Управление состоянием в современном Frontend’e

Или как отказаться от Redux и выиграть

—

ГУЗЕНКО АЛЕКСАНДР
ГЛАВНЫЙ ПРОГРАММИСТ

IT_ONE INNOVATION & TECHNOLOGIES

Давайте знакомиться

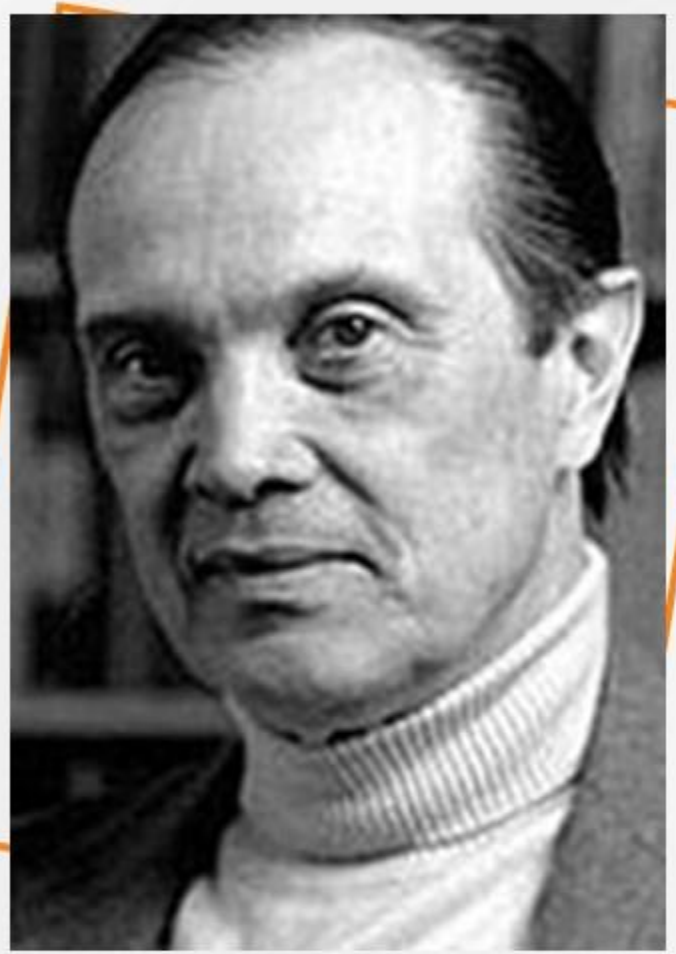
- Руковожу разработкой крупного банковского приложения
- Более 7 лет опыта работы в области
- Часто вижу излишнее усложнение кода

Почему это важно?

- Часто инструменты используются не по назначению
- Redux не панацея
- Излишнее усложнение приложений

	STATE MANAGEMENT	DATA FETCHING
Пример	Redux, MobX, Zustand …	Axios, React-query (TanStack Query), Rtk-query …
Зона ответственности	Сохранение и изменение данных	Запрос данных с сервера, кэширование и инвалидация кэша
Особенности	Требуют больше кода, сложнее для понимания НО дают больше гибкости	Мало кода, большая часть логики инкапсулирована внутри библиотеки

Закон природы «Кошелёк Миллера»



Джордж Миллер, психолог 1920-2012

> ***человеческая память не может запомнить и повторить более 7 ± 2 элементов***

- Использование инструментов не по назначению

ПРИВОДИТ К

- Усложнению проекта

ПРИВОДИТ К

- Потери эффективности

IT_ONE

Типы данных в приложении

	СОСТОЯНИЕ ПРИЛОЖЕНИЯ (APPLICATION STATE)	СОСТОЯНИЕ КОМПОНЕНТОВ (COMPONENT STATE)	СОСТОЯНИЕ СЕРВЕРНЫХ ДАННЫХ (SERVER DATA STATE)
Пример	Выбранный язык, тема, настройки	Значение инпута, текущий номер страницы, открытие и закрытие модальных окон	Данные о пользователе, списки элементов, детали объектов
На что могут повлиять изменения	Любые компоненты	текущий и вложенные компоненты	Любые компоненты
Предпочтительные инструменты	State Management библиотеки, Context	Props, Context	State Management библиотеки, Data-fetching библиотеки

Когда можно обойтись без State Management библиотек?

