**Redux Essentials, Part 4: Using Redux Data**

What You'll Learn

* Using Redux data in multiple React components
* Organizing logic that dispatches actions
* Writing more complex update logic in reducers

## Showing Single Posts

Крч у нас посты показываются в урезанном (чего? Не было такого) виде, когда текст длинный. Логично, что юзер захочет посмотреть весь. И вместо кнопки показать больше, мы открываем в новом пути этот пост. Ну лан, мы видели и такое. Это рил жизнь

### Creating a Single Post Page

Сделали компонент, который получает, как я пон, id нужного поста, потом ищет в сторе в разделе постов нужный пост по id, ну и показывает его внутренности, если находит, а если не находит, пишет, что не нашел.

import React from 'react'

import { useSelector } from 'react-redux'

export const SinglePostPage = ({ match }) => {

const { postId } = match.params

const post = useSelector(state =>

state.posts.find(post => post.id === postId)

)

if (!post) {

return (

<section>

<h2>Post not found!</h2>

</section>

)

}

return (

<section>

<article className="post">

<h2>{post.title}</h2>

<p className="post-content">{post.content}</p>

</article>

</section>

)

}

### Adding the Single Post Route

Обновили роутер

import { PostsList } from './features/posts/PostsList'

import { AddPostForm } from './features/posts/AddPostForm'

import { SinglePostPage } from './features/posts/SinglePostPage'

function App() {

return (

<Router>

<Navbar />

<div className="App">

<Switch>

<Route

exact

path="/"

render={() => (

<React.Fragment>

<AddPostForm />

<PostsList />

</React.Fragment>

)}

/>

<Route exact path="/posts/:postId" component={SinglePostPage} /> эта штука подставит в путь id поста

<Redirect to="/" />

</Switch>

</div>

</Router>

)

}

У нас почему-то нет компонента под оттедбный пост. Поэтому в списке постов, отдельно взятому добавили логику перехода на страницу поста

import React from 'react'

import { useSelector } from 'react-redux'

import { Link } from 'react-router-dom'

export const PostsList = () => {

const posts = useSelector(state => state.posts)

const renderedPosts = posts.map(post => (

<article className="post-excerpt" key={post.id}>

<h3>{post.title}</h3>

<p className="post-content">{post.content.substring(0, 100)}</p> а вот оно урезание

<Link to={`/posts/${post.id}`} className="button muted-button">

View Post

</Link>

</article>

))

return (

<section className="posts-list">

<h2>Posts</h2>

{renderedPosts}

</section>

)

}

Ну и в меню ссылочку добавили на посты в общем, чтобы назщад можно было возвращаться

import React from 'react'

import { Link } from 'react-router-dom'

export const Navbar = () => {

return (

<nav>

<section>

<h1>Redux Essentials Example</h1>

<div className="navContent">

<div className="navLinks">

<Link to="/">Posts</Link>

</div>

</div>

</section>

</nav>

)

}

## Editing Posts

Возможность редактировать посты.

Let's add a new <EditPostForm> component that has the ability to take an existing post ID, read that post from the store, lets the user edit the title and post content, and then save the changes to update the post in the store.

### Updating Post Entries

Все, что касается логики постов, лежит в куске про посты. Его и будем менять. Это же новый экшн, согласись? Пост отредачили типо

Что надо знать?

Id поста, который хотят отредачить (значит, надо найти пост по id)

Новые значения для заголовка и содержимого (значит, форму надо показать)

const postsSlice = createSlice({

name: 'posts',

initialState,

reducers: {

postAdded(state, action) {

state.push(action.payload)

},

postUpdated(state, action) { - новый планктончик-редьюсер

const { id, title, content } = action.payload – забрали новую инфу, уже отредаченнную

const existingPost = state.find(post => post.id === id) – нашли пост по id

if (existingPost) { если пост есть такой

existingPost.title = title то изменим его содержимое на новое, которое из формочки взяли

existingPost.content = content

}

}

}

})

export const { postAdded, postUpdated } = postsSlice.actions - не забываем экспортировать новичков

export default postsSlice.reducer

### Creating an Edit Post Form

Оч просто – новый компонент-форма. Т ынажал отредачить пост. После жтого ищется пост с нужным id, из него выдергивается контент и засовывается в форму. Все, можешь редачить. Диспатч ток когда уже форму сохранет юзер, а так все в локальном состоянии лежит.

