// Импортируем функции и типы из redux-toolkit и ваш API сервис

import { createSlice, createAsyncThunk, PayloadAction } from "@reduxjs/toolkit";

import { api } from "src/services/api";

// --- 1. createAsyncThunk ---

// Создаём асинхронный экшен для получения данных пользователя

export const fetchUser = createAsyncThunk(

"user/fetchUser", // Уникальное имя экшена

async (\_, { rejectWithValue }) => { // Для этого запроса не нужны аргументы, поэтому "\_"

try {

// Обращаемся к API для получения данных пользователя

const data = await api.fetchMe();

// Если запрос успешен, возвращаем объект с email, name, surname

return data;

} catch (error) {

// Если произошла ошибка, возвращаем её текст (это попадёт в state.error)

return rejectWithValue(error?.message || "Ошибка при загрузке профиля");

}

}

);

// --- 2. Тип для состояния пользователя ---

interface UserState {

email: string,

name: string,

surname: string,

loading?: boolean, // Поле для отслеживания загрузки (опционально)

error?: string | null, // Поле для сохранения ошибок (опционально)

}

// --- 3. Начальное состояние пользователя ---

const initialState: UserState = {

email: "",

name: "",

surname: "",

loading: false, // По умолчанию не загружается

error: null, // Ошибок нет

};

// --- 4. Создать slice пользователя ---

const userSlice = createSlice({

name: "user", // Имя среза (будет частью глобального состояния)

initialState, // Начальное состояние

reducers: { // Стандартные (синхронные) экшены

setUserField: (

state,

action: PayloadAction<{ field: keyof UserState; value: string }>

) => {

// setUserField позволяет менять отдельные поля пользователя

const { field, value } = action.payload;

state[field] = value;

},

resetUser: () => initialState, // Сбросить стейт пользователя в начальный вид

},

extraReducers: (builder) => { // Экшены для обработки асинхронных thunk'ов

builder

// fetchUser.pending — экшен, когда пошёл запрос

.addCase(fetchUser.pending, (state) => {

state.loading = true; // Включаем индикатор загрузки

state.error = null; // Сбрасываем возможную ошибку

})

// fetchUser.fulfilled — экшен, когда запрос успешен

.addCase(fetchUser.fulfilled, (state, action) => {

state.loading = false; // Загрузка завершена

state.email = action.payload.email; // Записываем e-mail

state.name = action.payload.name; // Записываем имя

state.surname = action.payload.surname; // Записываем фамилию

})

// fetchUser.rejected — экшен, если произошла ошибка при запросе

.addCase(fetchUser.rejected, (state, action) => {

state.loading = false; // Завершаем загрузку

// Сохраняем текст ошибки (action.payload содержит наш rejectWithValue)

state.error = action.payload as string || "Ошибка";

});

},

});

// --- 5. Экспортируем обычные экшены для использования в компонентах ---

export const { setUserField, resetUser } = userSlice.actions;

// --- 6. Экспортируем редьюсер — понадобиться для конфигурации store ---

export default userSlice.reducer;

import React, { useEffect } from "react";

import { useSelector, useDispatch } from "react-redux";

import { fetchUser } from "./path/to/userSlice"; // thunk

import type { RootState, AppDispatch } from "./store"; // если на TypeScript

const UserProfile: React.FC = () => {

const dispatch = useDispatch<AppDispatch>(); // тип только для TS. Можно просто useDispatch()

const { email, name, surname, loading, error } = useSelector((state: RootState) => state.user); // вытаскиваем из user среза

useEffect(() => {

dispatch(fetchUser());

}, [dispatch]);

if (loading) return <div>Загрузка...</div>;

if (error) return <div style={{color: "red"}}>Ошибка: {error}</div>;

return (

<div>

<h1>Профиль пользователя</h1>

<p>Email: {email}</p>

<p>Имя: {name}</p>

<p>Фамилия: {surname}</p>

</div>

);

};

export default UserProfile;

import { configureStore } from '@reduxjs/toolkit';

import userReducer from './path/to/userSlice'; // путь к твоему срезу

const store = configureStore({

reducer: {

user: userReducer, // ключ user, будет доступен в state.user

},

});

export default store;

// (Опционально, если используешь TypeScript)

export type RootState = ReturnType<typeof store.getState>;

export type AppDispatch = typeof store.dispatch;

### Что такое `rejectWithValue`?

`rejectWithValue` — это функция, которую ты получаешь вторым аргументом в асинхронной функции внутри `createAsyncThunk`.

