**1. Создание slice** для хранения данныхСначала создадим срез (slice), где будут храниться значения name и email.

// features/userSlice.js

import { createSlice } from "@reduxjs/toolkit";

// Начальное состояние

const initialState = {

name: "",

email: "",

};

// Создаём slice

const userSlice = createSlice({

name: "user",

initialState,

reducers: {

// Обновление name

setName: (state, action) => {

state.name = action.payload; // Перезаписываем значение name

},

// Обновление email

setEmail: (state, action) => {

state.email = action.payload; // Перезаписываем значение email

},

// Сброс полей (опционально)

resetUser: (state) => {

state.name = "";

state.email = "";

},

},

});

// Экспортируем действия

export const { setName, setEmail, resetUser } = userSlice.actions;

// Экспортируем reducer

export default userSlice.reducer;

**2. Настройка хранилища** Теперь нужно зарегистрировать userSlice в глобальном хранилище.

// app/store.js

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

import userReducer from "../features/userSlice"; // Импортируем reducer из среза

// Глобальное хранилище

export const store = configureStore({

reducer: {

user: userReducer, // Регистрируем userSlice

},

});

export default store;

**3. Подключение Redux к приложению** Подключим хранилище Redux к React-приложению с помощью компонента Provider

// index.js

import React from "react";

import ReactDOM from "react-dom";

import { Provider } from "react-redux";

import { store } from "./app/store";

import App from "./App";

ReactDOM.render(

<Provider store={store}>

<App />

</Provider>,

document.getElementById("root")

);

**4. Компонент с двумя инпутами** Создадим компонент, в котором будут два поля ввода (name и email), а также кнопка для вывода текущего состояния.

// components/UserForm.js

import React from "react";

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

import { setName, setEmail, resetUser } from "../features/userSlice";

const UserForm = () => {

const dispatch = useDispatch();

// Получаем текущие значения name и email из состояния

const { name, email } = useSelector((state) => state.user);

// Обработчики для обновления значений

const handleNameChange = (e) => {

dispatch(setName(e.target.value)); // Обновляем name через action

};

const handleEmailChange = (e) => {

dispatch(setEmail(e.target.value)); // Обновляем email через action

};

// Сброс значений

const handleReset = () => {

dispatch(resetUser());

};

return (

<div>

<h2>Redux Toolkit: Хранение данных из инпутов</h2>

<div>

<label>

Name:

<input type="text" value={name}

onChange={handleNameChange}

placeholder="Введите имя"

/>

</label>

</div>

<div>

<label>

Email:

<input type="email" value={email}

onChange={handleEmailChange}

placeholder="Введите email"

/>

</label>

</div>

<button onClick={handleReset}>Сброс</button>

<div>

<h3>Текущее состояние:</h3>

<p><strong>Name:</strong> {name}</p>

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

</div>

</div>

);

};

export default UserForm;

Как работает этот пример?

**1.Компонент UserForm:**

•При вводе текста в любое из полей (name или email), вызывается соответствующий экшен (setName или setEmail), передавая значение в payload.

•Данные (payload) сохраняются в глобальном состоянии через userSlice, а затем отображаются на экране.

•Кнопка "Сброс" вызывает экшен resetUser, который возвращает начальное состояние (name = "", email = "").

**2.React + Redux Toolkit:**

•useSelector: считывает данные из состояния Redux (значения name и email).

•useDispatch: отправляет экшены в глобальное хранилище для обновления состояния.Итог

•С помощью Redux Toolkit мы удобно организовали хранение значений name и email.

•createSlice упростил создание редюсеров и экшенов.

•dispatch обновляет хранилище, а useSelector позволяет удобно извлекать данные из состояния.