REDUX

Hello

Kamil Richert

Senior Software Engineer at Atlassian

How did it all start?

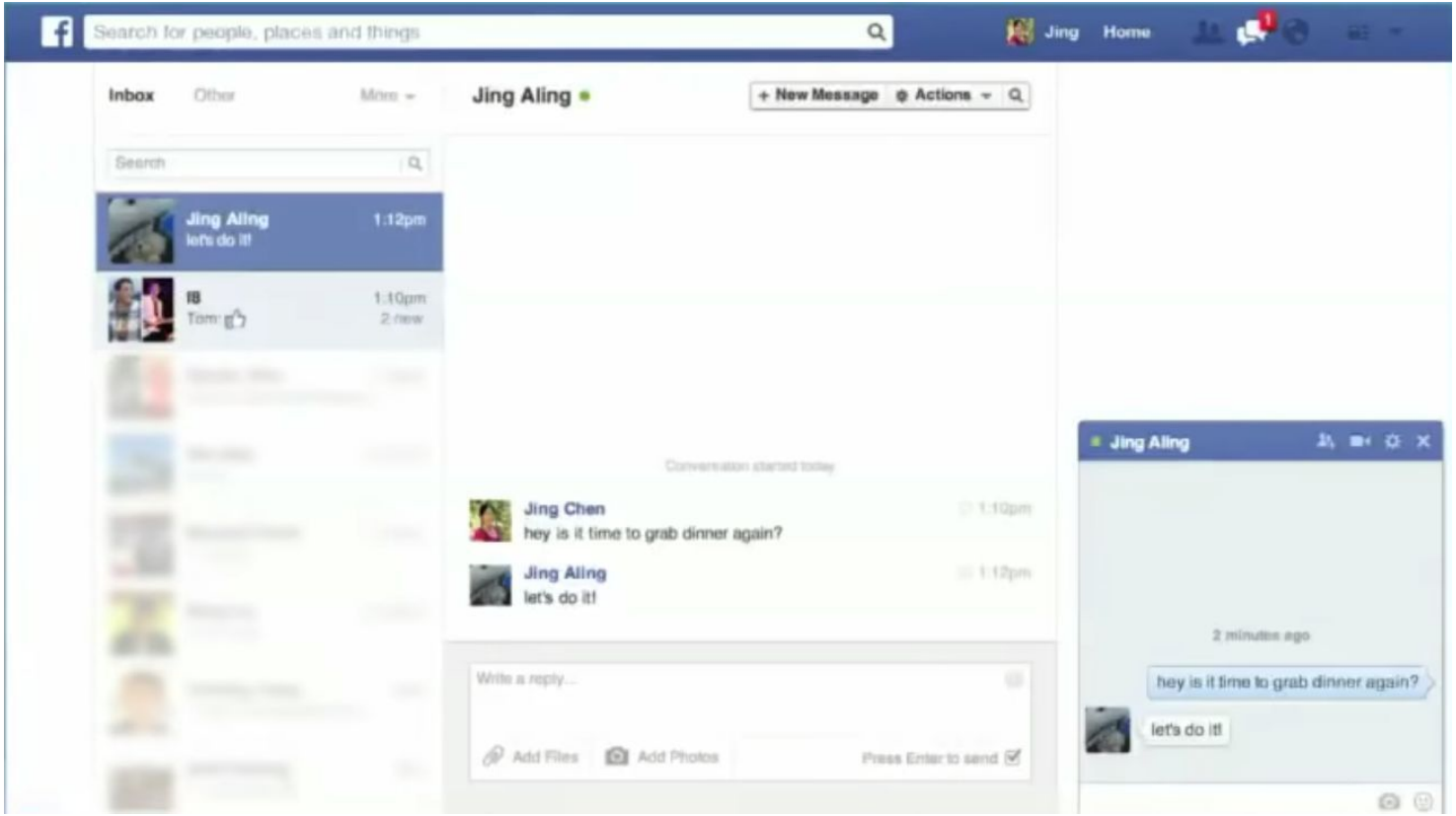
In the beginning there was Flux ...
an architecture / concept created by Facebook programmers to solve the problem of state management

State management problem? What does it mean?

