What is Redux?

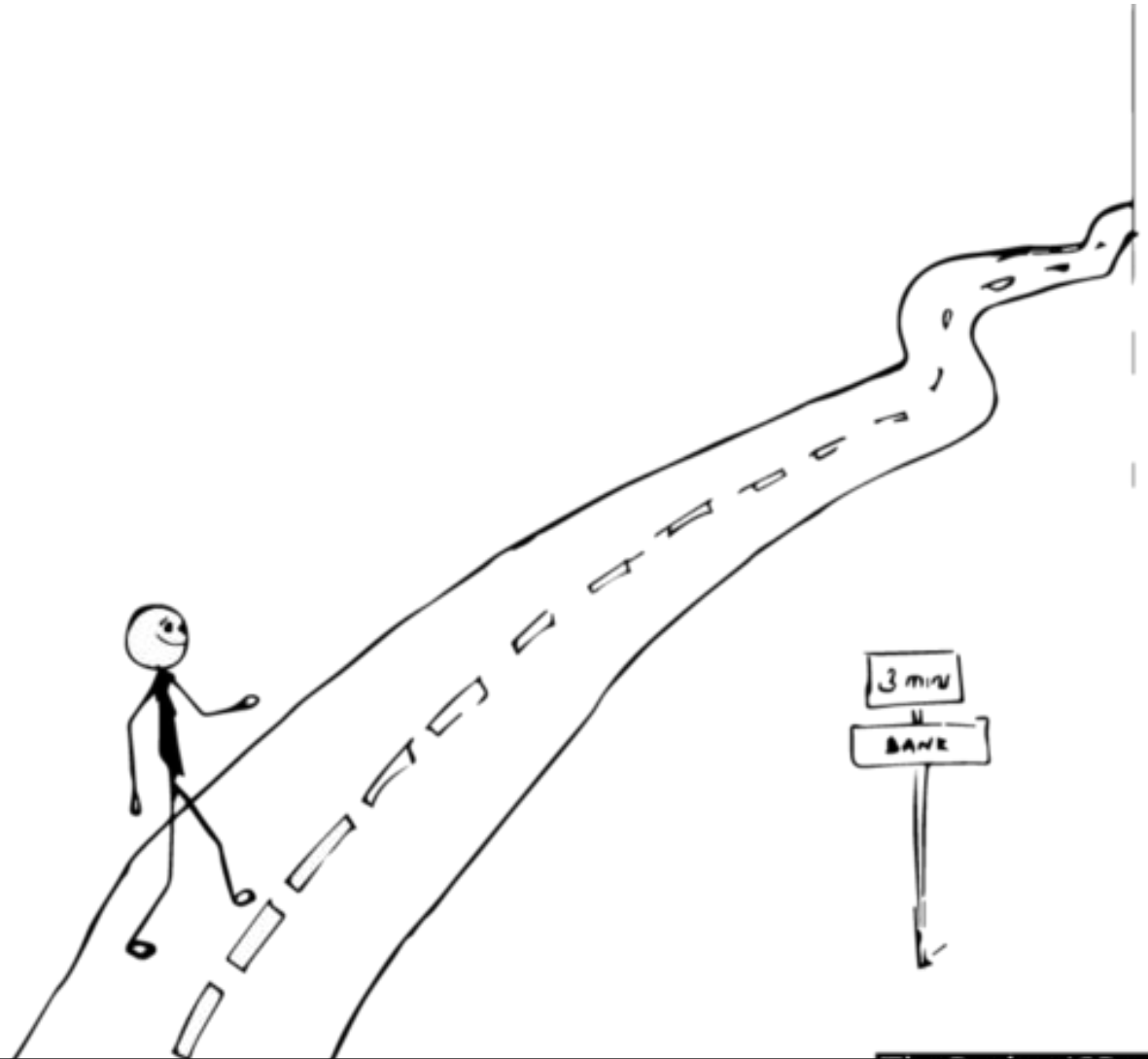
Redux is a predictable state container for JavaScript apps.

Why use Redux?

- As you already know, questions like “Why should you use A over B?” boil down to your personal preferences.
- I have built apps in production that don’t use Redux. I’m sure that many have done the same.
- For example, in larger apps with a lot of moving pieces, state management becomes a huge concern. Redux ticks that off quite well without performance concerns or trading off testability.

Explaining Redux to a 5 year Old

- Let’s consider an event you’re likely familiar with — going to the bank to withdraw cash. Even if you don’t do this often, you’re likely aware of what the process looks like.


3 min
BANK

You wake up one morning, and head to the bank as quickly as possible. While going to the bank there's just one intention / action you've got in mind: to WITHDRAW_MONEY.

When you get into the bank, you then go straight to the Cashier to make your request known.

Wait, you went to the Cashier?

Why didn’t you just go into the bank vault to get your money?

If only Young Joe got into the Vault. He’ll cart away with as much as he finds.

After all, it’s your hard earned money.

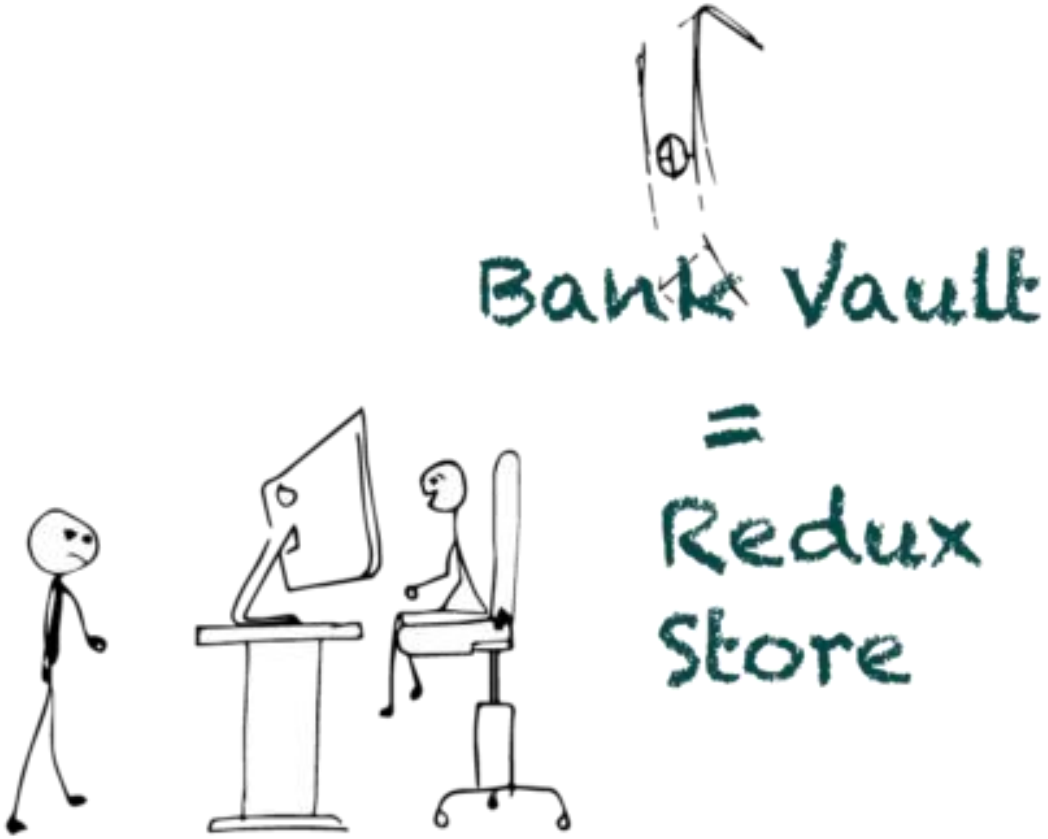
Well, like you already know, things don’t work that way. Yes, the bank has money in the vault, but you have to talk to the Cashier to help you follow a due process for withdrawing your own money.

