Rematch

Czyli jak uporządkować Reduxa

Jakub Drozdek

Aplikacja – prosty licznik

Counter: 0

Synchronous actions:

+1 -1

+5 -5

Asynchronous actions:

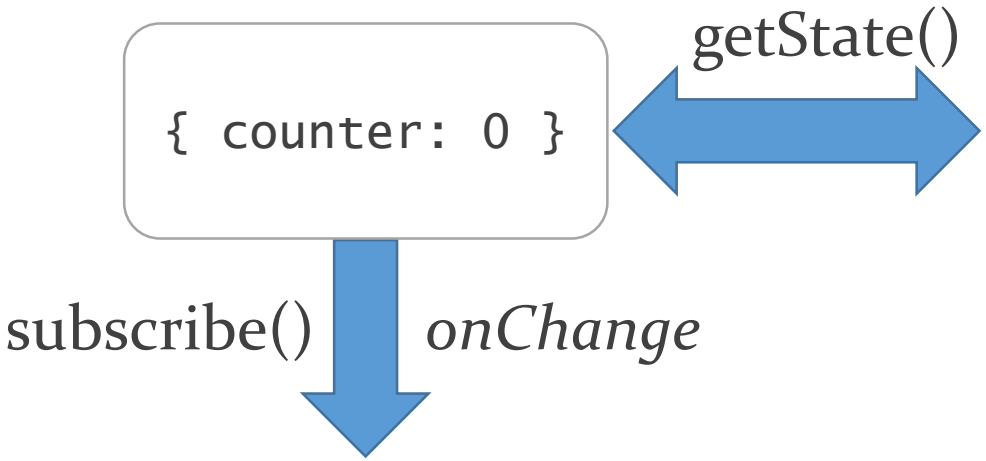
Valid +2 Invalid -1

REDUX

Stan aplikacji

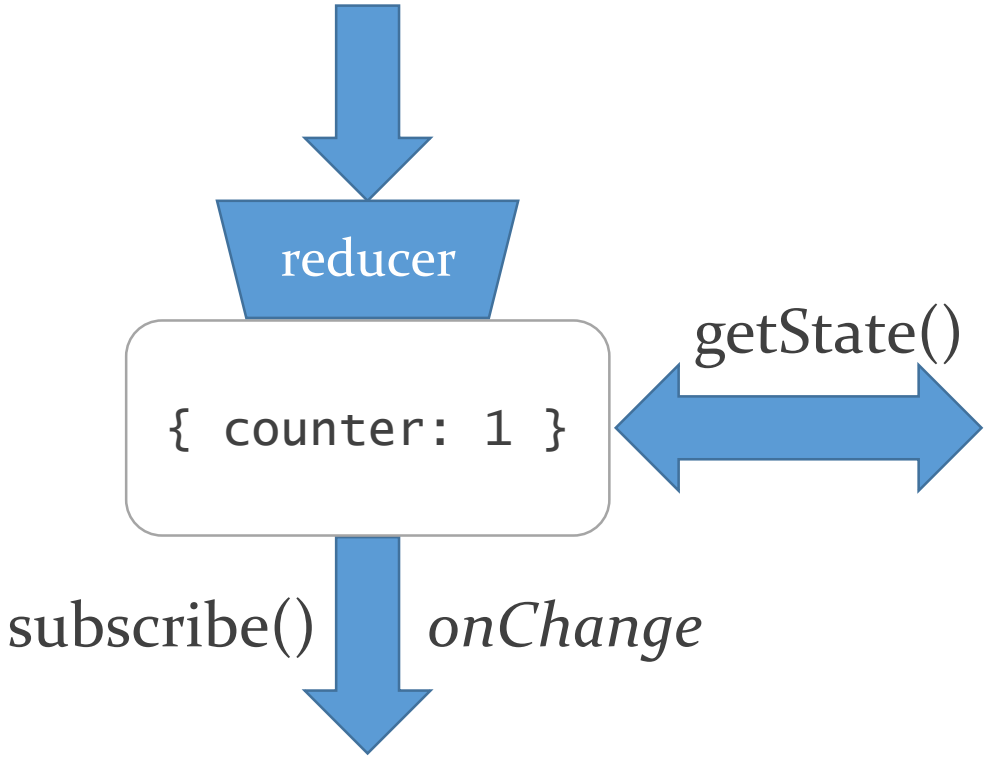
```
{ counter: 0 }
```

Odczyt stanu



Aktualizacja stanu

dispatch({ type: 'COUNTER/INCREMENT' })