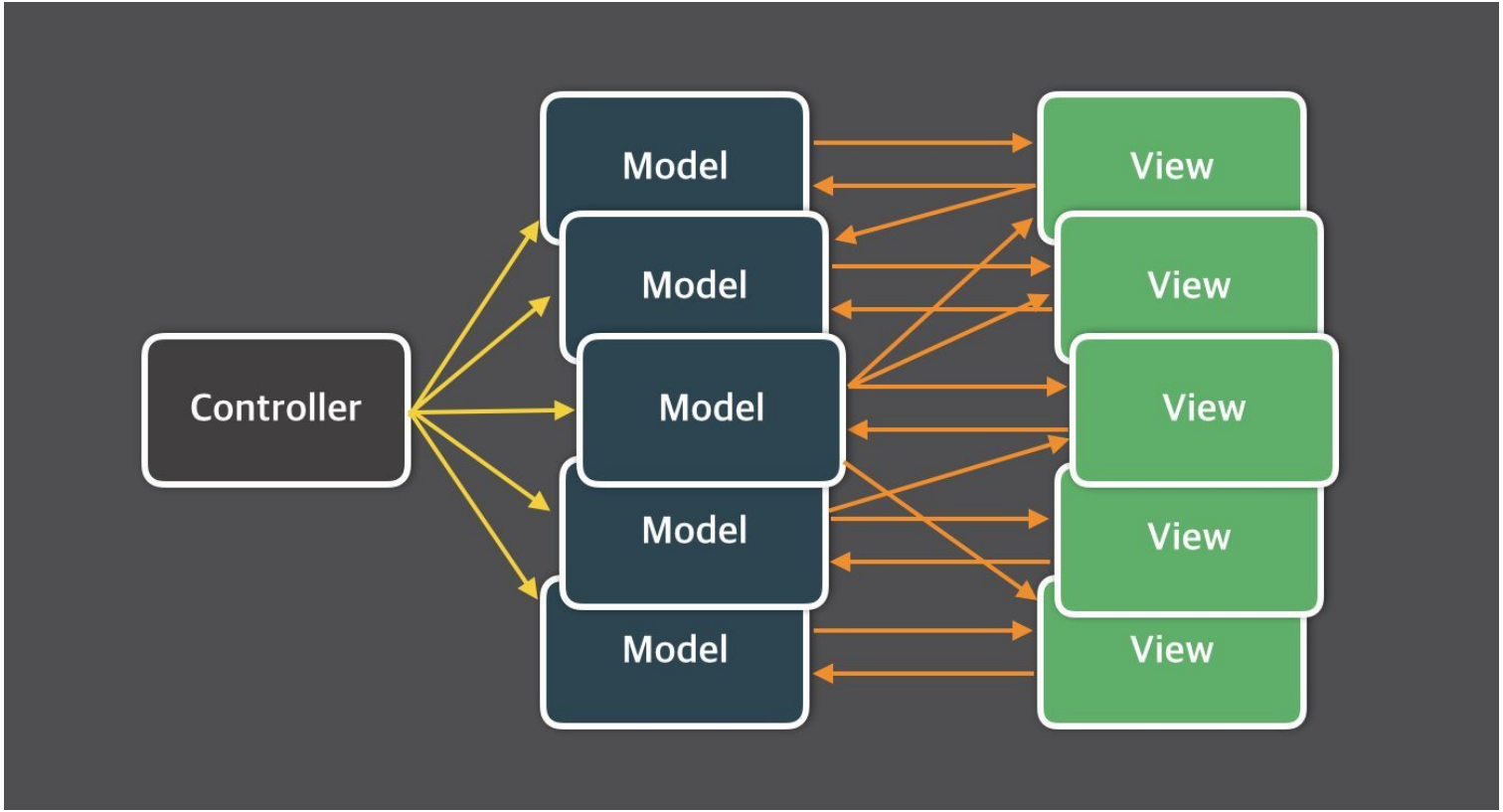
Addresses an application scalability (extensibility) issue as well
forces a one-way flow of data

How did this problem look for Facebook?