The Cashier, from their computer, then enters some commands and delivers your cash to you. Easy-peasy.

Here’s how you get money. Not from the Vault, sorry.

Now, how does Redux fit into this story?

The Bank Vault is to the bank what the Redux Store is to Redux.



- The bank vault keeps the money in the bank, right?

- Well, within your application, you don’t spend money. Instead, the state of your application is like the money you spend. The entire user interface of your application is a function of your state.

- Just like the bank vault keeps your money safe in the bank, the state of your application is kept safe by something called a store. So, the store keeps your “money” or state intact.

The Redux Store can be likened to the Bank Vault. It holds the state of your application — and keeps it safe.

This leads to the first Redux principle:

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

The Redux principle #1.

ONE application STATE OBJECT
managed by ONE STORE

2. Go to the bank with an action in mind.

- If you're going to get any money from the bank, you're going to have to go in with some intent or action to withdraw money.
- If you just walk into the bank and roam about, no one's going to just give you money. You may even end up been thrown out by the security. Sad stuff.
- The same may be said for Redux.
- Write as much code as you want, but if you want to update the state of your Redux application (like you do with setState in React), you need to let Redux know about that with an action.

Now, this leads to Redux principle #2.

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

What does that mean in plain language?

- When you walk to the bank, you go there with a clear action in mind. In this example, you want to withdraw some money.
- If we chose to represent that process in a simple Redux application, your action to the bank may be represented by an object.
- One that looks like this:

```
{
  type: "WITHDRAW_MONEY",
  amount: "$10,000"
}
```

In the context of a Redux application, this object is called an action! It always has a type field that describes the action you want to perform. In this case, it is WITHDRAW_MONEY.

Whenever you need to change/update the state of your Redux application, you need to dispatch an action.

The Redux principle #2.

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

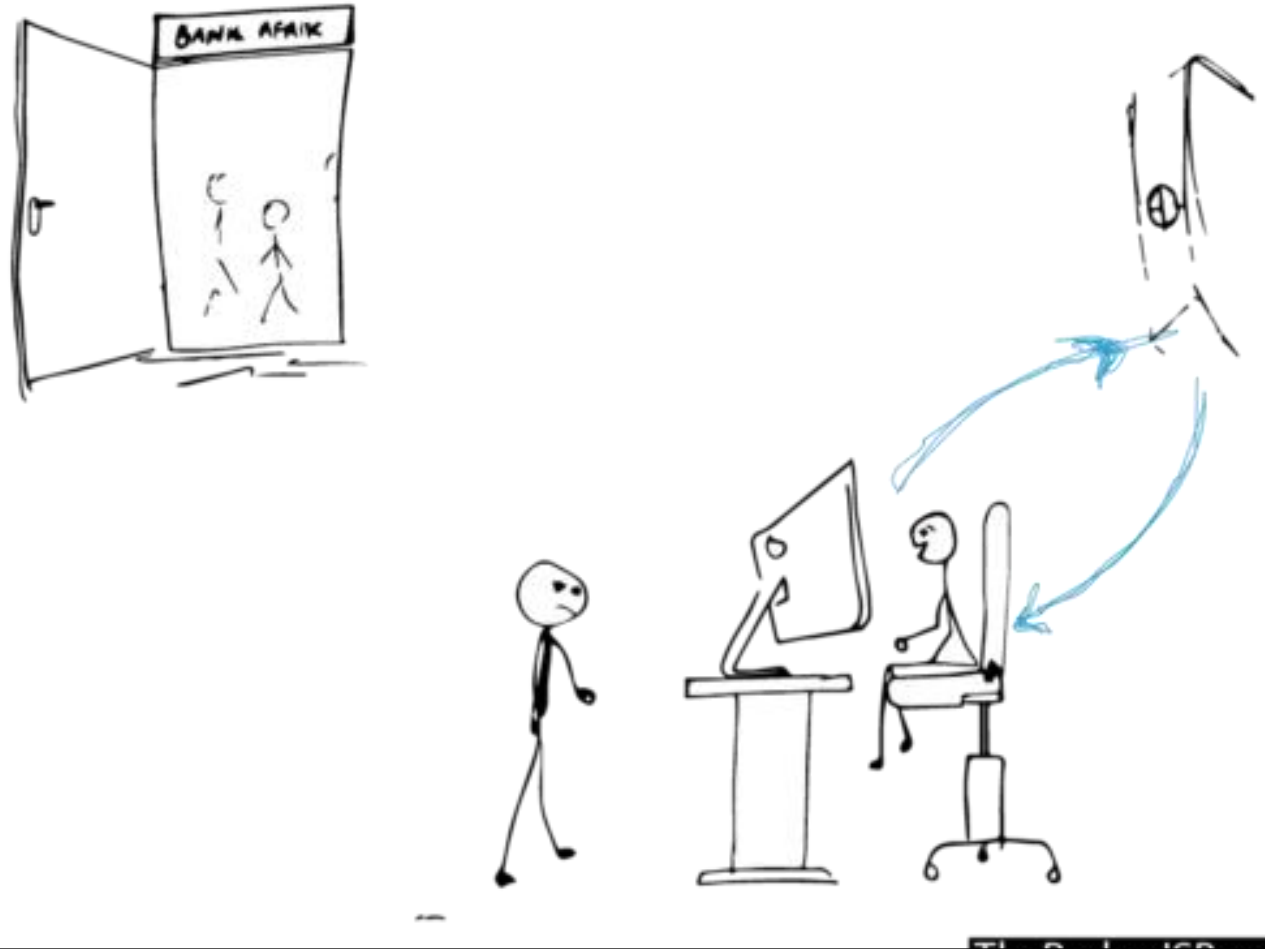
3. The Cashier is to the bank what the reducer is to Redux.

- Alright, take a step back.

- Remember that in the story above, you couldn’t just go straight into the bank vault to retrieve your money from the bank. No. You had to see the Cashier first.

- Well, you had an action in mind, but you had to convey that action to someone — the Cashier — who in turn communicated (in whatever way they did) with the vault that holds all of the bank’s money.

The Cashier and Vault communication!



- The same may be said for Redux.

Like you made your action known to the Cashier, you have to do the same in your Redux application. If you want to update the state of your application, you convey your action to the reducer — our own Cashier.

This process is mostly called dispatching an action.

Dispatch is just an English word. In this example, and in the Redux world, it is used to mean sending off the action to the reducers.

The reducer knows what to do. In this example, it will take your action to WITHDRAW_MONEY and ensure that you get your money.

