REDUX

Hello

Kamil Richert

Senior Software Engineer at Atlassian

Jak to się zaczeło?

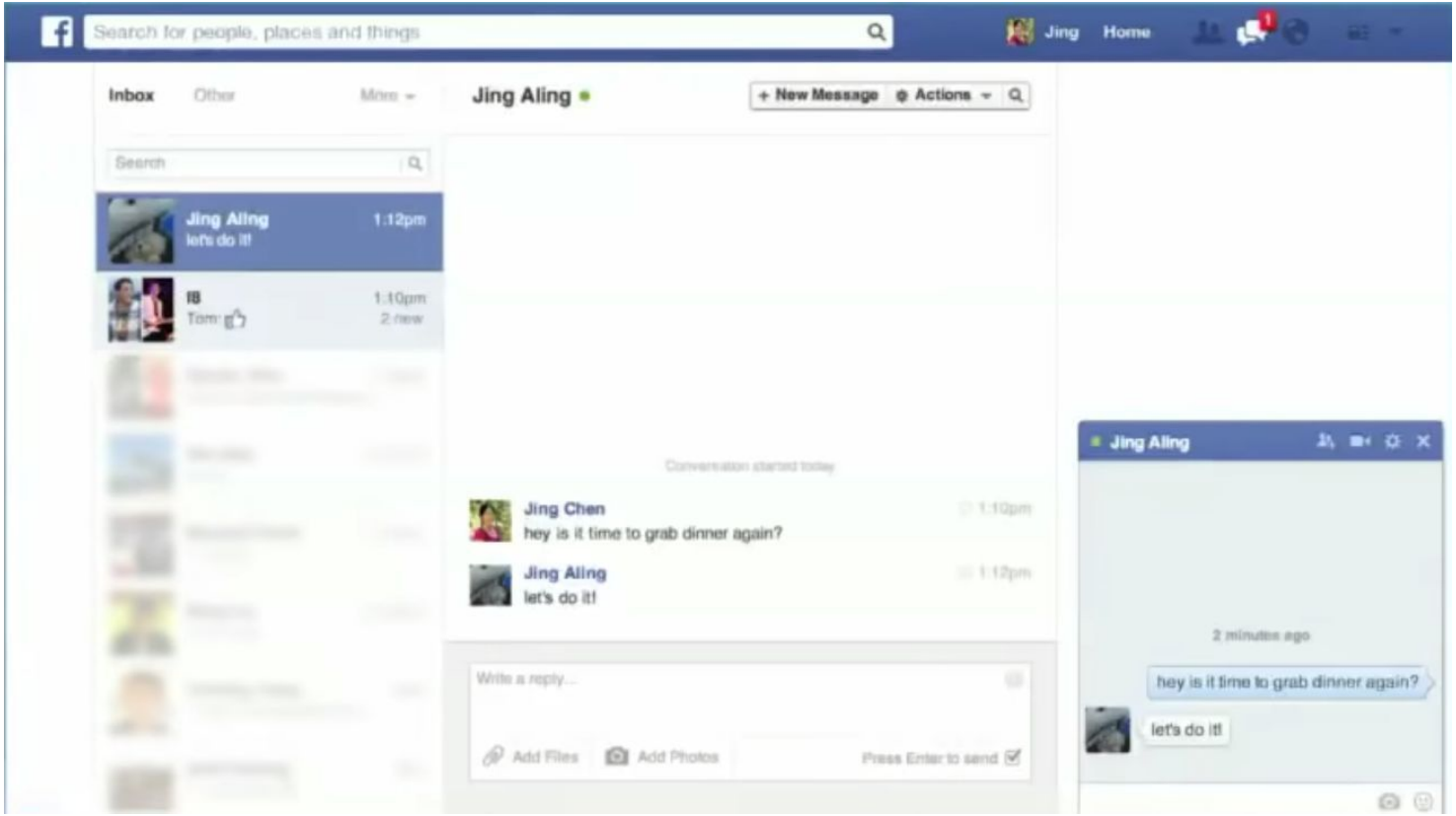
Na początku był Flux …
czyli architektura / koncept stworzony przez programistów Facebooka, który miał rozwiązać problem zarządzania stanem

Problem zarządzania stanem? Co to znaczy?

