

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
GRADUATION THESIS**

**Исследование эффективности функционирования клиентской части веб-приложений
с использованием актуальных библиотек управления состояниями в React**

Обучающийся / Student Ким Артур

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42081

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Веб-технологии 2021

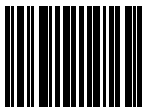
Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Руководитель ВКР/ Thesis supervisor Государев Илья Борисович, доцент, кандидат педагогических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Обучающийся/Student

Документ подписан	
Ким Артур	
26.05.2023	

(эл. подпись/ signature)

Ким Артур

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Государев Илья Борисович	
26.05.2023	

(эл. подпись/ signature)

Государев Илья
Борисович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**АННОТАЦИЯ
ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
SUMMARY OF A GRADUATION THESIS**

Обучающийся / Student Ким Артур

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42081

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Веб-технологии 2021

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Тема ВКР/ Thesis topic Исследование эффективности функционирования клиентской части веб-приложений с использованием актуальных библиотек управления состояниями в React

Руководитель ВКР/ Thesis supervisor Государев Илья Борисович, доцент, кандидат педагогических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

**ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ
DESCRIPTION OF THE GRADUATION THESIS**

Цель исследования / Research goal

Исследовать эффективность функционирования клиентской части веб-приложений с использованием актуальных библиотек управления состояниями в React.

Задачи, решаемые в ВКР / Research tasks

– составить обзор отечественных и зарубежных научных исследований, посвященных вопросам эффективности клиентской части веб-приложений с использованием актуальных библиотек управления состоянием в React, – определить методологию компонентного подхода и причины необходимости менеджеров управления состояниями, – выделить основные метрики менеджеров управления событиями в React, – выполнить тестирование быстродействия менеджеров управления событиями в одинаковых условиях, – получить и описать результаты, полученные в ходе исследования, – сформировать требования к будущему веб-сайту с информацией, полученной в ходе исследования, – разработать интерфейс веб-сайта с информацией по выбору библиотеки управления состояниями компонентов, полученной в ходе проведенного исследования.

Краткая характеристика полученных результатов / Short summary of results/findings


– рассмотрены отечественные и зарубежные научные источники на тему эффективности клиентской части веб-приложений с использованием актуальных библиотек управления состоянием в React, – дано определение методологии компонентного подхода и объяснены

причины необходимости менеджеров управления состояниями, – выделены основные метрики менеджеров управления событий в React, – протестировано быстроедействие менеджеров управления событий в одинаковых условиях, – получены и описаны результаты, полученные в ходе исследования, – сформированы требования к будущему веб-сайту с информацией, полученной в ходе исследования, – разработан, согласно сформированным ранее требованиям, интерфейс веб-сайта с информацией по выбору библиотеки управления состояниями компонентов, полученной в ходе проведенного исследования.

Наличие публикаций по теме выпускной работы / Publications on the topic of the thesis

1. Ким А. Оптимизация ре-рендеринга компонентов инструментами React // Научно-технические инновации и веб-технологии -2022. - № 1. - С. 4-10 (Статья)
2. Ким А. Выбор менеджера управления состояний в React // Научно-технические инновации и веб-технологии -2022. - № 2. - С. 30-34 (Статья)
3. Ким А. Анализ и оптимизация ре-рендеринга компонентов // Современное образование: традиции и инновации -2021. - № 4. - С. 69-75 (Статья; РИНЦ)

Обучающийся/Student

Документ подписан	
Ким Артур	
26.05.2023	

(эл. подпись/ signature)

Ким Артур

(Фамилия И.О./ name
and surname)

Руководитель ВКР/
Thesis supervisor

Документ подписан	
Государев Илья Борисович	
26.05.2023	

(эл. подпись/ signature)

Государев Илья
Борисович

(Фамилия И.О./ name
and surname)

**Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
ITMO University**

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ /
OBJECTIVES FOR A GRADUATION THESIS**

Обучающийся / Student Ким Артур

Факультет/институт/кластер/ Faculty/Institute/Cluster факультет программной инженерии и компьютерной техники

Группа/Group P42081

Направление подготовки/ Subject area 09.04.04 Программная инженерия

Образовательная программа / Educational program Веб-технологии 2021

Язык реализации ОП / Language of the educational program Русский

Статус ОП / Status of educational program

Квалификация/ Degree level Магистр

Тема ВКР/ Thesis topic Исследование эффективности функционирования клиентской части веб-приложений с использованием актуальных библиотек управления состояниями в React

Руководитель ВКР/ Thesis supervisor Государев Илья Борисович, доцент, кандидат педагогических наук, Университет ИТМО, факультет программной инженерии и компьютерной техники, доцент (квалификационная категория "ординарный доцент")

Основные вопросы, подлежащие разработке / Key issues to be analyzed

Техническое задание:

- рассмотреть отечественные и зарубежные научные источники на тему эффективности клиентской части веб-приложений с использованием актуальных библиотек управления состоянием в React,
- дать определение методологии компонентного подхода и объяснить причины необходимости менеджеров управления состояниями,
- выделить основные метрики менеджеров управления событий в React,
- протестировать быстродействие менеджеров управления событий в одинаковых условиях,
- получить и описать результаты, полученные в ходе исследования,
- сформировать требования к будущему веб-сайту с информацией, полученной в ходе исследования,
- разработать, согласно сформированным ранее требованиям, интерфейс веб-сайта с информацией по выбору библиотеки управления состояниями компонентов, полученной в ходе проведенного исследования.

Цель работы:

Исследовать эффективность функционирования клиентской части веб-приложений с использованием актуальных библиотек управления состояниями в React.

Задачи работы:

- составить обзор отечественных и зарубежных научных исследований, посвященных вопросам эффективности клиентской части веб-приложений с использованием актуальных библиотек управления состоянием в React,
- определить методологию компонентного подхода и причины необходимости менеджеров управления состояниями,
- выделить основные метрики менеджеров управления событиями в React,
- выполнить тестирование быстродействия менеджеров управления событиями в одинаковых условиях,
- получить и описать результаты, полученные в ходе исследования,
- сформировать требования к будущему веб-сайту с информацией, полученной в ходе исследования,
- разработать интерфейс веб-сайта с информацией по выбору библиотеки управления состояниями компонентов, полученной в ходе проведенного исследования.

Перечень подлежащих разработке вопросов:

Введение

1 Современное состояние проблемы

1.1 Анализ отечественных научных источников по теме исследования

1.2 Анализ зарубежных научных источников по теме исследования

2 Методология компонентного фронтенда

2.1 Подход SPA

2.2 Менеджеры управления состояниями

3 Проведение эксперимента для сравнения библиотек управления состоянием

3.1 Сравнение характеристик менеджеров управления событиями

3.2 Тестирование времени рендеринга в одинаковых условиях

3.3 Результаты эксперимента

4 Разработка рекомендательного веб-сайта

4.1 Требования к веб-сайту с рекомендациями по state-менеджерам

4.2 Проектирование будущего интерфейса

4.3 Разработка проекта

4.4 CI/CD и развертывание проекта

Заключение

Рекомендуемые материалы и пособия:

1) Banks A., Porcello E. Learning React: functional web development with React and Redux. – "" O'Reilly Media, Inc."" , 2017.

2) Mezzalana L. Mobx: Simple state management //Front-End Reactive Architectures. – Apress, Berkeley, CA, 2018. – С. 129-158.

3) Tharaka R. Recoil the next state management library for React [Электронный ресурс] Режим доступа: <https://medium.com/swlh/intro-to-recoil-d689a77c5f04>

4) Chinnathambi K. Learning React: A Hands-on Guide to Building Web Applications Using React and Redux. – Addison-Wesley Professional, 2018.

5) McFarlane T. Managing State in React Applications with Redux. – 2019.

Форма представления материалов ВКР / Format(s) of thesis materials:

Презентация

Дата выдачи задания / Assignment issued on: 06.02.2023

Срок представления готовой ВКР / Deadline for final edition of the thesis 31.05.2023

Характеристика темы ВКР / Description of thesis subject (topic)

Название организации-партнера / Name of partner organization: ООО «ДТ АЙТИ РУС»

Тема в области фундаментальных исследований / Subject of fundamental research: нет / not

Тема в области прикладных исследований / Subject of applied research: нет / not

СОГЛАСОВАНО / AGREED:

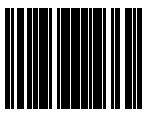
Руководитель ВКР/
Thesis supervisor

Документ подписан	
Государев Илья Борисович	
17.05.2023	

(эл. подпись)

Государев Илья
Борисович

Задание принял к
исполнению/ Objectives
assumed BY

Документ подписан	
Ким Артур	
18.05.2023	

(эл. подпись)

Ким Артур

Руководитель ОП/ Head
of educational program

Документ подписан	
Государев Илья Борисович	
19.05.2023	

(эл. подпись)

Государев Илья
Борисович

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ	4
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	5
ВВЕДЕНИЕ	7
1 СОВРЕМЕННОЕ СОСТОЯНИЕ ПРОБЛЕМЫ	12
1.1 Анализ отечественных научных источников по теме исследования .	13
1.2 Анализ зарубежных научных источников по теме исследования	18
2 МЕТОДОЛОГИЯ КОМПОНЕНТНОГО ФРОНТЕНДА	23
2.1 Подход SPA.....	23
2.1.1 SPA-фреймворки.....	25
2.1.2 Состояние компонента	29
2.2 Менеджеры управления состояниями.....	32
2.2.1 Необходимость в менеджерах управления состояниями	32
2.2.2 Библиотеки управления состояниями	35
3 ПРОВЕДЕНИЕ ЭКСПЕРИМЕНТА ДЛЯ СРАВНЕНИЯ БИБЛИОТЕК УПРАВЛЕНИЯ СОСТОЯНИЕМ	40
3.1 Сравнение характеристик менеджеров управления событиями	40
3.1.1 Документация.....	40
3.1.2 Популярность	41
3.1.3 Размер	42
3.1.4 Масштабируемость	43
3.2 Тестирование времени рендеринга в одинаковых условиях	44
3.2.1 Условия эксперимента	44
3.2.2 Замер времени рендеринга	45
3.2.3 Управление состояниями компонентов в Redux.....	47
3.2.4 Управление состояниями компонентов в MobX.....	49
3.2.5 Управление состояниями компонентов в Recoil.....	50
3.3 Результаты эксперимента	53
4 РАЗРАБОТКА РЕКОМЕНДАТЕЛЬНОГО ВЕБ-САЙТА	55
4.1 Требования к веб-сайту с рекомендациями по state-менеджерам	55

4.2 Проектирование будущего интерфейса.....	56
4.3 Разработка проекта	59
4.3.1 Инициализация	59
4.3.2 Стилизация	59
4.3.3 Маршрутизация	61
4.3.4 Менеджер управления состоянием	66
4.4 CI/CD и развертывание проекта	66
4.4.1 Тестирование	66
4.4.2 Деплой проекта на Github.....	69
4.4.3 CI/CD скрипт сборки для Github Actions.....	70
4.4.4 Создание докер-контейнера на базе репозитория	75
4.4.5 Загрузка докер образа в Docker Hub	76
ЗАКЛЮЧЕНИЕ	77
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	80

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ВКР – выпускная квалификационная работа

ГИА – государственная итоговая аттестация

ГЭК – государственная экзаменационная комиссия WEB – World Wide Web

SPA – Single Page Application

JS – JavaScript

TS – TypeScript

JSX – JavaScript XML

TSX – TypeScript XML

DOM – Document Object Model

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

JSON – JavaScript Object Notation

NPM – Node Package Manager

CI/CD – Continuous Integration and Continuous Deployment

ПО – программное обеспечение

YARN – Yet Another Resource Negotiator

API – Application Programming Interface

YML/YAML – Yet Another Markup Language

DevOps – Development Operations

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

Браузер – программа, предназначенная для просмотра веб-сайтов, гипертекстовый документов в Интернете.

