В этом видео мы с вами реализуем поиск по товарам в нашем магазине.

Для этого давайте добавим компонент Search, который будет полностью самодостаточным и будет фильтровать товары на странице.

Нам нужно создать class, так как у нас будет локальный стейт.

import React, {Component} from 'react'

class Search extends Component {

render () {

return (

<div>Search</div>

)

}

}

export default Search

Давайте добавим его в Sidebar.

import React from 'react'

import BasketCart from 'components/basketCart'

import Search from 'components/search'

const Sidebar = () => {

return (

<div>

<BasketCart />

<Search />

</div>

)

}

export default Sidebar

Теперь давайте напишем разметку для Search. Мы хотим сделать форму с одним инпутом и менять локальный стейт, когда мы меняем input.

import React, {Component} from 'react'

class Search extends Component {

render () {

return (

<div className='well blosd'>

<h3 className='lead'>Quick shop</h3>

<div className='input-group'>

<form onSubmit={this.handleSubmit}>

<input

onChange={this.handleChange}

type='text'

className='form-control'

/>

</form>

<span className='input-group-btn'>

<button className='btn btn-default'>

<span className='glyphicon glyphicon-search'/>

</button>

</span>

</div>

</div>

)

}

}

export default Search

Теперь нам нужно описать методы handleChange и handleSubmit и конструктор.

class Search extends Component {

constructor (props) {

super(props)

this.state = {

value: ''

}

this.handleChange = this.handleChange.bind(this)

this.handleSubmit = this.handleSubmit.bind(this)

}

handleChange (event) {

this.setState({

value: event.target.value

})

}

handleSubmit(event) {

event.preventDefault();

this.props.searchPhone(this.state.value)

}

render () {

...

}

}

export default Search

Если мы посмотрим в браузер, то у нас вывелся нас search, но мы еще не добавили екшен searchPhone, который будет срабатывать при сабмите формы.

Давайте добавим connect и mapDispatchToProps.

import {connect} from 'react-redux'

import {searchPhone} from 'actions'

const mapDispatchToProps = {

searchPhone

}

export default connect(null, mapDispatchToProps)(Search)

Теперь давайте добавим actionType

export const SEARCH\_PHONE = 'SEARCH\_PHONE'

импортируем actionType в actions и добавим наш новый екшен

import {

FETCH\_PHONES\_START,

FETCH\_PHONES\_SUCCESS,

FETCH\_PHONES\_FAILURE,

LOAD\_MORE\_PHONES\_START,

LOAD\_MORE\_PHONES\_SUCCESS,

LOAD\_MORE\_PHONES\_FAILURE,

FETCH\_PHONE\_BY\_ID\_START,

FETCH\_PHONE\_BY\_ID\_SUCCESS,

FETCH\_PHONE\_BY\_ID\_FAILURE,

ADD\_PHONE\_TO\_BASKET,

SEARCH\_PHONE

} from 'actionTypes'

export const searchPhone = (text) => dispatch => {

dispatch({

type: SEARCH\_PHONE,

payload: text

})

}

Этот екшен также синхронный, мы получаем на вход текст, который мы ищем и диспатчим наш екшен.

Так как поиск относится к phonesPage, то давайте обновим редьюсер этой страницы.

Для начала добавим в initialState поле search, которое будет пустое. И обработаем SEARCH\_PHONE actionType.

import R from 'ramda'

import {

FETCH\_PHONES\_SUCCESS,

LOAD\_MORE\_PHONES\_SUCCESS,

SEARCH\_PHONE

} from 'actionTypes'

const initialState = {

ids: [],

search: ''

}

export default (state = initialState, {type, payload}) => {

switch (type) {

case FETCH\_PHONES\_SUCCESS:

return R.merge(state, {

ids: R.pluck('id', payload)

})

case LOAD\_MORE\_PHONES\_SUCCESS:

const ids = R.pluck('id', payload)

return R.merge(state, {

ids: R.concat(ids, state.ids)

})

case SEARCH\_PHONE:

return R.merge(state, {

search: payload

})

default:

return state

}

}

Если мы посмотрим в браузер, то у нас при поиске строки и нажатии ентер отрабатывает наш екшен и обновляется reducer.

Теперь нам осталось только отфильтровать данные, которые выводятся на странице товаров по строке, которую мы искали.

Для этого нам достаточно только обновить селектор getPhones, и все должно заработать.

Так как теперь мы хотим применить еще и filter при выборе телефонов, то нам нужно обернуть нашу конструкцию в compose. Теперь мы можем внутри написать map, который у нас был до этого и добавить функцию applySearch, которая будет просто фильтровать полученные телефоны по имени. Для этого используем метод container, который проверить что то, что мы искали содержится в name.

export const getPhones = state => {

const applySearch = item => R.contains(

state.phonesPage.search,

R.prop('name', item)

)

const phones = R.compose(

R.filter(applySearch),

R.map(id => getPhoneById(state, id))

)(state.phonesPage.ids)

return phones

}

Если мы посмотрим в браузер, то search работает как задумано. Мы ищем Apple и получаем только те телефоны, у которых Apple встречается в названии.

Если у вас что-то не получается, либо возникли вопросы и комментарии, пишите их прямо под этим видео и я обязательно на них отвечу.