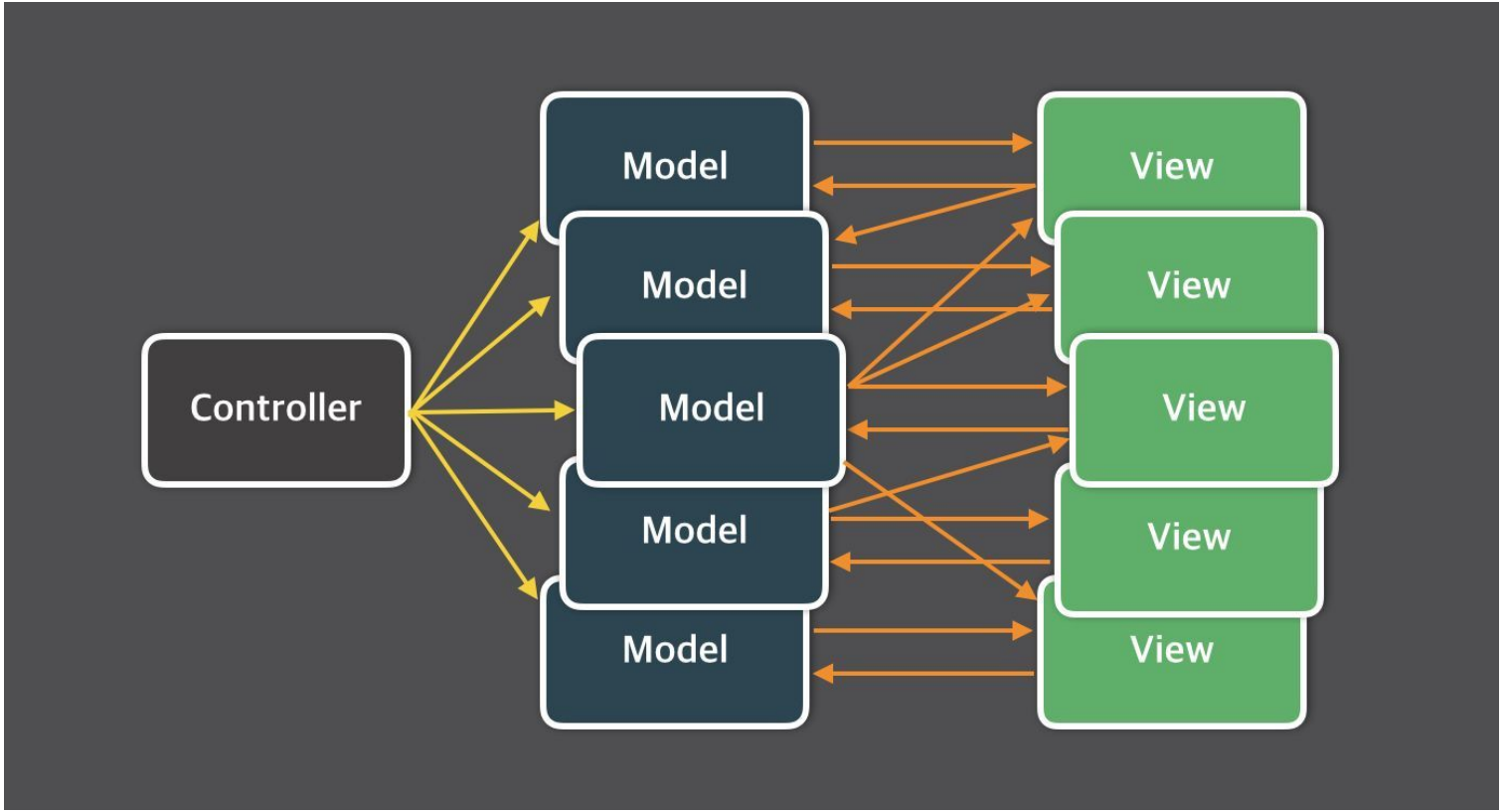
Rozwiązuje problem skalowalności (rozszerzalności) aplikacji oraz wymusza jednokierunkowy przepływ danych

Jak ten problem wyglądał u Facebooka?