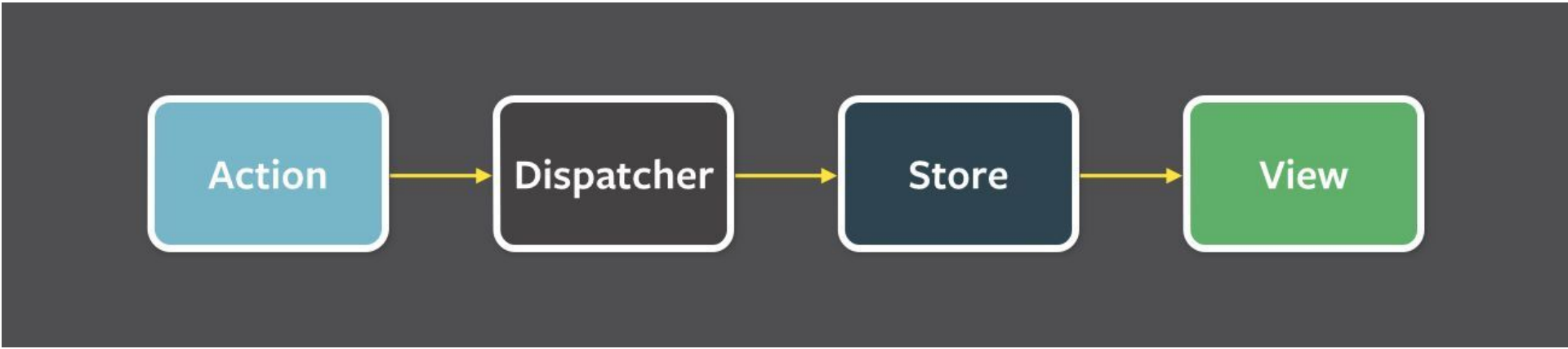
How did this problem look for Facebook?



