

PHASE 3

WEEK 2

DAY 1

План

1. Redux Toolkit
2. Создание Store
3. Slice = reducer + actions types + actions creators
4. createAsyncThunk — асинхронные actions creators
5. RTK Query

Redux Toolkit

Набор инструментов для упрощения и унификации работы с [Redux](#).

Также известен как RTK.

Redux Toolkit

Зачем всё это?

- **Асинхронный код** (fetch и др.) можно вынести из React-компонентов в action creators. Получатся так называемые **Thunk**.
- **Action types, action creators** генерируются автоматически — не нужно писать вручную.

Redux Toolkit

Зачем всё это?

- Проще описывать как меняется состояние после **fetch-запросов** (отправка, успех, ошибка).
- Можно “**мутировать**” **state** в reducer-функции — менять поля объектов напрямую, пушить в массивы и т. д.

RTK позаботится об **иммутабельности** за тебя.

Redux Toolkit

Установить RTK:

```
npm i @reduxjs/toolkit
```

Создание Store

Создание Redux store

```
import { createStore } from 'redux';  
import { composeWithDevTools } from 'redux-devtools-extension';  
import reducer from './rootReducer';  
  
const store = createStore(reducer, composeWithDevTools());  
  
export default store;  
  
export type RootState = ReturnType<typeof store.getState>;
```


Сырой Redux уже deprecated

// Использование Redux без Redux Toolkit - считается устаревшей практикой

```
import { createStore } from 'redux';  
import { composeWithDevTools } from 'redux-devtools-extension';  
import reducer from './rootReducer';
```

```
const store = createStore(reducer, composeWithDevTools());
```

```
export default store;
```

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



Создание store

```
import { configureStore } from '@reduxjs/toolkit';
import { useDispatch } from 'react-redux';
// Слайсы - это отдельные модули нашего приложения. У каждого слайса - свой редьюсер.
import tasksSlice from '../features/tasks/tasksSlice';
import authSlice from '../features/auth/authSlice';

const store = configureStore({
  // теперь функция combineReducers не нужна
  reducer: {
    auth: authSlice,
    tasks: tasksSlice,
  },
});

// для правильной типизации будем использовать useAppDispatch вместо
// useDispatch
export type AppDispatch = typeof store.dispatch;
export const useAppDispatch: () => AppDispatch = useDispatch;

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

export default store;
```



**Slice =
reducer +
actions types +
actions creators**

Slice

Теперь вместо того, чтобы отдельно описывать `reducer`, `actions types` и `actions creators` для каждого раздела приложения мы описываем только `slice`.

В слайсе тебе нужно реализовать все кейсы редьюсера (каждый в отдельной функции), а `actions types` и `actions creators` сгенерируются автоматически.

Кроме того в редьюсере можно (и нужно) мутировать стэйт!

Slice

```
// features/counter/counterSlice.ts
import { createSlice } from '@reduxjs/toolkit';
import CounterState from '../types/CounterState';

// начальный state
const initialState: CounterState = { count: 0 };

// объявляем slice с именем "counter"
const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    // отдельные кейсы редьюсера записываются в отдельные функции
    // здесь можно мутировать state, RTK создаст копию state автоматически
    plus: (state) => { state.count += 1; },
    minus: (state) => { state.count -= 1; },
  },
});

// RTK создаёт action creators для каждого кейса редьюсера, экспортируем их
export const { plus, minus } = counterSlice.actions;
// экспортом по умолчанию будет reducer
export default counterSlice.reducer;
```

Slice (использование во View)

```
// features/counter/Counter.tsx
import React from 'react';
import { useSelector } from 'react-redux';
import { selectCount } from '../selectors';
import { plus, minus } from '../counterSlice';
import { useAppDispatch } from '../../store';

export default function Counter(): JSX.Element {
  const count = useSelector(selectCount); // selectCount = (state) => state.count
  const dispatch = useAppDispatch();

  return (
    <div>
      <button onClick={() => dispatch(minus())}>-</button>
      <span>{count}</span>
      <button onClick={() => dispatch(plus())}>+</button>
    </div>
  );
}
```

createAsyncThunk

createAsyncThunk (API)

`createAsyncThunk` позволяет вынести асинхронную логику (например, `fetch`) из React-компонента в отдельные функции и запускать их как обычные экшены через `dispatch`. Разберём `createAsyncThunk` на примере загрузки с сервера списка задач.

```
// features/tasks/api.ts
import Task from '../types/Task';

export async function getTasks(): Promise<Task[]> {
  const result = await fetch('/api/tasks');
  return result.json();
}
```


createAsyncThunk (Slice)

```
// tasks/tasksSlice.ts
import {
  createAsyncThunk,
  createSlice,
} from '@reduxjs/toolkit';
import TasksState from '../types/TasksState';
import * as api from '../api';
```

```
// начальный state
const initialState: TasksState = {
  tasks: [],
  error: undefined,
};
```

```
// асинхронный action creator
export const loadTasks = createAsyncThunk(
  // здесь пишем action type (задаём сами)
  'tasks/loadTasks',
  () => {
    // api.getTasks возвращает Promise<Task[]>
    return api.getTasks();
  }
);
```

```
// slice объединяет в себе reducers, actions и action
creators
const tasksSlice = createSlice({
  name: 'tasks',
  initialState,
  // ...
  extraReducers: (builder) => {
    builder
      // показываем как меняется state если загрузка
      прошла успешно
      .addCase(loadTasks.fulfilled, (state, action) => {
        // здесь можно мутировать state
        // RTK создаст копию state автоматически
        state.tasks = action.payload;
      })
      .addCase(loadTasks.rejected, (state, action) => {
        // показываем как меняется state если загрузка
        прошла успешно
        state.error = action.error.message;
      });
  },
});

export default tasksSlice.reducer;
```

createAsyncThunk (Slice)

Обрати внимание: Результат `api.getTasks()` будет передан в `action.payload`. Если же `api.getTasks()` завершится с ошибкой, то ошибка будет передана в `action.error`.

```
export const loadTasks =
  createAsyncThunk(
    'tasks/loadTasks',
    () => {
      return api.getTasks();
    }
  );
```

```
extraReducers: (builder) => {
  builder
    .addCase(loadTasks.fulfilled, (state, action) => {
      state.tasks = action.payload;
    })
    .addCase(loadTasks.rejected, (state, action) => {
      state.error = action.error.message;
    });
}
```

createAsyncThunk (View)

```
// features/tasks/TasksList.tsx
import React, { useEffect } from 'react';
import { useSelector } from
'react-redux';
import { selectTasks, selectError } from
'./selectors';
import TaskView from './TaskView';
import { loadTasks } from './tasksSlice';
import { useAppDispatch } from
'../../store';
```

loadTasks - это **thunk**. Диспатчим его как обычный **action creator**.

```
export default function TasksList():
JSX.Element {
  const tasks = useSelector(selectTasks);
  const error = useSelector(selectError);
  const dispatch = useAppDispatch();

  useEffect(() => {
    dispatch(loadTasks());
  }, [dispatch]);

  if (error) return <div
className="error">{error}</div>;

  return (
    <div className="tasks list-group">
      {tasks.map((task) => (
        <TaskView key={task.id} task={task} />
      ))}
    </div>
  );
}
```

RTK Query

Также советуем познакомиться со встроенным в Redux Toolkit инструментом **RTK Query** (<https://redux-toolkit.js.org/rtk-query/overview>). Используя RTK Query ты описываешь только свой API (fetch-запросы к серверу), а reducers, actions types, actions creators и дополнительные удобные React-хуки генерируются автоматически.