Веб-сайт – совокупность веб-страниц, объединенных общим корневым адресом и доступных в Интернете через протоколы HTTP / HTTPS.

Frontend (Клиент) – часть веб-сайта, работающая на стороне пользователя (в браузере).

Фреймворк – компонент программной платформы, набор библиотек.

SPA – подход к разработке веб-сайта, при котором веб-сайт рассматривается как одна динамически изменяющаяся под запросы пользователя страница.

Компонент – это класс или функция, которая отображает пользовательский интерфейс.

Состояние компонента – данные компонента, связанные с ним и изменяющиеся с течением времени.

Оптимизация – выбор наилучшего (оптимального) варианта из множества возможных.

Менеджер управления состоянием (стейт-менеджер) – ПО для управления состоянием веб-приложения.

Хук – инструмент в React, который позволяет использовать состояние и другие возможности React без написания классов.

Рендеринг – изменение свойства или состояния компонента, которое влечет за собой перерисовку пользовательского интерфейса.

Ререндеринг – повторный рендеринг, вызванный действием пользователя.

Развертывание – процесс запуска программного приложения, системы или услуги в производство, который делает их доступными для использования конечными пользователями.

Тестирование – процесс оценки системы или ее компонентов с целью определить, удовлетворяет ли она заданным требованиям или нет.

Контейнер – изолированный, автономный и исполняемый пакет программного обеспечения, который включает в себя все необходимое для запуска части

программного обеспечения, включая код, среду выполнения, библиотеки, переменные среды и файлы конфигурации.

Масштабирование – процесс увеличения или уменьшения ресурсов, доступных программному приложению, системе или службе в ответ на изменения спроса.

Методология – систематический теоретический анализ методов, применяемых в области исследования.

Проектирование – процесс создания плана или спецификации конструкции объекта, системы или компонента.

ВВЕДЕНИЕ

В настоящее время, в эпоху бурного развития и перенасыщения рынка веб-технологий и, как следствие, повышения конкуренции, возникает проблема оптимизации веб-приложений. Каждая компания стремится сделать так, чтобы веб-сайт был быстрее, отзывчивее, но при этом не терял в функциональности и выполнял поставленные перед ним задачи в полной мере.

Главным инструментом в решении этой задачи для клиентской части веб-приложений стали современные фреймворки, особую место среди которых занял React.js. Фреймворки меняют старую стратегию, связанную с разработкой множества страниц под каждую задачу, и позволяют проектировать полноценные веб-сайты с использованием всего одной страницы и множеством компонентов в ней. Этот подход дает возможность не перезагружать веб-страницу после каждого действия пользователя, а обновлять только ту ее часть, где было осуществлено взаимодействие, то есть изменено состояние компонента.

При разработке больших высоконагруженных проектов, характеризующихся широким функционалом и множеством компонентов, существует проблема оптимизации управления состояниями сложных систем, с множеством последовательно вложенных компонентов. У вышеупомянутых фреймворков есть собственные инструменты для управления состояниями компонентов, однако их возможностей недостаточно для удобного взаимодействия с сложными состояниями. На помощь приходят сторонние библиотеки, имеющие в своем арсенале готовые решения и шаблоны под разные ситуации. Поскольку они ориентированы только под управление состояниями компонентов, их функционал в данной области намного шире, нежели встроенные решения фреймворков.

В настоящее время существует большое число различных библиотек для управления состояниями с разными подходами к хранению и изменению этих состояний. Такое обилие библиотек требует сформировать четкие критерии их применимости в плане оптимизации для разных целей и задач разработчика.

Актуальность работы определяется существенным недостатком информации о библиотеках управления состояниями компонентов в React. В настоящее время в русскоязычном сегменте Интернета крайне мало сведений о ключевых характеристиках библиотек управления состояниями, которые необходимы и полезны начинающим отечественным разработчикам в выборе между решениями в области менеджмента состояний. На данный момент существуют небольшие статьи преимущественно по Redux, однако информация об альтернативных библиотеках и их сравнении между собой практически отсутствует.

Целью ВКР является исследование эффективности функционирования клиентской части веб-приложений с использованием актуальных библиотек управления состояниями в React.

Объектом исследования являются современные библиотеки управления состоянием в связке с React.

Предметом исследования является эффективность внедрения библиотек, с точки зрения ключевых факторов работы с ними, такими как: скорость работы в равных условиях, размер, качество документации, популярность среди разработчиков, масштабируемости.

Задачи исследования:

- 1) составить обзор отечественных и зарубежных научных исследований, посвященных вопросам эффективности клиентской части веб-приложений с использованием актуальных библиотек управления состоянием в React,
- 2) определить методологию компонентного подхода и причины необходимости менеджеров управления состояниями,
- 3) выделить основные метрики менеджеров управления событиями в React,
- 4) выполнить тестирование быстродействия менеджеров управления событиями в одинаковых условиях,
- 5) получить и описать результаты, полученные в ходе исследования,

6) сформировать требования к будущему веб-сайту с информацией, полученной в ходе исследования,

7) разработать интерфейс веб-сайта с информацией по выбору библиотеки управления состояниями компонентов, полученной в ходе проведенного исследования.

Теоретические основы исследования. При анализе отечественных научных источников литературы было выявлено, что тема библиотек управления состояния компонентов в React в научном мире освещена достаточно скудно. Присутствует небольшое количество источников по библиотеке Redux, однако по другим библиотекам информации практически нет. В работах отечественных авторов (А.Э. Натроби́на, В.О. Федоров, И.С. Скороходов, А.Н. Тихомирова, Н.Н. Смекалин, С.Э. Шмаков, Е.В. Щенникова) приведена информация по библиотеке Redux, описывается основной принцип работы, положительные стороны, примеры использования. Небольшая сравнительная характеристика между библиотеками Redux и MobX приведена в работе «Сравнение библиотек Redux и mobx» (Н.С. Колышкина, Л.Е. Малкова), однако информации там крайне скудна, из-за чего трудно получить полную картину сравнения. В отдельных работах (А. Ким, А.Э. Натроби́на и В.О. Федоров) приводится дополнительная информация, касающаяся того сценария, как выполнить оптимизацию работы управления состоянием компонентов собственными силами React, благодаря использованию React Hooks и React Context Api.

В трудах зарубежных авторов (P. Podila, M. Weststrate, R. Wieruch, S. Slepner, E. Elrom, V. Subramanian, C. Györödi, R. Sotoc, S.L. Bangare, S. Gupta, M. Dalal, A. Inamdar, S. Hoque), тема раскрыта намного шире, приведена информация по самым актуальным библиотекам управления состоянием, их практическое применение, проектирование и разработка с использованием полноценных веб-приложений.

В рассмотренных источниках была приведена информация, касающаяся библиотек управления состоянием компонентов в React, как отечественными,

так и зарубежными специалистами. Малый объем опубликованного материала по библиотекам React в отечественных научных источниках, говорит о том, что данная тема исследований все еще является перспективным, требующим значительно большего развития, направлением веб-технологий в РФ. Приведенные факторы обуславливают актуальность темы ВКР.

Информационная база исследования: eLIBRARY, CyberLeninka, ResearchGate, Google Scholar, Medium.

Методы исследования.

Для решения задач исследования использовался комплекс теоретических и эмпирических взаимодополняющих методов исследования, среди которых ведущими были следующие методы: анализ, сравнение и обобщение, индукция и дедукция. Для проведения эксперимента задействованы следующие специфические методы: тестирование, наблюдение, замеры, анализ ключевых характеристик участников эксперимента, вычисление средних значений для результатов эксперимента.

Теоретическая значимость проведенного исследования состоит в систематизации информации о ключевых характеристиках при выборе библиотеки управления состояниями компонентов.

Практическая значимость состоит в возможности использования разработанного веб-сайта и результатов исследования в качестве опоры при выборе библиотеки управления состояниями компонентов для той или иной ситуации.

Достоверность теоретических и практических результатов исследования обеспечиваются проведенным анализом области исследования, использованием комплекса методов, адекватно цели и задачам исследования, его объекту и предмету исследования.

Результаты исследования. Проведенный анализ:

1) Отечественных научных источников литературы показал, что в настоящее время информация о библиотеках управления состоянием React.js ограничивается библиотекой Redux, другие же библиотеки либо освещены

очень поверхностно, либо вообще не описаны. Это говорит о том, что актуальность изучения библиотек управления состоянием компонентов является высокой, а информация, полученная в ходе настоящего исследования, внесет свой вклад в развитие отечественных веб-разработок.

2) Основных метрик библиотек управления состоянием позволил выявить, что наиболее удобная документация у библиотеки MobX; в настоящее время Redux лидирует по популярности использования; наименьший размер библиотеки в базовой конфигурации имеет Redux; Redux и Recoil хорошо подходят для масштабирования веб-приложений.

3) Скорости множественного рендеринга компонентов в равных условиях показал, что в равных условиях библиотека MobX работает быстрее среди выбранных.

4) Работы библиотеки Recoil позволил выявить, что на текущий момент библиотека Recoil крайне плохо адаптирована к работе множественного рендеринга компонентов с использованием циклов.

По теме ВКР опубликованы статьи, перечисленные в списке ниже:

1) Ким, А. Анализ и оптимизация ре-рендеринга компонентов / А. Ким // Современное образование: традиции и инновации. – 2021. – № 4. – С. 69-75. – EDN JPGJHQ. (опубликована).

2) Ким, А. Оптимизация ре-рендеринга компонентов инструментами React / А. Ким // Научно-технические инновации и веб-технологии. – 2022. – № 1. – С. 4-10. – EDN JHYSXA. (опубликована).

3) Ким, А. Выбор менеджера управления состояний в React / А. Ким // Научно-технические инновации и веб-технологии. – 2022. – № 2. – С. 30-34. – EDN DUSHAU. (принята к публикации).

1 СОВРЕМЕННОЕ СОСТОЯНИЕ ПРОБЛЕМЫ

На сегодняшний момент, основными инструментами создания клиентской части веб-приложений являются фреймворки. Они предоставляют разработчикам возможность отойти от традиционной формы многостраничного сайта, в которой приходилось создавать несколько отдельных страниц под каждую задачу, и перейти к более удобной и практичной одностраничной форме. Задачи выделяются не на каждую страницу, а на каждую составную часть страницы, называемую компонентом. Такие компоненты можно вкладывать друг в друга, создавая более сложные формы. При таком подходе разработчику достаточно написать определенное количество компонентов, после чего использовать их столько раз, сколько это необходимо, проектируя свое веб-приложение подобно конструктору. Таким образом, разрабатывается всего одна страница и множество компонентов, которые используются на этой странице по мере необходимости. С помощью данного подхода можно облегчить себе задачу разработки, используя множество готовых библиотек, компонентов и анимаций. Благодаря чему, удастся существенно сократить время и ресурсы, затрачиваемые на разработку веб-приложения.

