1. Поставити 2 пакета react-redux redux

npm i react-redux redux

2) Створюємо в папці src папку redux

3) в папці redux створюємо 3 папки store, actions, reducers

4) в index.js обгортаємо наш App в компонент Provider і передаємо йому пропсом store силку на обєкт storе, що буде створений в папці redux/store

import { Provider } from 'react-redux';

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>

, document.getElementById('root'));

5) Створюємо файл store.js який знаходиться за адресою redux/store/store.js;

6) імпортуємо в файл store.js метод createStore з бібліотеки redux

import { createStore } from 'redux'; //функция которая дает возможность создать глобальный объект сторе

7) підключаємо REDUX DEVTOOLS до проекта створюючи константу const DevTools = window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_();

const DevTools = window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_(); //в консоле сможем следить за изменениями стейта, инструмент только для нас для разработчиковю Инсрумент дает много возможностей (например сохранить историю кликов, которые привели к багу, или тестировщик так может сделать)

8) Створюємо 2 файла в папці redux/reducers

8.1) counterReducer.js - файл в якому створиться поле для глобального стора. (логика работы с полями)

8.2) index.js - всі редюсери обєднуються в один обєкт (куда будут импортиться все поля)

9) створюємо ф-ю reducer в якій через switch описуємо як буде мінятися стейт в залежності від типу екшена що запускається(від команди);

В counterReducer.js :

export default function count (state = 0, action) {

switch (action.type) {

case 'plus':

return state + 1;

case 'minus':

return state - 1;

case 'reload':

return 0;

default:

return state;

}

}

10) import { combineReducers } from 'redux' в файлі index.js що знаходиться в папці redux/reducers;

import {combineReducers} from 'redux'; // объединяет все поля в один объект

import count from '../reducers/counterReducer';

const rootReducer = combineReducers({

count: count,

})

export default rootReducer;

11) імпортуємо створений редюсер в файл index.js що знаходиться в папці redux/reducers;

12) створюємо змінну rootReducer куди записуємо виклик метода combineReducers для того щоб сформувати фінальний обєкт стейта всієї програми. (записуємо в обєкт ключі з імпортів що робимо в пункті №11); Не забути додати export default rootReducer для того щоб можна було цю змінну використати в пункті №13.

import {combineReducers} from 'redux'; // объединяет все поля в один объект

import count from '../reducers/counterReducer';

const rootReducer = combineReducers({

count: count,

})

export default rootReducer;

13) в store.js імпортуємо змінну rootReducer яка знаходиться за адресою redux/reducers/index.js;

import rootReducer from '../reducers/allReducers';

14) в файлі store.js створюємо змінну store куди записуємо виклик метода createStore(). Аргументами передаємо змінну rootReducer з пункта №13 і змінну DevTools з пункта № 7; І експортуємо змінну store для подальшого використання.

import { createStore } from 'redux'; //функция которая дает возможность создать глобальный объект сторе

import rootReducer from '../reducers/index';

const DevTools = window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ && window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_(); //в консоле сможем следить за изменениями стейта, инструмент только для нас для разработчиковю Инсрумент дает много возможностей (например сохранить историю кликов, которые привели к багу, или тестировщик так может сделать)

const store = createStore(rootReducer, DevTools);

export default store;

15) в файл index.js (ОСНОВНОЙ!!!) імпортуємо константу store з файла store.js і передаємо її пропсом в компонент Provider для того щоб підключити глобальний обєкт store до нашого App.

import React from 'react';

import ReactDOM from 'react-dom';

import { Provider } from 'react-redux'; // мы подключили

import store from './redux/store/store'; // мы подключили

import './index.css';

import App from './App';

import \* as serviceWorker from './serviceWorker';

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>

, document.getElementById('root'));

// If you want your app to work offline and load faster, you can change

// unregister() to register() below. Note this comes with some pitfalls.

// Learn more about service workers: http://bit.ly/CRA-PWA

serviceWorker.unregister();

16) Створюємо екшени - ф-ї що повертають обєкт з полем type в якому написана команда на яку має реагувати редюсер.

Команда в case редюсера має співпадати з командою в полі type для екшена.

В папке action создаем counterАction.js :

export const plus = () => {

return {

type: 'plus'

}

}

export const minus = () => ({

type: 'minus'

})

export const reset = {

type: 'reload'

}

17) Вся взаємодія між користувачем і програмою відбувається через екшени. Тому якщо ми хочемо міняти стейт через кліки чи інші події, то потрібно імпортувати екшени в компонент де буде відбуваєтися навішування на кнопки.

18) Якщо ми хочемо підєднати компонент до store нам потрібно імпортнути в компонент метод connect з пакета react-redux.

В компоненте Сounter.jsx

import { connect } from 'react-redux'; // мы подключили, подключили компонент к стейту

19) Створюємо ф-ю mapStateToProps яка приймає параметром store і повертає обєкт. В обєкті ми пишемо назву пропса за якою будемо звертатися до поля зі store в нашому компоненті.

Под капотом так работает:

props ()

mapStateToProps = {count: 0}

mapDispatchToProps = {plus: func, minus: func, reset: obj}

connect =

props = {

count: 0,

plus: func,

minus: func,

reset: obj,

}

20) Створюємо ф-ю mapDispatchToProps яка приймає параметром метод dispatch який буде активувати наші екшени. Ця ф-я також повертає обєкт з полями в яких записані ф-ї що викликають метод dispatch() аргументами якого є екшени що імпортовані в компонент в пункті № 17.

function mapStateToProps (state) {

return {

count: state.count,

}

}

function mapDispatchToProps (dispatch) {

return {

plus: function (number) {

dispatch(plus(number));

},

minus: function (number) {

dispatch(minus(number));

},

reset: function () {

dispatch(reset);

}

}

}

21) Щоб підєднати компонент до стора використовуємо метод connect що імпортували в компонент в пункті № 18. Цей метод приймає аргументами 2 ф-ї. Перший аргумент це ф-я mapStateToProps з пункта № 19 яка дає доступ компоненту до конкретного поля в глобальному стейті. Другий аргумент це ф-я mapDispatchToProps з пункта № 20, що дає доступ до активації екшенів через метод dispatch. Записується це так

connect(mapStateToProps,mapDispatchToProps)(назва компонента);

export default connect(mapStateToProps, mapDispatchToProps)(Counter);

22) Після connect ми маєм доступ до полів стора і екшенів через пропси. В тупому компоненті через props. В розумному через this.props.

render() {

console.log(this.props);

// let {count} = this.state;

return (

<div className='container'>

<p className="count">{this.props.count}</p>

<div className="buttons">

<button className="btn" onClick={() => this.props.minus(3)}>-</button>

<button className="btn" onClick={this.props.reset}>reset</button>

<button className="btn" onClick={() => this.props.plus(3)}>+</button>

</div>

</div>

);

}

}