WARSZTATY

DEMO

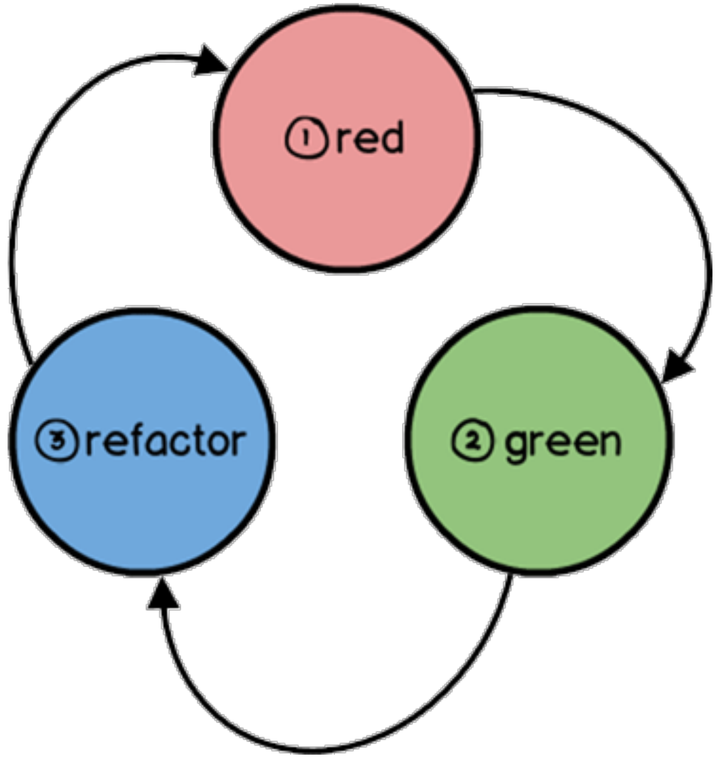
STACK

- React
- Redux
- Create React App
- Enzyme
- Jest
- Jasmine

TDD

CZYM JEST TDD?

- Test-Driven Development
- nie służy stricte testowaniu
- pomaga wyeliminować błędy w zrozumieniu działania



STRUKTURA TESTU

```
function add(a, b) { ... }

describe('Adding function', () => {

  it('adds two numbers', () => {
    expect(add(1, 2)).toBe(3);
  });

  it('fails when on wrong params given', () => {
    expect(add()).toBeNull();
  })

});
```

ENZYME

```
describe('My React Component', () => {
  it('renders a number passed through a prop', () => {
    // ?
  });
});
```

```
describe('My React Component', () => {

  it('renders a number passed through a prop', () => {
    const wrapper = render(<Component prop={2}/>,
                           document.getElementById('app'));

    expect(wrapper.find('span').text()).toEqual('2');
  });

});
```

```
import { shallow, mount } from 'enzyme';

describe('My React Component', () => {
  it('renders a number passed through a prop', () => {
    const wrapper = shallow(<Component prop={2}/>);

    expect(wrapper.find('span').text()).toEqual('2');
  });

  it('renders a number passed through a prop', () => {
    const connect = jest.fn();
    const wrapper = mount(<Component onconnect={connect}/>);
    expect(connect).toBeCalled();
  });
});
```

ZADANIE 1 – PIERWSZY KOMPONENT I TESTY

- Instalacja paczek
  - `yarn add enzyme`
  - `yarn add react-addons-test-utils`
- Test dla komponentu licznika osób na stronie
  - `Counter component displays a number passed to a prop`
  - `Counter component displays 0 if props hasn't been passed`
- Komponent licznika

http://airbnb.io/enzyme/docs/api/index.html

REDUX

CZYM JEST REDUX?

Redux jest przewidywalnym kontenerem stanu aplikacji JS.

TRZY ZASADY

- Single source of truth
- Stan jest tylko do odczytu
- Zmiany wykonujemy za pomocą „czystych” funkcji

ACTIONS & ACTION CREATORS

```
const ADD_TODO = 'todos/ADD_TODO';
```

```
{
  type: ADD_TODO,
  payload: { text: 'Build my first Redux app' }
}
```

```
const addTodo = (text) => ({
  type: ADD_TODO,
  payload: { text }
});
```

REDUCER

```
const initialState = [];

const todoReducer = (state = initialState, action) => {
  switch (action.type) {









    default:
      return state;
  }
};
```

REDUCER

```
const initialState = { todos: [] };

const todoReducer = (state = initialState, action) => {
  switch (action.type) {
    case ADD_TODO:
      return [
        ...state.todos,
        {
          text: action.payload.text,
          completed: false
        }
      ];

    default:
      return state;
  }
};
```

SKŁADANIE REDUCERÓW

```
import { combineReducers } from 'redux';
```

```
const appReducer = combineReducers({
  todo: todoReducer,
  routing: routerReducer
});
```

ZADANIE 2 – AKCJE I REDUCERY

- Tworzymy akcję do aktualizacji stanu licznika
- Tworzymy reducer, obsługujący aktualizację
- Pamiętamy o testach
  - `Users actions creates update action`
  - `Users reducer returns default state`
  - `Users reducer updates counter`

```
{
  "users": {
    "count": 2
  }
}
```

CONTAINER

CZYM JEST KONTENER?