Однако создавая клиентскую часть с помощью фреймворков, далеко не все задумываются о времени загрузки сайта, времени его ответа на запросы, плавность и комфортность использования. Актуальность оптимизации клиентской части веб-приложений на фоне быстро растущей конкуренции сейчас важна, как никогда. Зачастую, даже считанные секунды влияют на то, останется ли пользователь на сайте, или же предпочтет перейти на другой, более быстрый и отзывчивый.

Одним из главных параметров оптимизации клиентской части является управление состояниями тех компонентов, с помощью которых выстраивается интерфейс будущего веб-приложения. При использовании сложных, последовательно вложенных компонентов, правильный менеджмент их

состояний играет огромную роль в определении скорости работы всего веб-приложения.

В настоящее время существует множество разных библиотек и надстроек над фреймворками, позволяющие грамотно управлять состояниями компонентов. Самыми популярными и актуальными среди них являются Redux, MobX и Recoil. Благодаря им, значительно упрощается и автоматизируется передача состояний между компонентами. Однако зачастую трудно понять, какую именно библиотеку резонно использовать в той или иной ситуации, внедрение какой библиотеки принесет наибольшую пользу в плане оптимизации веб-приложения, с учетом важности временной составляющей ее работы, сложности изучения разработчиками документации и других немало важных факторов.

1.1 Анализ отечественных научных источников по теме исследования

Первое упоминание о библиотеках управления состояниями было найдено в работе А.Э. Натробиной и В.О. Федорова «Redux и новый React Context Api, как инструменты управления состоянием веб-приложения» [1]. В первой части статьи авторы обращают внимание на основную проблему взаимодействия сложных многовложенных компонентов без использования библиотек управления состояниями. Так как веб-приложение необходимо постоянно обновлять, улучшать и дорабатывать, компоненты в нем с каждым последующим изменением в коде будут наращивать свой уровень вложенности. Из-за чего с каждым увеличением их уровня вложенности, возникает проблема с оптимизацией передачи данных вниз по иерархии компонентов. Авторы отмечают, что без использования библиотек управления состояниями, приходится передавать данные последовательно по цепочке вниз от самого верхнего родителя до необходимого потомка. Компоненты становятся посредниками, в результате чего перегружаются лишней информацией, которая через них должна передаваться дальше, что противоречит самому термину компонента в React.

Далее, А.Э. Натробина и В.О. Федоров представляют общее описание и принцип работы одной из популярнейших на сегодняшний день библиотек управления состояниями компонентов – Redux. Отличительными характеристиками Redux являются:

- хранение состояния приложения в одном месте (что значительно упрощает централизованное управление данными, которые используются для правильной визуализации пользовательского интерфейса),
- однонаправленный поток данных в приложении [1].

Авторы отмечают, что архитектура Redux-приложения состоит из нескольких блоков, на основе которых строится идея упрощения управления состояниями компонентов и всего приложения в целом.

Основным блоком является единое и централизованное хранилище данных в приложении, именуемым Store (Хранилище). К данному блоку нельзя получить прямой доступ и изменить извне, поэтому сам процесс изменения состояния происходит в два этапа.

Команду о том, что нужно изменить определенные данные в хранилище отдает блок Actions (Действия). Если в приложении происходит событие, о котором должно узнать хранилище, создается специальный объект, в котором указывается, что какое именно событие произошло, и какие необходимые данные следует передать.

То, каким образом состояние приложения должно отреагировать на событие (действие) описывает блок Reducer (Уменьшитель). Это специальная функция, которая, приняв в себя данные от блока Actions, содержащие в себе информацию о произошедшем событии, возвращает обновленное состояние приложения. Авторы отмечают, что возвращается новый модифицированный объект, который полностью заменяет текущее состояние. Завершающим этапом потока данных в приложении является подписка данных компонентов на нужные поля в хранилище, благодаря чему обеспечивается актуальность хранящихся в компонентах данных.

Последним блоком является блок View (Вид). Это то, что видит и с чем взаимодействует пользователь при использовании приложения. Именно из него данные передаются в блок Actions и в него поступает информация о состоянии всего приложения из блока Store.

А.Э. Натроби́на и В.О. Федоров отмечают, что не всегда использование библиотек управления состоянием рационально. Так, если приложение небольшое и нужно передать значение компонента всего на несколько уровней вниз, удобнее воспользоваться «собственными силами» React, к примеру, React Context API.

В следующей работе Н.С. Колышкиной и Л.Е. Малковой «Сравнение библиотек redux и mobx», описываются определения термина «Состояние» и сравниваются две популярнейшие на текущий момент библиотеки управления состояниями в React – Redux и MobX [2]. Состоянием, по мнению авторов, является концепция, позволяющая разрабатывать компоненты в React, способные хранить и автоматически обновлять представления в соответствии с изменениями данных в них. Другим определением состояния Н.С. Колышкина и Л.Е. Малкова предлагают изменяемое хранилище данных компонента, где компоненты представляют собой автономные полнофункциональные блоки пользовательского интерфейса и логики.

В работе представляются общие описания двух библиотек, а также дается краткое описание порядка обработки состояний в обеих библиотеках. В Redux порядок выполнения обработки состояния, следующий: сначала информация с блока View помещается в блок Actions, далее, из него в блок Reducer, после в блок Store, откуда обработанная информация снова попадает в блок View. В MobX информация о изменении состояния сначала помещается с блока View в блок Actions, после в блок State, далее в Computed values, после чего вся обработанная там информация поступает вновь в блок View [2].

Авторы отмечают, что MobX позволяет писать меньше кода по сравнению с Redux, что упрощает его изучение, однако каждый раз, когда любые данные в приложении меняются, MobX выполняет перерендер (полное

обновление), что плохо сказывается на производительности в крупных приложениях.

В работе И.С. Скороходова и А.Н. Тихомировой «Исследование и сравнение современных реализаций Flux-архитектур разработки веб-приложений» отмечается, что согласно обсуждению разработчиков, на официальном форуме React, наиболее важными для них критериями в плане выбора библиотеки управления состояниями являются:

- величина сообщества – от количества сторонников и пользователей той или иной библиотеки зависит, будет ли проект активно развиваться и обновляться,

- простота – насколько легко освоить функционал библиотеки и начать работу с ней, сколько требуется времени, необходимого для чтения документации, кроме того, важно, что простой код легче поддерживать, и на его разработку расходуется меньше ресурсов,

- функциональность – как много возможностей предоставляет библиотека,

- качество документации – какой бы не была замечательной библиотека, если нет четкой и понятной документации к ней, с ней невозможно в полной мере работать [3].

Также авторы выделяют три основных принципа работы библиотеки Redux:

- состояние всего приложения хранится в едином хранилище,
- состояние приложения доступно только для чтения, то есть неизменяемо (для того, чтобы получить измененное состояние, необходимо взять исходное и наложить на него мутацию),

- изменения состояний осуществляются чистыми функциями (Редукторами), благодаря чему результат работы зависит только от переданных в них аргументов [3].

В работе А. Ким «Анализ и оптимизация рендеринга компонентов: к вопросу о подготовке будущих веб-разработчиков» описывается как можно оптимизировать передачу состояний в многовложенных компонентах собственными силами React [4].

В первой части статьи для оптимизации используется пропс `children`, благодаря которому в компоненте выделяется место, в которое можно передать другой компонент или даже несколько компонентов, такой подход позволяет менять порядок рендеринга, что позволяет не отрисовывать повторно ненужные компоненты. Метод основывается на том, что рендеринг компонентов в React выполняется по определенному пути, «от самого вложенного в структуру элемента к элементу, требующему внимания».

Во второй части статьи рассматривается метод, связанный с использованием компонента высшего порядка для повторного использования логики – мемо. Мемо после первого рендеринга компонента запоминает его значение по умолчанию, а после при повторном рендеринге проводит проверку с сопоставлением обновленных данных с исходными, и только в случае различия производит перерисовку.

В работе «Постановка и решение проблем с помощью `redux`» (Н.Н. Смекалин, С.Э. Шмаков, Е.В. Щенникова, Р.Е. Навошин, Д.Ю. Трифонова) описывается более подробно работа блока `Reducer` в библиотеке `Redux` [5]. Блок `Store` в `Redux`, обрабатывает только само дерево состояния. Когда ему нужно узнать, как действие изменяет состояние, он обращается к блоку `Reducer`. Данный блок состоит из корневого редюсера (`reducer`) и подконтрольных ему редюсеров (`reducers`). Основываясь на ключах объекта состояния, корневой забирает и разделяет состояние, после чего отправляет каждую часть подчиненным ему редюсерам, каждый из которых уже знает, что с ней делать. После, редюсеры передают свои копии обратно корневому редюсеру, который уже объединяет их вместе и формирует обновленный объект состояния. Далее, корневой редюсер передает полученный в ходе

преобразований объект состояния обратно блоку Store, а тот в свою очередь делает его новым текущим состоянием.

Е.С. Троценко в своей работе «Технологии разработки мобильных приложений для занятия фитнесом» пишет, что React в связке с MobX представляет оптимальное решение общих проблем разработки веб-приложений. React дает возможность оптимально отрисовывать интерфейс с помощью Virtual DOM, благодаря чему уменьшается количество дорогостоящих мутаций оригинального DOM. При помощи MobX синхронизируется состояние между React-компонентами, с использованием реактивной виртуальной зависимости графического состояния, обновляемого только когда это действительно необходимо [6].

1.2 Анализ зарубежных научных источников по теме исследования

В зарубежной работе «MobX Quick Start Guide: Supercharge the client state in your React apps with MobX» (P. Podila, M. Weststrate) говорится о том, что в MobX для управления состояниями применяется шаблон проектирования «Наблюдатель» [7]. Данный шаблон определяет зависимость «один ко многим» между объектами, благодаря чему при изменении одного объекта, все его зависимые объекты получают об этом уведомление и обновляются автоматически.

Авторы отмечают, что работа MobX состоит из триады потока данных: Действие, Состояние, Пользовательский интерфейс. Пользовательский интерфейс запускает действия, которые приводят к изменению состояния, из-за чего меняется и сам пользовательский интерфейс. В ходе изменения состояния могут появляться сайд-эффекты (side-effects). Сайд-эффекты – это внешние операции, которые запускаются из-за изменения состояния. Примерами сайд-эффектов могут служить такие действия, как: отправка данных на сервер, принятие данных с сервера, запуск таймера и т.д. Они обрабатываются отдельно от триады потока данных, благодаря чему устраняется дополнительная нагрузка на пользовательский интерфейс и, следовательно, на всю клиентскую часть приложения.

Для того, чтобы отслеживать определенные изменения состояния и выполнять соответствующие сайд-эффекты в MobX существуют два понятия: наблюдаемые и наблюдатели. Наблюдаемыми являются любые объекты JavaScript, за состояниями которых необходимо следить. Наблюдателями являются специальные методы MobX (API), которые позволяют анализировать изменения в состояниях, и на их основе выполнять сайд-эффекты.

