**Redux Essentials, Part 3: Basic Redux Data Flow**

Опять. Название о теории. Объяснение, что будет в главе – практика. Мб хоть слово про теорию скажете? Потому что у них опять непонятный фокус.

### Project Setup

Они подготовили заготовку проекта соц сети. Там есть стили, фейковое апи.

#### Creating a New Redux + React Project

М, как мило, фиговая документация рекомендует фиговый vite. Ну мб и не фиговый, но у меня он не работает на пк, как надо.

#### Exploring the Initial Project

Говорят, что где лежит

## Main Posts Feed

В соцсетях есть посты. Первая цель – вывести посты на экран.

### Creating the Posts Slice

Посты – смысловой кусок. Создадим слайс.

Создание отдельного файла

nside of src, create a new features folder, put a posts folder inside of features, and add a new file named postsSlice.js.

Сделали фейковый массив постов и сделали его начальным состоянием

import { createSlice } from '@reduxjs/toolkit'

const initialState = [

{ id: '1', title: 'First Post!', content: 'Hello!' },

{ id: '2', title: 'Second Post', content: 'More text' }

]

const postsSlice = createSlice({

name: 'posts',

initialState,

reducers: {}

})

export default postsSlice.reducer

А теперь внезапной, в нашей истории про поток данных, мы перенеслись в стор, который уже был пос\чтиготов. Запихнули руководителя отделов (редьюсор в главный редьюсер засунули)

import { configureStore } from '@reduxjs/toolkit'

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

export default configureStore({

reducer: {

posts: postsReducer

}

})

### Showing the Posts List

Мы создали компонент, где будет отображаться список постов

import React from 'react'

import { useSelector } from 'react-redux' – не забудь хук

export const PostsList = () => {

const posts = useSelector(state => state.posts) – вытащили посты

промэпились

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>

</article>

))

return (

<section className="posts-list">

<h2>Posts</h2>

{renderedPosts}

</section>

)

}

Запихнули компоненты в App

import React from 'react'

import {

BrowserRouter as Router,

Switch,

Route,

Redirect

} from 'react-router-dom'

import { Navbar } from './app/Navbar'

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

function App() {

return (

<Router>

<Navbar />

<div className="App">

<Switch>

<Route

exact

path="/"

render={() => (

<React.Fragment>

<PostsList />

</React.Fragment>

)}

/>

<Redirect to="/" />

</Switch>

</div>

</Router>

)

}

export default App

фрагмент, потому что на главной стр будет еще что-то

### Adding New Posts

#### Adding the New Post Form

Сделали форму для заполнения контента поста

Create AddPostForm.js in our posts folder

import React, { useState } from 'react'

export const AddPostForm = () => {

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

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

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

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

return (

<section>

<h2>Add a New Post</h2>

<form>

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

<input

type="text"

id="postTitle"

name="postTitle"

value={title}

onChange={onTitleChanged}

/>

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

<textarea

id="postContent"

name="postContent"

value={content}

onChange={onContentChanged}

/>

<button type="button">Save Post</button>

</form>

</section>

)

}

Ну и добавили компонент в эп

<Route

exact

path="/"

render={() => (

<React.Fragment>

<AddPostForm />

<PostsList />

</React.Fragment>

)}

/>

#### Saving Post Entries

Уже интереснее

Добавили плкнтончика-редьюсера в кусок о постах

const postsSlice = createSlice({

name: 'posts',

initialState,

reducers: {

postAdded(state, action) { - вот он красавец

state.push(action.payload)

}

}

})

export const { postAdded } = postsSlice.actions – импортнули экшены

export default postsSlice.reducer

#### Dispatching the "Post Added" Action

Теперь экшн надо диспатчить, когда форма отправляется

А еще постам нужен уникальный id, для этого библиотеку подтащили – nanoid

import React, { useState } from 'react'

import { useDispatch } from 'react-redux'

import { nanoid } from '@reduxjs/toolkit'

import { postAdded } from './postsSlice'

export const AddPostForm = () => {

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

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

const dispatch = useDispatch()

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

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

const onSavePostClicked = () => { - сделали обработчик, который и нат пустоту проверяет и диспатч делает

if (title && content) {

dispatch(

postAdded({

id: nanoid(), - вот библиотека работает

title, а тут инфу цепляем из локального стейта (там же это лежит)

content а тут инфу цепляем из локального стейта (там же это лежит) TYPE походу нет, потому что это все же через криэйторы…. хз

})

)

setTitle('')

setContent('')

}

}

return (

<section>

<h2>Add a New Post</h2>

<form>

{/\* omit form inputs \*/}

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

Save Post

</button>

</form>

</section>

)

}

Summary

* **Redux state is updated by "reducer functions"**:
  + Reducers always calculate a new state *immutably*, by copying existing state values and modifying the copies with the new data
  + The Redux Toolkit createSlice function generates "slice reducer" functions for you, and lets you write "mutating" code that is turned into safe immutable updates
  + Those slice reducer functions are added to the reducer field in configureStore, and that defines the data and state field names inside the Redux store
* **React components read data from the store with the useSelector hook**
  + Selector functions receive the whole state object, and should return a value
  + Selectors will re-run whenever the Redux store is updated, and if the data they return has changed, the component will re-render
* **React components dispatch actions to update the store using the useDispatch hook**
  + createSlice will generate action creator functions for each reducer we add to a slice
  + Call dispatch(someActionCreator()) in a component to dispatch an action
  + Reducers will run, check to see if this action is relevant, and return new state if appropriate
  + Temporary data like form input values should be kept as React component state. Dispatch a Redux action to update the store when the user is done with the form.

Ровно ничего про поток данных, ничего о структуре. Просто пересказали чужой туториал