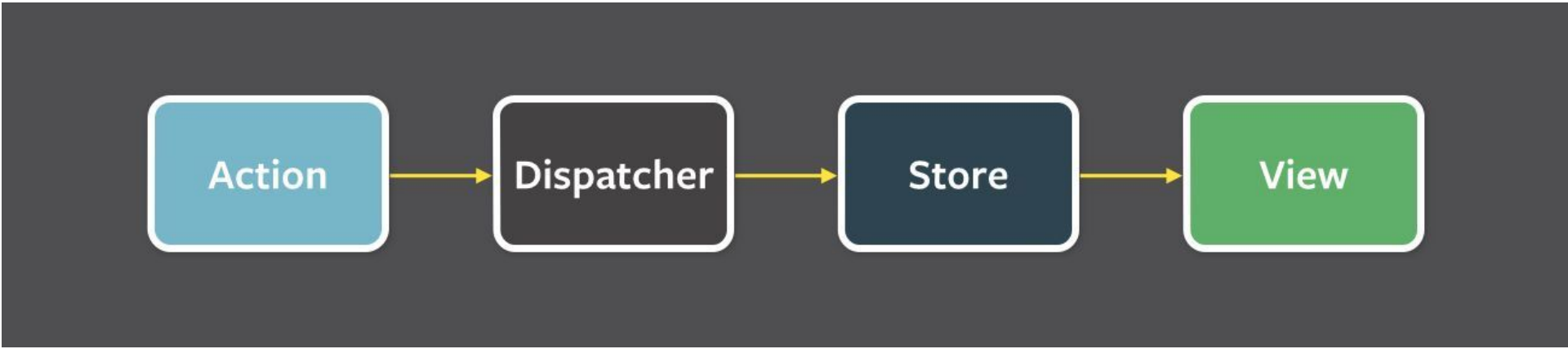
Jak ten problem wyglądał u Facebooka?