Применение API MobX упрощает выполнение некоторых сложных взаимодействий в пользовательском интерфейсе. Авторы отмечают, что MobX стремится предоставить декларативный, быстрый API для управления состояниями компонентов, без ущерба для простоты.

В работе также приводится сравнение между MobX и его более популярным аналогом Redux. MobX имеет некоторое концептуальное сходство с Redux в том, что касается обеспечения однонаправленного потока данных. Однако на этом сходство заканчивается. Механизм, принятый в MobX для управления всеми изменениями состояния пользовательского интерфейса и уведомления об этих изменениях других наблюдателей, кардинально отличается от того, что используется в Redux. Redux полагается на неизменяемые и зафиксированные точки состояния и сравнивает данные между двумя точками состояния для оценки изменений. MobX, напротив, успешно анализирует постоянно изменяемые состояния, благодаря использованию детализированной системы уведомлений для отслеживания изменений состояний.

MobX, в отличие от Redux, позволяет поддерживать асинхронные обновления состояний «из коробки», поэтому для управления ему не требуются дополнительные библиотеки промежуточного ПО. Для работы Redux с асинхронными обновлениями необходимо внедрять `redux thunk`, `redux-saga` или `redux-observable`, что затрудняет как саму разработку, так и увеличивает время на анализ документации для работы с ними.

В работе «Taming the State in React: Your journey to master Redux and Mobx» (R. Wieruch) приводится еще одно сравнение Redux и MobX [8]. Автор указывает, что Redux, ввиду долгого существования, имеет большое сообщество, динамичную систему библиотек и большой выбор лучших практик. MobX же является более новым решением относительно Redux, но дает совершенно другой подход к управлению состоянием в компонентах. Он позволяет с помощью своего API (методов наблюдателей) обновлять только те компоненты, которые зависят от наблюдаемого изменения состояния. Все остальное остается нетронутым. В масштабном приложении, благодаря такому подходу, при правильном использовании MobX, можно свести к минимуму обновления персонального интерфейса.

В работе «Recoil – Another React State Management Library?» (S. Slepner) приводится информация относительно нового инструмента для управления состоянием компонентов в React – библиотеки Recoil [9]. Основа Recoil – это атом. Атом – это объект, являющийся частью состояния, на который может подписаться любой компонент. Изменение значения атома приводит к повторному рендерингу всех компонентов, подписанных на него. В работе представлены основные API библиотеки Recoil. Автор отмечает, что ввиду того, что компанией-разработчиком Recoil и React является одна компания, API Recoil очень похож на использование простых хуков (функций, которые позволяют использовать состояние и другие возможности React без написания классов) что позволяет освоить данную библиотеку без каких-либо особых трудностей в понимании синтаксиса. Дополнительная информация относительно библиотеки Recoil была найдена в работе «React and Libraries: Your Complete Guide to React Ecosystem» (E. Elrom) [10]. В ней раскрывается еще одно понятие, селекторы. Селекторы в Recoil – это чистые функции, которые позволяют синхронно и асинхронно обрабатывать само состояние, формируемое атомами. Они могут принимать в себя атомы или другие селекторы. Селекторы используются для вычислений производных данных на основе состояния. Благодаря этому, можно избежать избыточности состояния,

так как в атомах хранится минимальный набор состояния, все остальное вычисляется при помощи функций селекторов. При этом атомы и селекторы имеют одинаковый интерфейс и являются взаимозаменяемыми.

Изучая материал относительно библиотек управления состоянием компонентов, нельзя не упомянуть официальную документацию этих библиотек.

Согласно официальной документации Redux, основными преимуществами данной библиотеки являются:

- предсказуемость: благодаря Redux можно писать приложения, которые ведут себя согласованно, выполняются как на клиентской, так и серверной стороне, а также легко тестируются,
- централизованность: централизация состояния и логики приложения обеспечивает функциональность работы с состоянием (отмена, повтор, сохранение и т.д.),
- отлаживаемость: инструменты Redux DevTools упрощают отслеживание, позволяют регистрировать изменения, отправлять отчеты об ошибках на сервер,
- гибкость: Redux работает с любым уровнем пользовательского интерфейса, имеет большую экосистему аддонов, в соответствии с потребностями разработчика [11].

Следуя официальной документации MobX, важными качествами данной библиотеки являются:

- простота: MobX предоставляет минималистичный, шаблонный код, который позволяет без каких бы то ни было специальных инструментов и дополнений обнаруживать изменения состояния и передать их туда, где это требуется,
- легкий оптимальный рендеринг: все изменения и использование данных отслеживаются во время выполнения, создавая дерево зависимостей, которое фиксирует все отношения между состоянием и выходными данными

– это гарантирует, что вычисления, связанные с изменением состояния, выполняются только при строгой необходимости, вследствие чего отсутствует необходимость вручную оптимизировать компоненты с помощью подверженных ошибкам и неоптимальных методов, таких как запоминание и селекторы,

– архитектурная свобода: MobX позволяет управлять состоянием приложения за пределами любой платформы пользовательского интерфейса, что делает код разьединенным, переносимым и, прежде всего, легко тестируемым [12].

Официальная документация Recoil представляет следующие достоинства библиотеки:

– минималистичность: Recoil прост и удобен, а также изначально направлен на работу с React, вследствие чего достигается высокая совместимость, быстродействие и гибкость в управлении состоянием в компонентах React,

– график потока данных: производные и асинхронные запросы обрабатываются чистыми функциями и атомами, что упрощает обработку состояний,

– наблюдение за несколькими частями приложения: Recoil позволяет реализовать сохраняемость, маршрутизацию, отладку с течением времени или отмену, отслеживая все изменения состояния в приложении, в том числе при разделении кода [13].

2 МЕТОДОЛОГИЯ КОМПОНЕНТНОГО ФРОНТЕНДА

2.1 Подход SPA

Долгое время, в качестве основного метода к построению клиентской части веб-сайтов использовался классический подход многостраничного сайта. Каждая страница представляла собой цельный монолитный объект, а сам сайт состоял из множества таких страниц. Такой подход имел много недостатков, главным из которых являлась полная загрузка всей страницы при переходе на нее, что значительно увеличивало общее время загрузки, а также не позволяло в полной мере снизить нагрузку на сеть. Каждое взаимодействие пользователя с сайтом, независимо от характера действия, вынуждало полностью заново загружать страницу. Как смягчающий фактор был кэш, но тем ни менее в силу своего объема и механизма работы многостраничного сайта даже он не позволял значительно оптимизировать работу сайта. Кэш позволял сохранить некоторые страницы у себя в памяти, но в силу монолитности страниц, нагрузка полной перезагрузки страницы с сети просто переходила на девайс пользователя.

Чуть позже, с появлением библиотеки JQuery начали появляться проблески современного подхода веб-разработки, однако в виду того, что данная библиотека в первую очередь направлена на создание динамического пользовательского интерфейса, она совершенно не подходила для обработки клиентских данных с веб-сайта.

Лишь в конце нулевых удалось воплотить в жизнь идею нового подхода разработки клиентской части сайтов – Single Page Application (одностраничное приложение; далее просто SPA). Данный подход позволял избавиться как от монолитности, так и от идеи множества страниц, что давало свободу в плане оптимизации веб-сайтов. Основная идея SPA заключается в том, что вместо нескольких страниц под каждую задачу используется всего одна. Причем эта одна страница сама динамически подстраивается под все действия пользователя и меняется в тех местах, где это необходимо. Весь секрет данной методологии скрывается в том, что сама страница не является монолитной, а

состоит из множества компонентов, если быть точным, из одного главного компонента (представляющим собой пустой холст динамической страницы) и множеством других подкомпонентов в нем, что вызываются по мере необходимости. Именно поэтому SPA, также в некоторых ситуациях называют компонентным подходом. Это меняет стратегию нагрузки на сеть, теперь вся клиентская часть сайта сразу загружается при переходе на сайт. После, происходит рендеринг необходимых компонентов, и перед нами появляется динамически собранная страница [14]. Таким образом, при необходимости перейти на другую страницу, не нужно ее заново загружать. Все «части» сайта уже загружены в виде компонентов и перезагружаются (ререндерятся) при необходимости. Это заметно снижает нагрузку на сеть, поскольку вся клиентская часть загружается сразу, после чего сеть используется только для взаимодействия с сервером. Так, значительная часть нагрузки на сеть переносится на девайс пользователя освобождая пространство для клиент-серверного взаимодействия (рисунок 1).

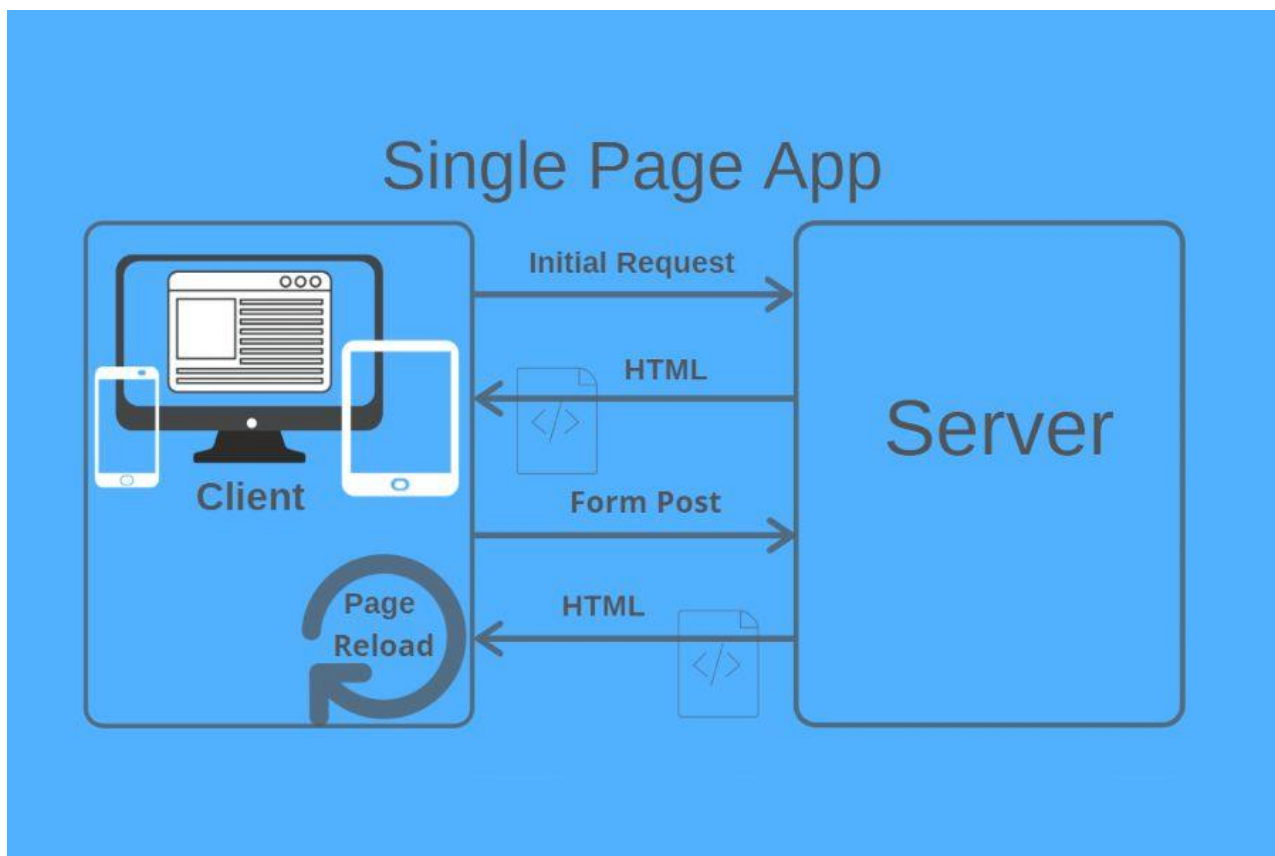


Рисунок 1 – Схема работы подхода SPA

На сегодняшний день, подход SPA является основным при разработке веб-приложений. Достаточно долгая история использования показала, что это действительно наиболее практичный способ разработки публичных и корпоративных веб-приложений с высокими требованиями производительности и поддержки кодовой базы [15]. Кроме того, в настоящее время методология SPA активно применяется при разработке мобильных и десктопных приложений, в виду того, что современные популярные кроссплатформенные фреймворки позволяют использовать те же технологии, что и при разработке веб-приложений [16].

