Redux Toolkit

▼ ДЗ

1. Добавить в UserDetail
   возможность редактировать пользователя (все кроме id)
   и удалить его
2. Добавить страницу с таблицей где отображаются студенты с оценками и посещаемостью как у вас в mystat, сделать свой slice studentsSlice с возможностью менять оценку и статус посещения

Redux: Основы и практическое применение

Что такое Redux?

Redux — это предсказуемый контейнер состояния для JavaScript-приложений. Он помогает управлять состоянием приложения централизованно, делая его предсказуемым и отладочным.

Основные принципы Redux

1. **Единственный источник правды** — всё состояние приложения хранится в одном store
2. **Состояние только для чтения** — единственный способ изменить состояние — отправить action
3. **Изменения происходят через чистые функции** — reducer'ы определяют, как изменяется состояние

Основные концепции Redux

1. Store (Хранилище)

Store — это объект, который содержит состояние всего приложения.

В нашем примере:

```
// store/store.ts
import { configureStore } from "@reduxjs/toolkit";
import { usersSlice } from "./slices/usersSlice";

export const store = configureStore({
   reducer: {
      users: usersSlice.reducer, // ← Здесь подключаем reducer
   },
})

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

Ключевые моменты:

- `configureStore` — современный способ создания store (Redux Toolkit)
- `RootState` — тип для всего состояния приложения
- `AppDispatch` — тип для функции dispatch

2. Action (Действие)

Action — это обычный JavaScript объект, описывающий что произошло в приложении.

В нашем примере:

```
// Когда мы вызываем:
dispatch(addUser({ name: 'Иван', email: 'ivan@example.com' }))

// Создается action вида:
{
```

```
    type: 'users/addUser',
    payload: { name: 'Иван', email: 'ivan@example.com' }
}
```

Структура Action:

- `type` — строка, описывающая тип действия
- `payload` — данные, необходимые для выполнения действия

3. Reducer (Редьюсер)

Reducer — это чистая функция, которая принимает текущее состояние и action, и возвращает новое состояние.

В нашем примере:

```
// store/slices/usersSlice.ts
export const usersSlice = createSlice({
    name: 'users',
    initialState,
    reducers: {
        // ↓ Каждый reducer обрабатывает определенный action
        addUser: (state, action: PayloadAction<Omit<User, 'id'>>) ⇒ {
            const newId = Math.max(...state.users.map(u ⇒ u.id)) + 1;
            state.users.push({
                id: newId,
                ...action.payload
            })
        },
        updateUser: (state, action: PayloadAction<User>) ⇒ {
            const index = state.users.findIndex(user ⇒ user.id === action.payload.
id);
            if (index !== -1) {
                state.users[index] = action.payload
            }
        },
        deleteUser: (state, action: PayloadAction<number>) ⇒ {
```

```
            state.users = state.users.filter(user => user.id !== action.payload);
        }
    }
})
```

4. Dispatch (Отправка)

Dispatch — это функция, которая отправляет action в store для обновления состояния.

В нашем примере:

```
// UsersPage.tsx
const dispatch = useDispatch<AppDispatch>();

const handleAddUser = () => {
    if (newUserName.trim() && newUserEmail.trim()) {
        // ↓ Отправляем action в store
        dispatch(addUser({
            name: newUserName.trim(),
            email: newUserEmail.trim()
        }))
        // ... остальной код
    }
}
```

Redux Toolkit (RTK)

В нашем примере используется **Redux Toolkit** — официальный, рекомендуемый способ написания Redux логики.

Преимущества RTK:

- **createSlice** — упрощает создание reducer'ов и actions
- **configureStore** — настройка store с хорошими значениями по умолчанию

- **Immer** — позволяет "мутировать" состояние (на самом деле создает новое)

Селекторы (Selectors)

Селекторы — функции для извлечения данных из состояния.

В нашем примере:

```
// Определение селекторов
export const selectAllUsers = (state: { users: UsersState }) ⇒ state.users.user
s;
export const selectUserById = (state: { users: UsersState }, userId: number) ⇒
   state.users.users.find(user ⇒ user.id === userId)

// Использование в компонентах
const users = useSelector((state: RootState) ⇒ selectAllUsers(state))
const user = useSelector((state: RootState) ⇒ selectUserById(state, Number(i
d)))
```

Подключение Redux к React

1. Подключение Provider

```
// main.tsx
import { Provider } from "react-redux";
import { store } from "./store/store.ts";

createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <Provider store={store}>  {/* ← Оборачиваем приложение */}
      <App />
    </Provider>
  </StrictMode>
);
```

2. Использование хуков

```
// В компонентах
import { useSelector, useDispatch } from 'react-redux';

const users = useSelector((state: RootState) ⇒ selectAllUsers(state)) // Чтени
e
const dispatch = useDispatch<AppDispatch>(); // Отправка actions
```

Практические примеры из кода

Добавление пользователя

```
// 1. Action Creator (автоматически создается с createSlice)
const addUserAction = addUser({ name: 'Иван', email: 'ivan@example.com'
});

// 2. Dispatch
dispatch(addUserAction);

// 3. Reducer обрабатывает action
addUser: (state, action) ⇒ {
   const newId = Math.max(...state.users.map(u ⇒ u.id)) + 1;
   state.users.push({ id: newId, ...action.payload })
}

// 4. Компонент получает обновленное состояние через useSelector
```

Удаление пользователя

```
// Action
dispatch(deleteUser(userId));

// Reducer
```

```
deleteUser: (state, action: PayloadAction<number>) ⇒ {
    state.users = state.users.filter(user ⇒ user.id !== action.payload);
}
```

Лучшие практики

1. **Используйте Redux Toolkit** — он упрощает код и предотвращает ошибки
2. **Создавайте селекторы** — для переиспользования логики доступа к данным
3. **Типизируйте всё** — используйте TypeScript для безопасности типов
4. **Разделяйте слайсы** — каждая область состояния в отдельном файле

Когда использовать Redux?

Используйте Redux когда:

- Много компонентов используют одно состояние
- Состояние часто обновляется
- Логика обновления состояния сложная
- Приложение средних/больших размеров

НЕ используйте Redux когда:

- Простое приложение с локальным состоянием
- Состояние редко меняется
- Только один компонент использует состояние