В этом уроке мы будет получать данные для страницы товаров.

Для начала давайте добавим разметку страницы в наш layout

import React from 'react'

const Layout = ({children}) => (

<div className='view-container'>

<div className='container'>

<div className='row'>

<div className='col-md-3'>

Sidebar

</div>

<div className='col-md-9'>

{children}

</div>

</div>

</div>

</div>

)

export default Layout

Если мы посмотрим в браузер, то у нас вывелись sidebar и phones.

Для вывода телефонов нам нужно их зафетчить при загрузке компонента phones. Поэтому stateless компонент нам здесь не подходит. Нам нужен класс с методом componentDidMount, где мы будем фетчить все данные.

import React, {Component} from 'react'

class Phones extends Component {

componentDidMount () {

this.props.fetchPhones()

}

render () {

return (

<div>Phones</div>

)

}

}

export default Phones

Для этого мы будем использовать екшен fetchPhones, который нам будет приходить из props.

Для того, чтобы екшен fetchPhones диспатчился, обернем класс Phones в connect.

import React, {Component} from 'react'

import {connect} from 'react-redux'

class Phones extends Component {

componentDidMount () {

this.props.fetchPhones()

}

render () {

return (

<div>Phones</div>

)

}

}

const mapDispatchToProps = {

fetchPhones

}

export default connect(null, mapDispatchToProps)(Phones)

В mapDispatchToProps мы используем сокращенную форму записи, и когда fetchPhones попадает с помощью mapDispatchToProps в connect, то в props прокидывается этот екшен, который оборачивается в dispatch.

Теперь нам нужно описать екшен fetchPhones. Для этого создадим папку actions и в index добавим метод fetchPhones.

export const fetchPhones = () => async dispatch => {

}

Так как у нас подключен в middleware redux-thunk, то мы можем писать асинхронные екшены. И мы будем для этого использовать async await из es7.

При фетче данных стандартной практикой является диспатч трех екшенов: start, success, fail.

Давайте создадим эти actionTypes для нашего метода fetchPhones в файле actionTypes.

export const FETCH\_PHONES\_START = 'FETCH\_PHONES\_START'

export const FETCH\_PHONES\_SUCCESS = 'FETCH\_PHONES\_SUCCESS'

export const FETCH\_PHONES\_FAILURE = 'FETCH\_PHONES\_FAILURE'

То есть при старте фейча телефонов у нас всегда будет диспатчится FETCH\_PHONES\_START, при success FETCH\_PHONES\_SUCCESS, а при ошибках FETCH\_PHONES\_FAILURE.

Хорошей практикой является всегда писать в екшенах одни и те же поля. Поле type, является обязательным, но остальные поля можно делать какими хочешь. И многие разработчики называют их как попало. Я рекомендую писать type, payload (где будут данные) и error (true) если нужен.

Давайте импортируем методы и добавим dispatch start.

import {

FETCH\_PHONES\_START,

FETCH\_PHONES\_SUCCESS,

FETCH\_PHONES\_FAILURE

} from 'actionTypes'

export const fetchPhones = () => async dispatch => {

dispatch({type: FETCH\_PHONES\_START})

}

Больше ничего передавать не нужно, так как мы просто говорим о том, что начался фетч данных.

Теперь мы хотим вызвать реальный фетч данных, но чтобы екшены ничего не знали о том, откуда приходят данные и что мы до этого с ними делаем. Я предпочитаю делать прослойку API, в которой все это будет обрабатываться. Поэтому для получения данных достаточно написать

const phones = await fetchPhonesApi()

Так как при получении данных, мы хотим обработать error, то обернем его в try catch.

try {

const phones = await fetchPhonesApi()

} catch (err) {

dispatch({

type: FETCH\_PHONES\_FAILURE,

payload: err,

error: true

})

}

В payload при ерроре мы записываем нашу ошибку и ставим поле error в true.

Теперь давайте добавим success екшен в код.

export const fetchPhones = () => async dispatch => {

dispatch({type: FETCH\_PHONES\_START})

try {

const phones = await fetchPhonesApi()

dispatch({

type: FETCH\_PHONES\_SUCCESS,

payload: phones

})

} catch (err) {

dispatch({

type: FETCH\_PHONES\_FAILURE,

payload: err,

error: true

})

}

}

Мы выстрелили екшен, что мы закончили успешно получать данные и тогда мы передаем данные, которые мы получили в payload. То есть при success конструкция всегда заключается из двух полей: type и payload.

Нам осталось только написать fetchPhonesApi метод. Для этого давайте создадим папку api, где опишем этот метод в файле index.

Этот метод должен возвращать Promise, так как мы там написали await. И это логично, так как получение данных тут будет потом асинхронное.