2.1.1 SPA-фреймворки

Фреймворк – это более расширенный, динамически пополняемый и многофункциональный вариант библиотеки языка программирования [17]. Они используются для упрощения разработки веб-сайтов, приложений, сервисов, в виду того, что имеют уже готовые модули и шаблоны проектирования в качестве решения большинства проблем. Это позволяет разработчику уделять меньше времени для создания базовых элементов и сосредоточиться непосредственно на более сложных и главных аспектах. Непосредственно в вебе фреймворки делятся на серверные (backend) и клиентские (frontend) (рисунок 2).

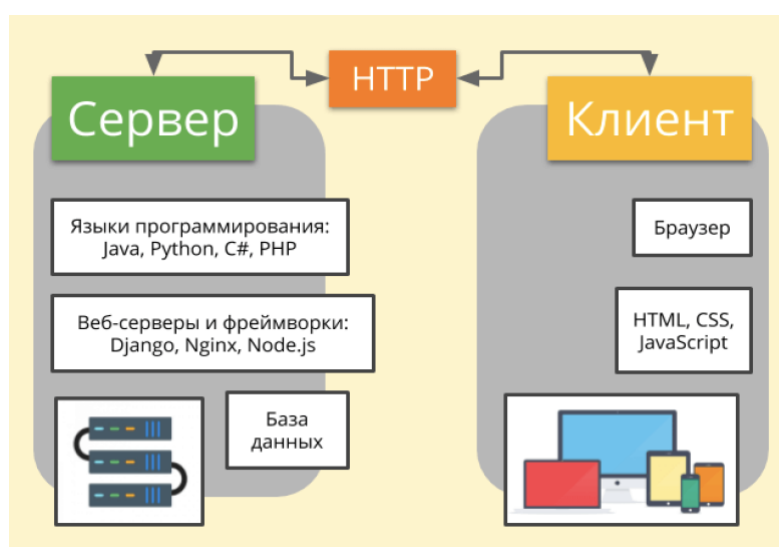


Рисунок 2 – Архитектура «Клиент-Сервер»

Серверные, как правило, позволяют работать с бизнес-логикой, хранением данных, обеспечением безопасности, иными словами, серверной частью приложений. Клиентские же, напротив, упрощают разработку пользовательского интерфейса и включают в себя как базовые HTML, CSS, JS языки, так и производные от них, к примеру, как JSX в фреймворке React. Именно в состав клиентских фреймворков и входят SPA фреймворки.

Таким образом, SPA фреймворк представляет собой разновидность клиентского фреймворка в основе структуры которого лежит вышеописанный подход SPA. Данный вид ПО помогает с легкостью создавать различные анимации и одностраничные веб-приложения.

В настоящее время актуальными современными SPA фреймворками является тройца: React, Vue, Angular.

React представляет собой SPA решение в разработке пользовательских интерфейсов (рисунок 3) [18-19]. В его основе лежит синтаксис языка JSX (JavaScript XML), что является расширением синтаксиса обычного JS, которое позволяет комбинировать вместе HTML шаблоны и JS конструкции. Несмотря на то, что в React в роли основного языка программирования выступает JSX, он также имеет поддержку языка TS (TypeScript). TS – это надстройка над языком JS, что добавляет строгую типизацию. Это позволяет избавиться от ряда ошибок связанных с типами данных. К тому же, благодаря использованию Virtual DOM под капотом React обеспечивает высокую производительность.

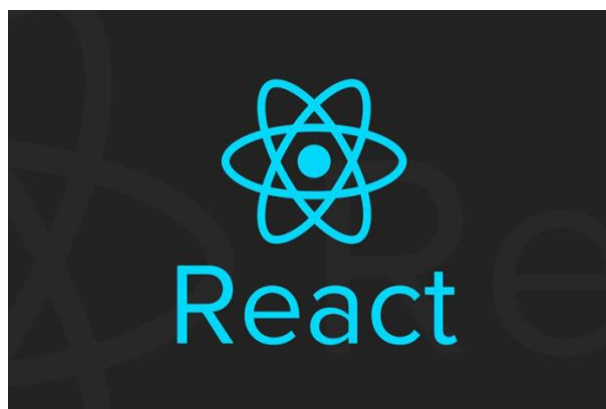
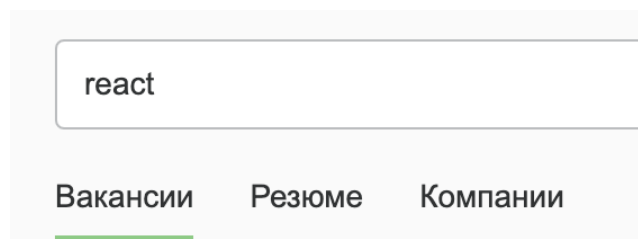


Рисунок 3 – Логотип React

Согласно крупному сервису вакансий в России HH.ru, React представляет собой наиболее популярное решение в области разработки одностраничных приложений. Именно требование знания этого фреймворка чаще всего встречается в вакансиях по России, это говорит о том, что в настоящее время React занимает лидирующие позиции в области клиентской веб-разработки (рисунок 4) [20].

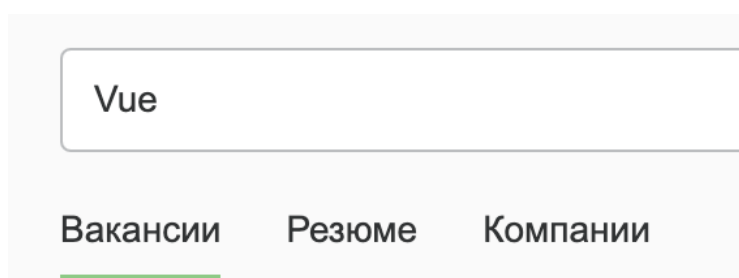


1 708 вакансий «react»

Рисунок 4 – Количество вакансий по запросу «React» в сервисе hh.ru

Актуальность, наибольшая востребованность, огромная поддержка со стороны сообщества, наличие подробной документации и множества дополнительных библиотек под разные задачи стали решающими факторами при выборе фреймворка для выполнения данной работы.

Вслед за React, на втором месте по популярности расположился не менее популярный фреймворк Vue (рисунок 5) [21].



821 вакансия «Vue»

Рисунок 5 – Количество вакансий по России по запросу «Vue» в сервисе hh.ru

Данный инструмент в отличии от React был разработан всего одним человеком, Эвансом Ю, тем ни менее в виду схожей с React концепцией, он

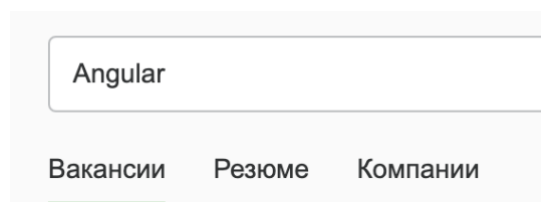
может вполне конкурировать с ним. Vue может похвастаться великолепной документацией (что делает изучение данного фреймворка проще чем React), низким порогом входа, легковесностью, поддержкой уже всем знакомых JSX и TS (рисунок 6) [22-24].



Рисунок 6 – Логотип Vue

Однако, в виду небольшого сообщества, меньшей востребованности и ограниченному количеству инструментов Vue в настоящее время уступает React.

Замыкает тройку лидеров фреймворк Angular (рисунок 7) [24].



607 вакансий «Angular»

Рисунок 7 – Количество вакансий по России по запросу «Angular»

Angular представляет свое виденье идеального SPA фреймворка от компании гиганта в области веб-разработки Google (рисунок 8) [24-25].



Рисунок 8 – Логотип Angular

Ключевой особенностью Angular, в отличие от своих конкурентов является то, что в качестве основного языка он использует TypeScript. Это является плюсом в виду того, что позволяет сразу избавиться от большинства типовых ошибок, однако данное решение увеличивает порог вхождения данного фреймворка. В Angular можно использовать обычный JS, тем не менее, сами разработчики отмечают, что делать это крайне нежелательно [25].

2.1.2 Состояние компонента

Как уже говорилось ранее, основной «боевой» единицей SPA подхода являются компоненты. Рассмотрим понятие компонента и его состояния на базе выбранного для данной работы фреймворка React.

Компонент – это базовый элемент веб-приложения. В React все состоит из них. Это позволяет многократно их использовать без лишнего переписывания кода. К тому же, компоненты можно вкладывать друг в друга, создавая тем самым более сложные много вложенные компоненты. Благодаря этому, структура веб-приложения становится легко разрабатываемой и масштабируемой. Однако, обратной стороной данного подхода становится сложность отслеживания данных, передаваемых между компонентами. В случае, если компонентов действительно много, становится крайне трудно уследить за всеми взаимодействиями между ними.

Компоненты могут хранить в себе два типа данных: пропсы и состояния (рисунок 9).

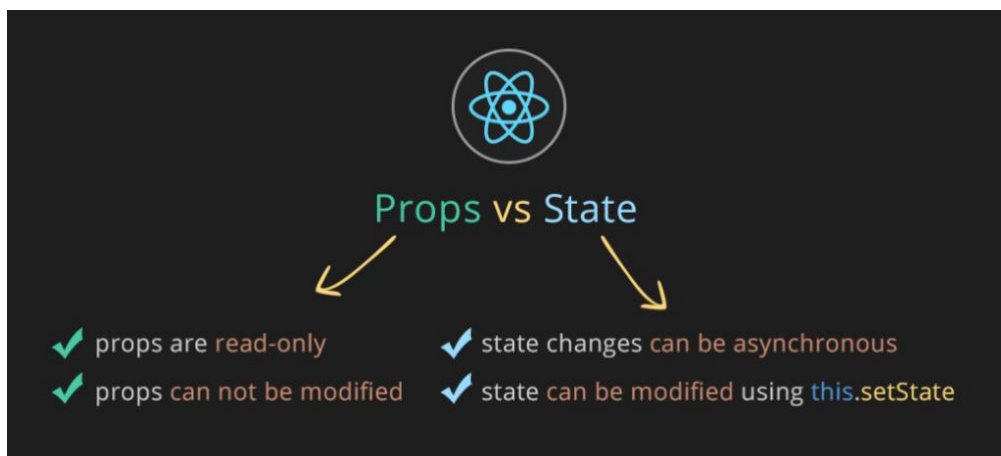


Рисунок 9 – Пропсы и состояния

Пропсы являются внешними данными компонента, переданными в него от других компонентов. Похожий механизм можно увидеть в параметрах функции. Именно путем пропсов в React реализуется передача данных между компонентами (от родительского к дочерним) (рисунок 10).