import React, { useState } from 'react'

import { useDispatch, useSelector } from 'react-redux'

import { useHistory } from 'react-router-dom'

import { postUpdated } from './postsSlice'

export const EditPostForm = ({ match }) => {

const { postId } = match.params

const post = useSelector(state => -----поиск поста по id

state.posts.find(post => post.id === postId)

)

const [title, setTitle] = useState(post.title) --- запихнули контент из существующего поста в форму

const [content, setContent] = useState(post.content)

const dispatch = useDispatch()

const history = useHistory() - это что??????

const onTitleChanged = e => setTitle(e.target.value)

const onContentChanged = e => setContent(e.target.value)

const onSavePostClicked = () => {

if (title && content) {

dispatch(postUpdated({ id: postId, title, content }))

history.push(`/posts/${postId}`) даже слова блин про это не сказали!

}

}

return (

<section>

<h2>Edit Post</h2>

<form>

<label htmlFor="postTitle">Post Title:</label>

<input

type="text"

id="postTitle"

name="postTitle"

placeholder="What's on your mind?"

value={title}

onChange={onTitleChanged}

/>

<label htmlFor="postContent">Content:</label>

<textarea

id="postContent"

name="postContent"

value={content}

onChange={onContentChanged}

/>

</form>

<button type="button" onClick={onSavePostClicked}>

Save Post

</button>

</section>

)

}

Сделали роутер для формы

import { PostsList } from './features/posts/PostsList'

import { AddPostForm } from './features/posts/AddPostForm'

import { SinglePostPage } from './features/posts/SinglePostPage'

import { EditPostForm } from './features/posts/EditPostForm'

function App() {

return (

<Router>

<Navbar />

<div className="App">

<Switch>

<Route

exact

path="/"

render={() => (

<React.Fragment>

<AddPostForm />

<PostsList />

</React.Fragment>

)}

/>

<Route exact path="/posts/:postId" component={SinglePostPage} />

<Route exact path="/editPost/:postId" component={EditPostForm} />

<Redirect to="/" />

</Switch>

</div>

</Router>

)

}

Ну и добавили в отдельный пост (который 1 на стр показывается) кнопчку для редактирования

import { Link } from 'react-router-dom'

export const SinglePostPage = ({ match }) => {

// omit other contents

<p className="post-content">{post.content}</p>

<Link to={`/editPost/${post.id}`} className="button"> т.е. редактирование происходит на отдельной стр вщ

Edit Post

</Link>

### Preparing Action Payloads

Как настроить, что будет внутри экшена?

Вот так если вручную, в простом редаксе

// hand-written action creator

function postAdded(title, content) {

const id = nanoid()

return {

type: 'posts/postAdded',

payload: { id, title, content }

}

}

А в RTK можно в куске прям указать колбэк, который подготоит экшн в таком виде, как ты хочешь

const postsSlice = createSlice({

name: 'posts',

initialState,

reducers: {

postAdded: {

reducer(state, action) {

state.push(action.payload)

},

prepare(title, content) {

return {

payload: {

id: nanoid(),

title,

content

}

}

}

}

// other reducers here

}

})

Now our component doesn't have to worry about what the payload object looks like - the action creator will take care of putting it together the right way. So, we can update the component so that it passes in title and content as arguments when it dispatches postAdded:

features/posts/AddPostForm.js

const onSavePostClicked = () => {  
 if (title && content) {  
 dispatch(postAdded(title, content))  
 setTitle('')  
 setContent('')  
 }  
}

а было вот как? Наверное, в бъект все завертывали

## Users and Posts

В соц сети еще и юзеры есть, не ток посты.