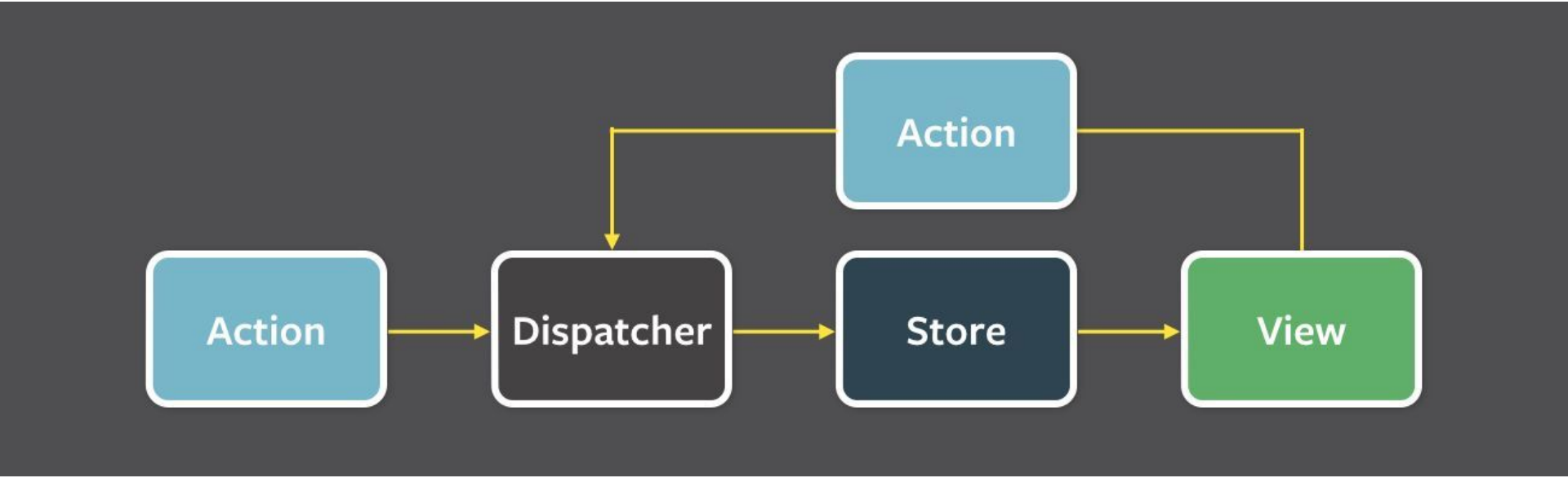
Dwukierunkowy przepływ danych



Jednokierunkowy przepływ danych



Jednokierunkowy przepływ danych



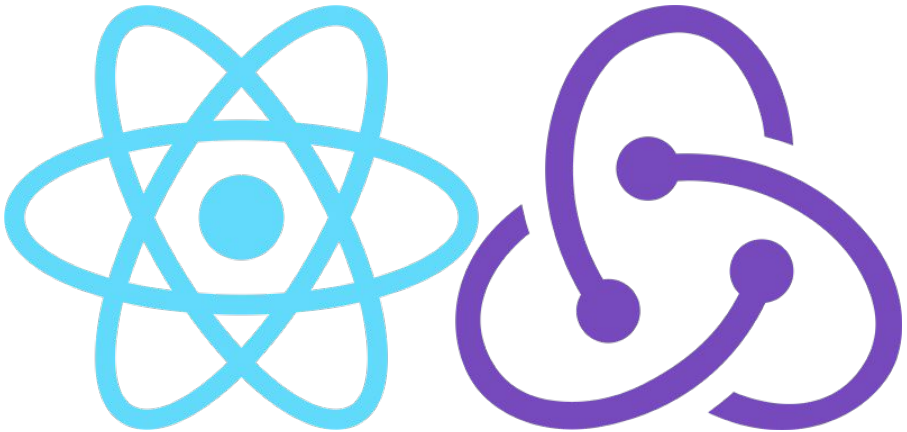
Objaśnienia

- **Action** – obiekt z danymi (np. wiadomość albo kliknięcie)
- **Dispatcher** – informuje konkretne store'y o akcji. Odpala callbacki aby poinformować store o akcjach.
- **Store** – zawiera stan i logikę aplikacji. Obsługuje akcje przesłane przez dispatcher.
- **View** – może być nim React, może być źródłem akcji (np. reakcja na akcję użytkownika)

