React & Redux

Оглавление:

React + Babel + Webpack

1. Создайте файл **package.json**
```
npm init -y
```

2. Установите **webpack**
```
npm install -g webpack
npm install --save-dev webpack webpack-cli
```

3. Установите **react**
```
npm install --save-dev react react-dom
```

4. Установите **babel**
```
npm install --save-dev babel-loader
npm install --save-dev babel-core
npm install --save-dev babel-preset-es2015
npm install --save-dev babel-preset-react
npm install --save-dev babel-preset-env
npm install --save-dev babel-preset-stage-0
```

5. Установите и запуск простого сервера **http-server** (чтобы остановить нажмите Ctrl+C)
```
npm install -g http-server
```

запуск сервера
```
http-server
```

6. Сборка билда
```
npx webpack --config webpack.config.js
```

7. Упрощенная сборка билда (если правило прописано в package.json)
```
npm run build
```

Структура проекта:

[project]
|---- **[dist]**
    |---- **index.html**
|---- **[src]**
    |---- **[components]**
        |---- **mycomponent.js**
    |---- **index.js**
|---- **package.json**
|---- **webpack.config.js**

Описание файла: **package.json**

```
{
  "name": "example-3",
  "version": "1.0.0",
  "description": "Test React",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "npx webpack --config webpack.config.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "babel-core": "^6.26.0",
    "babel-loader": "^7.1.4",
    "babel-preset-env": "^1.6.1",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "babel-preset-stage-0": "^6.24.1",
    "react": "^16.3.1",
    "react-dom": "^16.3.1",
    "webpack": "^4.5.0",
    "webpack-cli": "^2.0.14"
  }
}
```

После установки всех вышеперечисленных пакетов они будут записаны в package.json

---

Описание файла: **webpack.config.js**

```
const path = require('path');

module.exports = {
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'bundle.js',
        sourceMapFilename: 'bundle.map'
    },
    devtool: '#source-map',
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /(node_modules)/,
                loader: 'babel-loader',
                query: {
                    presets: ['env', 'stage-0', 'react']
                }
            }
        ]
    },
};
```

В конфиге прописан путь сборки всех скриптов и транспиляция кода с помощью Babel.

Описание файла: **index.html** (в папке dist)

```
<!DOCTYPE html>
<html>

<head>
  <meta name="viewport" content="minimum-scale=1.0, width=device-width,
                                          maximum-scale=1.0, user-scalable=no" />
  <meta charset="utf-8">
  <title>React Application</title>
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>

<body>
  <div id="react-container"></div>
  <script src="bundle.js"></script>
</body>

</html>
```

Подключаются библиотеки React и Babel. В теле страницы вызывается основной скрипт.

Описание файла: **index.js** (в папке src)

```
import React from 'react'
import { render } from 'react-dom'
import MyComponent from './components/mycomponent'

window.React = React;

render(
    <MyComponent title="React" subtitle="Фреймворк от Facebook"/>,
    document.getElementById("react-container")
);
```

Импортируется иблиотека React и модуль компонента. Далее происходит визуализация.

Описание файла: **mycomponent.js** (в папке src/components)

```
class MyComponent extends React.Component {
    constructor(props) {
        super(props);
    }

    render() {
        return (
            <div className="container" >
                <h1>{this.props.title}</h1>
                <p>{this.props.subtitle}</p>
            </div >
        );
    }
}

export default MyComponent
```

Основы React

Простой пример вывода сообщения Hello World!

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>Hello World</title>
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <div id="block"></div>
  <script type="text/babel">
      ReactDOM.render(
        <h1>Hello, world!</h1>,
        document.getElementById('root')
      );

      class Block extends React.Component {
        render() {
          return <h3>This is React!</h3>;
        }
      }

      ReactDOM.render(
        <Block />,
        document.getElementById('block')
      );
    </script>
</body>
</html>
```

Создание элемента **createElement** с последующей визуализацией

Простой пример создания тега <h1>

```
var element = React.createElement('h1', null, 'Hello World!!!');
ReactDOM.render(element, document.getElementById('div_message'));
```

Пример создания дерева элементов

```
var image = React.createElement('img', {src:
'http://localhost:8080/example/img/react.png'}); // передается свойство src
var titel = React.createElement('h1', null, 'React');
var subtitel = React.createElement('p', null, 'Библиотека');
var container = React.createElement('div', {className: 'container'}, image,
                                                        titel, subtitel);

ReactDOM.render(container, document.getElementById('root'));
```

Виртуальный DOM

Вместо того, чтобы взаимодействовать с DOM напрямую, мы работаем с его легковесной копией. Мы можем вносить изменения в копию, исходя из наших потребностей, а после этого применять изменения к реальному DOM.

```
var h1 = React.createElement('h1', null, 'Виртуальный DOM (JS)');
var div = React.createElement('div', null, h1);
var dom = ReactDOM.render(div, document.getElementById('test_dom'));
console.log(dom);
```

Создание компонента **React.Component**

Создание простого компонента <Block />

```
class Block extends React.Component {
	render() {
		return <h3>This is React!</h3>;
	}
}