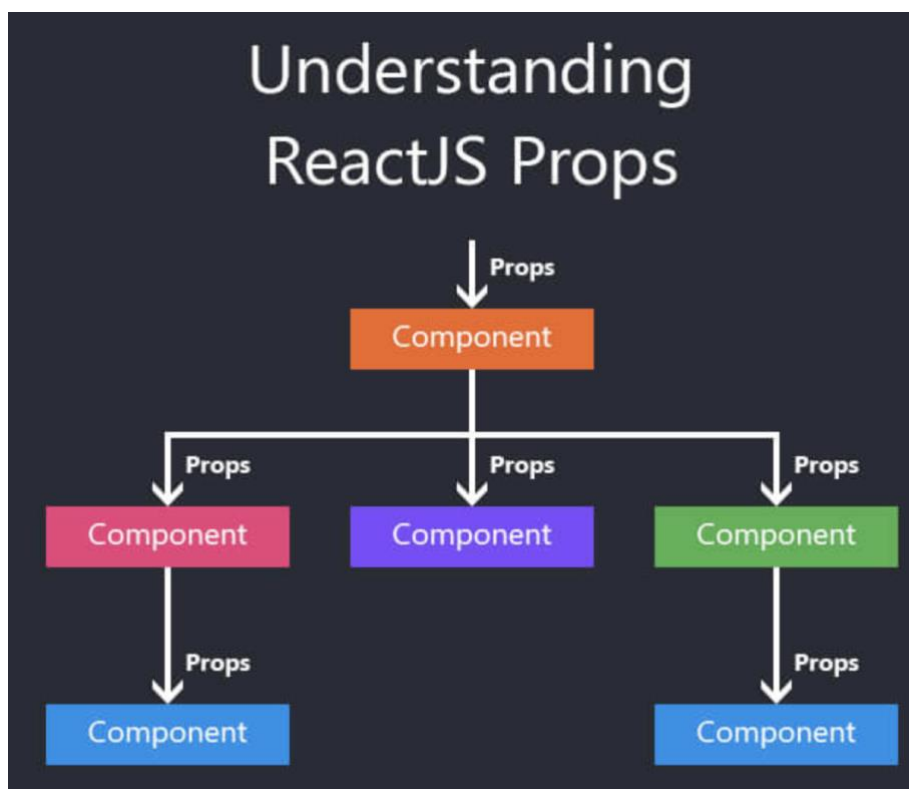


Рисунок 10 – Схема работы пропсов

Это позволяет вкладывать друг в друга компоненты и управлять их взаимодействием. В отличие от состояния, они не могут быть изменены внутри компонента и доступны только для чтения. Чтобы их поменять,

необходимо делать это в родительском компоненте и уже потом передавать в дочерний. Пропсы не всегда должны быть именно в виде данных, ими также могут быть функции обратного вызова (в этом случае передача данных осуществляется в обратном направлении, от дочернего к родительскому).

Состояние является внутренним (локальным) типом данных в компоненте, к которому нельзя получить доступ и изменить его извне. Ближайшим простым аналогом являются локальные переменные функции. Состояние представляет собой встроенный объект в React, что хранит в себе информацию о компоненте. Состояние может меняться со временем, независимо от выполнения программы (асинхронно) и каждый раз, когда это происходит, компонент перерисовывается (ререндерится). Такое изменение может быть вызвано в ответ на действия пользователей или системные события. Подобные изменения определяют поведение компонента и то, как он будет отображаться в веб-приложении. Обычно состояния используются для визуализации и отслеживания динамических изменений внутри компонента (рисунок 11).

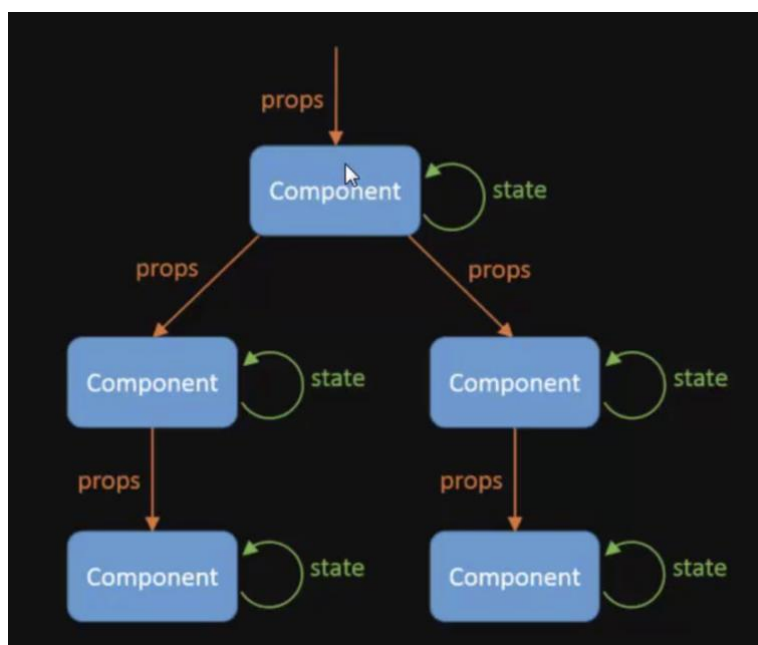


Рисунок 11 – Схема работы состояний и пропсов

Кроме того, состояние компонента также можно передать в качестве пропсов другим компонентам.

Стоит отметить, что наличие состояния для компонентов в React не является обязательным условием [26]. Так как оно увеличивает сложность и снижает предсказуемость компонента, в следствии чего там, где можно без него обойтись, предпочтительней его не использовать. Тем ни менее, обойтись совсем без состояний в интерактивном веб-приложении не получится.

2.2 Менеджеры управления состояниями

Подход SPA в корне поменял парадигму разработки клиентской части веб-приложений, в следствии чего часть проблем с оптимизацией загрузки легла на девайсы пользователей, что меняет траекторию развития веб-разработки и заставляет веб-разработчиков обратить большее внимание на рендеринг и ре-рендеринг компонентов, ключевой особенностью которых является управление их состоянием. Тут за дело вступают менеджеры управления состояниями, помогающие грамотно организовать работу компонентов. Они позволяют создавать глобальные состояния, доступ к которым можно получить из любой части веб-приложения.

Менеджеры управления состояниями (или state managers) – это библиотеки, обеспечивающие логику взаимодействия компонентов и их состояний в веб-приложении.

2.2.1 Необходимость в менеджерах управления состояниями

Несмотря на то, что в React уже есть базовые инструменты управления состоянием (React Context и React Hooks (useState и useReducer)), зачастую их бывает недостаточно.

В качестве простого примера, можно привести такую ситуацию: пусть существует веб-приложение с использованием React Context, и следующей структурой компонентов (рисунок 12).

```
      /-----CHILD_1-----CHILD_1_1
BASE---                                \--CHILD_1_2
      \-----CHILD_2
```

Рисунок 12 – Структура компонентов

В компоненте CHILD_1 имеется общее для его дочерних компонентов (CHILD_1_1 и CHILD_1_2) состояние. Тогда, следуя правилам React Context и используя React Hooks, через хук useContext можно получить это общее состояние внутри дочерних компонентов через контекст (рисунок 13).

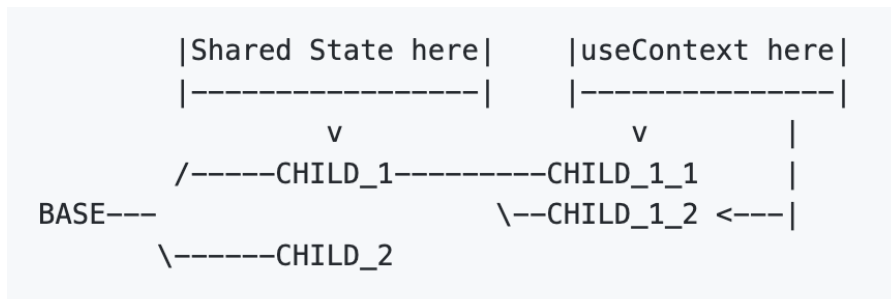


Рисунок 13 – Использование контекста

В данном случае, поскольку компонентов не так много и их вложенность небольшая, React своими силами и без всяких сторонних инструментов обеспечивает быстрый доступ к состояниям между компонентами, однако, как уже говорилось выше, на деле не все так просто.

Если, к примеру, необходимо расширить или добавить новую функцию в существующее приложение, причем для ее реализации из компонента CHILD_2 необходимо знать то самое общее состояние, о котором говорилось ранее, внутри компонента CHILD_1, то появляется проблема. Поскольку CHILD_2 не является потомком CHILD_1, то для того, чтобы CHILD_2 дать доступ к общему состоянию CHILD_1 необходимо переместить логику работы с состоянием в общий для CHILD_1 и CHILD_2 родительский компонент. В данном случае, им является компонент BASE. Тогда, используя хук useContext, можно, как и раньше дать доступ к общему состоянию для компонента CHILD_2.

С одной стороны, алгоритм простой, но в случае, если существует множество таких общих состояний и хранятся они не в одном компоненте, а нескольких, то процедура обеспечения доступа состояний к компонентам становится в разы сложнее. Главной проблемой такого подхода становится поиск общих состояний и того самого общего родителя.

В качестве решения проблемы, зная, что, в React все компоненты находятся внутри одного главного компонента Root, можно поместить все общие состояния внутрь него. Так, для доступа к этим состояниям достаточно применить хук `useContext`. Общие состояния внутри компонента Root можно реализовать через другой хук `useReducer` (данный хук позволяет создавать сложные, многосоставные состояния, с возможностью обеспечивать зависимость между ними). В итоге получается примерно такой код (рисунок 14).

```
const appReducer = (state, event) => {
  return {
    state1: reducer1(state.state1, event),
    state2: reducer2(state.state2, event),
  }
}

const AppStoreProvider = ({ children }) => {
  const [state, dispatch] = React.useReducer(appReducer, initialState);

  return(
    <AppDispatchContext.Provider value={dispatch}>
      <AppStateContext.Provider value={state}>
        {children}
      </AppStateContext.Provider>
    </AppDispatchContext.Provider>
  )
}
```

Рисунок 14 – Пример реализации общих состояний

Как можно увидеть данное решение мало чем отличается от имеющихся готовых библиотек управления состоянием, как к примеру Redux. В следствии чего не имеет смысла заново "собирать велосипед", данный механизм уже давно придуман и воплощен в современных стейт менеджерах и имеет подробную документацию и большую поддержку комьюнити. Конечно, всегда можно создавать собственные решения для управления состоянием на подобии вышеописанного, но зачастую это тем или иным образом сводится к уже готовому решению.