С помощью неё ты можешь вернуть свою кастомную ошибку в случае неудачи (ошибки запроса, валидации и т.д.).

Это позволяет твой экшену попадать в `.rejected` c переданным значением, которое потом будет доступно как `action.payload`.

\*\*Пример:\*\*

```js

export const fetchUser = createAsyncThunk(

"user/fetchUser",

async (\_, { rejectWithValue }) => {

try {

// ...код запроса

} catch (error) {

// Сюда попадёт твоя строка ошибки в action.payload у .rejected

return rejectWithValue("Что-то пошло не так!");

}

}

);

```

---

### Почему пропускаем первый аргумент — `\_`?

Внутрь thunk можно передать аргумент, который ты хочешь использовать в запросе.

Если тебе не нужен никакой параметр (например, userId), то ты этот аргумент просто не используешь. По соглашению, чтобы было видно "я его не использую", вместо имени пишут подчёркивание `\_`.

\*\*Пример:\*\*

```js

// Если бы аргумент был нужен, то:

createAsyncThunk("user/fetchUser", async (userId, tools) => { ... })

// В твоём случае аргумент вообще не нужен:

createAsyncThunk("user/fetchUser", async (\_, tools) => { ... })

```

---

\*\*Кратко:\*\*

- `rejectWithValue` нужен, чтобы прокинуть свою ошибку в redux.

- `\_` — просто "заглушка" вместо неиспользуемого аргумента.

Если появится необходимость передать что-то (например, id пользователя) — вместо `\_` пишешь имя.

Если хочешь — покажу примеры с передачей аргумента!

Отличный вопрос!

### 1. Имя `rejectWithValue`

Имя именно \*\*rejectWithValue\*\* – зашито в библиотеке redux-toolkit, его изменить нельзя.

Когда ты объявляешь async-функцию для `createAsyncThunk`, вторым параметром автоматически приходит объект с нужными "инструментами" — именно с этим именем.

\*\*Вот так правильно:\*\*

```js

createAsyncThunk('some/type', async (arg, { rejectWithValue }) => {

// ...

});

```

Если переписать на другое (например, `{ myReject }`), будет ошибка — внутри нет свойства с таким именем.

---

## 2. Пример с передачей аргумента

Ты можешь передать аргумент первым параметром при вызове thunk (например, id пользователя) и использовать в async-функции.

### Пример: Получить пользователя по ID

\*\*slice.ts\*\*

```js

export const fetchUserById = createAsyncThunk(

"user/fetchById",

// Первый параметр (userId) — это то, что ты передаёшь при dispatch(fetchUserById(123))

async (userId: number, { rejectWithValue }) => {

try {

const response = await fetch(`https://api.example.com/user/${userId}`);

if (!response.ok) {

// Можно вернуть ошибку

return rejectWithValue("Пользователь не найден");

}

return await response.json();

} catch (error) {

return rejectWithValue("Ошибка сети");

}

}

);

// В компоненте:

dispatch(fetchUserById(123)); // userId = 123 попадёт в функцию как первый параметр

```

---

### Пример 2: Аргумент-объект

```js

export const fetchUserByCredentials = createAsyncThunk(

"user/fetchByCredentials",

async ({ email, password }, { rejectWithValue }) => {

try {

const response = await fetch("/api/login", {

method: "POST",

body: JSON.stringify({ email, password }),

headers: { "Content-Type": "application/json" }

});

if (!response.ok) return rejectWithValue("Неверные данные");

return await response.json();

} catch {

return rejectWithValue("Ошибка сервера");

}

}

);

// Использование в компоненте:

dispatch(fetchUserByCredentials({ email: "user@mail.ru", password: "123456" }));

```

---

### Коротко

- Имя \*\*rejectWithValue\*\* менять нельзя.

- Первый аргумент в async-функции — то, что ты передашь при dispatch.

- Передавать можно всё что угодно: строку, число, объект.