

Тема занятия:

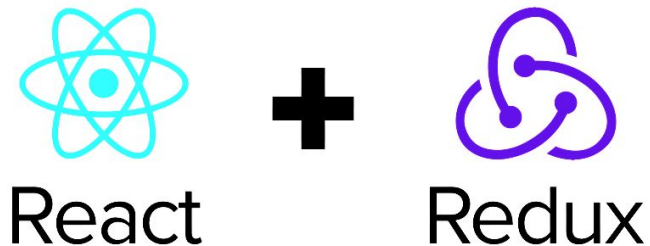
Introduction to Redux

Создание проекта React Redux



Redux - это управление состоянием приложения для JavaScript-приложений с пользовательским интерфейсом. Он является контейнером состояния, который управляет состоянием всего приложения в едином объекте, называемом хранилищем (store).

Redux Toolkit - это пакет, предназначенный для стандартного способа написания логики Redux . Он был создан, чтобы помочь решить сложности в написании логики, связанной с Redux



Установка

Рекомендуемый способ запуска новых приложений с помощью React и Redux Toolkit — использование официального шаблона

Redux Toolkit + TS для Vite

Ссылка на репозиторий:

<https://github.com/reduxjs/redux-templates/tree/master/packages/vite-template-redux>

Примечание: Vite — это инструмент сборки, цель которого — обеспечить более быструю и экономичную разработку современных веб-проектов.

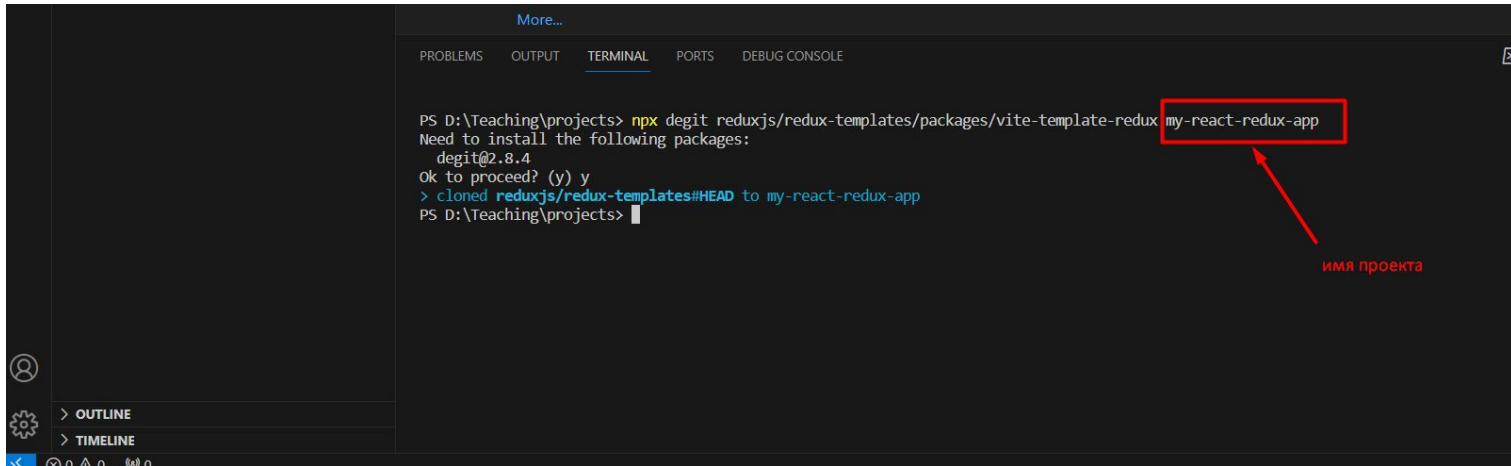


Установка

Шаг 1

Откройте VSCode, перейдите в папку, в которой будет лежать ваш проект. В терминале введите следующую команду

```
npx degit reduxjs/redux-templates/packages/vite-template-redux my-app
```



Примечание: у вас на компьютере уже должен быть установлен Node.js

Установка

Шаг 2

В VSCode перейдите в папку вашего проекта



Установка

Шаг 3

Открываем файл package.json и меняем название нашего проекта

my-react-redux-app > {} package.json > {} scripts

```
1  {
2    "name": "vite-template-redux",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite",
8      "start": "vite",
9      "build": "tsc && vite build",
10     "preview": "vite preview",
11     "test": "vitest",
12     "format": "prettier --write .",
13     "lint": "eslint .",
14     "type-check": "tsc"
15   }
```