ReactDOM.render(
	<Block />,
	document.getElementById('block')
);
```

Методы описаны оп ссылке: https://reactjs.org/docs/react-component.html

Встроенные методы:

constructor()
static getDerivedStateFromProps()
componentWillMount() / UNSAFE_componentWillMount()
render()
componentDidMount()

Методы обновления состояния:

componentWillReceiveProps() / UNSAFE_componentWillReceiveProps()
static getDerivedStateFromProps()
shouldComponentUpdate()
componentWillUpdate() / UNSAFE_componentWillUpdate()
render()
getSnapshotBeforeUpdate()
componentDidUpdate()

Размонтирование
Этот метод вызывается, когда компонент удаляется из DOM:

componentWillUnmount()

Обработка ошибок
Этот метод вызывается при возникновении ошибки во время рендеринга, в методе жизненного цикла или в конструкторе любого дочернего компонента.

componentDidCatch()

Остальные APIs

setState()
forceUpdate()

свойства класса
defaultProps
displayName

свойства экземпляра
props
state

constructor() - Конструктор для компонента React

static getDerivedStateFromProps() - вызывается после создания экземпляра компонента, а также при получении новых реквизитов. Он должен вернуть объект для обновления состояния или null, чтобы указать, что новые реквизиты не требуют каких-либо обновлений состояния.

componentWillMount() / UNSAFE_componentWillMount() - вызывается непосредственно перед установкой. Он вызывается перед render(), поэтому вызов setState() синхронно в этом методе не приведет к дополнительной визуализации.

render() - метод визуализации

componentDidMount() - вызывается сразу после установки компонента. Инициализация, требующая узлов DOM, должна описываться здесь.

componentWillReceiveProps() / UNSAFE_componentWillReceiveProps() - вызывается до того, как смонтированный компонент получит новые реквизиты. Рекомендуется использовать статический цикл getDerivedStateFromProps.

shouldComponentUpdate() - поведение по умолчанию заключается в повторном рендеринге при каждом изменении состояния

componentWillUpdate() / UNSAFE_componentWillUpdate() - вызывается непосредственно перед рендерингом при получении новых реквизитов или состояний. Используйте это как возможность выполнить подготовку до того, как произойдет обновление.

getSnapshotBeforeUpdate() - вызывается непосредственно перед тем, как последний обработанный вывод будет выполнен, например, DOM. Он позволяет вашему компоненту фиксировать текущие значения. Любое значение, возвращаемое этим жизненным циклом, будет передано как параметр componentDidUpdate ().

componentDidUpdate() - вызывается сразу после обновления. Этот метод не вызывается для первоначального рендеринга.

componentWillUnmount() - вызывается непосредственно перед размонтированием и уничтожением компонента. Выполните любую необходимую очистку в этом методе.

componentDidCatch() - это компоненты React, которые улавливают ошибки JavaScript в любом месте их дочернего дерева компонентов, регистрируют эти ошибки и отображают резервный интерфейс вместо разбитого дерева компонентов.

setState() - устанавливает изменения в состоянии компонента и сообщает React, что этот компонент и его дочерние элементы должны быть повторно отображены с обновленным состоянием. Это основной метод, который вы используете для обновления пользовательского интерфейса в ответ на обработчики событий и ответы сервера.

forceUpdate() - Принудительное обновление компонента. По умолчанию, когда состояние вашего компонента или реквизита изменяется, ваш компонент будет повторно отображать.

defaultProps - можно определить как свойство самого класса компонента, чтобы установить реквизиты по умолчанию для класса.

displayName - используется для отладки сообщений. Обычно вам не нужно явно указывать его, поскольку он выводится из имени функции или класса, который определяет компонент.

props - содержит реквизиты, которые были определены вызывающим элементом этого компонента.

state - Состояние содержит данные, специфичные для этого компонента, которые могут меняться со временем. Состояние определено пользователем, и оно должно быть простым объектом JavaScript.

Свойства компонента **props**

Компонентом называется объединённая группа элементов созданных с помощью createElement или класс React.Component

```
function MyComponentProps(props) {
    return (
        React.createElement('div', {className: 'container'},
            React.createElement('img', {src: props.imageUrl}),
            React.createElement('h1', null, props.title),
            React.createElement('p', null, props.subtitle),
        )
    );
}

var mycomponentprops = React.createElement(MyComponentProps, {
    title: "MyComponentProps React (JS)",
    subtitle: "Компонент со свойствами",
    imageUrl: "http://localhost:8080/example/img/react.png"
});

ReactDOM.render(mycomponentprops, document.getElementById('test_props'));
```

В компонент можно передать параметры которые будут доступны в props

---

Отладка с помощью React Developer Tools

Плагин для Chrome:
https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=ru&

Расширенный синтаксис JSX

Создание и визуализация элемента **ReactDOM.render**
Обычно скрипт JSX кода имеет расширение *.jsx

```
var jsx =
    <div className="container">
        <img src="http://localhost:8080/example/img/react.png" />
        <h1>React</h1>
        <p>Библиотека для создания пользовательского интерфейса</p>
    </div>;
ReactDOM.render(jsx, document.getElementById('root'));
```

Запуск скрипта на странице

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>Test React</title>
  <link rel="stylesheet" href="./src/style.css">
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>
<body>
  <h4>Для корректной работы запустите сервер: http-server</h4>
  <div id="root">Загрузка...</div>
  <script type="text/babel" src="./src/appjsx.jsx"></script>
</body>
</html>
```

Виртуальный DOM

```
var dom_jsx = (
    <div>
        <h1>Виртуальный DOM (JSX)</h1>
    </div>
);
ReactDOM.render(dom_jsx, document.getElementById('root'));
```

Компонент

```
function MyComponentJSX() {
    return (
        <div className="container" >
            <img src="http://localhost:8080/example/img/react.png" />
            <h1>MyComponent React</h1>
            <p>Компонент: Библиотека для создания пользовательского интерфейса</p>
        </div >
    );
}
ReactDOM.render(<MyComponentJSX />, document.getElementById('root'));
//ReactDOM.render(React.createElement(MyComponent, null),
                                                document.getElementById('root'));
```

Свойства компонента

```
function MyComponentPropsJSX(props) {
    return (
        <div className="container" >
            <img src={props.imageUrl} />
            <h1>{props.title}</h1>
            <p>{props.subtitle}</p>
        </div >
    );
}

ReactDOM.render(<MyComponentPropsJSX
    title="MyComponentProps React (JSX)"
    subtitle="Компонент со свойствами"
    imageUrl="http://localhost:8080/example/img/react.png" />,
document.getElementById('root'));
```

Состояние компонента **state**

Состояние хранится в state компонента созданного с помощью React.Component

Пример таймера (взять с официального сайта)

```
class Timer extends React.Component {
    constructor(props) {
        super(props);
        this.state = { seconds: 0 };
    }

    tick() {
        this.setState(prevState => ({
            seconds: prevState.seconds + 1
        }));
    }

    componentDidMount() {
        this.interval = setInterval(() => this.tick(), 1000);
    }

    componentWillUnmount() {
        clearInterval(this.interval);
    }

    render() {
        return (
            <div>
                Seconds: {this.state.seconds}
            </div>
        );
    }
}

ReactDOM.render(<Timer />, mountNode);
```