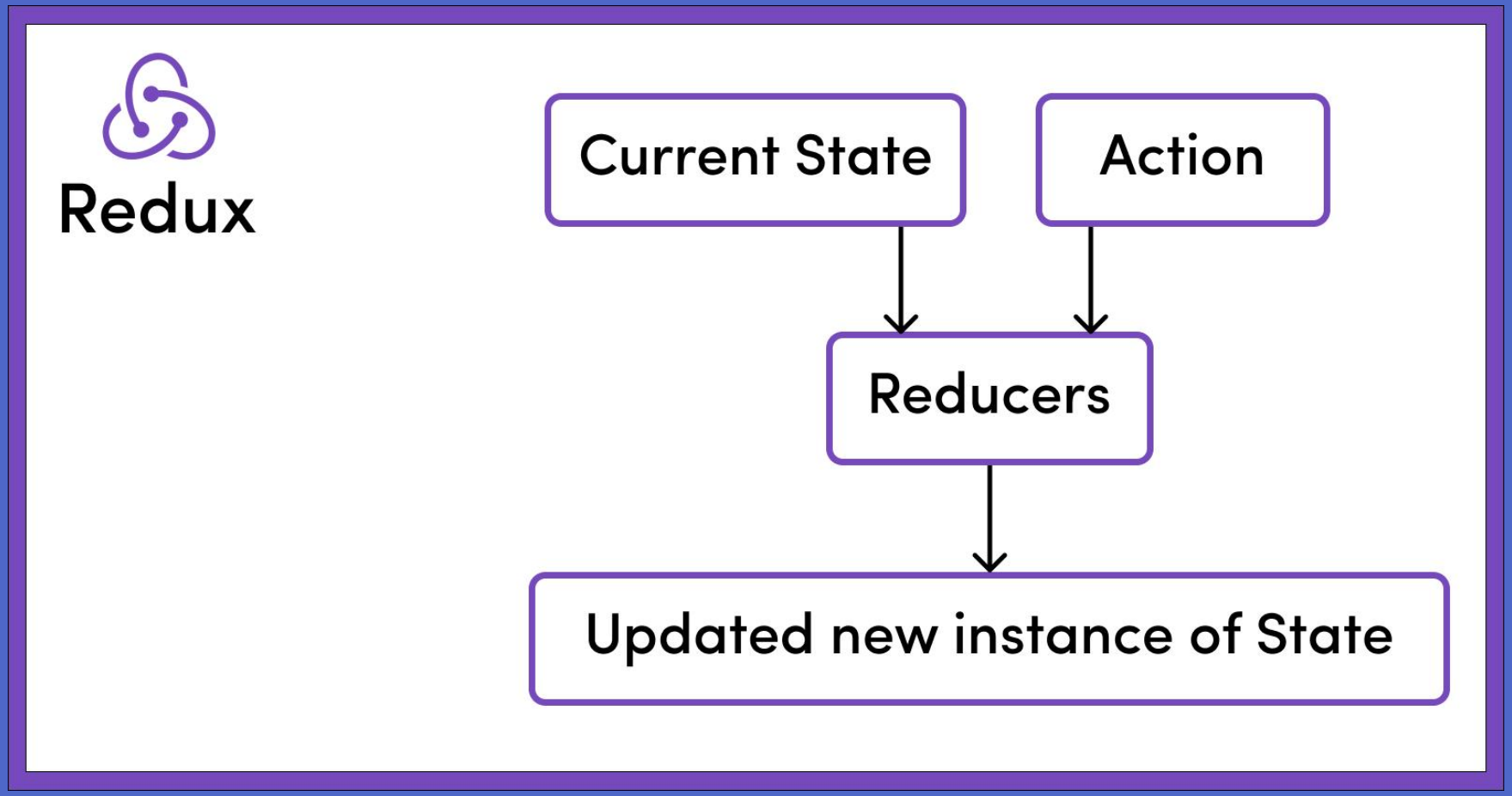
In Redux terms, the money you spend is your state. So, your reducer knows what to do, and it always returns your new state.

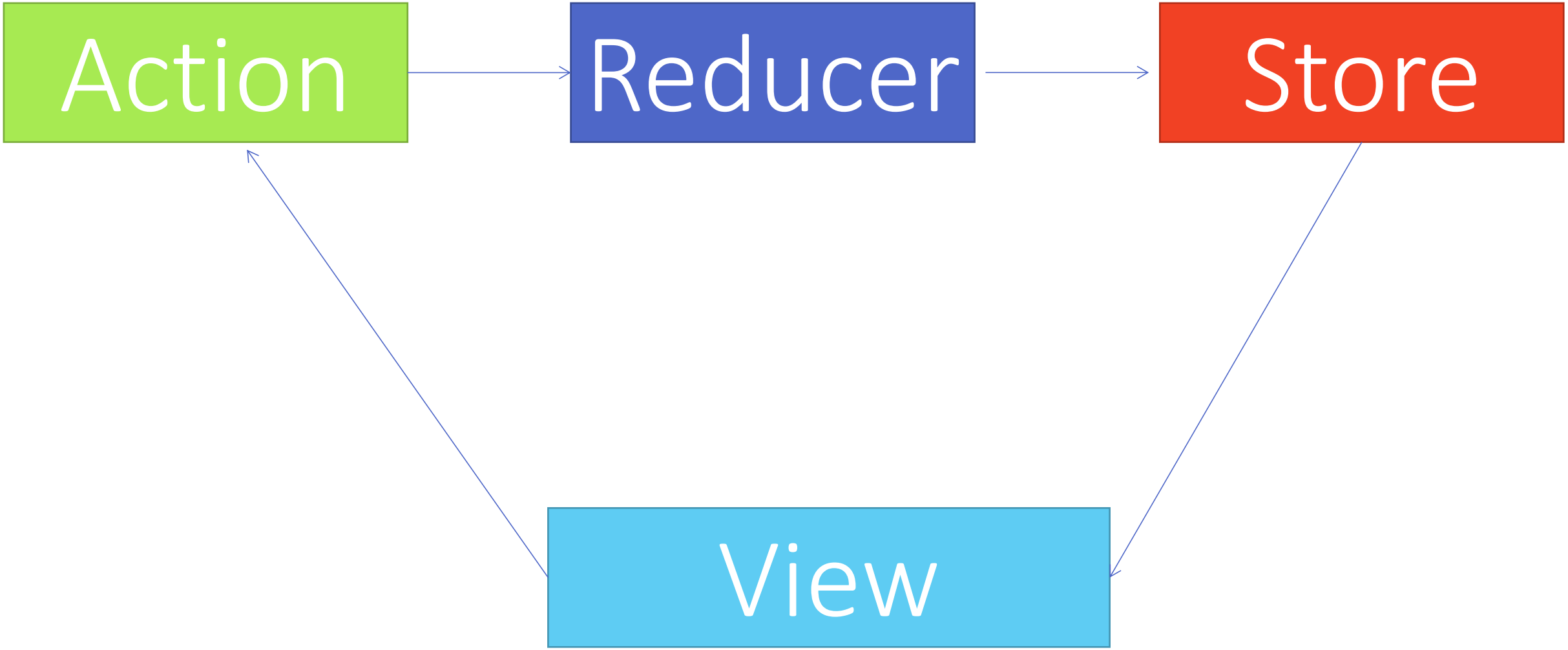
And this leads to the last Redux principle:

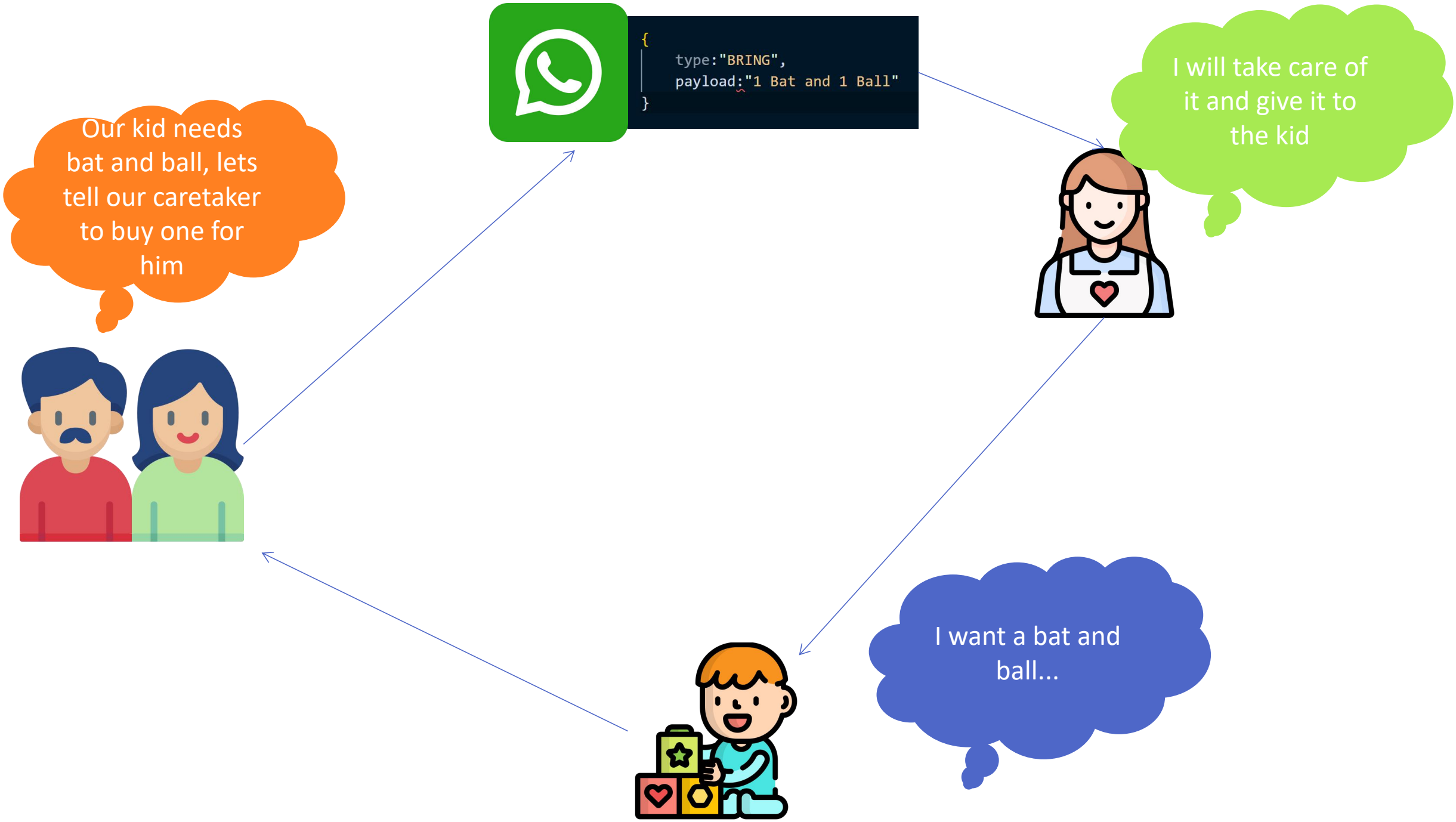
- To specify how the state tree is transformed by actions, you write pure reducers.

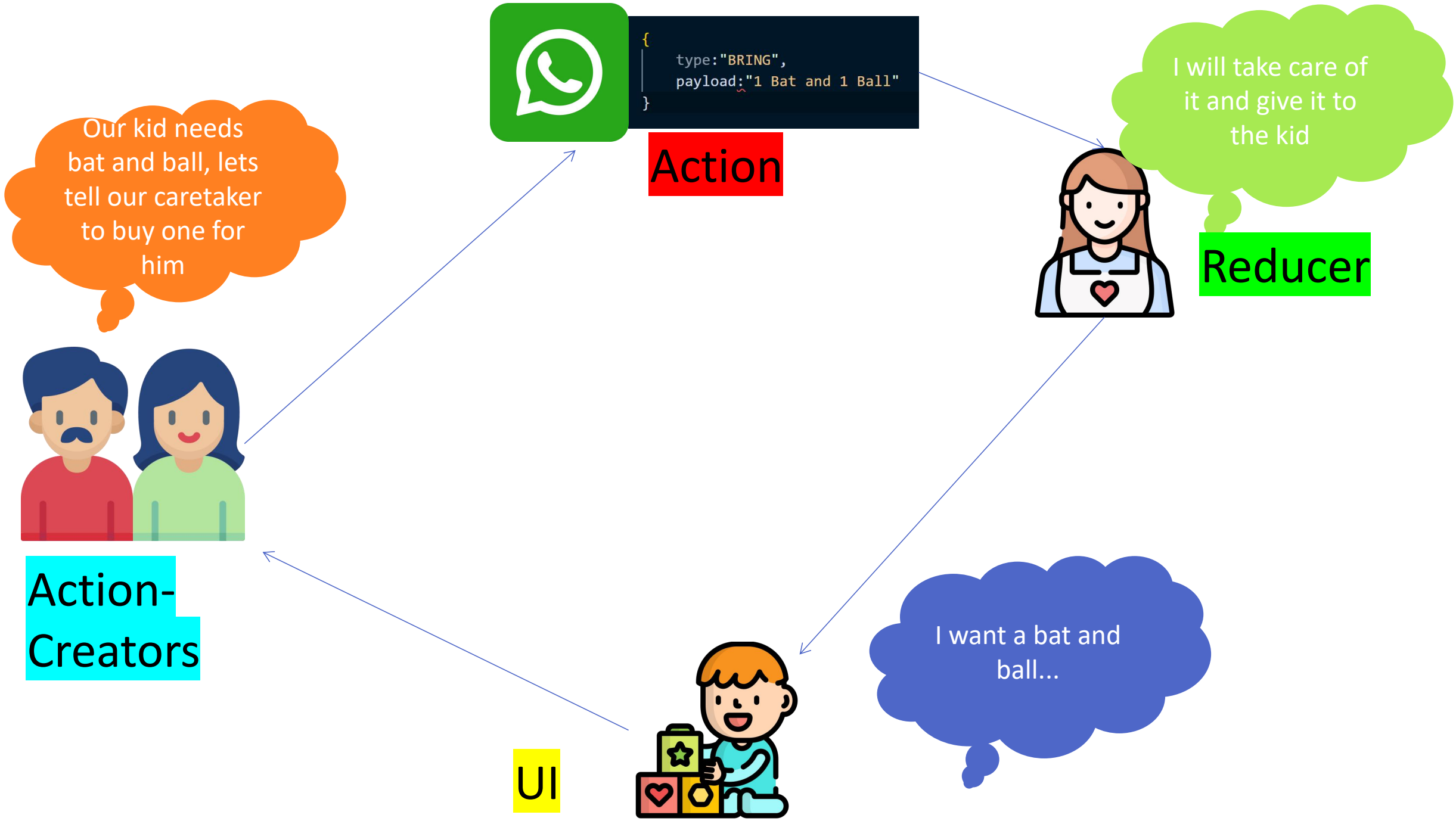
The Redux principle #3.