Redux – inicjalizacja store

```
1  // redux/store.js
2  import thunk from "redux-thunk";
3  import { createStore, combineReducers, applyMiddleware, compose } from "redux";
4
5  // Create root reducer
6  import counterReducer from "./counter";
7
8  const rootReducer = combineReducers({
9    counter: counterReducer
10 });
11
12 // Plug in middleware for Dev Tools
13 const composeEnhancers =
14   typeof window === "object" && window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
15     ? window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({})
16     : compose;
17
18 const middleware = composeEnhancers(applyMiddleware(thunk));
19
20 // Create store
21 const store = createStore(rootReducer, middleware);
22
23 export default store;
```

Redux – action types i action creators

```
// redux/counter/constants.js
export const PREFIX = "COUNTER";

export const INCREMENT = `${PREFIX}/INCREMENT`;
export const DECREMENT = `${PREFIX}/DECREMENT`;

export const INCREMENT_BY = `${PREFIX}/INCREMENT_BY`;
export const DECREMENT_BY = `${PREFIX}/DECREMENT_BY`;
```

```
// redux/counter/actions.js
import * as types from "./constants";

export const increment = () => ({
  type: types.INCREMENT
});
export const decrement = () => ({
  type: types.DECREMENT
});

export const incrementBy = value => ({
  type: types.INCREMENT_BY,
  value
});
export const decrementBy = value => ({
  type: types.DECREMENT_BY,
  value
});
```

```
dispatch(incrementBy(5)) === dispatch({ type: 'COUNTER/INCREMENT_BY', value: 5 })
```

Redux – reducer

```
1   // redux/counter/reducer.js
2   import { INCREMENT, INCREMENT_BY, DECREMENT, DECREMENT_BY } from "./constants";
3
4   const initialState = 0;
5
6   export default (state = initialState, action) => {
7     switch (action.type) {
8       case INCREMENT:
9         return state + 1;
10      case INCREMENT_BY:
11        return state + action.value;
12      case DECREMENT:
13        return state - 1;
14      case DECREMENT_BY:
15        return state - action.value;
16      default:
17        return state;
18    }
19  };
```

Redux – reducer + stan będący obiektem

```
// redux/counter/reducer.js
import {
  INCREMENT,
  INCREMENT_BY,
  DECREMENT,
  DECREMENT_BY,
} from "./constants";

const initialState = {
  count: 0,
  otherStuff: 123,
};
```

```
export default (state = initialState, action) => {
  switch (action.type) {
    case INCREMENT:
      return {
        ...state,
        count: state.count + 1
      };
    case INCREMENT_BY:
      return {
        ...state,
        count: state.count + action.value
      };
    case DECREMENT:
      return {
        ...state,
        count: state.count - 1
      };
    case DECREMENT_BY:
      return {
        ...state,
        count: state.count - action.value
      };
    default:
      return state;
  }
};
```

Redux – thunks – action creators

```
// redux/counter/actions.js
import * as types from "./constants";
import { wait } from "../../utils/promises";

// ...

export const incrementByAsyncRequest = () => ({
  type: types.INCREMENT_BY_ASYNC_REQUEST
});
export const incrementByAsyncSuccess = value => ({
  type: types.INCREMENT_BY_ASYNC_SUCCESS,
  value
});
export const incrementByAsyncFailure = error => ({
  type: types.INCREMENT_BY_ASYNC_FAILURE,
  error
});

export const incrementByAsync = value => async dispatch => {
  dispatch(incrementByAsyncRequest());

  await wait(2000);

  if (value > 0) {
    dispatch(incrementByAsyncSuccess(value));
  } else {
    dispatch(incrementByAsyncFailure("Value must be greater than 0."));
  }
};
```