### Adding a Users Slice

Сделали кусок юзеров

import { createSlice } from '@reduxjs/toolkit'

const initialState = [

{ id: '0', name: 'Tianna Jenkins' },

{ id: '1', name: 'Kevin Grant' },

{ id: '2', name: 'Madison Price' }

]

const usersSlice = createSlice({

name: 'users',

initialState,

reducers: {}

})

export default usersSlice.reducer

Добавили руководителя отделения юзеров – импортированный редьюсер из куска юзеров, в главный редьюсер, в большого босса при создании стора

import { configureStore } from '@reduxjs/toolkit'

import postsReducer from '../features/posts/postsSlice'

import usersReducer from '../features/users/usersSlice'

export default configureStore({

reducer: {

posts: postsReducer,

users: usersReducer

}

})

### Adding Authors for Posts

У постов вообще-то есть авторы. Как раз-таки эти юзеры. Но тут решили упростить и сделать некий список юзеров, из которого надо выбирать,Ю кто автор при добавлении поста.

Первое, что надо сделать, это изменить логику добавления постов. Потому что там пока про автора вщ никакой инфы

const postsSlice = createSlice({

name: 'posts',

initialState,

reducers: {

postAdded: {

reducer(state, action) {

state.push(action.payload)

},

prepare(title, content, userId) { --подготовительный колбек сделали

return {

payload: {

id: nanoid(),

title,

content,

user: userId - юзера добавили в экшн

}

}

}

}

// other reducers

}

})

В компоненте формы, с помощью которой делается добавление поста, внесли изменения. Добавили этот select с выпадющим списком юзеров

import React, { useState } from 'react'

import { useDispatch, useSelector } from 'react-redux'

import { postAdded } from './postsSlice'

export const AddPostForm = () => {

const [title, setTitle] = useState('')

const [content, setContent] = useState('')

const [userId, setUserId] = useState('')

const dispatch = useDispatch()

const users = useSelector(state => state.users)

const onTitleChanged = e => setTitle(e.target.value)

const onContentChanged = e => setContent(e.target.value)

const onAuthorChanged = e => setUserId(e.target.value) – новый стейт для силекта

const onSavePostClicked = () => {

if (title && content) {

dispatch(postAdded(title, content, userId)) – у нас же есть теперь prepare, заготовщик так сказать

setTitle('')

setContent('')

}

}

const canSave = Boolean(title) && Boolean(content) && Boolean(userId) – добавочная проверочка

const usersOptions = users.map(user => ( - вот он списочек для силекта

<option key={user.id} value={user.id}>

{user.name}

</option>

))

return (

<section>

<h2>Add a New Post</h2>

<form>

<label htmlFor="postTitle">Post Title:</label>

<input

type="text"

id="postTitle"

name="postTitle"

placeholder="What's on your mind?"

value={title}

onChange={onTitleChanged}

/>

<label htmlFor="postAuthor">Author:</label> --- тут сам силект

<select id="postAuthor" value={userId} onChange={onAuthorChanged}>

<option value=""></option>

{usersOptions}

</select>

<label htmlFor="postContent">Content:</label>

<textarea

id="postContent"

name="postContent"

value={content}

onChange={onContentChanged}

/>

<button type="button" onClick={onSavePostClicked} disabled={!canSave}>

Save Post

</button>

</form>

</section>

)

}

У поста должна быть какая-то строяка, где отобразится инфа об авторе. Создали отдельный компонент для этой строчки. А для отдельного поста в списке постов так и не создали. Гении

import React from 'react'

import { useSelector } from 'react-redux'

export const PostAuthor = ({ userId }) => {

const author = useSelector(state =>

state.users.find(user => user.id === userId)

)

return <span>by {author ? author.name : 'Unknown author'}</span>

}

## More Post Features

### Storing Dates for Posts

У постов еще пишут, когда их выложили. + показывают сначала свеженькие, т.е. в обратном порядке

Но как нам с датой работать? В Redux могут быть ток простые объекты. Называется, выстрелили себе в ногу

Но это решается – просто в строку сразу преобразуй и все

features/posts/postsSlice.js

postAdded: {

reducer(state, action) {

state.push(action.payload)

},

prepare(title, content, userId) {

return {

payload: {

id: nanoid(),

date: new Date().toISOString(),

title,

content,

user: userId,

},

}

},

},

Ты только глянь, сколько у нас уже надстроек!

