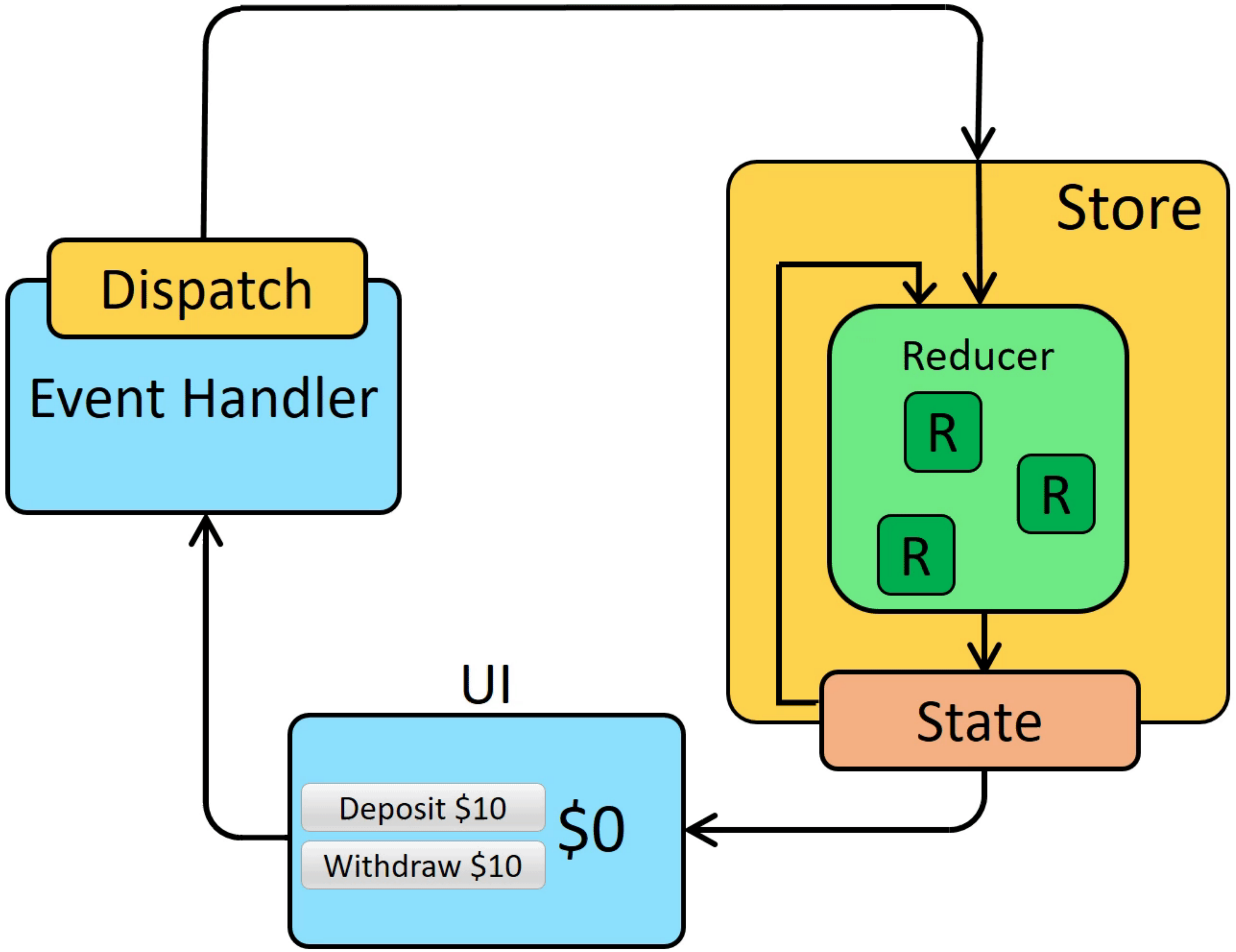


What is “Redux”?