Redux – thunks – reducer

```
// redux/counter/reducer.js
import {
  // ...
  INCREMENT_BY_ASYNC_SUCCESS,
  INCREMENT_BY_ASYNC_REQUEST,
  INCREMENT_BY_ASYNC_FAILURE
} from "./constants";

const initialState = {
  count: 0,
  isLoading: false,
  error: null
};
```

```
export default (state = initialState, action) => {
  switch (action.type) {
    // ...
    case INCREMENT_BY_ASYNC_REQUEST:
      return {
        ...state,
        isLoading: true,
        error: null
      };
    case INCREMENT_BY_ASYNC_SUCCESS:
      return {
        ...state,
        isLoading: false,
        count: state.count + action.value
      };
    case INCREMENT_BY_ASYNC_FAILURE:
      return {
        ...state,
        isLoading: false,
        error: action.error
      };
    default:
      return state;
  }
};
```

Redux – podpięcie do komponentu

```
1   import React, { Component } from "react";
2   import { connect } from "react-redux";
3
4   import { actions } from "./redux/counter";
5
6   class App extends Component {…
51  }
52
53  const mapState = state => ({
54    count: state.counter.count,
55    error: state.counter.error,
56    isLoading: state.counter.isLoading
57  });
58  const mapDispatch = dispatch => ({
59    increment: () => dispatch(actions.increment()),
60    incrementBy: value => dispatch(actions.incrementBy(value)),
61    decrement: () => dispatch(actions.decrement()),
62    decrementBy: value => dispatch(actions.decrementBy(value)),
63    incrementByAsync: value => dispatch(actions.incrementByAsync(value))
64  });
65
66  export default connect(
67    mapState,
68    mapDispatch
69  )(App);
```

Redux – podpięcie do komponentu z bindActionCreators

```
const newActions = {
  one: (payload) => dispatch(actions.one(payload)),
  two: (payload) => dispatch(actions.two(payload)),
  // ...
};

// ===

const newActions = bindActionCreators(actions, dispatch);
```

Redux – podpięcie do komponentu z bindActionCreators

```
1   import React, { Component } from "react";
2   import { bindActionCreators } from "redux";
3   import { connect } from "react-redux";
4
5   import { actions } from "./redux/counter";
6
7   class App extends Component {···
52  }
53
54  const mapState = state => ({
55    count: state.counter.count,
56    error: state.counter.error,
57    isLoading: state.counter.isLoading
58  });
59  const mapDispatch = dispatch => bindActionCreators(actions, dispatch);
60
61  export default connect(
62    mapState,
63    mapDispatch
64  )(App);
```