Пример состояния для компонента счетчик

```
class Counter extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            count: 1
        }
    }

    handleClick() {
        //this.setState(prevState => ({ count: prevState.count + 1 }));
        this.setState((prevState, props) => ({
            count: prevState.count + 1
        }));
        console.log('CLICK!', this.state.count);
    }

    render() {
        return (
            <div className="container" >
                <div className="count">{this.state.count}</div>
                <img src={this.props.imageUrl} onClick={this.handleClick.bind(this)} />
                <h1>{this.props.title}</h1>
                <p>{this.props.subtitle}</p>
            </div >
        );
    }
}

ReactDOM.render(
    (<div>
        <Counter title="React"
            subtitle="Фреймворк от Facebook"
            imageUrl="http://localhost:8080/example/img/react.png" />
        <Counter title="Angular 2"
            subtitle="Фреймворк от Google"
            imageUrl="http://localhost:8080/example/img/angular.png" />
    </div>),
    document.getElementById('root'));
```

Запуск скрипта на странице

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>Test React State</title>
  <link rel="stylesheet" href="./src/style.css">
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
</head>
<body>
  <div id="mountNode">Загрузка...</div>
  <div id="root">Загрузка...</div>
  <script type="text/babel" src="./src/state.jsx"></script>
</body>
</html>
```

Пакет Create React App

Основные команды:

1. Установка пакета **create-react-app**
```
npm install -g create-react-app
```

2. Создание проекта на основе пакета create-react-app
```
create-react-app project_name
```

3. Запуск сервера (остановка Ctrl+C)
```
npm start
```

4. Запуск в тестовом режиме
```
npm test
```

5. Сборка билда
```
npm run build
```

6. Копировать все конфиги и транзитивные зависимости (Webpack, Babel, ESLint и т. д.) в ваш проект. (данная команда необратима)
```
npm run eject
```

7. При повторном развертывании проета, чтобы установить все пакеты описанные в файле package.json введите команду
```
npm install
```

Создание компонента с выводом приветствия Hello World!

В папке [src] создаем папку [scripts] где будем хранить наши тестовые скрипты.
В папке [scripts] создаем и описываем файл HelloWorld.js

```
import React, { Component } from 'react';

class HelloWorld extends Component {
    render(){
        return (
            <div>Hello World!</div>
        );
    }
}

export default HelloWorld;
```

В файле App.js импортируем наш компонент и добавим его в render

```
import HelloWorld from './scripts/HelloWorld';

<HelloWorld />
```

Локальный state

Если вам нужно чтобы какие то изменения происходили только внутри конкретной компоненты, то всегда стоит использовать локальный стейт.

В папке [scripts] создаем и описываем файл Dropdown.js

```
import React, { Component } from 'react';

class Dropdown extends Component {
    constructor(props){
        super(props);
        this.state = { isOpened: false };
    }
    onClick(){
        this.setState({ isOpened: !this.state.isOpened });
    }
    render (){
        return (
            <div onClick={this.onClick.bind(this)}>
             Its dropdown (click me), isOpened: {this.state.isOpened.toString()}
            </div>
        );
    }
}
export default Dropdown;
```

Инициализируем стейт isOpened равный false. Далее при нажатии на текст, происходит событие onClick в котором меняется значение state.

В файле App.js импортируем наш компонент и добавим его в render

```
import Dropdown from './scripts/Dropdown';

<Dropdown />
```

Пробрасываем данные с помощью props

В папке [scripts] создаем и описываем файл Header.js

```
import React, { Component } from 'react';

class Header extends Component {
    render() {
        return (
            <div>
                {
                    this.props.items.map((item, index) => {
                        return <a href={item.link} key={index}>{item.label}</a>
                    })
                }
            </div>
        );
    }
}
export default Header;
```

В файле App.js импортируем наш компонент, описываем массив данных и передаем этот массив в компонент в качестве параметра. Компонент добавляется в render.

```
import Header from './scripts/Header';

const headerMenu = [
  {
    link: '/news', label: 'News'
  },
  {
    link: '/contacts', label: 'Contacts'
  },
  {
    link: '/about', label: 'About'
  }
];

class App extends Component {
  render() {
    return (
      <div className="App">
        <HelloWorld />
        <Dropdown />
        <Header items={headerMenu}/>
      </div>
    );
  }
}
```

Валидация react props с помощью PropTypes

Добавим проверку получаемых данных в файле Header.js.

Чтобы подключить PropTypes в react версии 15 и ниже нужно:

```
import React, { Component, PropTypes } from 'react';
```

Сначала подключаем PropTypes в react версии 16.

```
import PropTypes from 'prop-types';
```

Далее в нашем классе добавляем проверку типа получаемых данных через props

```
class Header extends Component {
    static propTypes = {
        items: PropTypes.array.isRequired
    }

    render() {
        return (
            <div>
                {
                    this.props.items.map((item, index) => {
                        return <a href={item.link} key={index}>{item.label}</a>
                    })
                }
            </div>
        );
    }
}
```

Примеры валидации типов:

```
items: PropType.array.isRequired                тип данных массив (перем. обязательная)
flag: PropTypes.bool                            тип данных булев (перем. не обязательная)
submit: PropType.func.isRequired                тип функция (перем. обязательная)
title: PropType.string.isRequired               тип строка (перем. обязательная)

type: PropType.oneOf(['news', 'photos'])        тип который равен строке new или photos

user: PropType.shape({                          проверка объекта с полями
      name: PropType.string,
      age: PropType.number
})
users: PropType.arrayOf(                        проверка массива типов
      PropType.shape({
            name: PropType.string,
            age: PropType.number
      })
)
```

Работа с формами

В папке [scripts] создаем и описываем форму файле RegistrationForm.js