Другими готовыми решениями от React, как упоминалось ранее, являются хуки `useState` и `useReducer`. Первый предназначен для создания простых локальных (привязанных строго к одному компоненту) состояний, и не рассматривается как альтернатива Redux, второй, напротив имеет схожий с Redux механизм работы и используется для создания состояний со сложной логикой. Тем ни менее хук `useReducer` в настоящее время не способен полностью заменить Redux по ряду причин. Redux позволяет создать единое глобальное хранилище состояний, доступ к которому можно получить из любой части веб-приложения, когда как `useReducer` создает просто одно независимое хранилище состояния, доступ к которому можно получить только из компонента, привязанного к нему. Можно поднять все состояния `useReducer` до компонента самого верхнего уровня и, таким образом, получить глобальное хранилище, однако такое хранилище будет иметь два существенных отличия от решения Redux. Во-первых, в React нет встроенной нативной функции для объединения всех редьюсеров, ее необходимо реализовывать самому (в Redux эта функция называется `combineReducers`). Во-вторых, каждое состояние, созданное с помощью `useReducer` имеет собственную функцию отправки типа действия (диспетчер). Объединяющая опция для всех диспетчеров в React в настоящий момент также отсутствует. Redux же напротив предоставляет одну общую функцию диспетчеризации, которая использует любое действие, предназначенное для любой функции редьюсера. Таким образом, связка `useState` и `useReducer`, в настоящее время, годится лишь для небольших проектов, и не может в полной мере заменить библиотеки управления состояниями.

2.2.2 Библиотеки управления состояниями

Наиболее известными решениями в области управления состояниями являются библиотеки Redux, MobX и относительно новая, но быстро набирающая популярность библиотека Recoil.

Redux представляет собой одно из первых решений в сфере управления состояниями в компонентах [11, 27]. Основная его идея заключается в

создании для веб-приложения одного централизованного хранилища (SingleStore), в котором хранятся все состояния компонентов. Благодаря этому, каждый компонент может получить доступ к нужному состоянию, не передавая его из родительского к дочернему, как это реализовано в React Context.

Redux является иммутабельной библиотекой (immutable), что означает, что состояния внутри нее, представляются как неизменяемые объекты («снимки замороженных во времени объектов»). Вместо того, чтобы изменять текущее состояние, Redux сравнивает текущее, а также находящиеся внутри него состояния между собой, после чего в случае, если находит нужное выбирает его, иначе создает новое состояние. Новые состояния образуются путем передачи текущего состояния через чистые функции, называемые редьюсерами. Таким образом, если входное внутрь редьюсера состояние равно выходному, то состояние не меняется, если же есть хоть какая-то разница, генерируется новое. Именно благодаря тому, что редьюсеры являются чистыми функциями, сравнение входных и выходных состояний обходится без побочных эффектов. Данное свойство библиотеки может быть полезно в случае, когда, к примеру, есть какая-то таблица с множеством значений, и в одной из ячеек было изменено значение. В таком случае повторная визуализация (ререндеринг) таблицы, в ответ на ее изменение быстрее чем искать обход всего состояния с целью найти и изменить в нем ту самую ячейку.

Стоит также отметить, что Redux позволяет создавать более чем одно хранилище, однако, как говорят сами разработчики данной библиотеки, это не рекомендуется, поскольку это расходится с лучшими практиками использования Redux.

Следующей знаменитой в мире веб-разработки библиотекой управления состоянием является MobX [12, 28]. Механизм данного стейт менеджера в корне отличается от Redux. Здесь в отличие от одного центрального хранилища в Redux, состояния располагаются внутри нескольких хранилищ

(MultiStore). Это позволяет ускорить работу с состояниями в отдельных небольших блоках веб-приложения, однако, замедляет его при обновлении всего приложения целиком.

Также немаловажным отличием будет то, что если Redux делается упор на функциональное программирование и чистые функции, то MobX основан на реактивном программировании (программировании с асинхронными потоками данных). Поэтому MobX является мутабельной библиотекой (mutable). Если в Redux состояния компонентов доступны только для чтения и изменить их можно было только путем явных действий, то в MobX для состояний компонентов доступны как чтение, так и запись. Таким образом, состояние в MobX может как самостоятельно мутировать в ответ на действия пользователя, так и путем явных принудительных действий. Это говорит о том, что MobX ориентирован на точечное обновление компонентов, только в тех местах, где это необходимо, вместо полного обновления всего состояния веб-приложения, как это реализовано в Redux. Подобное поведение может быть полезно при использовании формы обратной связи, когда после изменения какого-то поля, разумнее изменить только состояние, связанное с ним, вместо полного ререндеринга формы.

Еще одним немаловажным отличием MobX от Redux является отсутствие шаблонности в коде и как следствие меньший объём строк кода. В Redux для работы с состоянием необходимо создавать аж четыре файла (action.js, actionTypes.js, reducer.js, store.js). В MobX достаточно создать всего один файл, в котором описать класс для взаимодействия с состояниями компонентов.

Замыкает тройку наиболее популярных в мире библиотек стейт менеджер Recoil [13, 29]. Данное ПО представляет собой новый взгляд компании-разработчика React на управление состояниями в компонентах. Recoil – это попытка упростить синтаксис Redux, поменяв логику хранения и уменьшив размер необходимого кода. В связи с чем, Recoil имеет некоторые сходства с Redux в плане использования чистых функций и общего хранилища

состояний. Несмотря на то, что данная библиотека относительно новая, она быстро набирает популярность, в виду того, что сами разработчики React все чаще рекомендуют его в качестве решения в управлении состояний компонентов.

Методология библиотеки Recoil заключается в том, чтобы обозначить логику взаимодействия с состояниями компонентов через атомы и селекторы. Атом представляет собой часть состояния. Атомы обновляемы и на них можно подписывать компоненты. При обновлении атома, все компоненты, подписанные на него, также будут обновлены в соответствии с изменениями. Атомы могут использоваться сразу несколькими компонентами. Селекторы являются чистыми функциями, которые могут принимать в себя как атомы, так и другие селекторы (редьюсеры в Redux реализуют нечто похожее). Они используются для того, чтобы вычислять производные данные на основе состояния компонента. Это помогает избавиться от избыточности состояния, так как в атомах хранится лишь минимальная часть состояния. Остальное, все что можно вычислить, вычисляется отдельно внутри селекторов. Так как селекторы отслеживают подписанные на них компоненты, данный подход получается очень эффективным. В Recoil, как и в Redux имеется одно хранилище называемое RecoilStore, именно в нем располагаются атомы и селекторы. В целом Recoil позволяет пересмотреть логику работы Redux и перейти от сложного к простому, уменьшив количество кода и сохранив иммутабельность и единое хранилище от Redux. Однако, в настоящее время, Recoil врядли сможет полноценно заменить Redux, поскольку проект является новичком в данной области, у него не большая поддержка комьюнити и примеров практического использования не так много по отношению к тому же Redux, а также часть функций, судя по официальному сайту, все еще находится в разработке [30]. Тем ни менее, если текущего функционала и качества документации Recoil хватает, сами разработчики React рекомендуют обратить внимание на данное решение. К тому же, не стоит забывать, что

проект активно развивается и возможно когда-нибудь сможет покрыть весь функционал Redux.

Стоит отметить, что Recoil, в отличие от двух предшественников, позиционирует себя исключительно как решение для фреймворка React и не может быть использовано в других SPA-фреймворках.

3 ПРОВЕДЕНИЕ ЭКСПЕРИМЕНТА ДЛЯ СРАВНЕНИЯ БИБЛИОТЕК УПРАВЛЕНИЯ СОСТОЯНИЕМ

3.1 Сравнение характеристик менеджеров управления событиями

Для того, чтобы понять какой менеджер управления событиями лучше подойдет для того или иного случая, необходимо разобрать их общие характеристики. Ими является качество документации, популярность, размер, возможность масштабирования.

3.1.1 Документация

Качество документации напрямую влияет на то, как быстро разработчик сможет освоить ПО и использовать полученные знания в реальной разработке. Документация должна быть не перегружена лишней информацией, полностью пояснять функционал, кратко и понятно раскрывать механизм работы ПО, чтобы потенциальный разработчик мог с легкостью начать использовать его.

Redux, в современном своем представлении, является не просто одной библиотекой, это целая экосистема, состоящая из нескольких библиотек, не только для управления состоянием. В связи с чем документация Redux очень расширена, она состоит из нескольких частей, и для полноценного использования всей экосистемы Redux необходимо освоить их все хотя бы в базовом понимании [11]. Это усложняет обучение для начинающего разработчика.

MobX не имеет своей экосистемы, подобной Redux. Данная библиотека заточена только под управление состоянием компонентов, что делает ее документацию намного меньше чем в Redux. Однако это не означает, что ее функционал урезан по сравнению с Redux в направлении управления состоянием, данная библиотека вполне может составить достойную конкуренцию Redux. Документация MobX коротка и понятна, что делает ее изучение более быстрым в сравнении с Redux [12].

Recoil, в настоящее время, только начинает набирать обороты в мире веб-разработки. Как и MobX, Recoil разработан в первую очередь для управления состоянием, поэтому размер его документации не сильно

отличается от размера документации MobX [13]. Тем ни менее, на момент написания данного текста полноценного релиза Recoil еще не было (актуальная на текущий момент версия Recoil – 0.7.6), поэтому документацию Recoil в текущем ее виде нельзя назвать полноценной, несмотря на то, как активно продвигает данную библиотеку компания-создатель React [30].

3.1.2 Популярность

Несмотря на то, что популярность довольно субъективный фактор, от нее напрямую зависит продолжительность и качество выпускаемого продукта. Чем известнее библиотека, тем выше вероятность того, что она будет активно развиваться и поддерживаться. К тому же, популярность влияет и на количество комьюнити, которое использует библиотеку, что способствует нахождению багов, поддержке, дальнейшему продвижению.

Согласно общемировому ресурсу npm trends на первом месте по популярности, на правах одного из первых менеджеров управления состояний находится Redux [33-34]. Следом за ним с существенно низкой по сравнению с Redux популярностью располагает библиотека MobX, после которой с довольно низкими показателями находится Recoil.

Информация о популярности за все время использования в виде графика согласно ресурсу npm trends представлена на рисунке 15 (по вертикале располагается число загрузок через npm-менеджер, а по горизонтали год использования).

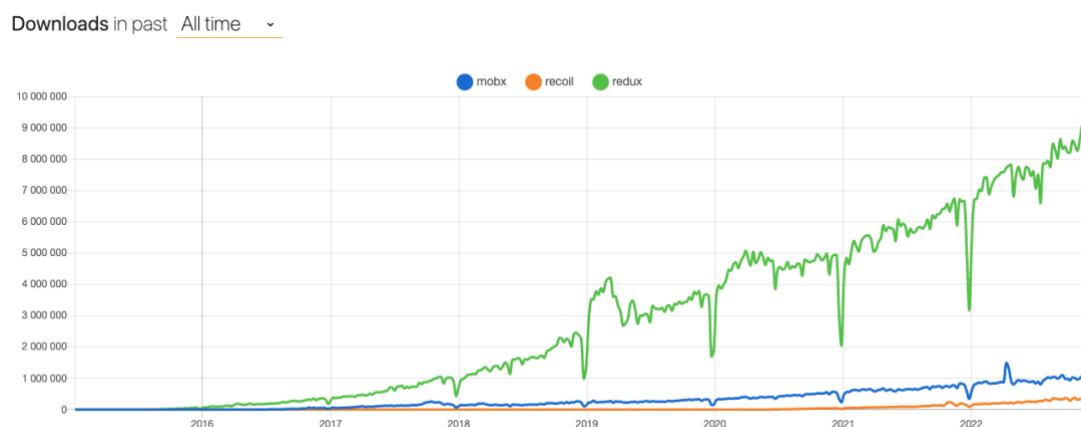


Рисунок 15 – График популярности библиотек

Кроме того, на сайте представлена сводка данных по библиотекам согласно известному веб-сервису Github (рисунок 16) [33-34].