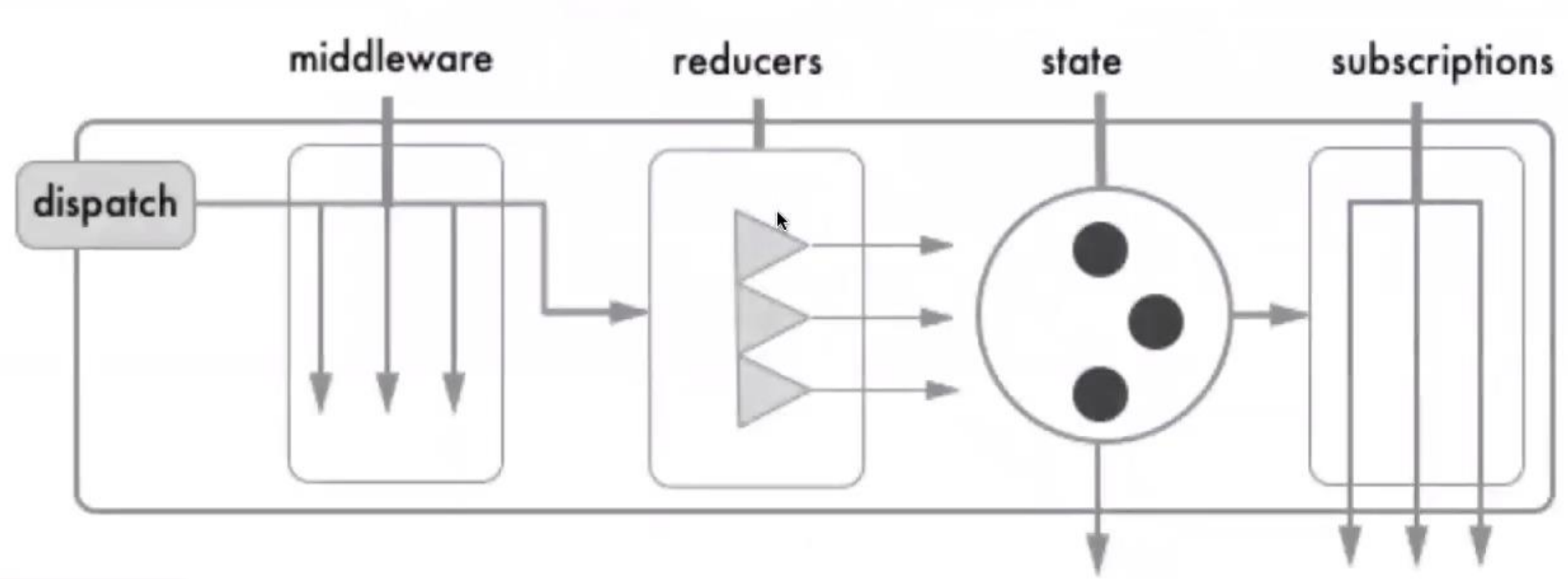
Redux – podpięty komponent

DEMO

REDUX vs. REMATCH

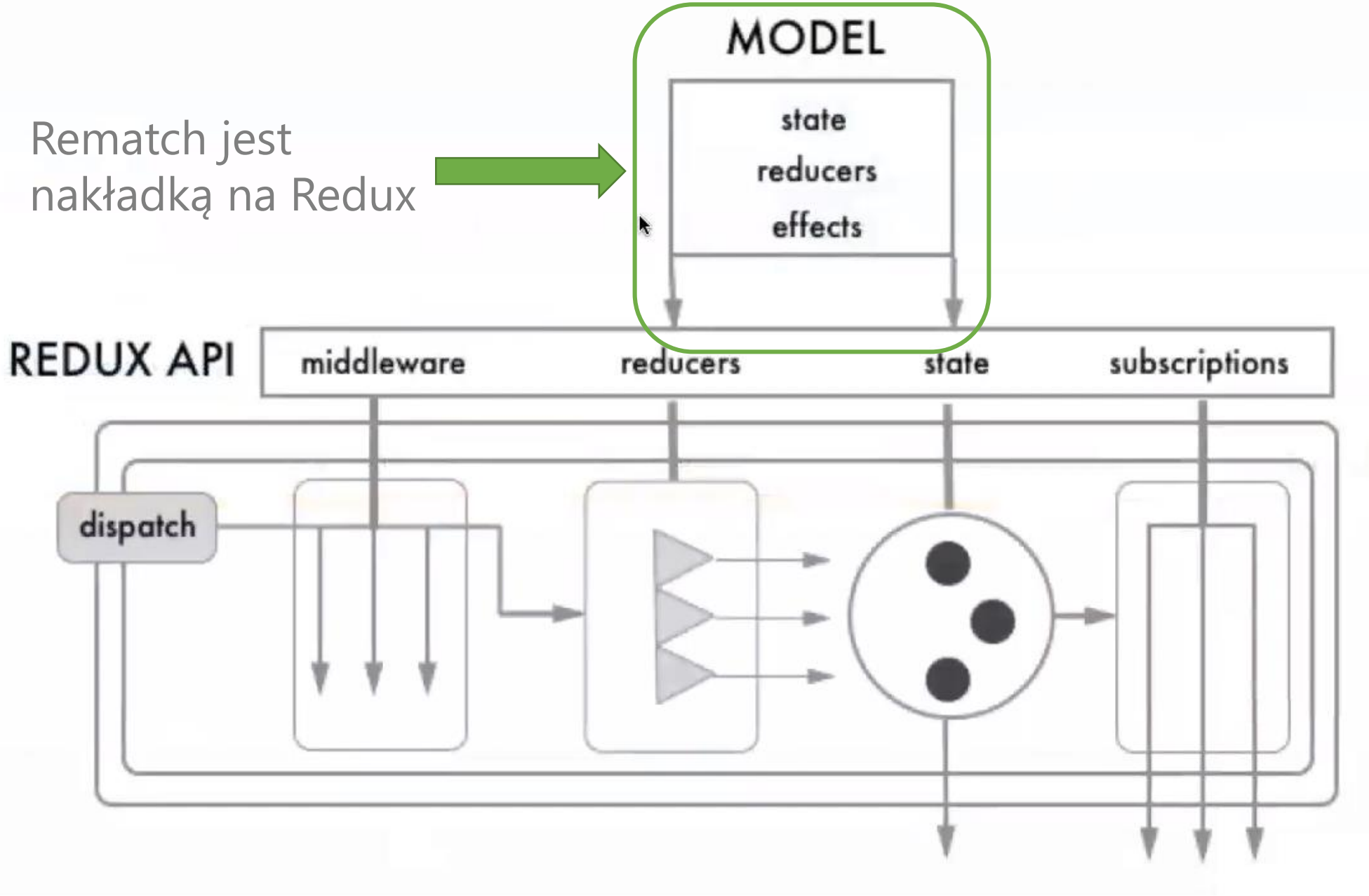
Czym jest Rematch?