FLUX to pomysł / architektura

REDUX to jego implementacja

REDUX



A Predictable State
Container for JS Apps

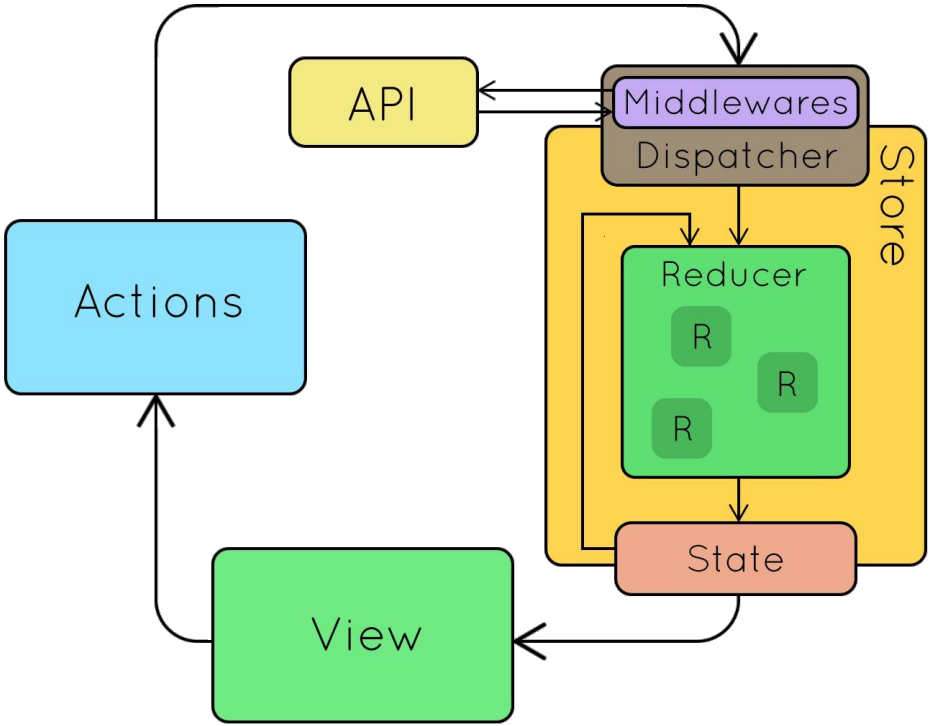
REDUX przykład bank

1. Zamiar (**ACTION**) pobrania pieniędzy (**WITHDRAW_MONEY**)
2. Podjedź do okienka (**REDUCER**) i poproś o pieniądze (**DISPATCH**)
3. Okienko / pracownik (**REDUCER**) „idzie” do skarbca (**STORE**) i wyciąg pieniądze.
1. Tylko okienko / pracownik (**REDUCER**) wie jak obsłużyć skarbiec (**STORE**) żeby wszystko się zgadzało (**STATE**).

REDUX przykład drink

1. Zamiar (**ACTION**) wypicia piwo (**HAVE_A_BEER**)
2. Podjedź do barmana (**REDUCER**) i poproś o drinka (**DISPATCH**)
3. Barman (**REDUCER**) „idzie” do półek (**STORE**) i wyciąg butelkę.
1. Tylko barman (**REDUCER**) wie jak wyciągnąć butelkę (**STORE**) żeby wszystko się zgadzało (**STATE**).

REDUX schemat



Z czego się składa REDUX?

- **ACTION** – obiekty posiadające typ, jako jedyne niosą dane oraz są odpalane za pomocą dispatch
- **REDUCER** – czyste funkcje, które określają jak się stan zmienia pod wpływem akcji
- **STORE** i **STATE –** obiekt przechowujący cały stan aplikacji, tylko do odczytu (jest niemutowalny), może na nim wykonać następujące operacje:
    - **subscribe** – nasłuchiwanie na zmiany stanu
    - **dispatch** – wysyła ACTION
    - **getState** – zwraca aktualny stan

Jak skorzystać z Reduxa?

1. Pobierz paczkę reduxa: **npm install redux**

2. Stwórz reducera (funkcję):

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

Jak skorzystać z Reduxa?

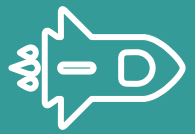
3. Stwórz obiekt store za pomocą funkcji createStore z paczki redux:
   const store = createStore(reducer);

4. Stwórz obiekt akcji:
   const action = { type: ‘action type’, payload: ‘payload’ };

5. Na obiekcie store wykonaj funkcję dispatch i przekaż w niej obiekt akcji:
   store.dispatch(action);

6. Sprawdź zmieniony stan poprzez odczytanie stanu z obiektu store.
   store.getState();

ZADANIE



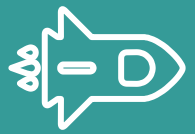
Stwórz w REDUX'ie **COUNTER**'a.