Base:
- Single global state
- Action objects dispatched once something changes
- Pure reducer function that determines the next state based on dispatched actions

Can also include:
- Action creator functions
- Middlewares for side-effects
- Thunk functions for business logic, async stuff and/or side-effects
- Selector functions, memoized selector functions…
- Devtools extension that allows time-bending

Source:
https://redux.js.org/introduction/why-rtk-is-redux-today#how-redux-toolkit-is-different-than-the-redux-core

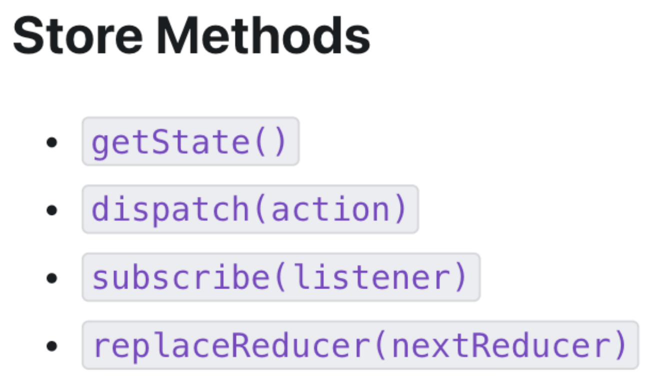

Store
Dispatch
Event Handler
Reducer
R
R
R
State
UI
Deposit $10
Withdraw $10
$0

Store

A store holds the whole **state** tree of your application.

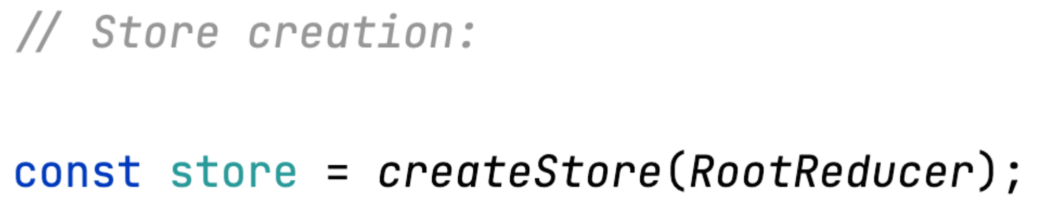
The only way to change the state inside it is to dispatch an action on it.

A store is not a class. It's just an object with a few methods on it. To create it, pass your root reducing function to createStore