Сделали отдельный компонент для инфы о дате. Туда бахнули библиотеку для работы с датой. И вообще не пояснили, как она работает, молодцы

features/posts/TimeAgo.js

import React from 'react'

import { parseISO, formatDistanceToNow } from 'date-fns'

export const TimeAgo = ({ timestamp }) => {

let timeAgo = ''

if (timestamp) {

const date = parseISO(timestamp)

const timePeriod = formatDistanceToNow(date)

timeAgo = `${timePeriod} ago`

}

return (

<span title={timestamp}>

&nbsp; <i>{timeAgo}</i>

</span>

)

}

### Sorting the Posts List

Ревертнем массивчик с постами?

features/posts/PostsList.js

// Sort posts in reverse chronological order by datetime string

const orderedPosts = posts.slice().sort((a, b) => b.date.localeCompare(a.date))

const renderedPosts = orderedPosts.map(post => {

return (

<article className="post-excerpt" key={post.id}>

<h3>{post.title}</h3>

<div>

<PostAuthor userId={post.user} />

<TimeAgo timestamp={post.date} />

</div>

<p className="post-content">{post.content.substring(0, 100)}</p>

<Link to={`/posts/${post.id}`} className="button muted-button">

View Post

</Link>

</article>

)

})

Здесь я мало что пон

features/posts/postsSlice.js

import { createSlice, nanoid } from '@reduxjs/toolkit'

import { sub } from 'date-fns'

const initialState = [

{

// omitted fields

content: 'Hello!',

date: sub(new Date(), { minutes: 10 }).toISOString()

},

{

// omitted fields

content: 'More text',

date: sub(new Date(), { minutes: 5 }).toISOString()

}

]

### Post Reaction Buttons

А на посты можно ставить реакции в соцсетях. И тут решили добавить тоже

Создали отдельный компонент

features/posts/ReactionButtons.js

Тут по сути просто отрисовали

import React from 'react'

const reactionEmoji = {

thumbsUp: '👍',

hooray: '🎉',

heart: '❤️',

rocket: '🚀',

eyes: '👀'

}

export const ReactionButtons = ({ post }) => {

const reactionButtons = Object.entries(reactionEmoji).map(([name, emoji]) => {

return (

<button key={name} type="button" className="muted-button reaction-button">

{emoji} {post.reactions[name]}

</button>

)

})

return <div>{reactionButtons}</div>

}

Реакции у нас на что? На почты. Что отвечает за посты? Кусок постов. Надо что сделать, если новые действия появляются? Новый планктончик-редьюсер

const postsSlice = createSlice({

name: 'posts',

initialState,

reducers: {

reactionAdded(state, action) {

const { postId, reaction } = action.payload --- пойми, что за реакция, пойми что за пост

const existingPost = state.find(post => post.id === postId) найди пост

if (existingPost) {

existingPost.reactions[reaction]++ -- измени именно его реакции

}

}

// other reducers

}

})

export const { postAdded, postUpdated, reactionAdded } = postsSlice.actions

Кусок изменили, пора и компонент реакций поменять

import React from 'react'

import { useDispatch } from 'react-redux'

import { reactionAdded } from './postsSlice'

const reactionEmoji = {

thumbsUp: '👍',

hooray: '🎉',

heart: '❤️',

rocket: '🚀',

eyes: '👀'

}

export const ReactionButtons = ({ post }) => {

const dispatch = useDispatch()

const reactionButtons = Object.entries(reactionEmoji).map(([name, emoji]) => {

return (

<button

key={name}

type="button"

className="muted-button reaction-button"

onClick={() =>

dispatch(reactionAdded({ postId: post.id, reaction: name }))

}

>

{emoji} {post.reactions[name]}

</button>

)

})

return <div>{reactionButtons}</div>

}

Summary

* **Any React component can use data from the Redux store as needed**
  + Any component can read any data that is in the Redux store
  + Multiple components can read the same data, even at the same time
  + Components should extract the smallest amount of data they need to render themselves
  + Components can combine values from props, state, and the Redux store to determine what UI they need to render. They can read multiple pieces of data from the store, and reshape the data as needed for display.
  + Any component can dispatch actions to cause state updates
* **Redux action creators can prepare action objects with the right contents**
  + createSlice and createAction can accept a "prepare callback" that returns the action payload
  + Unique IDs and other random values should be put in the action, not calculated in the reducer
* **Reducers should contain the actual state update logic**
  + Reducers can contain whatever logic is needed to calculate the next state
  + Action objects should contain just enough info to describe what happened