- Отсутствие Aplication State (или он очень простой)
  +
- Нет сложной бизнес логики

Преимущества отсутствия State Management библиотек в пользу Data-fetching библиотек

- Кеширование и инвалидация кеша
- Отсутствие boilerplate кода
- Освобождается оперативная память программиста
- Codegen инструменты (для swagger/GraphQL)

Вывод

State Management библиотеки нужны НЕ ВСЕГДА!

Что делать, если State Management библиотеки мне необходимы?

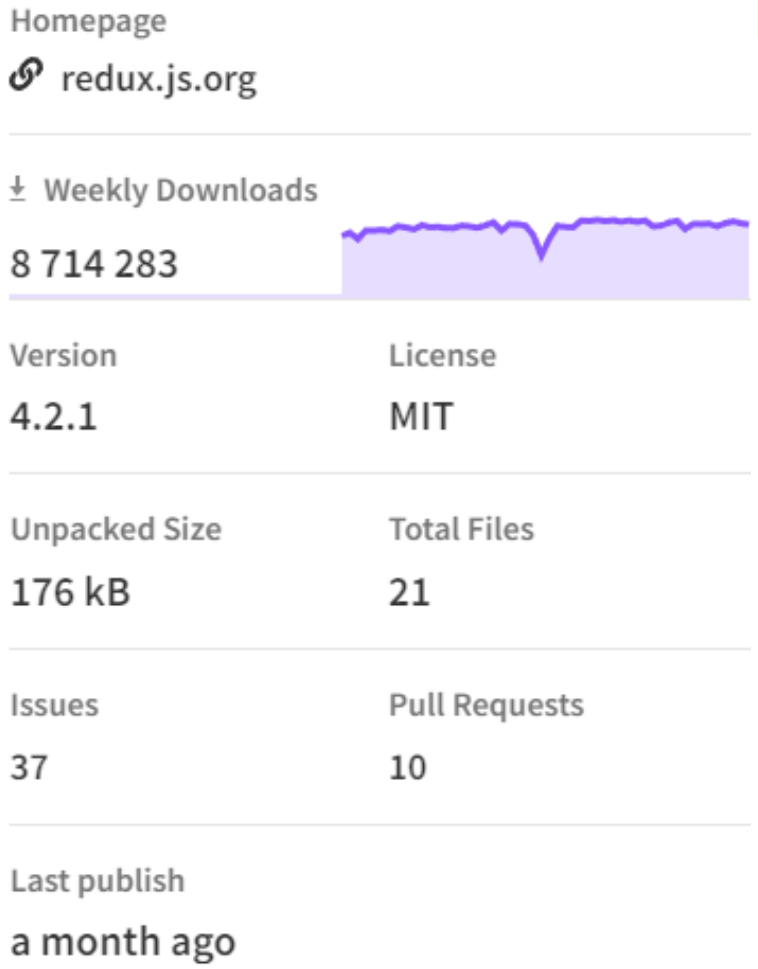

$k
Сколково
CARICATURA.RU

Redux

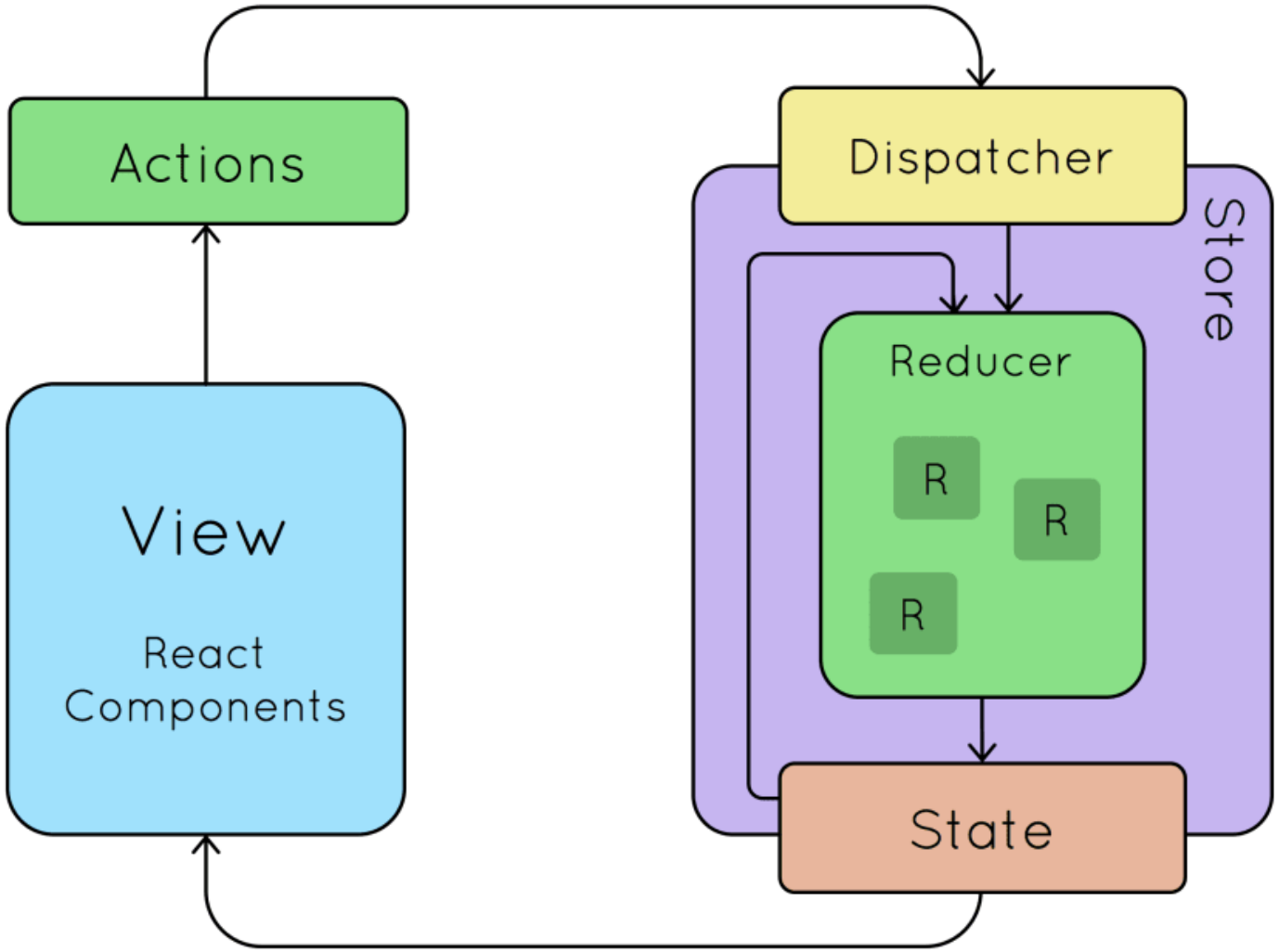
- Большое комьюнити
- Инфраструктура
- Стабильность

НО

- Много boilerplate кода
- Надо следить за мемоизацией
- Плохо подходит для модульной архитектуры