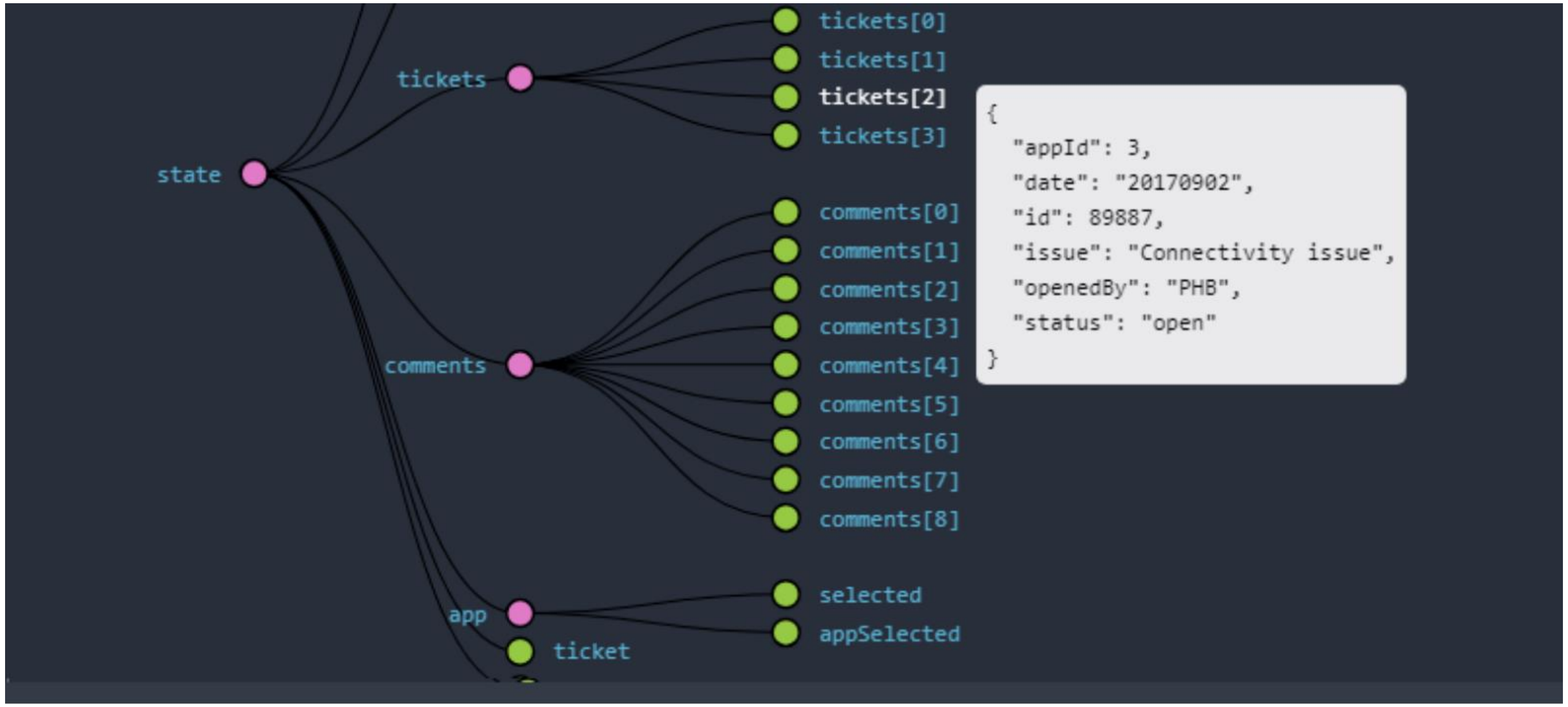


Store Methods

- `getState()`
- `dispatch(action)`
- `subscribe(listener)`
- `replaceReducer(nextReducer)`



```
// Store creation:

const store = createStore(RootReducer);
```



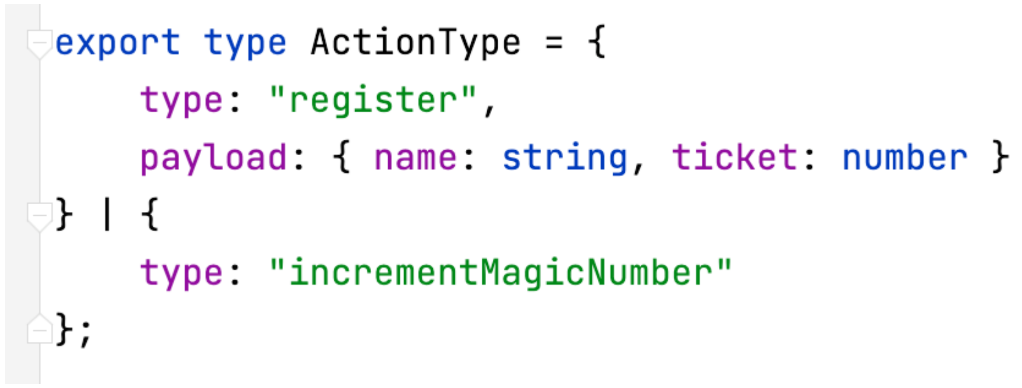
```
{
  "appId": 3,
  "date": "20170902",
  "id": 89887,
  "issue": "Connectivity issue",
  "openedBy": "PHB",
  "status": "open"
}
```