1. Stwórz funkcje (reducer) counter i obsłuż akcje: INCREMENT, DECREMENT, RESET.
2. Stwórz store za pomocą funkcji **createStore** i użyj w niej stworzonego reducera.
3. Użyj obiektu **window** by "wyciągnąć" store do konsoli i sprawdź czy reducer jest dobrze zaimplementowany poprzez wywołania **dispatch** na store z odpowiednią akcją.

REDUX DEVTOOLS

- dodaj redux devtools do google chrome
- dodaj window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__() jako 2 argument funkcji createStore

ZADANIE



Stworzymy w REDUX'ie **BANK** z przykadu nr 1.

1. Stwórz funkcje (reducer) bank i obsłuż akcje: DEPOSIT, WITHDRAW, WITHDRAW_ALL, BALANCE. Limit dla wpłaty to 1000zł.
2. Stwórz store za pomocą funkcji **createStore** i użyj w niej stworzonego reducera.
3. Użyj obiektu **window** by "wyciągnąć" store do konsoli i sprawdź czy reducer jest dobrze zaimplementowany poprzez wywołania **dispatch** na store z odpowiednią akcją.

REACT-REDUX

paczka ułatwiająca użycie Redux’a w React’cie

REACT-REDUX
Co najważniejsze?

Provider, useDispatch, useSelector, useStore

react-redux

Provider

komponent Reactowy, który daje dostęp do stora Reduxowego wszystkim dzieciom-komponentom, które użyją connect()

- przyjmuje props store

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

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

react-redux

useSelector

Umożliwia wyodrębnienie danych ze stanu Redux za pomocą funkcji selektora.

Funkcja selektora powinna być czysta, ponieważ potencjalnie jest wykonywana wiele razy i w dowolnym momencie.

Selektor zostanie wywołany z całym stanem magazynu Redux jako jedynym argumentem.

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

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

react-redux

useDispatch

Ten hook zwraca odwołanie do funkcji dispatcj z Redux. Możesz go użyć do wysyłania akcji.

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

const dispatch = useDispatch()

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

react-redux

useStore

Ten hook zwraca odwołanie do tego samego **store** Redux'a, który został przekazany jako props do komponentu <**Provider**>.

Nie powinien być często używany. useSelector() powinien być podstawowym wyborem.

Uwaga! Komponent nie zaktualizuje się automatycznie, jeśli zmieni się stan sklepu przy użyciu tego hook'a.

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

const store = useStore()

const state = store.getState()
```

REDUX – podłączenie do Reacta

1. Stwórz store za pomocą funckji **createStore** z paczki redux oraz stwórz reducer który przekażesz jako argument do **createStore.**

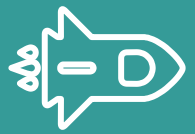
2. Zainstaluj paczkę react-redux.

3. “Owrapuj” aplikację komponentem **Provider** z paczki react-redux i przekaż do niego stworzony store.

4.Użyj **useSelector** by otrzymać dane z Redux’a.

5. Aby wykonać akcję wykorzystaj zwrot z hook’a **useDispatch**.

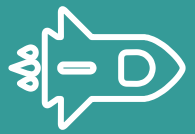
ZADANIE



Stwórzmy aplikację **COUNTER** przy pomocy React-Redux.

1. Zaczniemy od funkcji (reducer) counter i obsłuż w niej akcje: INCREMENT, DECREMENT, RESET.
2. Stwórzmy **store** w pliku *store.ts* i wykorzystaj w nim stworzony reducer z kroku 1.
3. Dodaj action creactory dla każdej akcji (funkcje tworzące obiekt akcji).
4. Dodaj komponent Provider z przekazanym storem w *index.ts*
5. Wywołaj funkcję **dispatch** z action creatorami na kliknięcie przycisków.

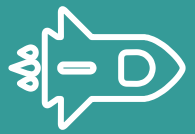
ZADANIE



Stwórz aplikację **WYPOŻYCZALNI** przy pomocy React-Redux.

1. Stwórz funkcje (reducer) rentalOffice i obsłuż akcje: dodawania, usuwania, zaznaczania wypożyczenia i zwrócenia.
2. Wspólnie zaktualizujemy **store** w pliku *store.ts* by obsługiwał 2 reducery. Naprawmy po tej operacji Countera.
3. Dodaj action creactory dla każdej akcji (funkcje tworzące obiekt akcji).
4. Wywołaj funkcję **dispatch** z action creatorami na kliknięcie przycisków i submit formularza.

ZADANIE



Stworzymy aplikację **SKLEPU** przy pomocy React-Redux.

1. Stwórz funkcje (reducer) shopCart i obsłuż akcje: dodawania do koszyka i usuwania z niego.
2. Zaktualizuj **store** w pliku *store.ts* by obsługiwał 3 reducery.
3. Dodaj action creactory dla każdej akcji (funkcje tworzące obiekt akcji).
4. Wywołaj funkcję **dispatch** z action creatorami na kliknięcie przycisków w sklepie.
5. Wykorzystaj zawartość redux by aktualizować koszyk.

REDUX-THUNK

A co jeżeli akcje mają się dziać asynchronicznie?

Czym jest ten thunk?
Funkcją. Thunk jest specjalną nazwą dla funkcji, która jest zwracana przez inną funkcję.

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

Aby wykonać **asynchroniczne** akcje musimy wykorzystać middleware redux-thunk.

Możemy dodać ten middleware do store'a za pomocą funkcji **applyMiddleware** z reduxa.

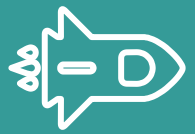
```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
import rootReducer from './reducers/index';