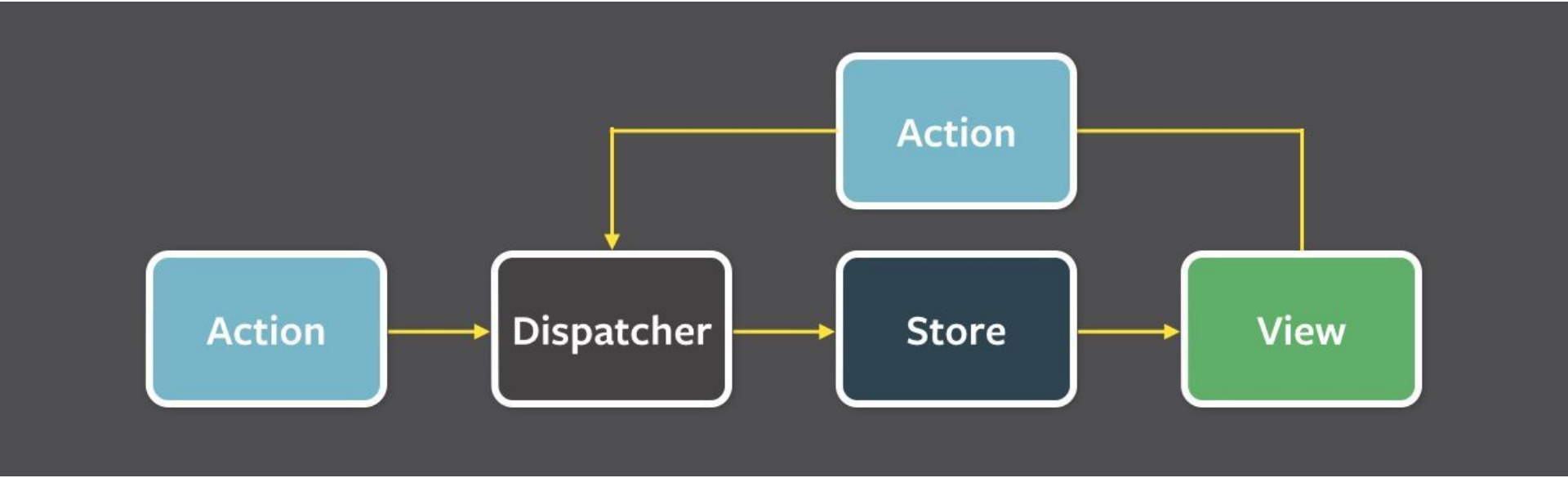
Two-way data flow



One-way data flow



One-way data flow



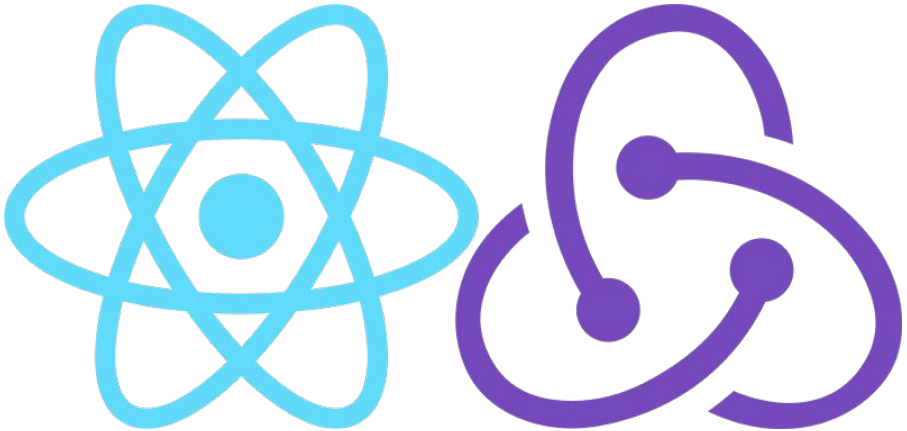
Explanations

- **Action** – data object (e.g. message or click)
- **Dispatcher** – informs specific stores about the action. Launches callbacks to inform the store about the stock
- **Store** – contains the state and logic of the application. Supports actions sent by dispatcher.
- **View** – it can be React, it can be the source of an action (e.g. reaction to user action)