Actions

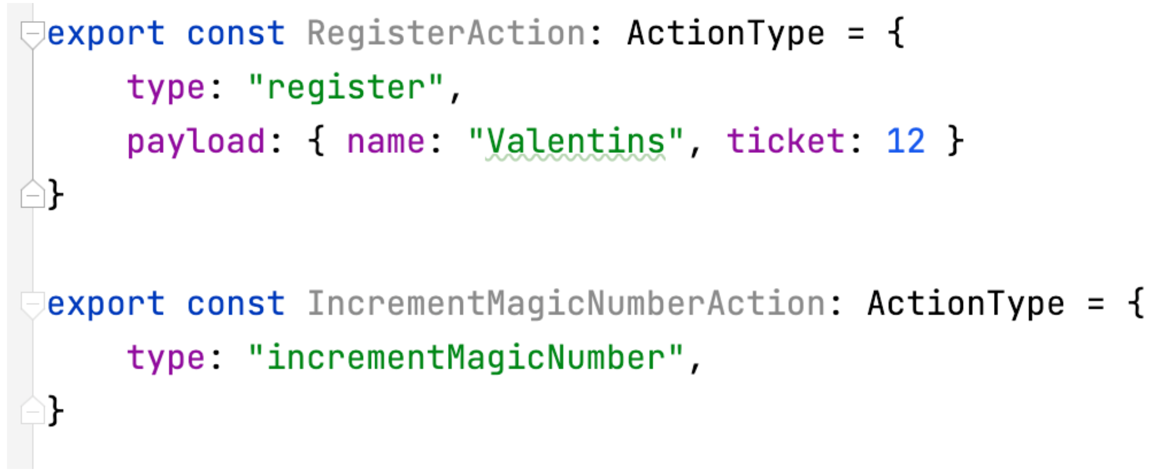
An *action* is a plain object that represents an intention to change the state. Actions are the only way to get data into the store. Any data, whether from UI events, network callbacks, or other sources such as WebSockets needs to eventually be dispatched as actions.

Actions must have a *type* field that indicates the type of action being performed.

Other than *type*, the structure of an action object is really up to you.



```
export type ActionType = {
    type: "register",
    payload: { name: string, ticket: number }
} | {
    type: "incrementMagicNumber"
};
```

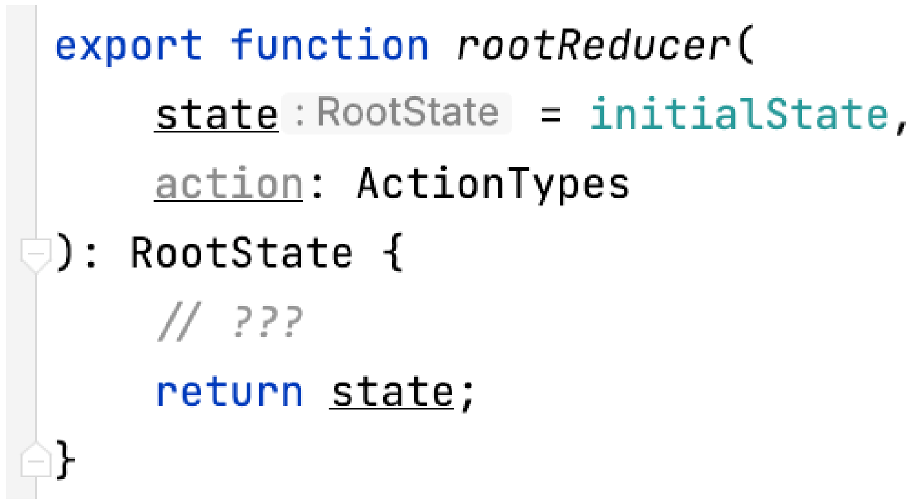


```
export const RegisterAction: ActionType = {
    type: "register",
    payload: { name: "Valentins", ticket: 12 }
}

export const IncrementMagicNumberAction: ActionType = {
    type: "incrementMagicNumber",
}
```

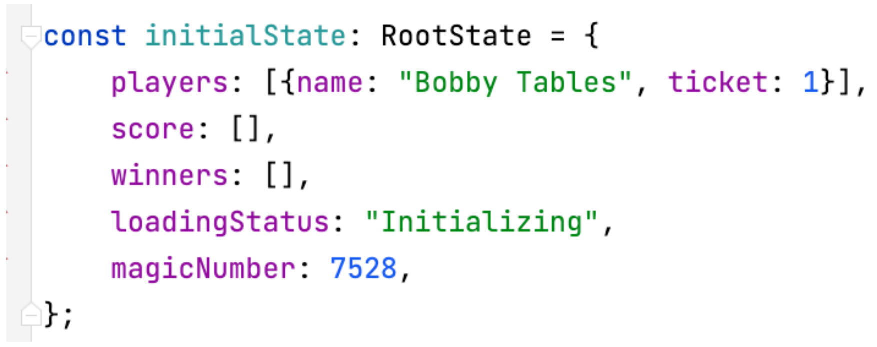
Reducer

In Redux, the accumulated value is the state object, and the values being accumulated are actions. Reducers calculate a new state given the previous state and an action.