меняем на my-react-redux-app

Установка

Шаг 4

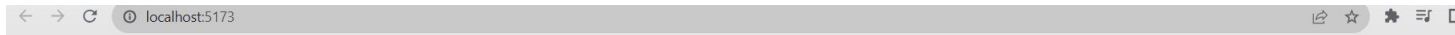
Устанавливаем все пакеты,
указанные в package.json с
помощью команды **npm install**.
Это может занять некоторое время

```
my-react-redux-app > {} package.json > {} devDependencies
1  {
2    "name": "my-react-redux-app",
3    "private": true,
4    "version": "0.0.0",
5    "type": "module",
6    "scripts": {
7      "dev": "vite",
8      "start": "vite",
9      "build": "tsc && vite build",
10     "preview": "vite preview",
11     "test": "vitest",
12     "format": "prettier --write .",
13     "lint": "eslint .",
14     "type-check": "tsc"
15   },
16   "dependencies": {
17     "@reduxjs/toolkit": "^1.8.1",
18     "react": "^18.2.0",
19     "react-dom": "^18.2.0",
20     "react-redux": "^8.0.1"
21   },
22   "devDependencies": {
23     "@testing-library/dom": "^9.2.0",
24     "@testing-library/jest-dom": "^5.11.4",
25     "@testing-library/react": "^14.0.0",
26     "@testing-library/user-event": "^14.2.5",
27     "@types/react": "^18.0.15",
28     "@types/react-dom": "^18.0.6",
29     "@types/testing-library__jest-dom": "^5.14.5",
30     "@vitejs/plugin-react": "^4.0.0",
31     "eslint": "^8.0.0",
32     "eslint-config-react-app": "^7.0.1",
33     "eslint-plugin-prettier": "^4.2.1",
34     "jsdom": "^21.1.0",
35     "prettier": "^2.7.1",
```


Установка

Шаг 5

Запускаем проект с помощью команды **npm start**. Как результат мы увидим уже созданный с помощью Redux Toolkit счётчик