Redux

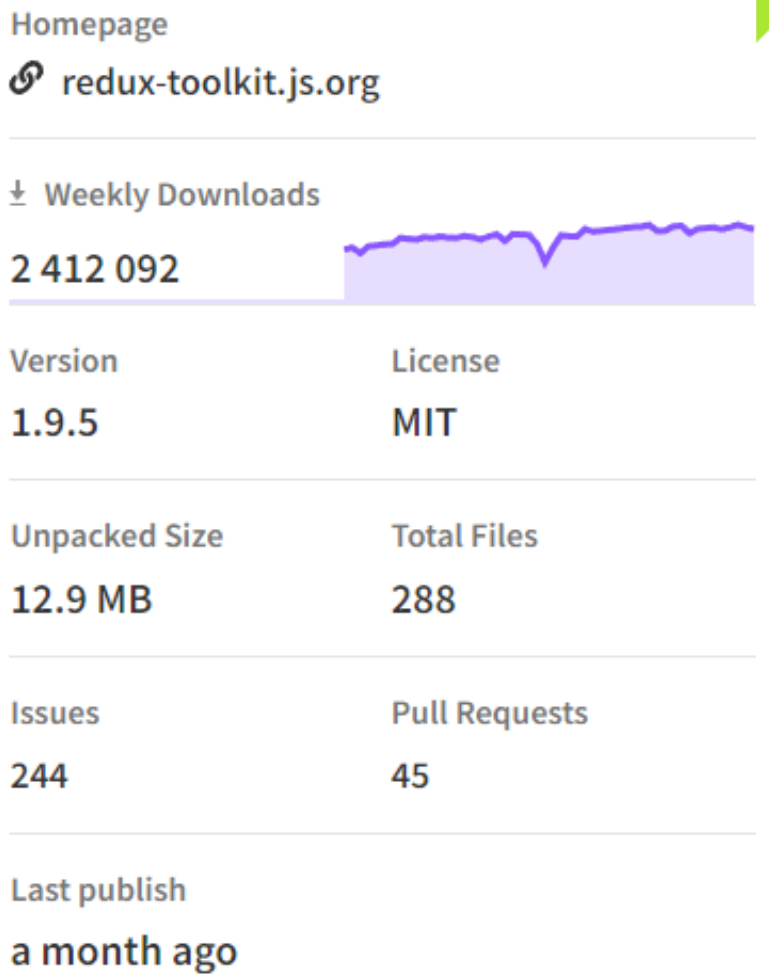

Actions
Dispatcher
Store
Reducer
R
R
R
View
React
Components
State

Reduxjs/toolkit

Все преимущества и недостатки redux

+

- Меньше boilerplate кода
- Удобное новое API

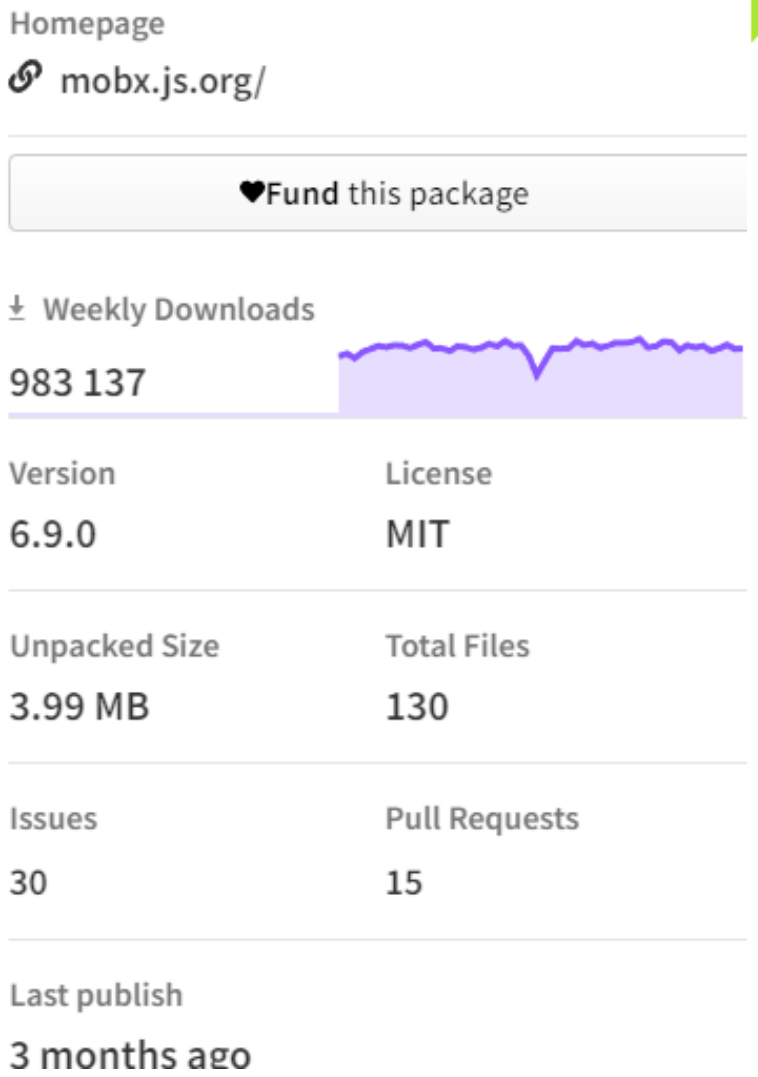


MobX

- Модульность
- boilerplate-free
- Нет необходимости следить за мемоизацией

НО

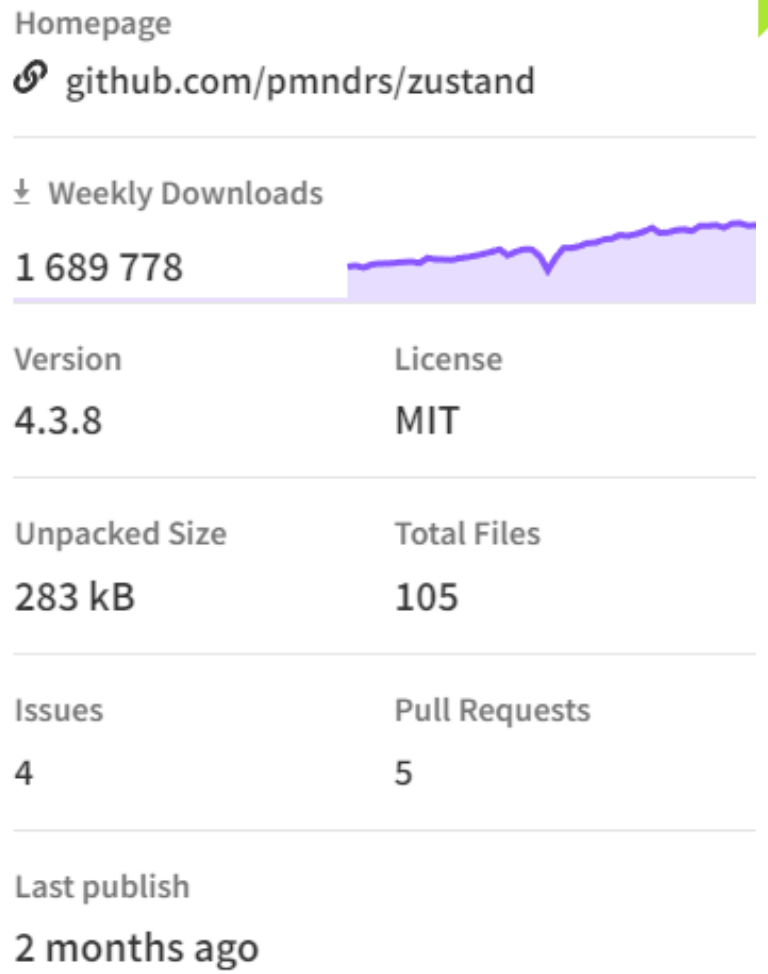
- Инструменты отладки уступают в удобстве Redux DevTools
- Непривычный реактивный подход с мутабельным состоянием



```
import React from "react"
import ReactDOM from "react-dom"
import { makeAutoObservable } from "mobx"
import { observer } from "mobx-react-lite"

class Timer {
    secondsPassed = 0

    constructor() {
        makeAutoObservable(this)
    }

    increaseTimer() {
        this.secondsPassed += 1
    }
}

const myTimer = new Timer()

// A function component wrapped with `observer` will react
// to any future change in an observable it used before.
const TimerView = observer(({ timer }) => <span>Seconds passed: {timer.secondsPassed}</span>)

ReactDOM.render(<TimerView timer={myTimer} />, document.body)

setInterval(() => {
    myTimer.increaseTimer()
}, 1000)
```