To specify how the state tree is transformed by actions, you write pure reducers.


Redux
Current State
Action
Reducers
Updated new instance of State


Action
Reducer
Store
View


Our kid needs bat and ball, lets tell our caretaker to buy one for him
{
type:"BRING",
payload:"1 Bat and 1 Ball"
}
I will take care of it and give it to the kid
I want a bat and ball...


{
type:"BRING",
payload:"1 Bat and 1 Ball"
}
Action
I will take care of it and give it to the kid
Reducer
Our kid needs bat and ball, lets tell our caretaker to buy one for him
Action-Creators
I want a bat and ball...
UI

Hum kuch prabandh karte hai....

```
{
    type:"GET",
    payload:"10 guns"
}
```

Hum nipat lenge Kaleen Bhaiya

Humko Katta chahiye Papa...

Hum kuch prabandh karte hai....

Action-Creators

```
{
    type:"GET",
    payload:"10 guns"
}
```

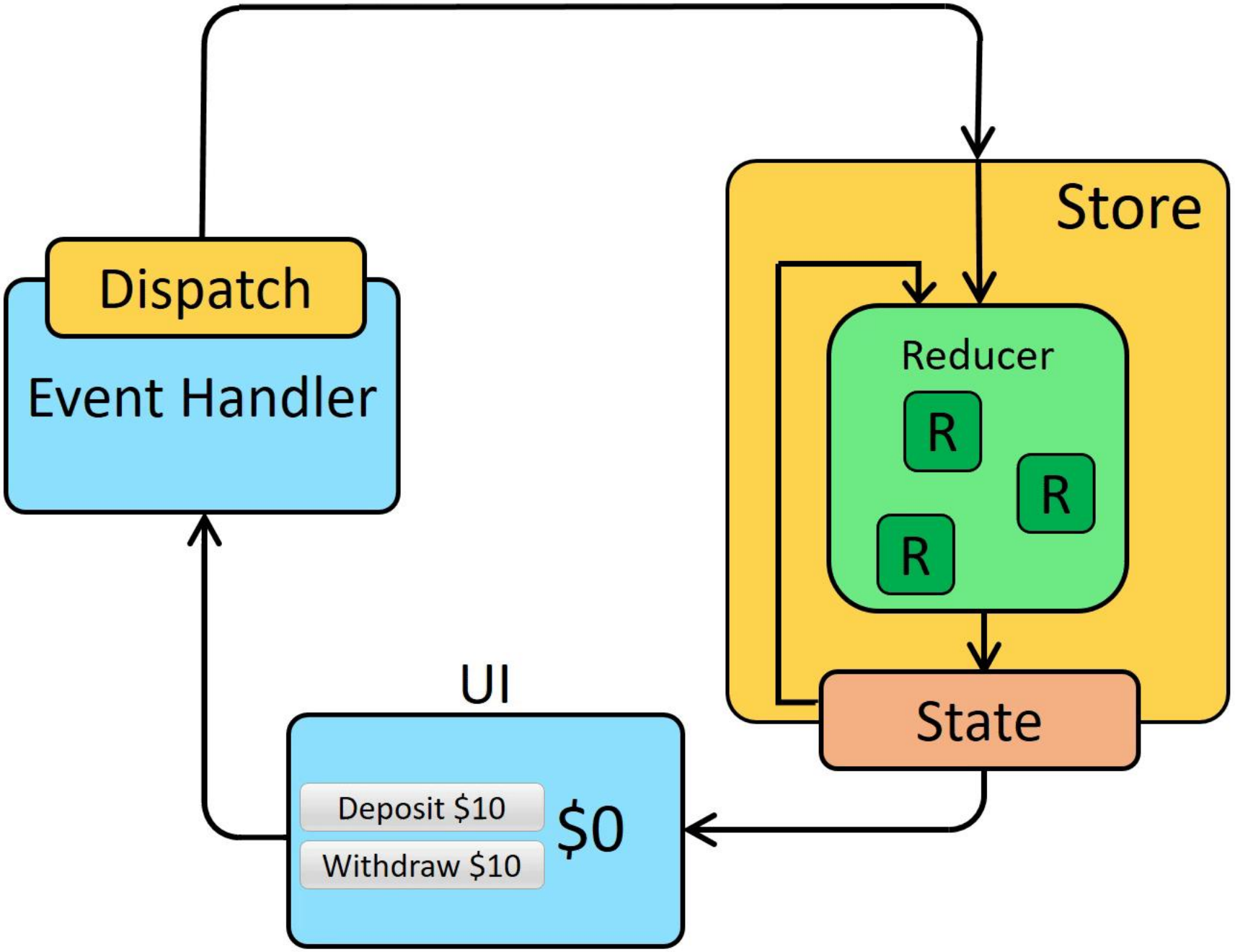
Action

Hum nipat lenge Kaleen Bhaiya

Reducer

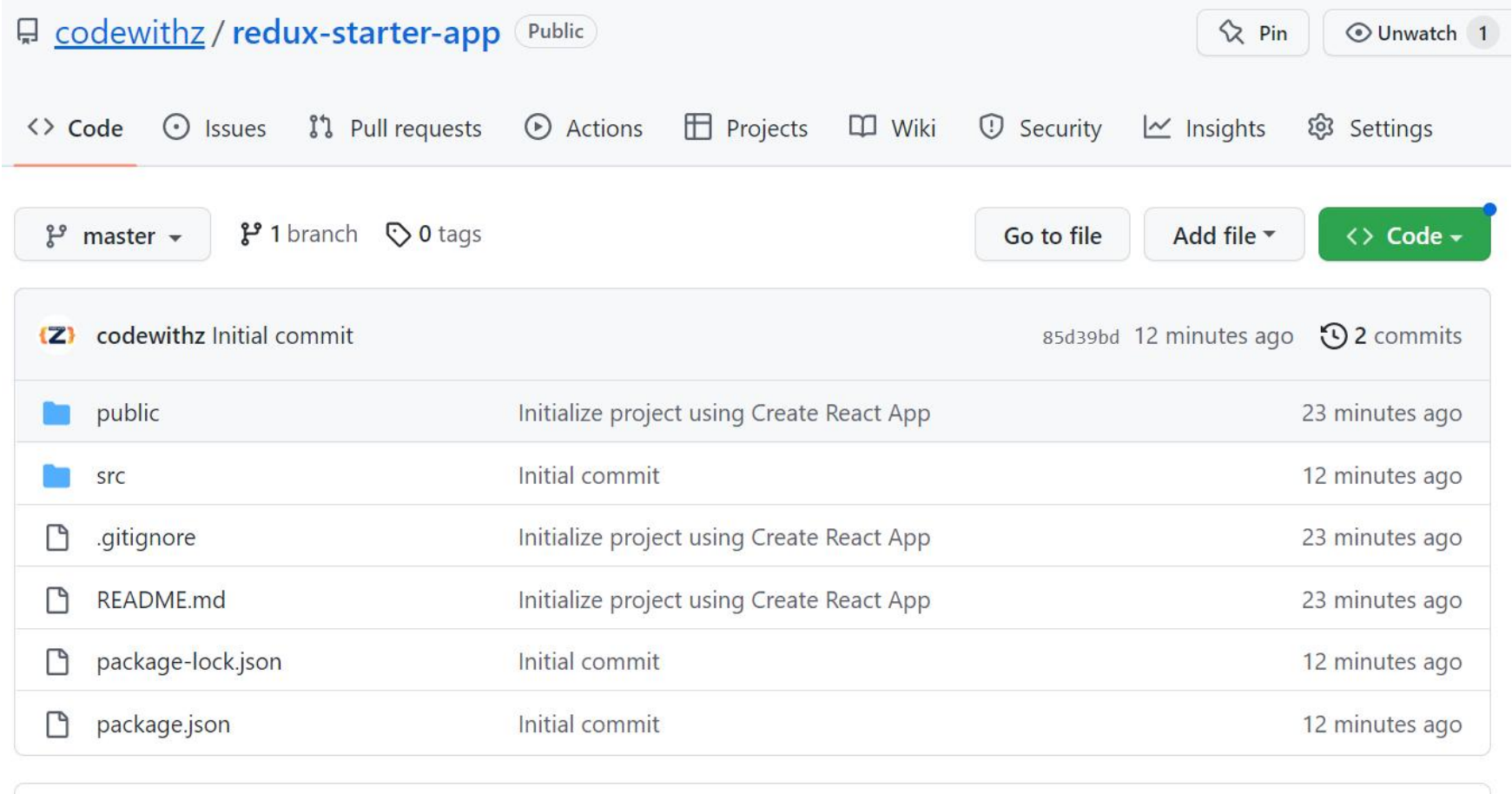
Humko Katta chahiye Papa...