FLUX is an idea / architecture

REDUX is its implementation

REDUX



A Predictable State
Container for JS Apps

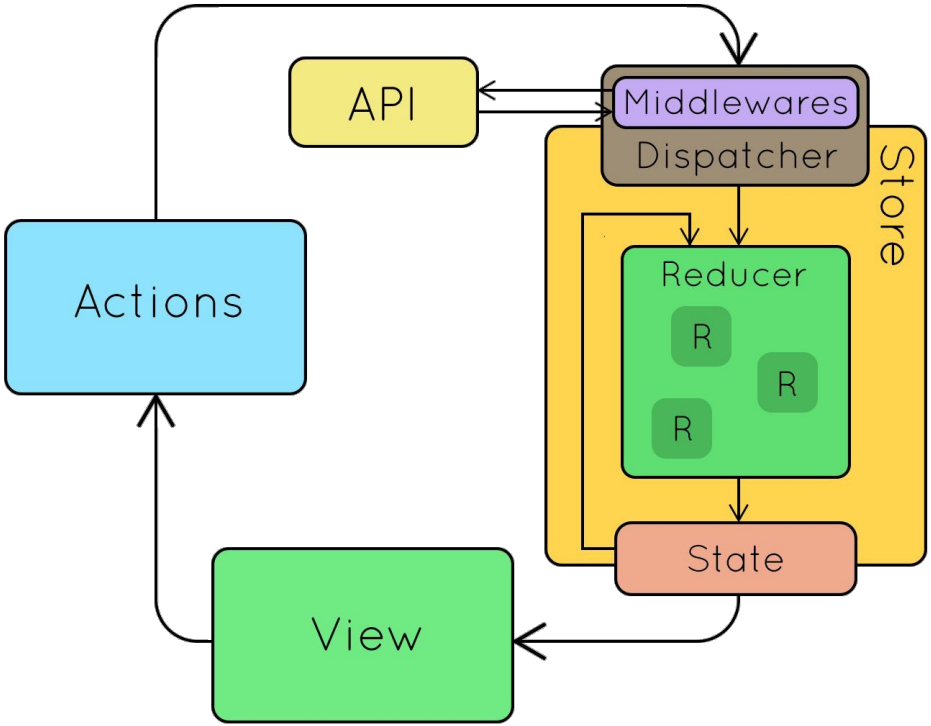
REDUX bank example

1. Intention (**ACTION**) to get money (**WITHDRAW_MONEY**)
2. Go to the window (**REDUCER)** and ask for money (**DISPATCH**)
3. A person in window / employee (**REDUCER**) "goes" to the vault (**STORE**) and extract money.
4. Only the window / employee (**REDUCER**) knows how to handle the vault (**STORE**) so that everything is correct (**STATE**).

REDUX drink example

1. Intention (**ACTION**) to drink beer (**HAVE_A_BEER**)
2. Go to the bartender (**REDUCER**) and ask for a drink (**DISPATCH**)
3. Bartender (**REDUCER**) "goes" to the shelves (**STORE**) and lifts the bottle.
4. Only the bartender (**REDUCER**) knows how to take out a bottle (**STORE**) so that everything is right (**STATE**).

REDUX scheme



What does REDUX consist of?

- **ACTION** – objects of type are the only ones to carry data and are fired with dispatch
- **REDUCER** – pure functions that determine how the state changes under the influence of an action
- **STORE** and **STATE –** an object that stores the entire state of the application, read-only (it is immutable), can perform the following operations on it:
    - **subscribe** – listening for state changes
    - **dispatch** – sends ACTION
    - **getState** – returns current state

How to start with Redux?

1. Install redux package: **npm install redux**

1. Create reducer (function):

```
function reducer(state, action) {
        switch(action.type) {
                case TYPE:
                        return <changed state>
                default
                        return state
        }
}
```

How to start with Redux?

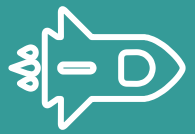
3. Create a store object using the createStore function from the redux package:
   const store = createStore(reducer);

4. Create action object:
   const action = { type: 'action type', payload: 'payload' };

5. Perform the dispatch function on the store object and pass the action object in it:
   store.dispatch(action);

6. Check the changed state by reading the state from the store object.
   store.getState();

TASK



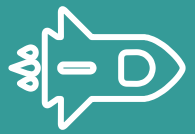
Create a COUNTER in REDUX.

1. Create functions (reducer) counter and handle actions: INCREMENT, DECREMENT, RESET.
2. Create a store using the createStore function and use the created reducer in it.
3. Use the window object to "extract" the store to the console and check that the reducer is well implemented by calling dispatch to the store with the appropriate action.

REDUX DEVTOOLS

- add redux devtools to google chrome
- add window .__ REDUX_DEVTOOLS_EXTENSION__ &&
  window .__ REDUX_DEVTOOLS_EXTENSION __ () as the 2nd
  argument of the createStore function

TASK



We will create a BANK in REDUX from examples.