Schemat działania Redux



Czym jest Rematch?

Rematch jest nakładką na Redux



Co daje Rematch?

- Redukuje boilerplate
- Narzuca dobre praktyki
- Udostępnia pluginy rozwiązujące typowe problemy
- Jest łatwy w konfiguracji

Redux – inicjalizacja store (przypomnienie)

```
1  // redux/store.js
2  import thunk from "redux-thunk";
3  import { createStore, combineReducers, applyMiddleware, compose } from "redux";
4
5  // Create root reducer
6  import counterReducer from "./counter";
7
8  const rootReducer = combineReducers({
9    counter: counterReducer
10 });
11
12 // Plug in middleware for Dev Tools
13 const composeEnhancers =
14   typeof window === "object" && window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
15     ? window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({})
16     : compose;
17
18 const middleware = composeEnhancers(applyMiddleware(thunk));
19
20 // Create store
21 const store = createStore(rootReducer, middleware);
22
23 export default store;
```

Rematch – inicjalizacja store

```
// redux/store.js
import { init } from "@rematch/core";

import * as models from "./models";

const store = init({
  models
});

export default store;
```

Redux – action types i creators, reducer (przypomnienie)

```
// redux/counter/constants.js
export const PREFIX = "COUNTER";

export const INCREMENT = `${PREFIX}/INCREMENT`;
export const DECREMENT = `${PREFIX}/DECREMENT`;

export const INCREMENT_BY = `${PREFIX}/INCREMENT_BY`;
export const DECREMENT_BY = `${PREFIX}/DECREMENT_BY`;
```

```
// redux/counter/actions.js
import * as types from "./constants";

export const increment = () => ({
  type: types.INCREMENT
});
export const decrement = () => ({
  type: types.DECREMENT
});

export const incrementBy = value => ({
  type: types.INCREMENT_BY,
  value
});
export const decrementBy = value => ({
  type: types.DECREMENT_BY,
  value
});
```

```
// redux/counter/reducer.js
import { INCREMENT, INCREMENT_BY, DECREMENT, DECREMENT_BY } from "./constants";

const initialState = 0;

export default (state = initialState, action) => {
  switch (action.type) {
    case INCREMENT:
      return state + 1;
    case INCREMENT_BY:
      return state + action.value;
    case DECREMENT:
      return state - 1;
    case DECREMENT_BY:
      return state - action.value;
    default:
      return state;
  }
};
```