- „pomost” pomiędzy komponentem, a reduxem
- umożliwia użycie z reduxem już istniejących komponentów bez potrzeby ingerowania w ich kod
- ułatwia separację zadań
  - pobieranie/przetwarzanie danych
  - prezentacja danych

CONNECT()

```
connect(
   [mapStateToProps],
   [mapDispatchToProps],
   [mergeProps],
   [options]
)
```

odpalana przy każdej
zmianie zawartości store'a

zazwyczaj nie
są potrzebne ;-)

```
import { connect } from 'react-redux';
import { ToDo } from './todo.component';
import { addTask } from '../todo/todo.actions';

const mapStateToProps = (state) => ({
  list: state.taskList
});

const mapDispatchToProps = (dispatch) => ({
  add: task => dispatch(addTask(task))
});

export const ToDoContainer = connect(
  mapStateToProps,
  mapDispatchToProps
)(ToDo);
```

ZADANIE 3 - REDUX + REACT

- Tworzymy kontener dla naszego komponentu

TESTOWANIE ZACHOWANIA KOMPONENTÓW

TESTOWANIE WYWOŁANIA FUNKCJI

```
const myProps = {
  mockFn: jest.fn()
};


myProps.mockFn(10);


expect(myProps.mockFn).toBeCalledWith(10);
```

SYMULOWANIE ZMIANY PARAMETROW

```
const connect = jest.fn();

const wrapper = shallow(<SomeComponent displayButton={false} />);

expect(wrapper.find('button').length).toBe(0);

wrapper.setProps({displayButton: true});

expect(wrapper.find('button').length).toBe(1);
```

SYMULOWANIE INTERAKCJI

```
const clear = jest.fn();

const wrapper = shallow(<MyComponent clearFields={clear} />);

wrapper.find(,button').simulate('click');

expect(clear).toBeCalled();
```

ZADANIE 4 – PRZYGOTOWANIE DO IMPLEMENTACJI KOMUNIKACJI Z SOCKETEM

- Komponent wyświetlający aktualny stan połączenia z socketem (**connected**, **not-connected**, **connecting**)
- Komponent pozwalający wykonać akcję podłączenia się do systemu
- Akcje i reducery do komunikacji z socketem
- Pamiętamy o testach :)

```
{
  "socket": {
    "status": "connected"
  }
}
```

WEBSOCKET

OD CZEGO ZACZYNAMY ?

```
const ws = new WebSocket(url);
```

PRZECHOWYWANIE ADRESU SERWERA

```
REACT_APP_SOCKET_URL=ws://localhost:8080
```

```
const ws = new WebSocket(process.env.REACT_APP_SOCKET_URL);
```

METODY

```
ws.onmessage = ({data}) => {...};
```

```
ws.onopen = () => {...};
```

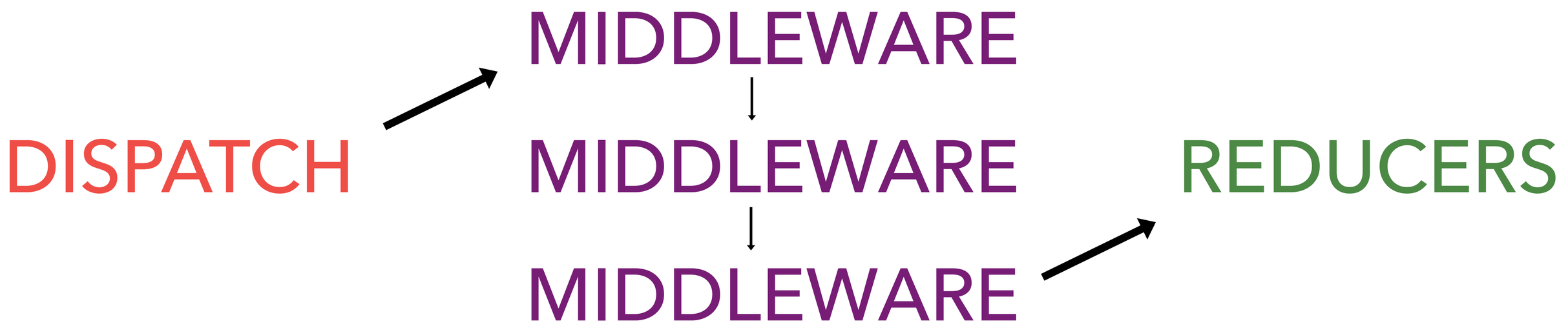
PRZEKAZYWANIE INFORMACJI

```
{
  data: '{"type":"users","count":10}'
}
```

WEBSOCKET

+

REDUX


DISPATCH
MIDDLEWARE
MIDDLEWARE
MIDDLEWARE
REDUCERS

MIDDLEWARE

```
export const middleware = function(store) {
  return function (next) {
    return function (action) {
     next(action);
    }
  }
};
```

WERSJA ES6

```
export const socketMiddleware = store => next => action => {...}
```

INTEGRACJA Z REDUX

```
const store = createStore(
  appReducer,
  compose(
    applyMiddleware(
      socketMiddleware,
      routerMiddleware(browserHistory)
    ),
    window.devToolsExtension ? window.devToolsExtension() : f => f
  )
);
```

MOCKOWANIE WEBSOCKETU

```
window.WebSocket = window.WebSocket || function () {};
```

MOCKOWANIE TWORZENIA OBIEKTU

```
const myFakeObject = {
  mockFn: jest.fn()
};

window.MySuperObject = jest.fn(() => myFakeObject);
```

ZADANIE 5 – KOMUNIKACJA Z SOCKETEM

- Tworzymy middleware obsługujący komunikację z websocketem i reduxem
- Pamiętamy o testach :)

Adres serwera: ws://192.168.88.84:8080

```
{
  data: '{"type":"users","count":10}'
}
```