1. Create functions (reducer) bank and handle actions: DEPOSIT, WITHDRAW, WITHDRAW_ALL, BALANCE. The deposit limit is PLN 1,000.
2. Create a store using the createStore function and use the created reducer in it.
3. Use the window object to "extract" the store to the console and check that the reducer is well implemented by calling dispatch to the store with the appropriate action.

REACT-REDUX

package facilitating the use of Redux in React

REACT-REDUX

What is the most important?

Provider, useDispatch, useSelector, useStore

react-redux

Provider

React component that gives access to the Redux stack to all the component children who use useSelector hook

- accepts props store

```
import { Provider } from ‘react-redux’;

const App = () => (
        <Provider store={store}>
                <Content />
        </Provider>
);
```

react-redux

useSelector

Extracts data from Redux state using a selector function.

The selector function should be clean as it potentially gets executed multiple times and at any point in time.

The selector will be called with the entire Redux stock state as the only argument.

```
import { useSelector } from 'react-redux'

const selectedData = useSelector(
        (state) => state.counter
);
```

react-redux

useDispatch

This hook returns a reference to the dispatcj function from Redux. You can use it to send actions.

```
import { useDispatch } from 'react-redux'

const dispatch = useDispatch()

const onClick= () =>
      dispatch({ type: 'increment' })
```

react-redux

useStore

This hook returns a reference to the same store Redux that was passed as props to the <Provider> component.

It should not be used often. useSelector() should be the primary choice.

Note! The component will not update automatically if the state of the store changes using this hook.

```
import { useStore } from 'react-redux'

const store = useStore()

const state = store.getState()
```

REDUX – how to add it to React

1. Create a store using the createStore function from the redux package and create a reducer that you pass as an argument to createStore.

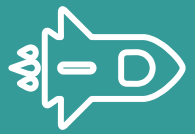
2. Install the react-redux package.

3. "Ovrap" the application with the Provider component from the react-redux package and transfer the created store to it.

4. Use useSelector to get data from Redux.

5. To perform the action, use the return from useDispatch hook.

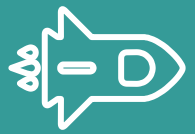
TASK



Let's create a COUNTER application with React-Redux.

1. We will start with the (reducer) counter function and handle the following actions in it: INCREMENT, DECREMENT, RESET.
2. Let's create a store in the store.ts file and use the reducer from step 1.
3. Add action creator for each action (functions that create an action object).
4. Add Provider component with provided store in index.ts
5. Call the dispatch function with action creators on the click of buttons

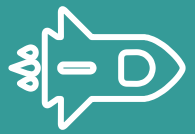
TASK



Create a **RENTALS** app with React-Redux.

1. Create rentalOffice (reducer) functions and handle the following actions: add, delete, select rental and return.
2. Together we will update the store in the store.ts file to support 2 reducers. Let's fix Counter after this operation.
3. Add action creator for each action (functions that create an action object).
4. Call the dispatch function with action creators on clicking the buttons and submit the form.

TASK



We will create a STORE application using React-Redux.

1. Create shopCart (reducer) functions and handle actions: adding to the cart and removing from it.
2. Update store in store.ts to support 3 reducers.
3. Add action creator for each action (functions that create an action object).
4. Call the dispatch function with action creators on clicking buttons in the store.
5. Use redux content to update the cart.

REDUX-THUNK

What if the actions are supposed to happen asynchronously?

What is the thunk?
Function. Thunk is a special name for a function that is returned by another function.

https://daveceddia.com/what-is-a-thunk/

THUNK

```
function yell (text) {
	console.log(text + '!')
}

function thunkedYell (text) {
	return function thunk () {
		console.log(text + '!')
	}
}
const thunk = thunkedYell('bonjour') // no action yet.
thunk() // 'bonjour!'
```

https://medium.com/fullstack-academy/thunks-in-redux-the-basics-85e538a3fe60

REDUX-THUNK

To perform asynchronous actions we need to use the redux-thunk middleware.