- 0 +

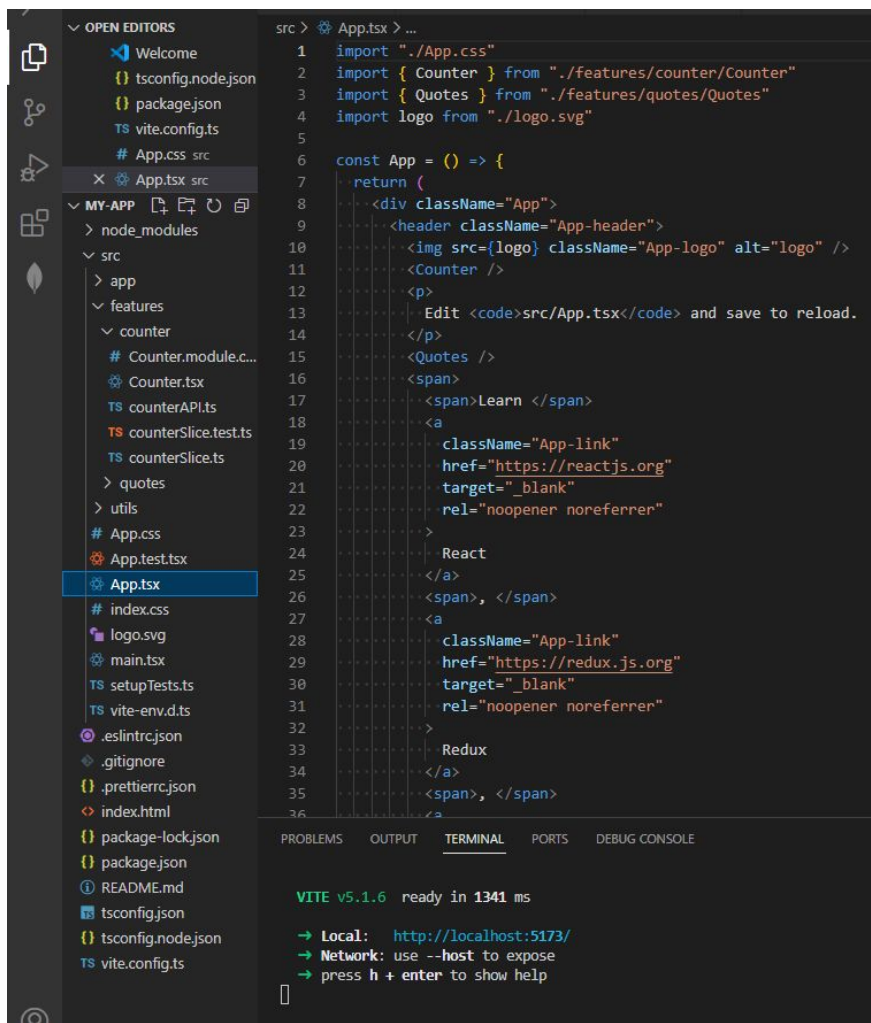
2 Add Amount Add Async Add If Odd

Edit src/App.tsx and save to reload.

Learn [React](#), [Redux](#), [Redux Toolkit](#), and [React Redux](#)

Структура проекта

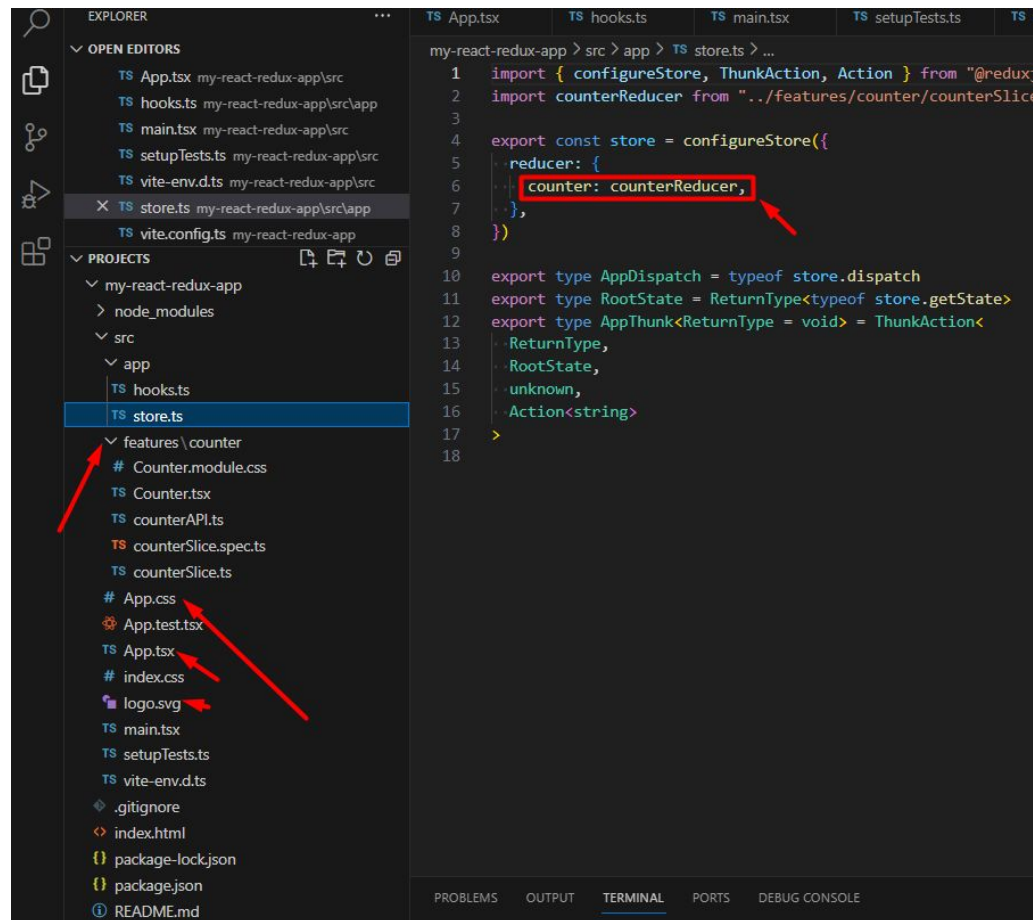
- У нас появился новый файл с настройкой vite - vite.config.ts
- Созданный counter находится в папке features
- В папке app лежит файл с настройкой store и файл с вспомогательными хуками hooks.ts



Очистка проекта

Чтобы удалить созданный counter, нужно:

- очистить файл App.tsx,
- удалить папку features,
- удалить counterReducer из объекта reducer в файле store,
- удалить файл logo.svg, удалить App.css



Redux introduction



Рассмотрим основные понятия и схему работы Redux

Инициализация:

- Хранилище (Store): Здесь хранится все состояние приложения, например, счет в игре.