UI


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

Clone the starter file from following git repo

https://github.com/codewithz/redux-starter-app

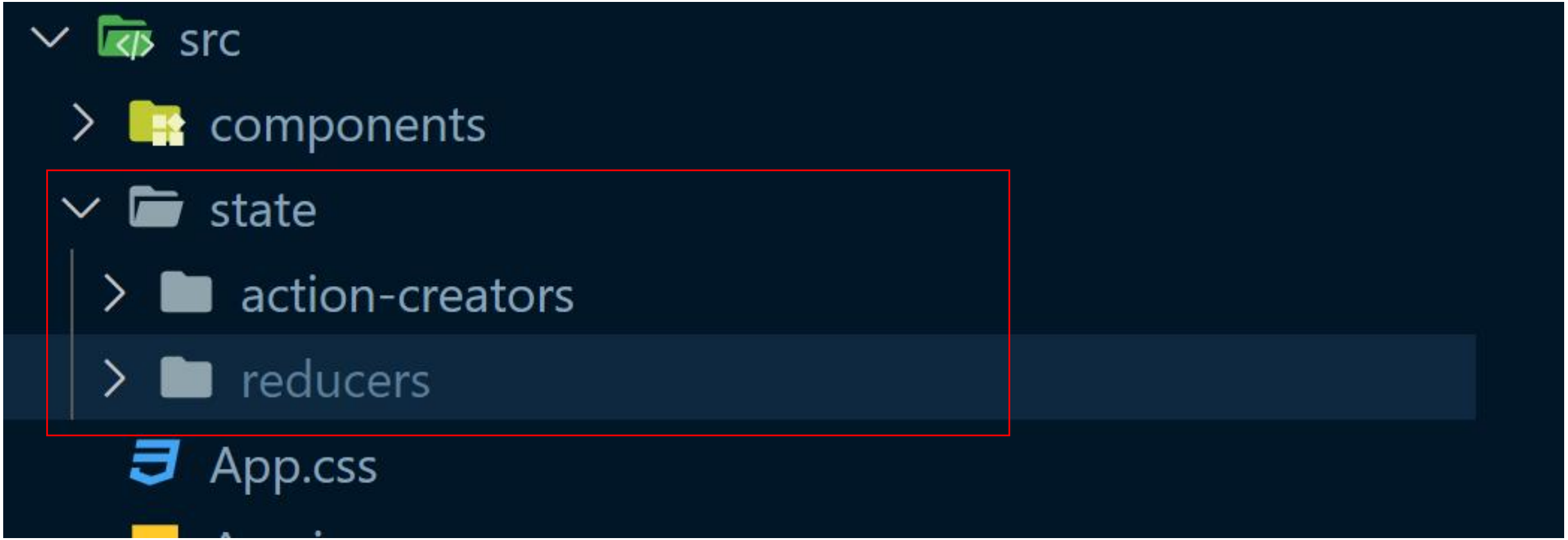


Run

npm install

inside the folder in a terminal

Create a folder by name state and inside it create two folders reducers and action-creators



Action Creator is the place where we write the code which will be responsible for generating actions

action is passed an object containing the type of action and payload for the action

Create a file by name index.js in action-creators and write the above marked code in it

EXPLORER

REDUX-COMPLETE-APP

- node_modules
- public
- src
  - components
  - state
    - action-creators
      - index.js
    - reducers
  - App.css
  - App.js
  - App.test.js
  - index.css
  - index.js
  - logo.svg

OUTLINE

TIMELINE

index.js

src > state > action-creators > index.js > ...

```js
export const depositMoney = (amount) => {
  return (dispatch) => {
    dispatch({
      type: "deposit",
      payload: amount,
    });
  };
};

export const withdrawMoney = (amount) => {
  return (dispatch) => {
    dispatch({
      type: "withdraw",
      payload: amount,
    });
  };
};
```

Create a file by name amountReducer.js in reducers folder and write folllowing code in it.

EXPLORER

index.js | amountReducer.js

src > state > reducers > amountReducer.js > default

```
export default reducer = (state = 0, action) => {
  if (action.type === "deposit") {
    return state + action.payload;
  }
  else if (action.type === "withdraw") {
    return state - action.payload;
  }
   else {
    return state;
  }
};
```

REDUX-COMPLETE-APP
- node_modules
- public
- src
  - components
  - state
    - action-creators
      - index.js
    - reducers
      - amountReducer.js
  - App.css
  - App.js
  - App.test.js
  - index.css
  - index.js

There can be multiple reducers in an app taking care of different type of actions. So we need to make combine all the reducers using combineReducers()

npm install redux react-redux redux-thunk

```
PS E:\Local d\Zartab\CodeWithZAcademy\react\redux-complete-app> npm i redux react-redux redux-thunk

added 9 packages, and audited 1477 packages in 4s

228 packages are looking for funding
  run `npm fund` for details

6 high severity vulnerabilities
```

