В этом уроке мы с вами добавим кнопку "Показать больше товаров".

Внизу страницы товаров у нас будет кнопка load more, и при нажатии на нее будет выстреливать екшен и посылаться запрос, чтобы получить больше товаров и они будут добавляться в редьюсер и на страницу.

Давайте для начала добавим разметку в нашу страницу телефонов

render () {

const {phones, loadMorePhones} = this.props

return (

<div>

<div className='books row'>

{phones.map((phone, index) => this.renderPhone(phone, index))}

</div>

<div className='row'>

<div className='col-md-12'>

<button

onClick={loadMorePhones}

className='pull-right btn btn-primary'

>

Load More

</button>

</div>

</div>

</div>

)

}

Мы с вами добавили кнопке метод onClick, по которому должен вызываться екшен, который загрузит еще товаров и деструктурировали loadMorePhones из this.props.

Давайте добавим этот метод в mapDispatchToProps

const mapDispatchToProps = {

fetchPhones,

loadMorePhones

}

и импортируем из екшенов

import {fetchPhones, loadMorePhones} from 'actions'

Теперь давайте опишем его. Мы можем полностью скопипастить fetchPhones метод и заменить типы

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

dispatch({type: LOAD\_MORE\_PHONES\_START})

try {

const phones = await fetchPhonesApi()

dispatch({

type: LOAD\_MORE\_PHONES\_SUCCESS,

payload: phones

})

} catch (err) {

dispatch({

type: LOAD\_MORE\_PHONES\_FAILURE,

payload: err,

error: true

})

}

}

И импортируем типы

import {

FETCH\_PHONES\_START,

FETCH\_PHONES\_SUCCESS,

FETCH\_PHONES\_FAILURE,

LOAD\_MORE\_PHONES\_START,

LOAD\_MORE\_PHONES\_SUCCESS,

LOAD\_MORE\_PHONES\_FAILURE

} from 'actionTypes'

И добавим их в actionTypes

export const LOAD\_MORE\_PHONES\_START = 'LOAD\_MORE\_PHONES\_START'

export const LOAD\_MORE\_PHONES\_SUCCESS = 'LOAD\_MORE\_PHONES\_SUCCESS'

export const LOAD\_MORE\_PHONES\_FAILURE = 'LOAD\_MORE\_PHONES\_FAILURE'

Если мы посмотрим в браузер, то увидим, что у нас выстреливают эти три екшена. Но в реальном приложении мы бы хотели передавать в API offset, чтобы каждый раз загружать новую пачку записей, а не одни и те же записи.

Так как в store у нас хранится список idшников телефонов на странице, то мы можем прямо в екшене получить количество товаров, которые у нас уже выведены на странице и передать в API.

Давайте в selectors добавим новый метод, который будет возвращать количество отрендереных товаров.

export const getRenderedPhonesLength = state => R.length(state.phonesPage.ids)

И теперь мы можем вторым аргументом в екшене получить функцию getState, которая при вызове вернет нам текущий стейт.

export const loadMorePhones = () => async (dispatch, getState) => {

const offset = getRenderedPhonesLength(getState())

dispatch({type: LOAD\_MORE\_PHONES\_START})

try {

const phones = await loadMorePhonesApi({offset})

dispatch({

type: LOAD\_MORE\_PHONES\_SUCCESS,

payload: phones

})

} catch (err) {

dispatch({

type: LOAD\_MORE\_PHONES\_FAILURE,

payload: err,

error: true

})

}

}

Также я переименовал метод из API и теперь мы вызываем loadMorePhonesApi, куда передаем offset. То есть к какого елемента нам нужно возвращать данные.

Теперь нужно импортировать эти 2 метода

import {getRenderedPhonesLength} from 'selectors'

import {

fetchPhones as fetchPhonesApi,

loadMorePhones as loadMorePhonesApi

} from 'api'

Теперь давайте опишем метод в API

import phones from './mockPhones'

export const fetchPhones = async () => {

return new Promise(resolve => {

resolve(phones)

})

}

export const loadMorePhones = async ({offset}) => {

return new Promise(resolve => {

resolve(phones)

})

}

И хоть сейчас мы не используем этот offset, так как у нас всего 6 товаров, но при реальном API мы бы в url передали бы offset и получили бы новые данные.

Теперь давайте добавим success action в наши редьюсеры.

Сначала в phones, где он будет делать такую же логику, что и fetchPhones.

import R from 'ramda'

import {

FETCH\_PHONES\_SUCCESS,

LOAD\_MORE\_PHONES\_SUCCESS

} from 'actionTypes'

const initialState = {}

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

switch (type) {

case FETCH\_PHONES\_SUCCESS:

const newValues = R.indexBy(R.prop('id'), payload)

return R.merge(state, newValues)

case LOAD\_MORE\_PHONES\_SUCCESS:

const moreValues = R.indexBy(R.prop('id'), payload)

return R.merge(state, moreValues)

default:

return state

}

}

И в phonesPage. Но в phonesPage мы хотим не перетереть все idшники, а склеить 2 массива. Поэтому используем метод Ramda concat.

import R from 'ramda'

import {

FETCH\_PHONES\_SUCCESS,

LOAD\_MORE\_PHONES\_SUCCESS

} from 'actionTypes'

const initialState = {

ids: []

}

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)

})

default:

return state

}

}

Если мы посмотрим в браузер, то у нас работает кнопка loadMore и загружает нам больше телефонов. В devtools мы видим, что в редьюсере phones количество телефонов не увеличивается, так как они уникальные и просто перетирают предыдущие, а в phonesPage просто добавляются новые idшники. Компонент нам вообще не пришлось менять, так как он просто читает данные.

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