Действие (Action):

- Пользователь зарабатывает очко, и это событие называется "Увеличить счет".

Диспатч (Dispatch):

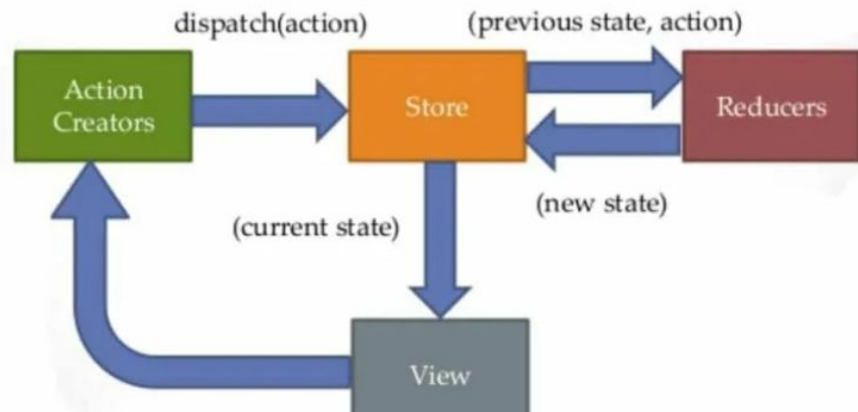
- Диспатч отправляет действие "Увеличить счет" в хранилище.

Редьюсер (Reducer):

- Редьюсер обрабатывает действие и обновляет состояние хранилища. Например, увеличит текущий счет.

Подписчик (Subscriber):

- Подписчик отслеживает изменения в хранилище. Если счет увеличился, подписчик может обновить интерфейс, чтобы отразить новый счет.



Предметный пример работы Redux

Давайте представим, что ваше веб-приложение — это большая коробка с игрушками. Каждая игрушка — это какая-то информация или часть вашего приложения.



- Хранилище (**Redux Store**): Это место, где вы кладете все свои игрушки, чтобы они были в одном месте и легко доступны.
- Действия (**Actions**): Это какие-то инструкции, чтобы что-то сделать с игрушками. Например, "Добавь новую игрушку" или "Измени цвет этой игрушки".
- Редьюсер (**Reducer**): Это специальные инструкции, как изменить игрушки в ответ на действия. Если вы получаете инструкцию "Добавь новую игрушку", редьюсер знает, как это сделать.
- Диспатч (**Dispatch**): Это как почтовая служба, которая разносит ваши инструкции (действия) редукторам (инструкциям по изменению игрушек).
- Подписчики (**Subscribers**): Это люди, которые следят за изменениями в коробке с игрушками. Если что-то меняется (например, добавляется новая игрушка), они могут быстро узнать и обновить информацию.

Зачем нужен Redux?

- **Упрощение управления состоянием:** Когда ваши приложения становятся сложными и содержат большое количество состояния (например, данные пользователя, настройки, текущее состояние интерфейса и др.), управление этим состоянием может стать сложной задачей. Redux предоставляет паттерн и инструменты для более предсказуемого и управляемого управления состоянием приложения.
- **Централизация состояния:** Redux использует одно центральное хранилище, где хранится весь стейт вашего приложения.
- **Предсказуемость изменений:** Состояние в Redux изменяется только через действия (actions), и эти изменения обрабатываются редьюсерами (reducers)
- **Легкость отладки и тестирования:** Redux обеспечивает простоту отслеживания изменений и, соответственно,



Настройка Redux Toolkit для React проекта

1. Создание функции `createAppSlice` в файле `createAppSlice.ts`

```
import { asyncThunkCreator, buildCreateSlice } from "@reduxjs/toolkit"

// `buildCreateSlice` allows us to create a slice with async thunks.
export const createAppSlice = buildCreateSlice({
  creators: { asyncThunk: asyncThunkCreator },
})
```


Настройка Redux Toolkit для React проекта

2. Создание slice (в файле counterSlice.ts) для управления состоянием счётчика. **Slice** - это часть хранилища, включающая в себя редьюсер, который управляет некоторой частью состояния, а также связанные с этим редьюсером