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

REDUX-THUNK

Przykład wykorzystania redux-thunk:

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

function incrementAsync() {
  return (dispatch) => {
        // Yay! Możemy wywołać asynchroniczne akcję z dispatch
        setTimeout(() => { dispatch(increment()); }, 1000);
  };
}
```

ZADANIE



Dodaj przyciski w **COUNTER**’erze, które wykonują daną akcję po 3 sekundach (dodawanie, odejmowanie, reset). Wartość po 3 sekundach ma się zmienić o 5.

Pokaż spinner w momencie wywołania akcji synchronicznej do momentu jej wykonania.

RESELECT

Paczka ułatwiająca pracę z selektorami w reduxie (mapStateToProps)

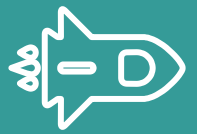
RESELECT

```
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 }
```

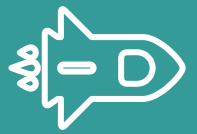
ZADANIE



Wykorzystując paczkę **reselect** przenieś logikę z komponentu do stanu:

1. Z komponentu ShoppingCart wynieś obliczanie całkowitej kwoty do selektora.
2. Z komponentu Shop wynieś sprawdzenie czy dany element jest już w koszyku.

ZADANIE



Dodatkowe:

Dodaj możliwość dodawania więcej niż 1 egzemplarza danego produktu - dodaj nową pozycję do obiektu w tablicy, np *amount*.

Przy zakupie co najmniej 5 elementów danego egzemplarza dodaj dodatkową zniżkę 5%. Wykorzystaj selector by odczytywać czy dodatkowa zniżka powinna być wliczana.

REDUX TOOLKIT

Oficjalny zestaw narzędzi do wydajnego rozwoju Redux

REDUX TOOLKIT

Pakiet Redux Toolkit ma być standardowym sposobem pisania logiki Redux. Został pierwotnie stworzony, aby pomóc w rozwiązaniu trzech typowych problemów związanych z Redux:

„Konfiguracja store Redux jest zbyt skomplikowana”

„Muszę dodać wiele pakietów, aby Redux zrobił coś użytecznego”

„Redux wymaga zbyt dużego kodu standardowego”

REDUX TOOLKIT

Co zawiera?

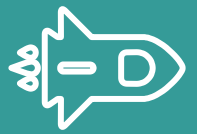
- configureStore(): zapewnia uproszczone opcje konfiguracji i wartości domyślne. (zawiera redux-thunk i umożliwia korzystanie z rozszerzenia Redux DevTools)
- createReducer(): pozwala dostarczyć tabelę typów akcji do funkcji reducera, zamiast pisać instrukcje switch.
- createAction(): generuje funkcję tworzenia akcji dla podanego ciągu typu akcji.
- createSelector(): z biblioteki Reselect, ponownie wyeksportowane w celu ułatwienia użycia.
- createSlice(): akceptuje obiekt funkcji reducer, nazwę slice i wartość stanu początkowego oraz automatycznie generuje reducer z odpowiednimi action creatorami i typami akcji

  i wiele innych.

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
```