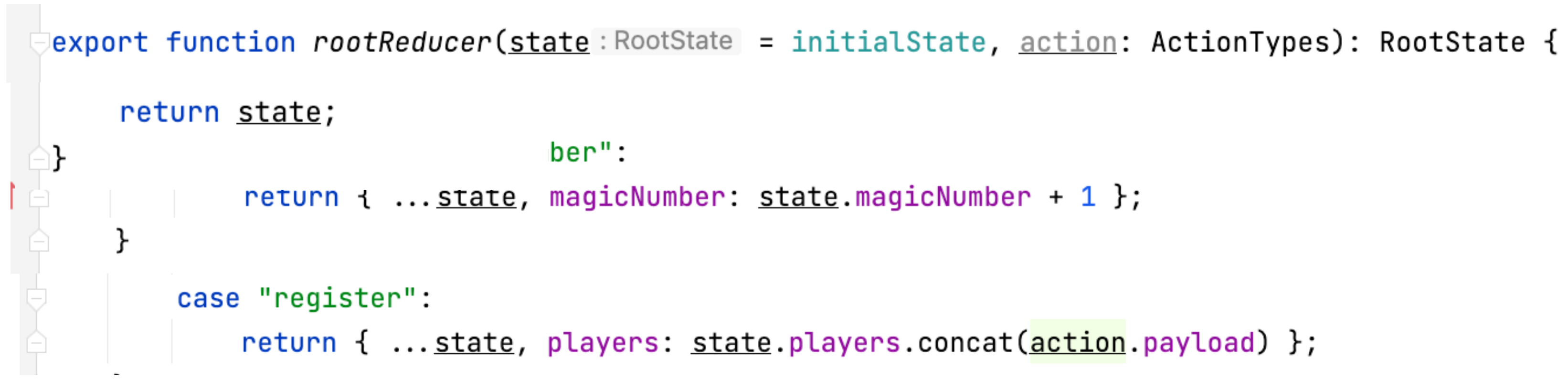
A *reducer* (also called a *reducing function*) is a function that accepts an accumulation and a value and returns a new accumulation. They are used to reduce a collection of values down to a single value. Reducers must be *pure functions*—functions that return the exact same output for given inputs. They should also be free of side-effects. This is what enables exciting features like hot reloading and time travel.



```
export function rootReducer(
    state: RootState = initialState,
    action: ActionTypes
): RootState {
    // ???
    return state;
}
```



```
const initialState: RootState = {
    players: [{name: "Bobby Tables", ticket: 1}],
    score: [],
    winners: [],
    loadingStatus: "Initializing",
    magicNumber: 7528,
};
```



```
export function rootReducer(state : RootState = initialState, action: ActionTypes): RootState {

    return state;
}                                    ber":
                return { ...state, magicNumber: state.magicNumber + 1 };
    }
            case "register":
                return { ...state, players: state.players.concat(action.payload) };
```

Do’s:

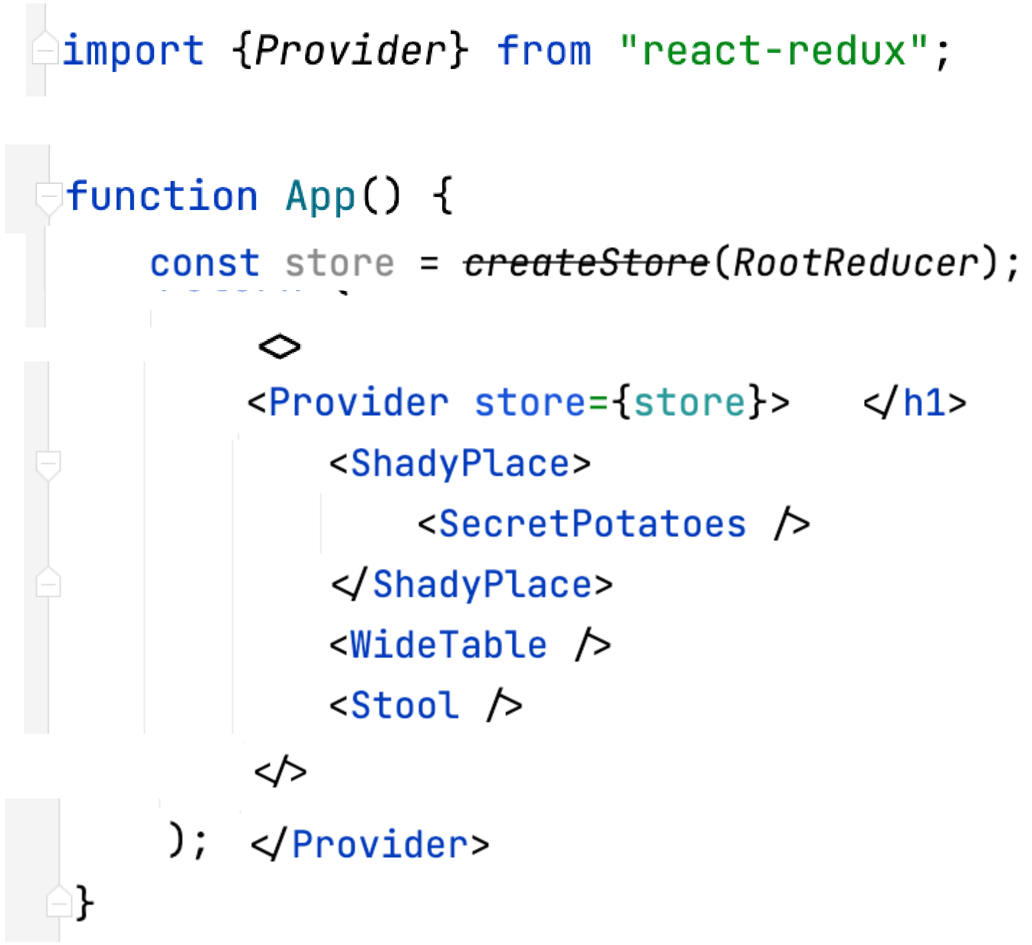
- ✓ Pure function
- ✓ Create new state

Don’ts:

- ✗ No side-effects
- ✗ including API calls
- ✗ No mutating state

www.evolution.com

React



```
import {Provider} from "react-redux";

function App() {
    const store = ~~createStore~~(RootReducer);
            <>
            <Provider store={store}>    </h1>
                <ShadyPlace>
                    <SecretPotatoes />
                </ShadyPlace>
                <WideTable />
                <Stool />
            </>
        );  </Provider>
}
```