Zustand

- Модульность
- boilerplate-free
- Работает на понятных всеми хуках



```
import { create } from 'zustand'

const useStore = create((set) => ({
  bears: 0,
  increasePopulation: () => set((state) => ({ bears: state.bears + 1 })),
  removeAllBears: () => set({ bears: 0 }),
}))
```

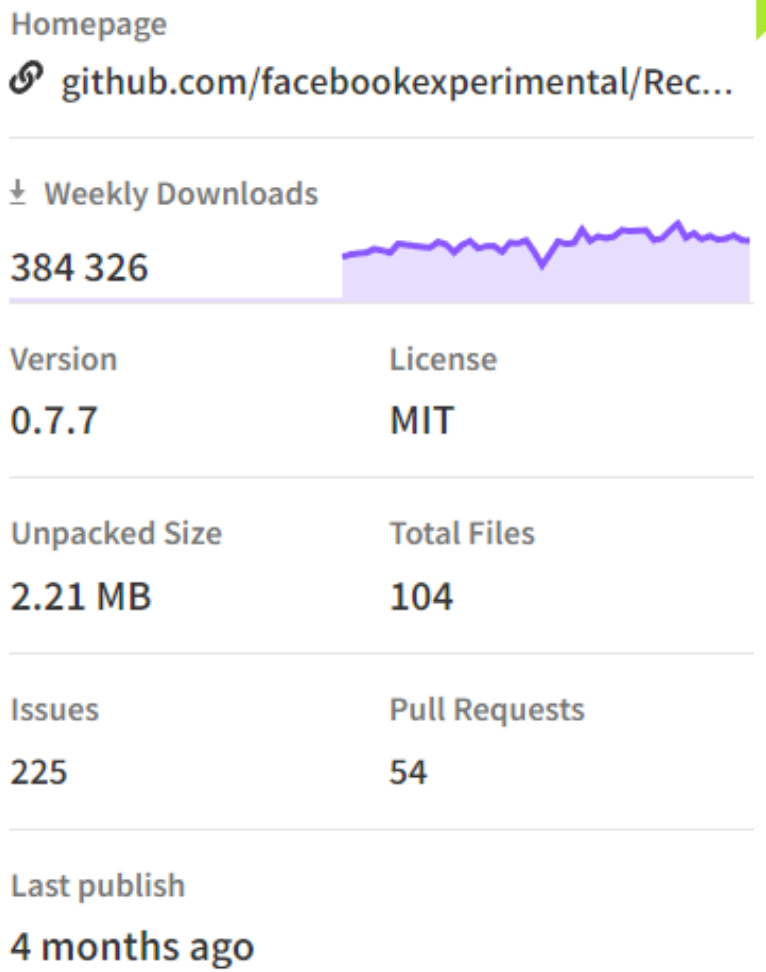
```
function BearCounter() {
  const bears = useStore((state) => state.bears)
  return <h1>{bears} around here ... </h1>
}

function Controls() {
  const increasePopulation = useStore((state) => state.increasePopulation)
  return <button onClick={increasePopulation}>one up</button>
}
```

Recoil

- Модульность
- Простота
- Поддержка команды facebook
- Решает ограничения useState и Context API