ZADANIE



Przepisz reducer od **rentalOffice** na taki co będzie używał redux toolkit.

REDUX podsumowanie

Największe zalety Redux'a:

1. Jednokierunkowy przepływ danych
2. Przewidywalny
3. Skalowalność
4. Łatwość testowania
5. Rozwiązuje problem z props drilling
6. Łatwy dostęp do stanu aplikacji z każdego miejsca w kodzie

REDUX SAGA

to biblioteka, której celem jest tworzenie efektów ubocznych aplikacji

REDUX SAGA

Saga jest jak osobny wątek w twojej aplikacji, który jest wyłącznie odpowiedzialny za skutki uboczne. Redux-saga jest middleware Reduxa, co oznacza, że ma dostęp do pełnego stanu aplikacji i może również wysyłać akcje.

Wykorzystuje funkcję ES6 o nazwie generatory. Dzięki temu asynchroniczny kod wygląda jak standardowy synchroniczny kod JavaScript.

Więcej o generatorach:
https://github.com/gajus/gajus.com-blog/blob/master/posts/the-definitive-guide-to-the-javascript-generators/index.md

REDUX SAGA

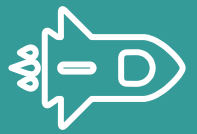
Wybrane efekty:

- **put** - tworzy opis efektu, który instruuje oprogramowanie pośredniczące, aby zaplanować wysłanie akcji do store. To może nie być natychmiastowe, ponieważ inne zadania mogą znajdować się w kolejce zadań sagi lub nadal być w toku.
- **take** - tworzy opis efektu, który nakazuje oprogramowaniu pośredniczącemu czekać na określoną akcję w store. Generator zostaje zawieszony do czasu wywołania akcji zgodnej ze wzorcem.
- **takeEvery** - tworzy sagę przy każdej akcji wysłanej do store, która pasuje do przekazanego wzorca
- **takeLatest** - rozwidla sagę przy każdej akcji wysłanej do store, która pasuje do wzorca i automatycznie anuluje wszystkie wcześniej rozpoczęte zadania sagi, jeśli nadal są uruchomione.
- **call** - tworzy opis efektu, który instruuje oprogramowanie pośredniczące, aby wywołało przekazaną funkcję

REDUX SAGA

```
import { call, put, takeEvery, takeLatest } from 'redux-saga/effects'

function* fetchUser(action) {
  try {
     const user = yield call(Api.fetchUser, action.payload.userId);
     yield put({type: "USER_FETCH_SUCCEEDED", user: user});
  } catch (e) {
     yield put({type: "USER_FETCH_FAILED", message: e.message});
  }
}

function* mySaga() {
 yield takeLatest("USER_FETCH_REQUESTED", fetchUser); // takeEvery
}


export default mySaga;
```

ZADANIE



Wykonaj to samo zadanie dla asynchronicznych przycisków, ale z użyciem Redux Saga.

Stwórz nowe akcje asynchroniczne, na które będziesz nasłuchiwać w saga i w subskrypcji wywołaj odpowiednią akcję do pokazania Spinnera, odczekaj 3 sekundy i zmień wartość countera o 5.

Dzięki

Znajdziecie mnie:
https://www.linkedin.com/in/kamil-richert/
https://github.com/krichert