Rematch – model

```
1   // redux/models/counter.js
2   export default {
3     state: 0,
4     reducers: {
5       increment(state) {
6         return state + 1;
7       },
8       decrement(state) {
9         return state - 1;
10      },
11      incrementBy(state, payload) {
12        return state + payload;
13      },
14      decrementBy(state, payload) {
15        return state + payload;
16      }
17    }
18  };
```

Rematch – akcje asynchroniczne = efekty

```
// redux/models/counter.js
import { wait } from "../../utils/promises";

export default {
  state: {
    count: 0,
    isLoading: false,
    error: null
  },
  reducers: {
    // ...
    incrementByAsyncRequest(state) {
      return { ...state, isLoading: true, error: null };
    },
    incrementByAsyncSuccess(state, payload) {
      return { ...state, isLoading: false, count: state.count + payload };
    },
    incrementByAsyncFailure(state, payload) {
      return { ...state, isLoading: false, error: payload };
    }
  },
  effects: {
    async incrementByAsync(payload) {
      this.incrementByAsyncRequest();

      await wait(2000);

      if (payload > 0) {
        this.incrementByAsyncSuccess(payload);
      } else {
        this.incrementByAsyncFailure("Value must be greater than 0.");
      }
    }
  }
};
```

Rematch – dispatch

```
import store from "../store";
const { dispatch } = store;

dispatch({ type: "counter/incrementBy", payload: 5 });
// ===
dispatch.counter.incrementBy(5);
```

```
dispatch[model][action](payload)
```

Rematch – podpięcie do komponentu

Redux

```
1   import React, { Component } from "react";
2   import { connect } from "react-redux";
3
4   import { actions } from "./redux/counter";
5
6   class App extends Component {…
51  }
52
53  const mapState = state => ({
54    count: state.counter.count,
55    error: state.counter.error,
56    isLoading: state.counter.isLoading
57  });
58  const mapDispatch = dispatch => ({
59    increment: () => dispatch(actions.increment()),
60    incrementBy: value => dispatch(actions.incrementBy(value)),
61    decrement: () => dispatch(actions.decrement()),
62    decrementBy: value => dispatch(actions.decrementBy(value)),
63    incrementByAsync: value => dispatch(actions.incrementByAsync(value))
64  });
65
66  export default connect(
67    mapState,
68    mapDispatch
69  )(App);
```

Rematch

```
1   import React, { Component } from "react";
2   import { connect } from "react-redux";
3
4   class App extends Component {…
49  }
50
51  const mapState = state => ({
52    count: state.counter.count,
53    error: state.counter.error,
54    isLoading: state.counter.isLoading
55  });
56  const mapDispatch = ({ counter }) => ({
57    increment: counter.increment,
58    decrement: counter.decrement,
59    incrementBy: counter.incrementBy,
60    decrementBy: counter.decrementBy,
61    incrementByAsync: counter.incrementByAsync
62  });
63
64  export default connect(
65    mapState,
66    mapDispatch
67  )(App);
```

Rematch – podpięcie do komponentu

Redux

```
1   import React, { Component } from "react";
2   import { bindActionCreators } from "redux";
3   import { connect } from "react-redux";
4
5   import { actions } from "./redux/counter";
6
7   class App extends Component {…
52  }
53
54  const mapState = state => ({
55    count: state.counter.count,
56    error: state.counter.error,
57    isLoading: state.counter.isLoading
58  });
59  const mapDispatch = dispatch => bindActionCreators(actions, dispatch);
60
61  export default connect(
62    mapState,
63    mapDispatch
64  )(App);
```

Rematch

```
1   import React, { Component } from "react";
2   import { connect } from "react-redux";
3
4   class App extends Component {…
49  }
50
51  const mapState = state => ({
52    count: state.counter.count,
53    error: state.counter.error,
54    isLoading: state.counter.isLoading
55  });
56  const mapDispatch = ({ counter }) => counter;
57
58  export default connect(
59    mapState,
60    mapDispatch
61  )(App);
```

REMATCH – co jeszcze?