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { createAppSlice } from "../../app/createAppSlice"

interface CounterState {
  value: number;
}

const initialState: CounterState = { value: 0 };

export const counterSlice = createAppSlice(
  ({
    name: 'counter',
    initialState,
    reducers: create => ({
      increment: create.reducer(state => { state.value += 1; }),
      decrement: create.reducer(state => { state.value -= 1; }),
      incrementByAmount: create.reducer((state, action: PayloadAction<number>) => {
        state.value += action.payload;
      }),
    }),
    selectors: {
      selectCount: counter => counter.value,
    },
  });
export const { increment, decrement, incrementByAmount } = counterSlice.actions;
export const { selectCount } = counterSlice.selectors
```

Настройка Redux Toolkit для React проекта

3. Создание файла store.ts для настройки хранилища

```
import type { Action, ThunkAction } from "@reduxjs/toolkit"
import { combineSlices, configureStore } from "@reduxjs/toolkit"
import { counterSlice } from "../features/counter/counterSlice"

const rootReducer = combineSlices(counterSlice)
export type RootState = ReturnType<typeof rootReducer>

// The store setup is wrapped in `makeStore` to allow reuse
// when setting up tests that need the same store config
export const makeStore = (preloadedState?: Partial<RootState>) => {
  const store = configureStore({
    reducer: rootReducer,
    preloadedState,
  })
  return store
}

export const store = makeStore()
```

↓
продолжение

Настройка Redux Toolkit для React проекта

3. Создание файла store.ts для настройки хранилища

```
export type AppStore = typeof store
export type AppDispatch = AppStore["dispatch"]
export type AppThunk<ThunkReturnType = void> = ThunkAction<
  ThunkReturnType,
  RootState,
  unknown,
  Action
>
```

Настройка Redux Toolkit для React проекта

4. В файле App.tsx подключаем хранилище к нашему React-приложению с помощью обёртки

```
// App.tsx
import React from 'react';
import { Provider } from 'react-redux';
import store from './app/store';
import HomePage from './pages/HomePage';

function App() {
  return (
    <Provider store={store}>
      <div className="App">
        <HomePage />
      </div>
    </Provider>
  );
}

export default App;
```

Настройка Redux Toolkit для React проекта

5. Теперь мы можем использовать счётчик в нашем компоненте. Например, в файле **Counter.tsx**:

```
import { useState } from "react"
import { useDispatch, useSelector } from "react-redux"
import styles from "./Counter.module.css"
import {
  decrement,
  increment,
  incrementByAmount,
  selectCount
} from "./counterSlice"

function Counter () {
  const dispatch = useDispatch()
  const count = useSelector(selectCount)
  const [incrementAmount, setIncrementAmount] = useState<number>(0)

  const incrementValue = Number(incrementAmount) || 0
```

↓
продолжение

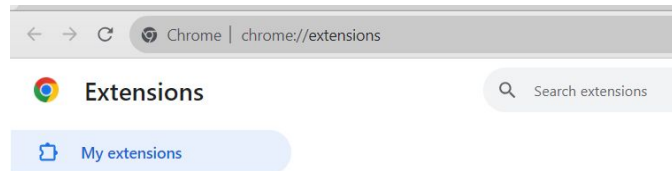
Настройка Redux Toolkit для React проекта

5. Теперь мы можем использовать счётчик в нашем компоненте. Например, в файле **Counter.tsx**:

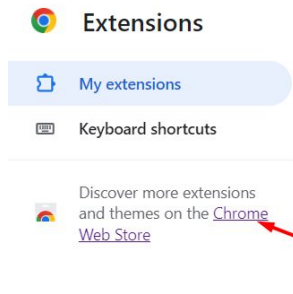
```
return (  
  <div>  
    <div>  
      <button onClick={() => dispatch(decrement())}>-</button>  
      <span>{count}</span>  
      <button onClick={() => dispatch(increment())}>+</button>  
    </div>  
    <div>  
      <input  
        value={incrementAmount}  
        type="number"  
        onChange={e => {  
          setIncrementAmount(e.target.value)  
        }}  
      />  
      <button onClick={() => dispatch(incrementByAmount(incrementValue))}>  
        Add Amount  
      </button>  
    </div>  
  </div>  
)  
}
```

Установка расширения для браузера Redux dev tools

1. Перейдите на страницу расширений в Chrome.



2. Перейдите в раздел поиска расширений



Установка расширения для браузера Redux dev tools

3. Найдите и установите расширение Redux DevTools

1)

2)



Add to Chrome

4. Установленное расширение будет отображаться в инспекторе в браузере (F12)