Create a file from index.js and add a combineReducers

EXPLORER

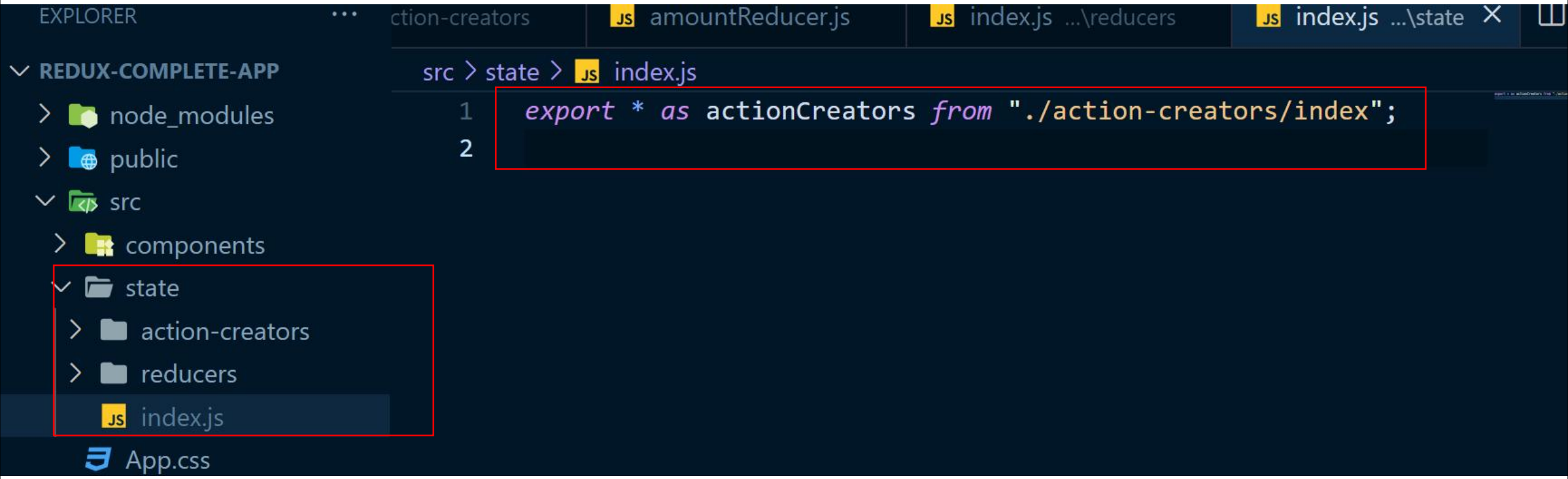
REDUX-COMPLETE-APP
- node_modules
- public
- src
  - components
  - state
    - action-creators
      - index.js
    - reducers
      - amountReducer.js
      - index.js
  - App.css

index.js ...\action-creators | amountReducer.js | index.js

src > state > reducers > index.js > ...

```
1  import { combineReducers } from "redux";
2  import amountReducer from "./amountReducer";
3
4  export default reducers = combineReducers({
5    amount: amountReducer,
6  });
7
```

Create a file by name index.js in state folder and write the following code to export all the action creators from state



Lets create our store

Create a file by name store.js in state and write following code

EXPLORER

REDUX-COMPLETE-APP

- node_modules
- public
- src
  - components
  - state
    - action-creators
    - reducers
    - index.js
    - store.js
  - App.css
  - App.js
  - App.test.js
  - index.css
  - index.js

amountReducer.js | index.js ...\reducers | index.js ...\state | store.js

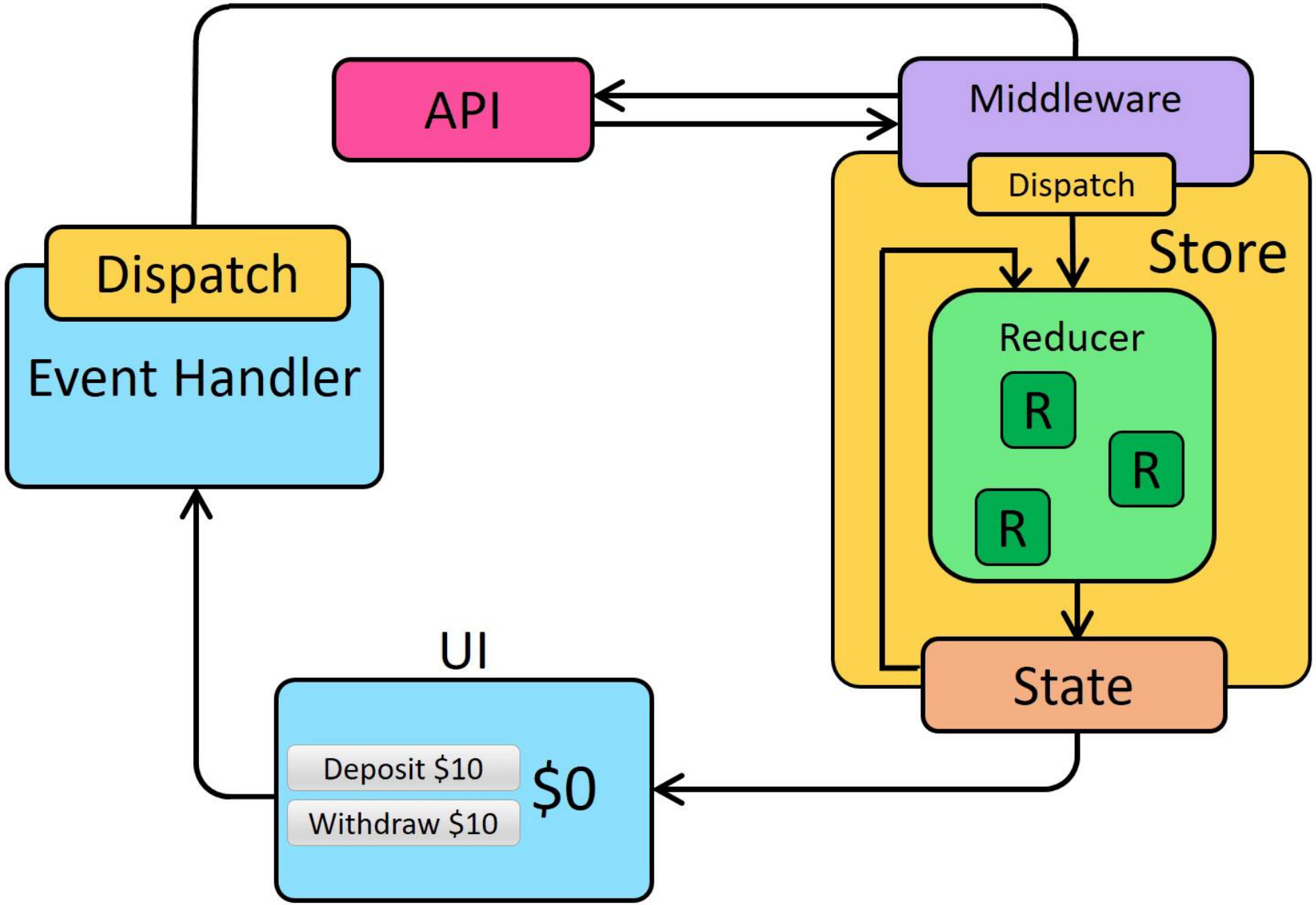
src > state > store.js > store

```js
import { applyMiddleware, createStore } from "redux";
import thunk from "redux-thunk";
import reducers from "./reducers";
// We are creating our store here
export const store = createStore(
    reducers, //here we specify all our reducers
    {}, //This object represent initialState
    applyMiddleware(thunk) //applyMiddleware helps to apply
                           //middle which helps to take care of
                           // async functions
    );
```