Rematch – reagowanie na akcje z innego modelu

```
1   // redux/models/counter.js
2   export default {
3     state: 0,
4     reducers: {
5       increment(state) {
6         return state + 1;
7       },
8       "otherModel/actionName"(state, payload) {
9         return state + payload;
10      }
11    }
12  };
```

LUB

```
1   // redux/models/counter.js
2   export default {
3     state: 0,
4     reducers: {
5       increment: state => state + 1,
6       "otherModel/actionName": (state, payload) => state + payload
7     }
8   };
```

Rematch – wywoływanie akcji z innego modelu

```
1  // redux/models/counter.js
2  import store from "../store";
3  import { wait } from "../../utils/promises";
4
5  export default {
6    state: 0,
7    reducers: {
8      incrementBy: (state, payload) => state + payload
9    },
10   effects: {
11     async incrementByAsync(payload) {
12       this.incrementBy(payload);
13       await wait(2000);
14       store.dispatch.otherModel.actionName(payload);
15     }
16   }
17 };
```

LUB

```
1  // redux/models/counter.js
2  import { wait } from "../../utils/promises";
3
4  export default {
5    state: 0,
6    reducers: {
7      incrementBy: (state, payload) => state + payload
8    },
9    effects: dispatch => ({
10     async incrementByAsync(payload) {
11       this.incrementBy();
12       await wait(2000);
13       dispatch.otherModel.actionName(payload);
14     }
15   })
16 };
```

Rematch – pluginy

- **Loading** – automatycznie ustawia *{ loading: true }* po odpaleniu dowolnego efektu;
- **Persist** – pozwala zapisać część stanu aplikacji do localStorage i przywrócić po odświeżeniu strony;
- **Updated** – zapisuje timestamp każdej zmiany w store, dzięki czemu można decydować, czy odpalić ponownie kosztowne efekty;
- **Selectors** – integracja z biblioteką reselect, która zapamiętuje obliczony stan dla danych wartości parametrów;
- **React Navigation** – integracja z react-navigation, pozwala przechodzić do różnych widoków z poziomu reduxa;
- **Immer** – integracja z immer.js, odpowiednik ImmutableJS, który blokuje bezpośrednią modyfikację nawet dla typów prostych;

Rematch – używanie pluginów - @rematch/loading

konfiguracja

```
// redux/store.js
import { init } from "@rematch/core";
import createLoadingPlugin from "@rematch/loading";

import * as models from "./models";

const loading = createLoadingPlugin({
  // plugin config
});

const store = init({
  models,
  plugins: [loading]
});

export default store;
```

użycie

```
// App.js
const mapState = state => ({
  // true, when effect "counter/incrementByAsync" is running
  isLoading: state.loading.effects.counter.incrementByAsync,

  // or

  // true, when any effect from model "counter" is running
  isLoading: state.loading.models.counter,

  // or

  // true, when any effect from any model is running
  isLoading: state.loading.global,
});
```

Rematch – używanie pluginów - @rematch/select

bez biblioteki reselect

```
// redux/models/cart.js
export default {
  name: "cart",
  state: [
    { price: 42.0, amount: 3 },
    { price: 11.0, amount: 2 },
  ],
  selectors: {
    total() {
      return (rootState, props) =>
        rootState.cart.reduce((a, b) => a + b.price * b.amount, 0);
    }
  }
};
```

Zasada działania selektorów w reselect

```
const selector = createSelector(
  getterForA,
  getterForB,
  (a, b) => expensiveCalculations(a, b)
)

selector() // calls expensiveCalculations(a, b)
selector() // returns result from cache, immediately
```

