**Введение**

Классический подход к формам в React заключается в создание нескольких свойств и валидации во время submit.

Это удобно, когда полей мало, но если форма содержит 3 и более полей, то начнется путаница, особенно в валидации.

Однако если в проекте используется Redux, то можно облегчить работу с формами специальном библиотекой redux-form.

Библиотека redux-form работает с формами так:

1. Необходимо создать компонент специально для формы
2. В этом компоненте поля нужно создавать через специальный компонент Field
3. Этим полям нужно давать name, которые потом будут использоваться, как ключ в объекте, который получит обработчик формы
4. Компонент формы нужно обвернуть в HOC reduxForm, передав объект с уникальным именем по ключу form.
5. Использовать компонент форму можно в любой другом компоненте, но при создании необходимо передать функцию обработчик submit, который получит объект из 3 пункта.

**Подключение**

Прежде всего, надо скачать библиотеку, сделать это можно через npm:

npm i redux-form

Теперь необходимо добавить reducer из этого и подключить в store

import { createStore, combineReducers } from 'redux'

import { reducer } from 'redux-form'

const store = createStore(

combineReducers({

form: reducer

})

)

Здесь нужно подметить, что для работы redux-form сам reducer должен подключен, как form.

**Создание основы**

Сначала нужно создать отдельный компонент формы, допустим, FormLogin:

import React from 'react'

import { Field, reduxForm } from 'redux-form'

function FormLogin(props) {

const { handleSubmit } = props

return (

<form onSubmit={handleSubmit}>

<button type='submit'>Submit</button>

</form>

)

}

export default reduxForm({ form: 'login' })(FormLogin)

В этом файле есть функция-компонент, которую при экспорте обворачивают reduxForm.

Это необходимо, чтоб в пропсах можно было получить все необходимые функции и свойства. Сначала нам нужен только обработчик формы – handleSubmit. В самой форме есть только кнопка submit.

Теперь этой формой можно пользоваться так:

import FormLogin from './Form/FormLogin'

function App() {

function submit(obj) {

console.log('form', obj)

}

return <div>

<h1>Redux Form</h1>

<FormLogin onSubmit={submit}/>

</div>

}

Как говорилось ранее при использовании нужно передать обработчик формы, т.е. функция submit из App – это функция props.handleSubmit.

**Создание полей**

Для полей, как говорилось ранее есть отдельный компонент Field. Ему необходимо передать два пропса, ранее упомянутый пропс name и component:

<Field

name='name'

component='input'

type='text'

placeholder='name'

/>

Все остальные пропсы будут переданы в новый элемент input. Теперь, если нажать на submit, то можно будет увидеть в консоли значение этого поля.

Хорошей практикой является создание многоразовых компонентов – функций, которые возвращают значение для пропса component:

function textInput(props) {

return <input

className='Input'

type='text'

{...props.input}

placeholder={props.input.name}

/>

}

<Field

name='name'

component={textInput}

/>

<Field

name='lastname'

component={textInput}

/>

Преимущество видно сразу, т.к. стал короче.

**Начальное значение форм**

Устанавливать изначальное значение формы можно при помощи пропса initialValues, которому необходимо передать объект с начальным значением:

const defaulValues = {

name: 'Вася',

lastname: 'Пупкин'

}

<FormLogin

onSubmit = { submit }

initialValues = { defaulValues }

/>

Кроме того пользователю нужно дать возможность самому возвращаться к начальному значению. Это можно сделать с помощью функции props.reset

<button onClick={props.reset}>Reset</button>

**Валидация одной функцией**

В redux-form есть несколько способов валидировать формы.

Самый распространенный способ - это создать отдельную функцию validate и зарегистрировать:

function validate(inputs) {

const err = {}

const { name, lastname } = inputs

if (!name)

err.name = 'Имя пустое'

else if (name.length < 2 || name.length > 20)

err.name = 'Имя слишком короткое/длинное'

if (!lastname)

err.lastname = 'Фамилия пуста'

else if (lastname.length < 2 || lastname.length > 20)

err.lastname = 'Фамилия слишком короткая/длинная'

return err

}

export default reduxForm({

form: 'login',

validate

})(FormLogin)

Сам принцип валидации в такой функции простой. Нужно вернуть объект с ошибкой, обязательно указав на место. Теперь если эта функция вернет не пустой объект, то сделать submit формы не получится.

Но пользователь не увидит сообщение об ошибке, т.к. они нигде не выводятся. Для этого нужно изменить функцию textInput:

function textInput(props) {

const { meta } = props

return <>

<input

className='Input'

type='text'

{...props.input}

placeholder={props.input.name}

/>

{meta.error && meta.touched &&

<div>

{meta.error}

</div>

}

</>

}

Сама ошибка хранится в пропсе meta.

**Валидация Field-Level Validation**

Другой вид валидации – это валидация на уровне поля. В отличии от способа выше он заключается в том, что есть множество функций валидации, которые можно комбинировать.

function requiredInput(input) {

return input

? undefined

: 'Значение пустое'

}

<Field

name='name'

component={textInput}

validate={[requiredInput]}

/>

<Field

name='lastname'

component={textInput}

validate={[requiredInput]}

/>

**Использование других полей**

Во время валидации нужно получить значения других полей. В первом способе это не трудно. Если же валидация происходит по 2 варианту, то если нужно получить все поля всегда 2 параметр с ними.

function matchPassword(input, allInputs) {

return input === allInputs.password

? undefined

: 'Пароль не совпадает'

}

<Field

name='password'

component={passwordInput}

validate={[requiredInput, matchPassword]}

/>

<Field

name='confirm-password'

component={passwordInput}

validate={[matchPassword]}

/>

**Submit validate**

Последнее что надо упомянуть – это валидация в момент отправки формы. Это можно сделать ошибку SubmissionError:

const names = [

'admin', 'Alex', 'Mark'

]

function submit(obj) {

if (names.includes(obj.name))

throw new SubmissionError({

name: 'Имя'

})

console.log('form', obj)

}