```
import React, { Component } from 'react';

class RegistrationForm extends Component {

    constructor(props) {
        super(props);
        this.state = { email: '' };
        this.onChange = this.onChange.bind(this);
        this.onSubmit = this.onSubmit.bind(this);
    }

    onSubmit(event) {           // сработает при нажатии Enter или Button
        event.preventDefault();
        console.log('onSubmit', 'EMAIL:', this.state.email);
    }

    onChange(event){            // стработает при изменении Input
        this.setState({
            email: event.target.value              // меняем значение state
        });
    }

    render() {
        return (
            <form onSubmit={this.onSubmit}>
                <input
                    type="text"
                    placeholder="E-mail"
                    value={this.state.email}
                    onChange={this.onChange}
                />
```

```
                    <button type="submit">Save</button>
                </form>
                );
            }
        }

export default RegistrationForm;
```

В файле App.js импортируем наш компонент и добавим его в render

```
import RegistrationForm from './scripts/RegistrationForm';

class App extends Component {
  render() {
    return (
      <div className="App">
        <RegistrationForm />
      </div>
    );
  }
}
```

CSS компонентный подход

В папке [scripts] создаем и описываем файл стиля RegistrationForm.css

```
.inputForm {
    width: 300px;
    height: 25px;
    margin: 0 auto;
}

.buttonForm {
    width: 50px;
    height: 31px;
}
```

В папке [scripts] отредактируем файл RegistrationForm.js

сначала подключим стиль

```
import './RegistrationForm.css';
```

теперь необходимо добавить к элементам формы атрибут класс

```
render() {
        return (
            <form onSubmit={this.onSubmit}>
                <input
                    type="text"
                    placeholder="E-mail"
                    value={this.state.email}
                    onChange={this.onChange}
                    className="inputForm"
                />
                <button type="submit" className="buttonForm">Save</button>
            </form>
            );
        }
    }
```

Обращение к DOM элементу в React с помощью атрибута ref
(использовать данное свойство нужно с осторожностью)

В папке [scripts] создаем и описываем файл TestRef.js
свойство ref получает объект в котором он объявлен

```
import React, { Component } from 'react';

class TestRef extends Component {
    onClick() {
        console.log('submit', this.testInput, this.testInput.value);
    }

    render() {
        return (
            <div>
                <input
                    type="text"
                    placeholder="test"
                    ref={ (input) => this.testInput = input }
                />
                <button onClick={this.onClick.bind(this)}>Submit</button>
            </div>
        )
    }
}

export default TestRef;
```

В файле App.js импортируем наш компонент и добавим его в render

```
import TestRef from './scripts/TestRef';

class App extends Component {
  render() {
    return (
      <div className="App">
        <TestRef />
      </div>
    );
  }
}
```

Реализация роутинга в React с помощью библиотеки react-router
Установите библиотеку командой: `npm install --save react-router-dom`

В папке [scripts/router] создаем и описываем файл About.js

```
import React, { Component } from 'react';
class About extends Component {
    render(){
        return (
            <div>This is about!</div>
        );
    }
}
export default About;
```

В папке [scripts/router] создаем и описываем файл News.js

```
import React, { Component } from 'react';
class News extends Component {
    render(){
        return (
            <div>This is a news!</div>
        );
    }
}
export default News;
```

В папке [scripts/router] создаем и описываем файл меню Menu.js
Импортируем Link из библиотеки react-router-dom

```
import React, { Component } from 'react';
import { Link } from 'react-router-dom';

class Menu extends Component {
    render(){
        return (
            <div>
                <ul>
                    <li><Link to='/'>Home</Link></li>
                    <li><Link to='/news'>News</Link></li>
                    <li><Link to='/about'>About</Link></li>
                </ul>
            </div>
        );
    }
}

export default Menu;
```

В файле index.js импортируем BrowserRouter, Route. Подключаем компоненты About и News. Далее описываем блок роутеров BrowserRouter в котором три роутера home, news и about.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

import { BrowserRouter, Route } from 'react-router-dom';
import About from './scripts/router/About';
import News from './scripts/router/News';

ReactDOM.render(
    <BrowserRouter>
        <div>
            <Route path="/" component={App} />
            <Route path="/news" component={News} />
            <Route path="/about" component={About} />
        </div>
    </BrowserRouter>,
    document.getElementById('root')
);
registerServiceWorker();
```

В файле App.js импортируем наш компонент и добавим его в render

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

import HelloWorld from './scripts/HelloWorld';
import Dropdown from './scripts/Dropdown';
import Header from './scripts/Header';
import RegistrationForm from './scripts/RegistrationForm';
import TestRef from './scripts/TestRef';
import Menu from './scripts/router/Menu';

const headerMenu = [
  {
    link: '/news',
    label: 'News'
  },
  {
    link: '/contacts',
    label: 'Contacts'
  },
  {
    link: '/about',
    label: 'About'
  }
];

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
        <br />
        <HelloWorld />
        <br />
        <Dropdown />
        <br />
        <Header items={headerMenu}/>
        <br />
        <RegistrationForm />
        <br />
        <TestRef />
        <br />
        <Menu />
      </div>
    );
  }
}

export default App;
```

Теперь на странице можно с помощью меню переключаться между home, news и about.

Функциональные компоненты

В папке [scripts] создаем и описываем файл FuncApp.js который по скоей сути App.js
Вместо класса используется чистая функция.