НО

- Нет API для выноса отделения бизнес логики от компонента
- Относительно молодой инструмент



```
const fontSizeState = atom({
  key: 'fontSizeState',
  default: 14,
});
```

```
function FontButton() {
  const [fontSize, setFontSize] = useRecoilState(fontSizeState);
  return (
    <button onClick={() => setFontSize((size) => size + 1)} style={{fontSize}}>
      Click to Enlarge
    </button>
  );
}
```

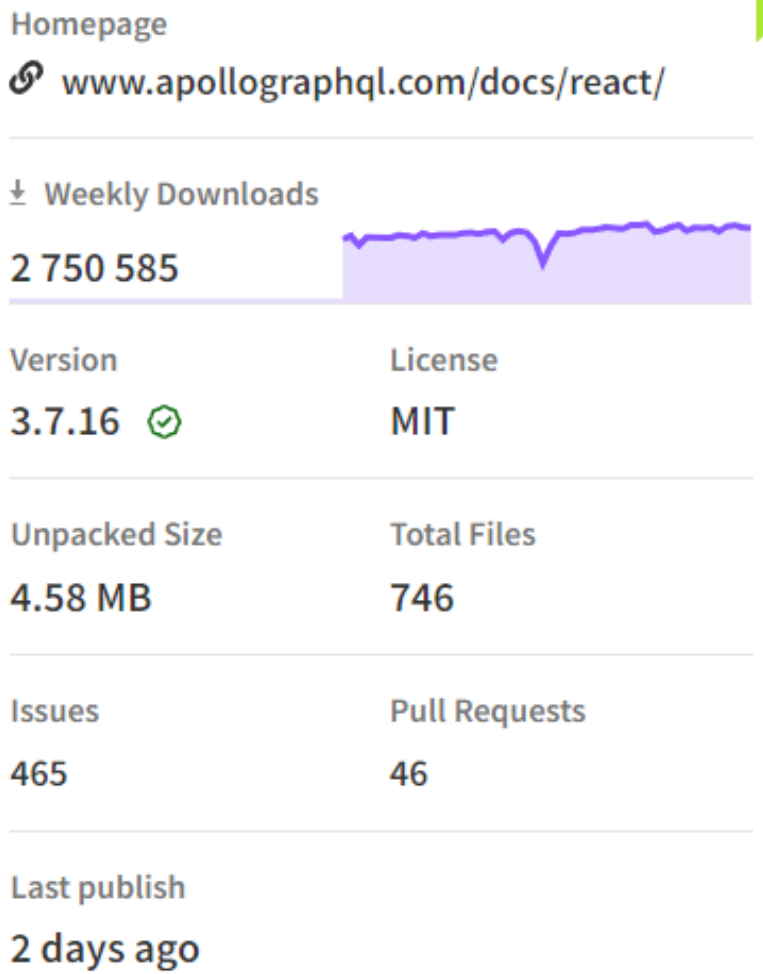
```
function Text() {
  const [fontSize, setFontSize] = useRecoilState(fontSizeState);
  return <p style={{fontSize}}>This text will increase in size too.</p>;
}
```

@apollo/client

- Одно из лучших решений для GraphQL
- Поддержка React из коробки
- State management + Data-fetching

НО

- Ориентирован в основном на GraphQL



Ситуация	Предпочтительное решение
Нет сложной бизнес логики	Data Fetching вместо State Managemen
Нет сложной бизнес логики + есть ресурсы для экспериментов	Recoil
Используете GraphQL	@apollo/client
Модульное решение + реактивность	MobX
Модульное решение + Hooks API	Zustand
Большое приложение со сложной бизнес логикой	Redux или Reduxjs/toolkit

Спасибо за внимание!

КОНТАКТЫ

MANKEY.SN@GMAIL.COM

—

HTTPS://WWW.LINKEDIN.COM/IN/ALEKSANDR-GUZENKO/

IT_ONE INNOVATION & TECHNOLOGIES