We can add this middleware to the store using the applyMiddleware function from redux.

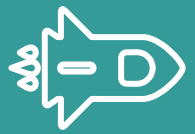
```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers/index';

const store = createStore(rootReducer, applyMiddleware(thunk));
```

REDUX-THUNK

Example:

```
function increment() {
  return {
    type: ‘INCREMENT’,
  };
}

function incrementAsync() {
  return (dispatch) => {
        // Yay! We can invoke an asynchronous action with dispatch
        setTimeout(() => { dispatch(increment()); }, 1000);
  };
}
```

TASK



Add buttons in **COUNTER** that perform the action after 3 seconds (add, subtract, reset).

Show spinner at the moment of invoking a synchronized action until it is executed.

RESELECT

package facilitating work with selectors in redux (mapStateToProps)

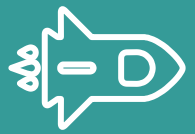
RESELECT

```
import { createSelector } from 'reselect'

const shopItemsSelector = state => state.shop.items
const taxPercentSelector = state => state.shop.taxPercent

const subtotalSelector = createSelector(
  shopItemsSelector,
  items => items.reduce((subtotal, item) => subtotal + item.value, 0)
)

const taxSelector = createSelector(
  subtotalSelector,
  taxPercentSelector,
  (subtotal, taxPercent) => subtotal * (taxPercent / 100)
)
```

RESELECT

```
const totalSelector = createSelector(
  subtotalSelector,
  taxSelector,
  (subtotal, tax) => ({ total: subtotal + tax })
)

const exampleState = {
  shop: {
    taxPercent: 8,
    items: [ { name: 'apple', value: 1.20 }, { name: 'orange', value: 0.95 } ]
  }
}

console.log(subtotalSelector(exampleState)) // 2.15
console.log(taxSelector(exampleState))      // 0.172
console.log(totalSelector(exampleState))    // { total: 2.322 }
```

TASK



Using the reselect package, move the logic from the component to the state:

1. From the ShoppingCart component, transfer the calculation of the total amount to the selector.
2. From the Shop component, check if the item is already in the cart.

REDUX TOOLKIT

The official set of tools for efficient development of Redux

REDUX TOOLKIT

The Redux Toolkit is intended to be the standard way to write Redux logic. It was originally created to help solve three common Redux problems:

"Store Redux configuration is too complicated"

"I have to add a lot of packages for Redux to do something useful"

"Redux requires too much standard code"

REDUX TOOLKIT

What is included?

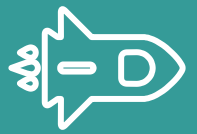
- **configureStore**(): Provides simplified configuration options and defaults. (includes redux-thunk and enables the use of Redux DevTools extension)
- **createReducer**(): allows you to provide an action type table to the reducer function, instead of writing switch statements.
- **createAction**(): Generates a create action function for the specified action type string.
- **createSelector**(): From the Reselect library, re-exported for ease of use.
- **createSlice**(): accepts a reducer object, slice name and initial value, and automatically generates a reducer with the appropriate action creators and action types

and many others.

REDUX TOOLKIT

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

const initialState = {
  value: 0,
}

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => { state.value += 1 },
    decrement: (state) => { state.value -= 1 },
    incrementByAmount: (state, action) => { state.value += action.payload },
  },
})

export const { increment, decrement, incrementByAmount } = counterSlice.actions

export default counterSlice.reducer
```

TASK



Rewrite the reducer from rentalOffice to what will be used by redux toolkit.

REDUX summary

The biggest advantages of Redux:

1. One-way data flow
2. Predictable
3. Scalability
4. Ease of testing
5. Solves the problem with props drilling
6. Easy access to application status from anywhere in the code

Thanks!

You can find me:

https://www.linkedin.com/in/kamil-richert/

https://github.com/krichert