```
import React from 'react';
import logo from '../logo.svg';
import '../App.css';

const FuncApp = (props) => {
    return (
        <div className="App">
            <header className="App-header">
                <img src={logo} className="App-logo" alt="logo" />
                <h1 className="App-title">Welcome to React</h1>
            </header>
            <p className="App-intro">
                To get started, edit <code>src/App.js</code> and save to reload.
            </p>
        </div>
    );
}

export default FuncApp;
```

В файле index.js импортируем новый компонент и в роутере меняем App на FuncApp

```
import FuncApp from './scripts/FuncApp';

ReactDOM.render(
    <BrowserRouter>
        <div>
            <Route path="/" component={FuncApp} />
            <Route path="/news" component={News} />
            <Route path="/about" component={About} />
        </div>
    </BrowserRouter>,
    document.getElementById('root')
);
registerServiceWorker();
```

В результате все работает как прежде, только теперь мы использовали функцию вместо класса.

Основы Redux

Базовый пример работы с Redux. А так же стэк Redux + Babel + WebPack.

Установка

1. Создайте файл **package.json**

```
npm init -y
```

2. Установите **webpack**

```
npm install -g webpack
npm install --save-dev webpack webpack-cli
```

3. Установите **babel**

```
npm install --save-dev babel-loader babel-core babel-preset-es2015
babel-preset-react babel-preset-env babel-preset-stage-0
```

4. Установите **redux**

```
npm install --save redux
```

Сборка билда

1. Сборка билда

```
npx webpack --config webpack.config.js
```

2. Упрощенная сборка билда (если правило прописано в package.json)

```
"scripts": {
    "build": "npx webpack --config webpack.config.js"
},
```

команда для сборки

```
npm run build
```

Сервер

1. Установите и запуск простого сервера

```
npm install -gbabel-preset-react http-server
```

запуск сервера

```
http-server
```

Структура проекта:

[project]
|---- **[dist]**
    |---- **index.html**
|---- **[src]**
    |---- **index.js**
|---- **package.json**
|---- **webpack.config.js**

Описание файла: **package.json**

```
{
  "name": "basic",
  "version": "1.0.0",
  "description": "",
  "main": "webpack.config.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "npx webpack --config webpack.config.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "babel-core": "^6.26.0",
    "babel-loader": "^7.1.4",
    "babel-preset-env": "^1.6.1",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "babel-preset-stage-0": "^6.24.1",
    "webpack": "^4.6.0",
    "webpack-cli": "^2.0.15"
  },
  "dependencies": {
    "redux": "^4.0.0"
  }
}
```

После установки всех вышеперечисленных пакетов они будут записаны в package.json

---

Описание файла: **webpack.config.js**

```
const path = require('path');

module.exports = {
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'bundle.js',
        sourceMapFilename: 'bundle.map'
    },
    devtool: '#source-map',
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /(node_modules)/,
                loader: 'babel-loader',
            }
        ]
    },
};
```

В конфиге прописан путь сборки всех скриптов и транспиляция кода с помощью Babel.

Описание файла: **index.html** (в папке dist)

```
<!DOCTYPE html>
<html>

<head>
  <meta name="viewport" content="minimum-scale=1.0, width=device-width,
                                 maximum-scale=1.0, user-scalable=no" />
  <meta charset="utf-8">
  <title>React Application</title>
</head>

<body>
  <h2>Hello Redux!</h2>
  <script src="bundle.js"></script>
</body>

</html>
```

Подключаются библиотеки React и Babel. В теле страницы вызывается основной скрипт.

Описание файла: **index.js** (в папке src)

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

// Обработка хранилища
function playList(state = [], action){
    console.log(state, action);

    if(action.type === 'ADD'){
        return [
            ...state,
            action.payload
        ]
    }
    return state;
}

// Создание хранилища
const store = createStore(playList);

// подписываемся на событие изменения хранилища
store.subscribe(()=>{
    console.log('subscribe', store.getState());
});

// Поменяем значение в хранилище
store.dispatch({type: 'ADD', payload: 'track 1'});
store.dispatch({type: 'ADD', payload: 'track 2'});
store.dispatch({type: 'ADD', payload: 'track 3'});
```

Импортируется иблиотека Redux. Далее описывается функция обработки хранилища, создаем само хранилище, подписываемся на событие изменения в хранилище. Добавляем данные в хранилище.

Основные методы.

createStore(*function*) – создание хранищища. В качестве параметра передается функции.

subscribe(*function*) – обработка события изменений в хранилище.

dispatch(*object*) — дабота с данными хранилища (добавление данных).

Соединение Redux и React

Базовый пример работы react и redux

Установка

1. Создайте файл **package.json**

```
npm init -y
```

2. Установите **webpack**

```
npm install -g webpack
npm install --save-dev webpack webpack-cli
```

3. Установите **babel**

```
npm install --save-dev babel-loader babel-core babel-preset-es2015
babel-preset-react babel-preset-env babel-preset-stage-0
```

4. Установите **react**

```
npm install --save react react-dom
```

5. Установите **redux**

```
npm install --save redux
```

6. Установите **react-redux**

```
npm install --save react-redux
```

Сборка билда

1. Сборка билда

```
npx webpack --config webpack.config.js
```

2. Упрощенная сборка билда (если правило прописано в package.json)

```
"scripts": {
    "build": "npx webpack --config webpack.config.js"
},
```

команда для сборки

```
npm run build
```

Сервер

1. Установите и запуск простого сервера

```
npm install -g http-server
```

запуск сервера (чтобы остановить нажмите Ctrl+C)

```
http-server
```

Структура проекта:

[project]
|---- **[dist]**
    |---- **index.html**
|---- **[src]**
    |---- **[components]**
        |---- **mycomponent.js**
    |---- **index.js**
|---- **package.json**
|---- **webpack.config.js**

Описание файла: **package.json**

```
{
  "name": "connect_react_redux",
  "version": "1.0.0",
  "description": "Steck: React + Redux + Webpack + Babel",
  "main": "webpack.config.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "npx webpack --config webpack.config.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "babel-core": "^6.26.3",
    "babel-loader": "^7.1.4",
    "babel-preset-env": "^1.6.1",
    "babel-preset-es2015": "^6.24.1",
    "babel-preset-react": "^6.24.1",
    "babel-preset-stage-0": "^6.24.1",
    "webpack": "^4.6.0",
    "webpack-cli": "^2.0.15"
  },
  "dependencies": {
    "react": "^16.3.2",
    "react-dom": "^16.3.2",
    "react-redux": "^5.0.7",
    "redux": "^4.0.0"
  }
}
```

После установки всех вышеперечисленных пакетов они будут записаны в package.json

---

Описание файла: **webpack.config.js**

```
const path = require('path');
module.exports = {
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'bundle.js',
        sourceMapFilename: 'bundle.map'
    },
```

```
    devtool: '#source-map',
    module: {
        rules: [
            {
                test: /\.js$/,
                exclude: /(node_modules)/,
                loader: 'babel-loader',
                query: {
                    presets: ['env', 'stage-0', 'react']
                }
            }
        ]
    },
};
```

В конфиге прописан путь сборки всех скриптов и транспиляция кода с помощью Babel.