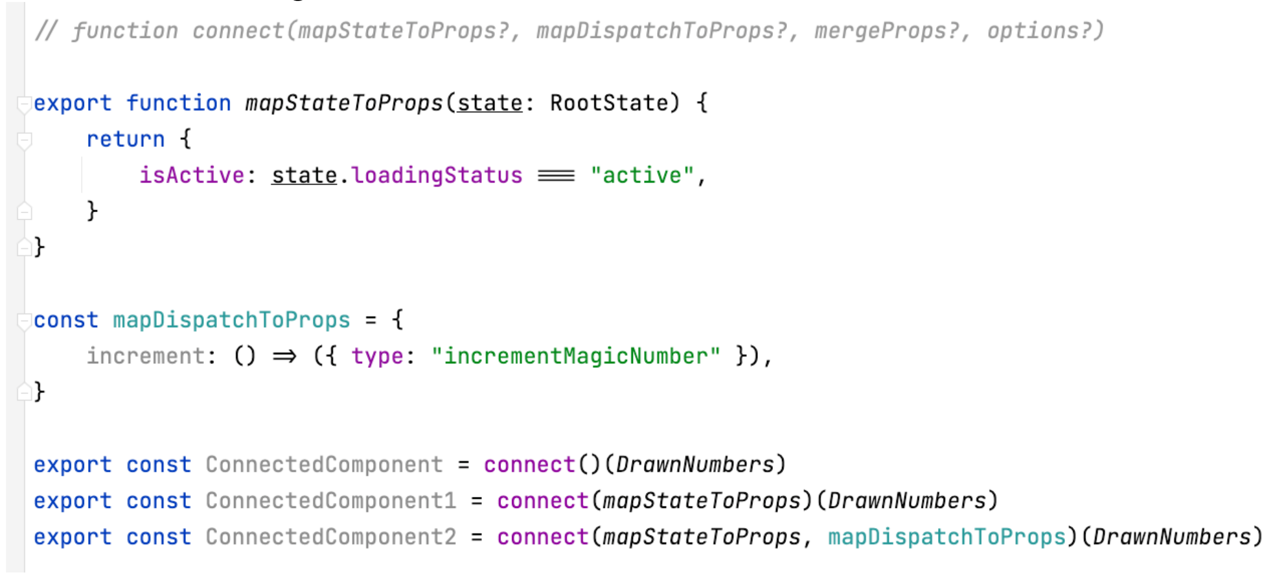
Evolution Engineering | ACCELERATING EVOLUTION

www.evolution.com

Reading state

Old:*

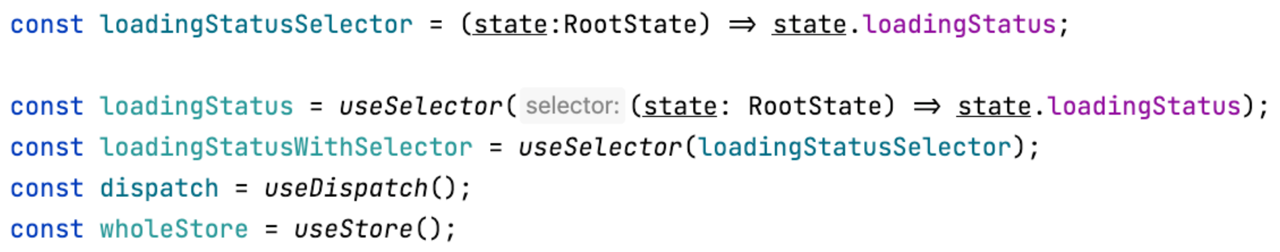
using *connect* from *‘react-redux’*



```
// function connect(mapStateToProps?, mapDispatchToProps?, mergeProps?, options?)

export function mapStateToProps(state: RootState) {
    return {
        isActive: state.loadingStatus === "active",
    }
}

const mapDispatchToProps = {
    increment: () => ({ type: "incrementMagicNumber" }),
}

export const ConnectedComponent = connect()(DrawnNumbers)
export const ConnectedComponent1 = connect(mapStateToProps)(DrawnNumbers)
export const ConnectedComponent2 = connect(mapStateToProps, mapDispatchToProps)(DrawnNumbers)
```

Modern:

using *hooks API* from *‘react-redux’*



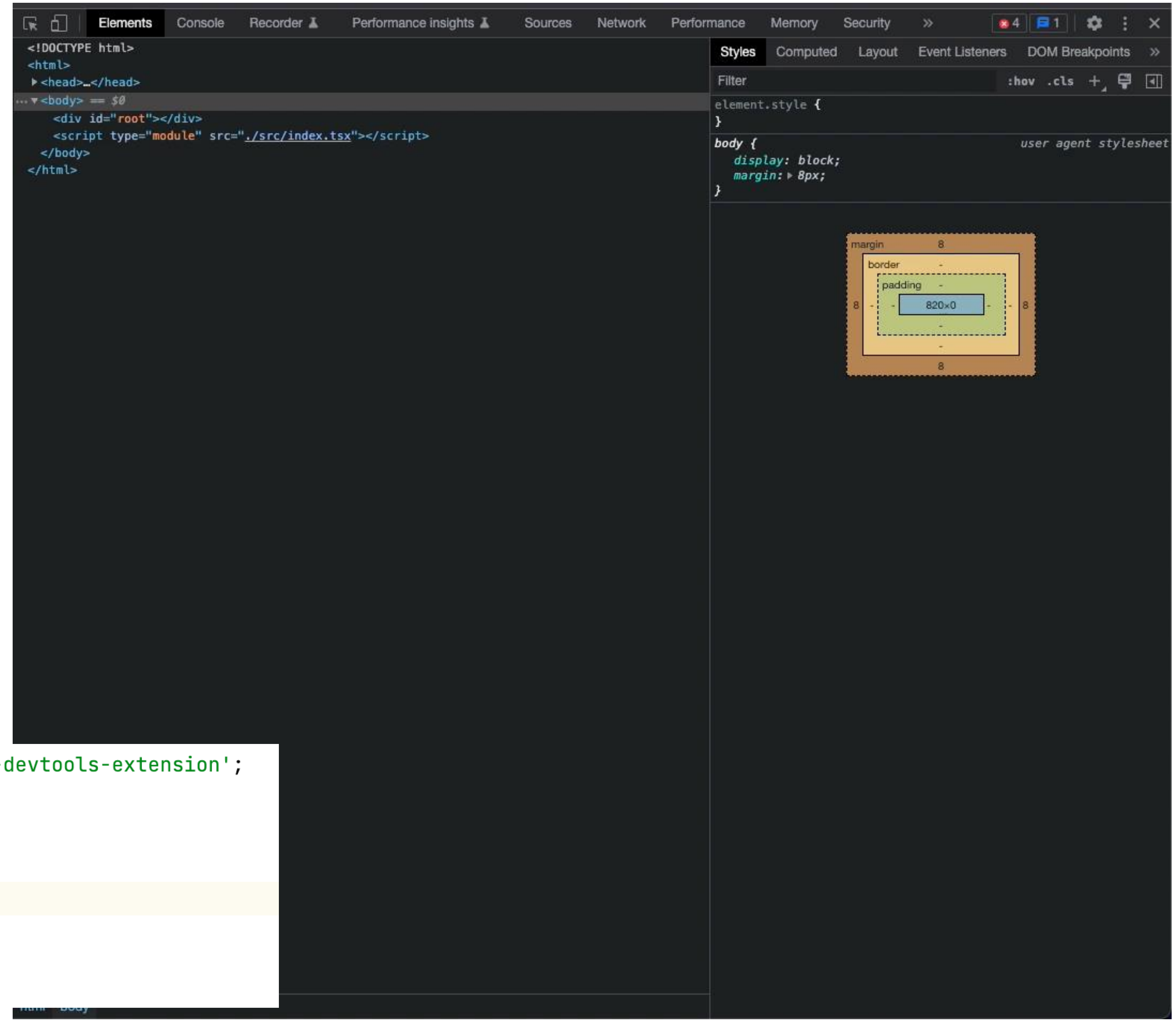
```
const loadingStatusSelector = (state:RootState) => state.loadingStatus;

const loadingStatus = useSelector( selector: (state: RootState) => state.loadingStatus);
const loadingStatusWithSelector = useSelector(loadingStatusSelector);
const dispatch = useDispatch();
const wholeStore = useStore();
```

connect() API: https://react-redux.js.org/api/connect

Hooks API: https://react-redux.js.org/api/hooks

Redux Devtools

Source: https://github.com/reduxjs/redux-devtools



```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body> == $0
    <div id="root"></div>
    <script type="module" src="./src/index.tsx"></script>
  </body>
</html>
```

```
element.style {
}
body {                                  user agent stylesheet
   display: block;
   margin: ▶ 8px;
}
```

```
import { composeWithDevTools } from 'redux-devtools-extension';

function App() {
    const store = ~~createStore~~(
        RootReducer,
        composeWithDevTools()
    );
```

HOMEWORK

Evolution Engineering | ACCELERATING EVOLUTION