Lets go to index.js in src and wrap whole <App/> under a <Provider> which will make the store object available to whole application

src > index.js > ...

```
1   import React from "react";
2   import ReactDOM from "react-dom/client";
3   import "./index.css";
4   import App from "./App";
5   import reportWebVitals from "./reportWebVitals";
6   import "bootstrap/dist/css/bootstrap.css";
7   import { Provider } from "react-redux";
8   import { store } from "./state/store";
9
10  const root = ReactDOM.createRoot(document.getElementById("root")
11  root.render(
12    <React.StrictMode>
13      <Provider store={store}>
14        <App />
15      </Provider>
16    </React.StrictMode>
17  );
```


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

Oops a small error!!

```
ERROR in [eslint]
src\state\reducers\amountReducer.js
  Line 1:16:  'reducer' is not defined  no-undef

src\state\reducers\index.js
  Line 4:16:  'reducers' is not defined  no-undef

Search for the keywords to learn more about each error.

webpack compiled with 1 error and 1 warning
```

Another fix needed!!
In index.js --> reducer fix the export

src > state > reducers > index.js > ...

```
1  import { combineReducers } from "redux";
2  import amountReducer from "./amountReducer";
3
4  const reducers = combineReducers({
5    amount: amountReducer,
6  });
7
8  export default reducers;
9
```

src > state > reducers > amountReducer.js > ...

```javascript
const reducer = (state = 0, action) => {
  if (action.type === "deposit") {
    return state + action.payload;
  } else if (action.type === "withdraw") {
    return state - action.payload;
  } else {
    return state;
  }
};

export default reducer;
```

Lets start using the state from the store into the Navbar and other components

amountReducer.js | Navbar.jsx | index.js ...\reducers | index.js ..

src > components > Navbar.jsx > Navbar

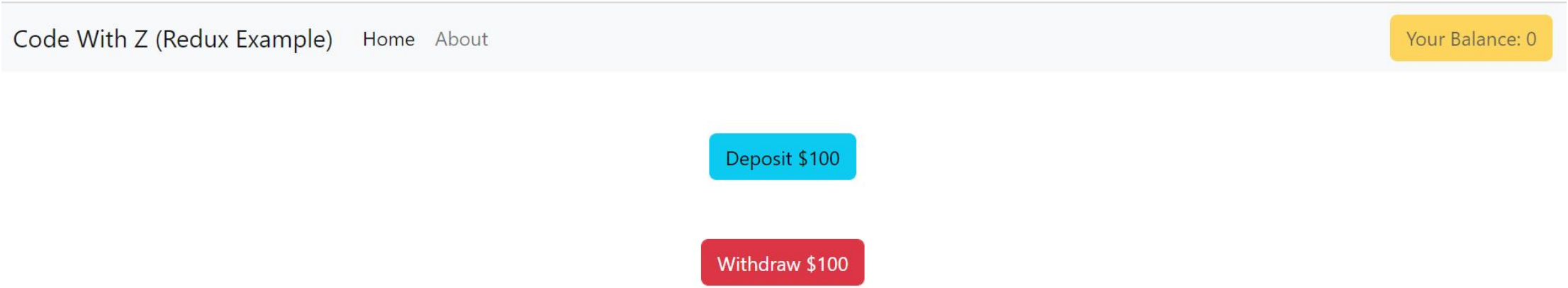
```
1   import React from "react";
2   import { useSelector } from "react-redux";
3
4   export default function Navbar() {
5     const amount = useSelector((state) => state.amount);
6     return (
7       <nav className="navbar navbar-expand-lg bg-light">
8         <div className="container-fluid">
9           <a className="navbar-brand" href="#">
10            Code With Z (Redux Example)
11          </a>
12          <button
```

In Navbar.jsx make the following change

```
            </li>
          </ul>

          <button disabled={true} className="btn btn-warning">
            Your Balance: {amount}
          </button>
        </div>
      </div>
    </nav>
```

Now the amount is displayed from state and note hardcoded



Lets put things into action now

Go to WithdrawDepositComponent and link them to function on onClick

src > components > WithdrawDepositComponent.jsx > WithdrawDepositComponent

```jsx
import React from "react";

export default function WithdrawDepositComponent() {
  const withdrawAmount = () => {};
  const depositAmount = () => {};
  return (
    <div className="container" style={{ marginTop: "50px" }}>
      <button
      className="btn btn-info"
      onClick={depositAmount}
      >Deposit $100</button>
      <br /><br /><br />
      <button
      className="btn btn-danger"
      onClick={withdrawAmount}
      >Withdraw $100</button>
    </div>
  );
}
```

Import useDispatch and actionCreators in WithdrawDepositComponent.jsx

src > components > WithdrawDepositComponent.jsx > WithdrawDepositComponent

```
1  import React from "react";
2  import { useDispatch } from "react-redux";
3  import { actionCreators } from "../state";
4
5  export default function WithdrawDepositComponent() {
6    const withdrawAmount = () => {};
7    const depositAmount = () => {};
```

useDispatch with dispatch an action based on the action we choose and it now update the state

src > components > WithdrawDepositComponent.jsx > WithdrawDepositComponent > depositAmount

```
1   import React from "react";
2   import { useDispatch } from "react-redux";
3   import { actionCreators } from "../state";
4
5   export default function WithdrawDepositComponent() {
6     const dispatch = useDispatch();
7
8     const withdrawAmount = () => {
9       dispatch(actionCreators.withdrawMoney(100));
10    };
11
12    const depositAmount = () => {
13      dispatch(actionCreators.depositMoney(100));
14    };
15    return (
```

Code With Z (Redux Example) Home About Your Balance: 400

Deposit $100

Withdraw $100

Now we can use bindActionCreators to just use the functions from reducer and make the call directly

src > components > WithdrawDepositComponent.jsx > WithdrawDepositComponent

```
1   import React from "react";
2   import { useDispatch } from "react-redux";
3   import { actionCreators } from "../state";
4
5   export default function WithdrawDepositComponent() {
6     const dispatch = useDispatch();
7
8     // const withdrawAmount = () => {
9     //   dispatch(actionCreators.withdrawMoney(100));
10    // };
11
12    // const depositAmount = () => {
13    //   dispatch(actionCreators.depositMoney(100));
14    // };
```

I have commented the withdrawMoney and depositMoney in component and now I will use a bindActionCreators function which gives the actionCreators as well as dispatch , so based on the method you are invoking from action it will dispatch it implicitly.

src > components > WithdrawDepositComponent.jsx > WithdrawDepositComponent

```
1   import React from "react";
2   import { useDispatch } from "react-redux";
3   import { bindActionCreators } from "redux";
4   import { actionCreators } from "../state";
5
6   export default function WithdrawDepositComponent() {
7     const dispatch = useDispatch();
8
9     const { withdrawMoney, depositMoney } = bindActionCreators(
10      actionCreators,
11      dispatch
12    );
```

src > components > WithdrawDepositComponent.jsx > WithdrawDepositComponent

```
17
18      // const depositAmount = () => {
19      //   dispatch(actionCreators.depositMoney(100));
20      // };
21      return (
22        <div className="container" style={{ marginTop: "50px" }}>
23          <button className="btn btn-info" onClick={() => depositMoney(100)}>
24            Deposit $100
25          </button>
26          <br />
27          <br />
28          <br />
29          <button className="btn btn-danger" onClick={() => depositMoney(100)}>
30            Withdraw $100
31          </button>
32        </div>
33      );
34    }
```