Описание файла: **index.html** (в папке dist)

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="minimum-scale=1.0, width=device-width,
                                 maximum-scale=1.0, user-scalable=no" />
  <title>React & Redux Application</title>
  <script src="https://unpkg.com/react@16/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
</head>

<body>
  <div id="root"></div>
  <script src="bundle.js"></script>
</body>

</html>
```

Подключаются библиотеки React и Babel. В теле страницы вызывается основной скрипт.

Описание файла: **index.js** (в папке src)

```
/**
 * Импортируем библиотеки react и redux
 */

import React from 'react';
import ReactDOM from 'react-dom';
import { createStore } from 'redux';
import { Provider } from 'react-redux';

import MyComponent from './components/mycomponent';


/**
 * Хранилище
 */
```

```
function playList(state = [], action) { // функция редюсер (reducer)
    if(action.type === 'ADD_TRACK'){
        return [
            ...state,
            action.payload
        ];
    }
    return state;
}

const store = createStore(playList);    // создание хранилища

/**
 * Визуализация компонента с помощью React.
 * (в провайдер передаем хранилище)
 */
ReactDOM.render(
    <Provider store={store}>
        <MyComponent title="React" subtitle="Фреймворк от Facebook"/>
    </Provider>,
    document.getElementById('root')
);
```

Импортируется иблиотеки React, Redux и модуль компонента. Описывается обработчик данных хранилища (редюсер). Создается хранилище. Компонент оборачивается в провайдер. Провайдер получает в качестве параметра наше хранилище. Теперь все элементы внутри провайдера имеют доступ к хранилищу через props.

Описание файла: **mycomponent.js** (в папке src/components)

```
import React, { Component } from 'react';
import { connect } from 'react-redux';

class MyComponent extends React.Component {

    constructor(props) {
        super(props);
    }

    onButtonClick() {
        console.log('ADD TRACK', this.trackInput.value);
        this.props.onAddTrack(this.trackInput.value);
        this.trackInput.value = '';
    }

    getInput(input) {
        this.trackInput = input;
    }

    render() {
        console.log(this.props.testStore);
        return (
            <div>
                <h1>{this.props.title}</h1>
                <p>{this.props.subtitle}</p>
                <br />
                <input type="text" ref={this.getInput.bind(this)} />
                <button onClick={this.onButtonClick.bind(this)}>
```

```
                    Add track</button>
                <ul>
                    {this.props.testStore.map((value, index) =>
                        <li key={index}>{value}</li>
                    )
                    }
                </ul>
            </div>
        );
    }
}

export default connect(
    state => ({
        testStore: state
    }),
    dispatch => ({
        onAddTrack: (trackName) => {
            dispatch({ type: 'ADD_TRACK', payload: trackName });
        }
    })
)(MyComponent);
```

Функции **connect()** позволяет установить свзять между React и Redux.
Более подробное описание стрекруты **connect** приводится ниже

```
const mapStateToProps = (state, ownProps) => {
    return {
        testStore: state
    };
}

const mapDispatchToProps = (dispatch) => {
    return {
        onAddTrack: (trackName) => {
            dispatch({ type: 'ADD_TRACK', payload: trackName });
        }
    }
};

export default connect(mapStateToProps, mapDispatchToProps)(MyComponent);
```

Функция **mapStateToProps** – работает со state.
Функция **mapDispatchToProps** — работает с данными хранилища.

Отладка с помощию Redux DevTools

Плагин для Chrome: https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd?hl=ru

```
const store = createStore(playList, window.__REDUX_DEVTOOLS_EXTENSION__ &&
window.__REDUX_DEVTOOLS_EXTENSION__({
    serialize: true
}));
```

Данный плагин работает если приложение запущено через сервер http-server

Метод combineReducers

Данный пример показывает комбинацию редьюсеров.

1. Установка пакета create-react-app
   ```
   $ npm install -g create-react-app
   ```
2. Создание проекта на основе пакета create-react-app
   ```
   $ create-react-app project_name
   ```
3. Установите redux
   ```
   npm install --save redux
   ```
4. Установите react-redux
   ```
   npm install --save react-redux
   ```

1. Запуск сервера
   ```
   $ npm start
   ```
2. Запуск в тестовом режиме
   ```
   npm test
   ```
3. Сборка билда
   ```
   npm run build
   ```
4. Копировать все конфиги и транзитивные зависимости (Webpack, Babel, ESLint и т. д.)
   ```
   npm run eject
   ```

После того как проект будет создан, необходимо создать папку **./src/reducers** в которой будут хранится редюсеры приложения.

В этой папке **./src/reducers** создайте файл **index.js** который служит контейнером редюсеров. Подключается библиотека redux и два редюсера tracks и lists

```
import { combineReducers } from 'redux';
import tracks from './tracks';
import lists from './lists';

export default combineReducers({
    tracks,
    lists
});
```

В папке **./src/reducers** создайте файл **tracks.js** который является редюсером треков

```
const initialState = [
    'Track 1',
    'Track 2'
];

export default function playList(state = initialState, action){
    if(action.type === 'ADD_TRACK'){
        return [
            ...state,
            action.payload
        ]
    }else if(action.type === 'DELETE_TRACK'){
        return state;
    }
    return state;
}
```

В папке **./src/reducers** создайте файл **lists.js** который является редюсером листов