Rematch – używanie pluginów - @rematch/select

wstrzykiwanie zależności

```
// redux/models/cart.js
export default {
  name: "cart",
  state: [
    { price: 42.0, amount: 3 },
    { price: 11.0, amount: 2 },
  ],
  selectors: (slice, createSelector, hasProps) => ({
    // ...
  })
};
```

Rematch – używanie pluginów - @rematch/select

lokalny stan – slice()

```
1   // redux/models/cart.js
2   export default {
3     name: "cart",
4     state: [
5       { price: 42.0, amount: 3 },
6       { price: 11.0, amount: 2 }
7     ],
8     selectors: (slice, createSelector, hasProps) => ({
9       total() {
10        return slice(cart =>
11          cart.reduce((a, b) => a + b.price * b.amount, 0)
12        );
13      }
14    })
15  };
```

Rematch – używanie pluginów - @rematch/select

kilka parametrów – createSelector()

```
1   // redux/models/cart.js
2   export default {
3     name: "cart",
4     state: [
5       { price: 42.0, amount: 3 },
6       { price: 11.0, amount: 2 }
7     ],
8     selectors: (slice, createSelector, hasProps) => ({
9       total() {
10        return createSelector(
11          slice,
12          (state, props) => props.shipping,
13          (cart, shipping) =>
14            cart.reduce((a, b) => a + b.price * b.amount, shipping)
15        );
16      }
17    })
18  };
```

Rematch – używanie pluginów - @rematch/select

Selektory innych modeli – createSelector() + parametr models

```
// redux/models/cart.js
export default {
  name: "cart",
  state: [
    { id: 1, price: 42.0, amount: 3 },
    { id: 2, price: 11.0, amount: 2 },
    { id: 3, price: 17.1, amount: 5 }
  ],
  selectors: (slice, createSelector, hasProps) => ({
    cart() {
      return slice(state => state);
    },
    sortByHot(models) {
      return createSelector(
        this.cart,
        models.popularity.pastDay,
        (cart, hot) =>
          cart.sort((a, b) => hot[a.id] > hot[b.id])
      );
    }
  })
};
```

Rematch – używanie pluginów - @rematch/select

Selektor z parametrem – hasProps()

```
1   // redux/models/cart.js
2   export default {
3     name: "cart",
4     state: [
5       { id: 1, price: 42.0, amount: 3 },
6       { id: 2, price: 11.0, amount: 2 },
7       { id: 3, price: 17.1, amount: 5 }
8     ],
9     selectors: (slice, createSelector, hasProps) => ({
10      expensiveFilter: hasProps(function(models, lowerLimit) {
11        return slice(cart => cart.filter(product => product.price > lowerLimit));
12      }),
13      wouldGetFreeShipping() {
14        return this.expensiveFilter(20.0);
15      }
16    })
17  };
```

Rematch – pisanie własnych pluginów

```
import { init } from "@rematch/core";
import createLoadingPlugin from "@rematch/loading";

import * as models from "./models";

const myCoolPlugin = {
  config: { /* ... */ }, // merges into init config.
  expose: { /* ... */ }, // A shared object for plugins to communicate with each other.
  init: expose => ({
    onModel: model => { // called every time a model is created.
      // do something
    },
    middleware: store => next => action => {
      // do something
      return next(action)
    },
    onStoreCreated: store => { // Run last, after the store is created.
      // do something
    }
  })
}

const loading = createLoadingPlugin();

const store = init({
  models,
  plugins: [loading, myCoolPlugin]
});

export default store;
```

Rematch – zaawansowana konfiguracja store

```
// redux/store.js
import { init } from "@rematch/core";

import * as models from "./models";

const store = init({
  models,
  plugins: [],
  redux: {
    initialState: {},
    reducers: {},
    middlewares: [],
    enhancers: [],
    rootReducers: {},
    devtoolOptions: {},

    // overwrites
    combineReducers: fn,
    createStore: fn,
  }
});

export default store;
```

GitHub

https://github.com/jakubdrozdek/redux-rematch-demo

- Licznik w czystym Reduxie
- Licznik w Rematch
- Apka z demo Rematcha (loading, selectors)