На данные момент давайте для простоты добавим файл с моковыми данными. Создадим файл mockPhones в папке api и экспортируем из него массив наших телефонов. Каждый елемент массива имеет id, categoryId и кучу дополнительных полей.

export default [

{

id: "1",

categoryId: "1",

name: "Apple iPhone 5c",

description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultricies lorem odio, at laoreet tellus sodales in. Nullam maximus eros ut tortor ultricies rutrum. Aliquam euismod lacus non est egesta",

price: 823,

image: "/uploads/iphone5c-selection-hero-2013.png",

cpu: "1.3GHz Apple A6",

camera: "8mp (3264x2448)",

size: "124.4mm x 59.2mm x 8.97mm (4.9 x 2.33 x 0.35)",

weight: "132 grams (4.7 ounces) with battery",

display: "4.0 326 pixel density",

battery: "1480 mAh",

memory: "16GB, 32GB and RAM 1 GB"

},

{

id: "2",

categoryId: "1",

name: "Apple iPhone 6",

description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultricies lorem odio, at laoreet tellus sodales in. Nullam maximus eros ut tortor ultricies rutrum. Aliquam euismod lacus non est egesta",

price: 953,

image: "/uploads/51u6y9Rm8QL.\_SY300\_.jpg",

cpu: "1.3GHz Apple A6",

camera: "8mp (3264x2448)",

size: "124.4mm x 59.2mm x 8.97mm (4.9 x 2.33 x 0.35)",

weight: "132 grams (4.7 ounces) with battery",

display: "4.0 326 pixel density",

battery: "1480 mAh",

memory: "16GB, 32GB and RAM 1 GB"

},

{

id: "3",

categoryId: "4",

name: "Lenovo A6000",

description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultricies lorem odio, at laoreet tellus sodales in. Nullam maximus eros ut tortor ultricies rutrum. Aliquam euismod lacus non est egesta",

price: 764,

image: "/uploads/\_35%20(1).JPG",

cpu: "1.3GHz Apple A6",

camera: "8mp (3264x2448)",

size: "124.4mm x 59.2mm x 8.97mm (4.9 x 2.33 x 0.35)",

weight: "132 grams (4.7 ounces) with battery",

display: "4.0 326 pixel density",

battery: "1480 mAh",

memory: "16GB, 32GB and RAM 1 GB"

},

{

id: "4",

categoryId: "5",

name: "Nokia Lumia 1520",

description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultricies lorem odio, at laoreet tellus sodales in. Nullam maximus eros ut tortor ultricies rutrum. Aliquam euismod lacus non est egesta",

price: 674,

image: "/uploads/Lumia1520-Front-Back-png.png",

cpu: "1.3GHz Apple A6",

camera: "8mp (3264x2448)",

size: "124.4mm x 59.2mm x 8.97mm (4.9 x 2.33 x 0.35)",

weight: "132 grams (4.7 ounces) with battery",

display: "4.0 326 pixel density",

battery: "1480 mAh",

memory: "16GB, 32GB and RAM 1 GB"

},

{

id: "5",

categoryId: "3",

name: "HTC One",

description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultricies lorem odio, at laoreet tellus sodales in. Nullam maximus eros ut tortor ultricies rutrum. Aliquam euismod lacus non est egesta",

price: 674,

image: "/uploads/htc-one-m7-802w-dual-sim-silver.jpg",

cpu: "1.3GHz Apple A6",

camera: "8mp (3264x2448)",

size: "124.4mm x 59.2mm x 8.97mm (4.9 x 2.33 x 0.35)",

weight: "132 grams (4.7 ounces) with battery",

display: "4.0 326 pixel density",

battery: "1480 mAh",

memory: "16GB, 32GB and RAM 1 GB"

},

{

id: "6",

categoryId: "2",

name: "Samsung Galaxy S6",

description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultricies lorem odio, at laoreet tellus sodales in. Nullam maximus eros ut tortor ultricies rutrum. Aliquam euismod lacus non est egesta",

price: 674,

image: "/uploads/Agnes\_Case\_for\_Samsung\_Galaxy\_S6\_(1)\_\_92643\_thumb.jpg",

cpu: "1.3GHz Apple A6",

camera: "8mp (3264x2448)",

size: "124.4mm x 59.2mm x 8.97mm (4.9 x 2.33 x 0.35)",

weight: "132 grams (4.7 ounces) with battery",

display: "4.0 326 pixel density",

battery: "1480 mAh",

memory: "16GB, 32GB and RAM 1 GB"

}

]

Теперь давайте импортируем этот массив телефонов и вернем промис, который вернет этот массив.

import phones from './mockPhones'

export const fetchPhones = async () => {

return new Promise(resolve => {

resolve(phones)

})

}

Давайте импортируем fetchPhones в наши екшены, но с названием fetchPhonesApi

import {fetchPhones as fetchPhonesApi} from 'api'

И импортируем наш екшен в компоненте Phones.

import {fetchPhones} from 'actions'

Если мы посмотрим в браузер в redux-devtools, то мы увидим, что при загрузке страницы у нас выстрелило 2 екшена: первый только с полем type, а второй еще и с данными.

Давайте попробуем поменять resolve на reject, чтобы проверить что срабатывает failure, если у нас будет какая-то ошибка.

import phones from './mockPhones'

export const fetchPhones = async () => {

return new Promise((resolve, reject) => {

// resolve(phones)

reject('error')

})

}

Если мы посмотрим в браузер, то мы получили 2 екшена: start и failure. Все работает, поэтому давайте вернем resolve.

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