```
const initialState = [
    'List 1',
    'List 2'
];

export default function playList(state = initialState, action){
    if(action.type === 'ADD_LIST'){
        return [
            ...state,
            action.payload
        ]
    }else if(action.type === 'DELETE_LIST'){
        return state;
    }
    return state;
}
```

Оба редюсера добавляют данные в хранилища, каждый в свой массив tracks и lists.

Отредактируем файл **index.js** в папке **./src**

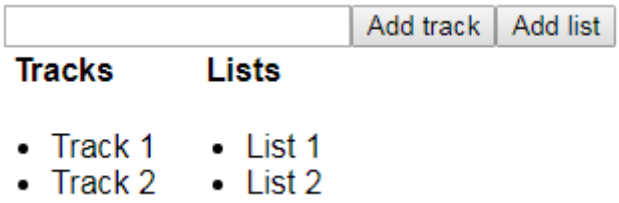
Подключим библиотеку redux, подключим ранее созданные редюсеры, создадим хранилище. В визуализации <App /> обернем в провайдер в который передадим хранилище.

```
/* Подключаем react */
import React from 'react';
import ReactDOM from 'react-dom';

import './index.css';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

/* Подключаем redux */
import { createStore } from 'redux';
import { Provider } from 'react-redux';

/* Подключаем наши редюсеры и создаем хранилище */
import reducer from './reducers';
const store = createStore(reducer);

/* Визуализация компонента через провайдер с передачей в него хранилища */
ReactDOM.render(
    <Provider store={store}>
        <App />
    </Provider>,
    document.getElementById('root')
);

registerServiceWorker();
```

Дальше необходимо создать компонент для отображения и ввода данные в хранилище.

Создаем папку **./src/components** в которой будет описан компонент TrackList

В папке **./src/components** создайте файл **TrackList.js**
Компонент будет отображать текстовое поле с двумя кнопками добавления треков и лостов



Создаем компонент на основе React предварительно импортировав эту библиотеку.
Замет импортируем connect для объединения React с Redux. Вызываем конструктор.
Описываем действия для кнопок (обе кнопки вызывают функции добавления данных в хранилище) Так же мы с помощью ref получаем объект Input и сохраняем его в переменную.

```
import React, { Component } from 'react';
import { connect } from 'react-redux';

class TrackList extends Component {
    constructor(props) {
        super(props);
    }

    onTrackButtonClick(event) {
        this.props.onAddTrack(this.trackInput.value);
        this.trackInput.value = '';
    }
    onListButtonClick(event){
        this.props.onAddList(this.trackInput.value);
        this.trackInput.value = '';
    }

    getInput(input) {
        this.trackInput = input;
    }

    render() {
        console.log(this.props.store);
        return (
            <div>
                <input type="text" ref={this.getInput.bind(this)} />
                <button onClick={this.onTrackButtonClick.bind(this)}>Add track</button>
                <button onClick={this.onListButtonClick.bind(this)}>Add list</button>
                <table>
                    <thead>
                        <tr>
                            <th>Tracks</th>
                            <th>Lists</th>
                        </tr>
                    </thead>
                    <tbody>
                        <tr>
                            <td>
                                <ul>
                                    {
                                        this.props.store.tracks.map((value, index) =>
                                            <li key={index}>{value}</li>
                                        )
                                    }
```

```
                            </ul>
                        </td>
                        <td>
                            <ul>
                                {
                                    this.props.store.lists.map((value, index) =>
                                        <li key={index}>{value}</li>
                                    )
                                }
                            </ul>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    );
  }
}

/* Получаем данные из хранилища */
const mapStateToProps = (state, ownProps) => {
    return {
        store: state
    };
}

/* Обновляем данные в хранилище */
const mapDispatchToProps = (dispatch) => {
    return {
        onAddTrack: (trackName) => {
            dispatch({ type: 'ADD_TRACK', payload: trackName });
        },
        onAddList: (listName) => {
            dispatch({ type: 'ADD_LIST', payload: listName });
        }
    }
};

/* Соединяем компонент с хранилищем */
export default connect(mapStateToProps, mapDispatchToProps)(TrackList);
```

В визуализации отображаются данные находящиеся в хранилище.
В конце скрипта мы экспортируем ранее импортируемую функцию connect в которой передаем для обработчика и сам компонент с непосредственным вызовом этой функции.

mapStateToProps – получает хранилище целиком, то есть все его данные. (это объект состоящий из двух массивов tracks и lists).

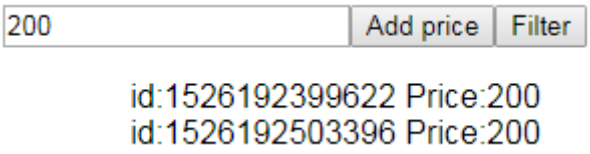
mapDispatchToProps – вызывает метод dispatch который необходим для работы с данными в хранилище (добавление, изменение и удаление). В качестве параметров передается тип и данные.

Отредактируем файл **App.js** в папке **./src**. Импортируем и вызываем компонент.

```
import TrackList from './components/TrackList';
<TrackList />
```

Поиск данных в хранилище.

Необходимо создать компонент который позволит добавлять и фильтровать цены.



В папке **./src/reducers** создайте файл **prices.js** редюсер добавления цен

```
const initialState = [];
export default function pricesList(state = initialState, action){
    if(action.type === 'ADD_PRICE'){
        return [
            ...state,
            action.payload
        ]
    }
    return state;
}
```

В папке **./src/reducers** создайте файл **filterPrices.js** редюсер возвращает строку фильра

```
const initialState = "";
export default function filterPrices(state = initialState, action){
   if(action.type === 'FILTER_PRICE'){
        return action.payload;
    }
    return state;
}
```

В папке **./src/reducers** файл **index.js** который служит контейнером редюсеров, подключаем два редюсера prices и filterPrices

```
import { combineReducers } from 'redux';
import tracks from './tracks';
import lists from './lists';
import prices from './prices';
import filterPrices from './filterPrices';
export default combineReducers({
    tracks,
    lists,
    prices,
    filterPrices
});
```

В папке **./src/components** создайте файл **PriceFilter.js**

Создаем компонент на основе React предварительно импортировав эту библиотеку. Замет импортируем connect для объединения React с Redux. Вызываем конструктор. Описываем действия для кнопок (добавление данных и фильтр данных) Так же мы с помощью ref получаем объект Input и сохраняем его в переменную.

```
import React, { Component } from 'react';
import { connect } from 'react-redux';

class PriceFilter extends Component {
    constructor(props) {
        super(props);
    }
    getInput(input) {
        this.trackInput = input;
    }
    onAddButtonClick(event) {
        this.props.onAddPrice(this.trackInput.value);
        this.trackInput.value = '';
    }
    onFilterButtonClick(event) {
        this.props.onFilterPrice(this.trackInput.value);
    }
    render() {
        return (
            <div>
                <input type="text" ref={this.getInput.bind(this)} />
                <button onClick={this.onAddButtonClick.bind(this)}>Add price</button>
                <button onClick={this.onFilterButtonClick.bind(this)}>Filter</button>
                <ul>
                    {
                        this.props.store.map((value, index) =>
                          <li key={index}>id:{value.id} {value.name}:{value.price}</li>
                        )
                    }
                </ul>
            </div>
        );
    }
}

const mapStateToProps = (state, ownProps) => {
    return {
        store: state.prices.filter(value => value.price.includes(state.filterPrices))
    };
}

const mapDispatchToProps = (dispatch) => {
    return {
        onAddPrice: (price) => {
            const priceObj = {
                id: Date.now().toString(),
                name: 'Price',
                price: price
            }
            dispatch({ type: 'ADD_PRICE', payload: priceObj });
        },
        onFilterPrice: (value) => {
            dispatch({ type: 'FILTER_PRICE', payload: value });
        }
    }
};

export default connect(mapStateToProps, mapDispatchToProps)(PriceFilter);
```

Асинхронные екшены с помощью redux-thunk

Установка пакета redux-thunk

```
$ npm install redux-thunk --save
```

В папке **./src/reducers** файл **index.js** подключаем метод applyMiddleware и библиотеку redux-thunk. При создании хранилища вторым параметром передаем applyMiddleware(thunk)

```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';
const store = createStore(reducer, applyMiddleware(thunk));
```

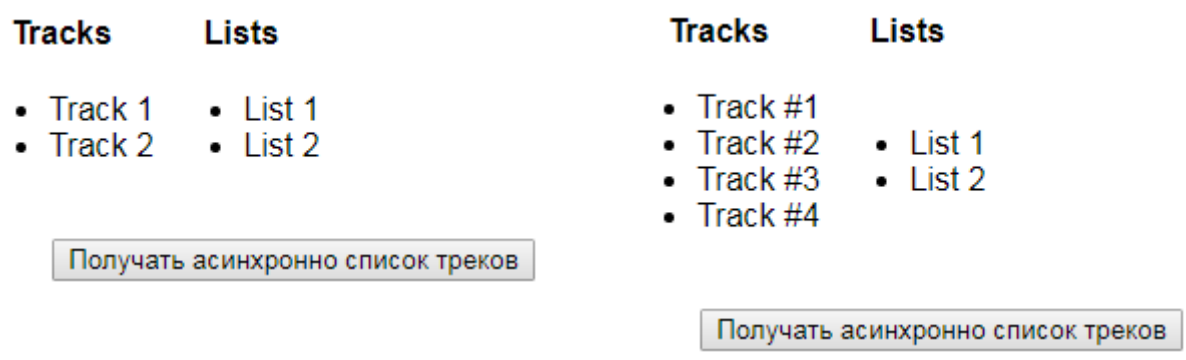
Необходимо создать папку **./src/actions** и в ней файл **track.js** (это action, он выполняет dispatch данных в хранилище по истечению таймера.)

```
var dataTracks = ['Track #1', 'Track #2', 'Track #3', 'Track #4'];

export default function asyncGetTracks (dispatch) {
    return dispatch => {
        setTimeout(() => {
            console.log('I got tracks');
            dispatch({ type: 'GET_TRACKS', payload: dataTracks });
        }, 2000)
    }
}
```

В папке **./src/reducers** изменим файл **tracks.js** добавим обработку GET_TRACKS

```
const initialState = ['Track 1', 'Track 2'];

export default function playList(state = initialState, action){
    if(action.type === 'ADD_TRACK'){
        return [
            ...state,
            action.payload
        ]
    }else if(action.type === 'GET_TRACKS'){
        return action.payload;
    }
    return state;
}
```

В папке **./src/components** изменим файл **TrackList.js** добавим кнопку получания данных



Импортируем ранее созданый action находящийся в папке **./src/actions** файл **track.js**
В разделе визуализации добавляем кнопку с вызовом функции onGetTracks которая описана ниже в функции mapDispatchToProps

```
import React, { Component } from 'react';
import { connect } from 'react-redux';

import asyncGetTracks from '../actions/tracks';

class TrackList extends Component {
    ...
    render() {
        return (
            <div>
                ...
                <br />
                <div>
                    <button onClick={this.props.onGetTracks}>
                          Получать асинхронно список треков
                    </button>
                </div>
            </div>
        );
    }
}

/* Получаем данные из хранилища */
const mapStateToProps = (state, ownProps) => {
    return {
        store: state
    };
}

/* Обновляем данные в хранилище */
const mapDispatchToProps = (dispatch) => {
    return {
        onAddTrack: (trackName) => {
            dispatch({ type: 'ADD_TRACK', payload: trackName });
        },
        onAddList: (listName) => {
            dispatch({ type: 'ADD_LIST', payload: listName });
        },
        onGetTracks: () => {
            dispatch(asyncGetTracks(dispatch));
        }
    }
};

/* Соединяем компонент с хранилищем */
export default connect(mapStateToProps, mapDispatchToProps)(TrackList);
```

В файле **App.js** импортируем наши компонент и добавяем их в render

```
import TrackList from './components/TrackList';
import PriceFilter from './components/PriceFilter';

<TrackList />
<PriceFilter />
```