Stats












			Stars	Issues	Version	Updated [?]	Created [?]	Size
	mobx	  	25 929	27	6.7.0	20 days ago	7 years ago	minzipped size 16.4 KB
	recoil	 	18 045	210	0.7.6	2 months ago	3 years ago	minzipped size 23.4 KB
	redux	  	58 948	58	4.2.0	8 months ago	11 years ago	minzipped size 1.6 KB

Рисунок 16 – Сводка данных по библиотекам согласно ресурсу Github

Можно заметить, что наибольшее количество нерешенных проблем имеет еще не вышедшая в полноценный релиз библиотека Recoil (210 активных проблем), что сильно отражается на ее популярности.

3.1.3 Размер

Размер используемой библиотеки является важным параметром при разработке любого проекта. Чем больше она по размеру, тем дольше будет первая загрузка сайта, что плохо, поскольку первичное впечатление от сайта зачастую становится решающим фактором в том, чтобы остаться на нем или покинуть его.

Так как версии и размер библиотек различаются по своему функционалу и требованиям к проекту, имеет смысл рассмотреть базовые конфигурации. Информация о размере минимальных базовых версий библиотек согласно ресурсу NPM trends:

- 16.4 кб mobx,
- 23.4 кб recoil,
- 1.6 кб redux [33, 34].

Как можно увидеть, библиотека redux в базовой версии имеет размер в 1.6 кб, что существенно меньше, в сравнении с выбранными конкурентами. Несмотря на кажущуюся незначительность размеров каждой из библиотек на деле даже 1 кб памяти может заметно отразиться на времени первого

рендеринга веб-сайта, а учитывая, что зачастую библиотек используется несколько размер становится одним из определяющих факторов.

3.1.4 Масштабируемость

Еще одним значимым параметром при выборе библиотеки управления состоянием является возможность дальнейшего масштабирования, так как это напрямую влияет на то, как быстро разработчики смогут перестроить свое веб-приложение под новый функционал.

Redux отлично подходит для масштабирования. Это обуславливается тем, что основной механизм работы с состояниями в данной библиотеке – это чистые функции [35-36]. Чистая функция всегда возвращает одинаковый результат при том же наборе аргументов в ней. Это делает состояние в Redux неизменным (немутабельным) и предсказуемым. Таким образом, вместо изменения состояния (мутации состояния), Redux возвращает новое состояние. Всякий раз, когда необходимо обновить хранилище состояний, отправляется экшен (action), который передается редюсеру (reducer) и в зависимости от типа этого экшена редюсер обновляет состояние неизменным образом. Это упрощает добавление нового функционала и вносит ясность при разработке. Кроме того, Redux хранит все состояния в одном глобальном хранилище, что позволяет в случае необходимости быстро обращаться к состояниям независимо от размера проекта [37]. Redux позволяет размещать состояния в несколько хранилищ, однако сами разработчики этого не рекомендуют поскольку это расходится с оптимальной работой данной библиотеки [38].

В MobX состояние изменчиво (мутабельно) [39]. Это происходит за счет того, что в основе его работы лежит объектно-ориентированное (использование классов и объектов) и реактивное программирование (изменение одного потока данных влечет автоматическое изменение другого) [38-39]. Все состояния находятся внутри наблюдаемых объектов. Компоненты подписываются на эти наблюдаемые объекты для доступа к состояниям и вычисляемым значениям. Наблюдаемые объекты могут быть построены на

основе других классов-хранилищ состояний путем объектно-ориентированного наследования. Из-за чего возможна огромная вложенность таких классов. Возникает риск ошибки, из-за того, что один из таких классов может быть изменен. Кроме того, состояние можно менять, напрямую обратившись к полю наблюдаемого объекта в любой части проекта. Такое приложение сложнее тестировать и поддерживать, поскольку оно не всегда возвращает предсказуемый результат. Это значительно усложняет масштабирование проекта, особенно, если над проектом работала одна команда, а после необходимости дополнить функционал за него взялась другая команда.

В Recoil хранение состояния основано на атомах и селекторах [40-41]. Это напоминает упрощенный механизм работы экшенов и редюсеров в Redux. Подобно редюсерам, селекторы являются чистыми функциями. Благодаря этому состояние в Recoil, как и в Redux неизменно (немутабельно). Это дает recoil те же преимущества в плане масштабирования что и у Redux. Кроме того, Recoil поддерживает параллельную обработку запросов к изменению состояний, что безусловно положительно складывается при масштабировании веб-приложения.

3.2 Тестирование времени рендеринга в одинаковых условиях

3.2.1 Условия эксперимента

В качестве эксперимента предлагается протестировать время, за которое компонент прорендерится 100 000 раз. Условия рендеринга для чистоты эксперимента будут одинаковыми, каждая библиотека будет работать с одним и тем же приложением, с одним и тем же вызовом. Возможность оптимизации в расчет не берется, поскольку главная цель эксперимента сравнить стандартные возможности менеджеров управления событий.

Приложение представляет собой своеобразный счетчик баланса, в котором можно добавить, убавить или полностью обнулить баланс, кроме того, имеется возможность указать количество прогонов (сколько раз необходимо выполнить вычисления и рендеринг компонента) (рисунок 17).

Баланс: 10001200 руб

Добавить Убавить Очистить

Введите количество прогонов

Запуск теста производительности

Тест: **Добавить к балансу(+100руб) 100000 раз**

Результат: 101 мс

Рисунок 17 – Приложение для эксперимента

Для того, чтобы не дублировать несколько раз код замера под каждую операцию, кнопка запуска теста производительности выполняет только операцию добавления. Процесс тестирования состоит в том, что операция добавления выполняется 100 000 раз, а хранение и обработка состояния компонента отрисовки баланса полностью ложится на плечи вышеупомянутых менеджеров управления состояний.

3.2.2 Замер времени рендеринга

Замер времени рендеринга проводится с помощью встроенного в язык JavaScript (а также в TypeScript) интерфейса Performance (рисунок 18-20) [42-43].

```
const PerfTest: React.FC<PerfTestProps> = ({ add }) => {
  const [result, setResult] = React.useState<string>('-');
  const [isDisabled, setIsDisabled] = React.useState<boolean>(false);

  const [count, setCount] = React.useState<number>(100_000);

  function runPerfTest() {
    setIsDisabled(true);
    const timeStart = performance.now();

    timerId = window.setInterval(() => {
      for (let i = 0; i <= count; i++) {
        add(100);
      }
    });

    window.clearInterval(timerId);
    const timeEnd = performance.now();
    setResult(` ${Math.round(timeEnd - timeStart)} мс`);
    setIsDisabled(false);
  };
}
```

Рисунок 18 – Тестирование времени рендеринга компонента для Redux

```
const PerfTest: React.FC<PerfTestProps> = observer(({ balanceState }) => {
  const [result, setResult] = React.useState<string>('-');
  const [isDisabled, setIsDisabled] = React.useState<boolean>(false);

  const [count, setCount] = React.useState<number>(100_000);

  function runPerfTest() {
    setIsDisabled(true);
    const timeStart = performance.now();

    timerId = window.setInterval(() => {
      for (let i = 0; i <= count; i++) {
        balanceState.add(100);
      }

      window.clearInterval(timerId);
      const timeEnd = performance.now();
      setResult(` ${Math.round(timeEnd - timeStart)} мс`);
      setIsDisabled(false);
    });
  }
});
```

Рисунок 19 – Тестирование времени рендеринга компонента для MobX

```
const PerfTest: React.FC<PerfTestProps> = () => {
  const [result, setResult] = React.useState<string>('-');
  const [isDisabled, setIsDisabled] = React.useState<boolean>(false);

  const [count, setCount] = useRecoilState(CountState);
  let [balance, setBalance] = useRecoilState(BalanceState);

  function runPerfTest() {
    setIsDisabled(true);
    const timeStart = performance.now();

    timerId = window.setInterval(() => {
      for (let i = 0; i <= count; i++) {
        balance += 100;
        setBalance(balance);
      }

      window.clearInterval(timerId);
      const timeEnd = performance.now();
      setResult(` ${Math.round(timeEnd - timeStart)} мс`);
      setIsDisabled(false);
    });
  }
};
```

Рисунок 20 – Тестирование времени рендеринга компонента для Recoil

Функция взаимодействия с состоянием вызывается одинаково внутри цикла для каждого из трех менеджеров управления состояний. Это позволяет

определить стандартные возможности библиотек, без какой-либо оптимизации.

3.2.3 Управление состояниями компонентов в Redux

Как упоминалось ранее, для управления состояниями компонентов в Redux используется связка actions, reducers, store.

Код для actions представлен на рисунке 21.

```
import {
  BalanceActionType,
  AddAction,
  WithdrawAction,
  ClearAction,
} from './types';

export const add = (amount: number): AddAction => ({
  type: BalanceActionType.ADD,
  payload: amount,
});

export const withdraw = (amount: number): WithdrawAction => ({
  type: BalanceActionType.WITHDRAW,
  payload: amount,
});

export const clear = (amount: number): ClearAction => ({
  type: BalanceActionType.CLEAR,
  payload: amount,
});
```

Рисунок 21 – Код для actions

Определение типов для удобства было вынесено в отдельный файл (рисунок 22).

```

import { Action } from 'redux'; 410 (gzipped: 263) You, 5 месяцев назад

You, 5 месяцев назад | 1 author (You)
export interface BalanceState {
  balance: number;
}

export enum BalanceActionType {
  ADD = 'ADD',
  WITHDRAW = 'WITHDRAW',
  CLEAR = 'CLEAR',
}

You, 5 месяцев назад | 1 author (You)
export interface AddAction extends Action {
  type: BalanceActionType.ADD;
  payload: number;
}

You, 5 месяцев назад | 1 author (You)
export interface WithdrawAction extends Action {
  type: BalanceActionType.WITHDRAW;
  payload: number;
}

You, 5 месяцев назад | 1 author (You)
export interface ClearAction extends Action {
  type: BalanceActionType.CLEAR;
  payload: number;
}

export type BalanceAction = AddAction | WithdrawAction | ClearAction;

```

Рисунок 22 – Определение типов

Код для reducers представлен на рисунке 23.

```

import { BalanceAction, BalanceActionType, BalanceState } from './types';

const initialState: BalanceState = {
  balance: 1000,
};

export const balanceReducer = (
  state: BalanceState = initialState,
  action: BalanceAction
): BalanceState => {
  switch (action.type) {
    case BalanceActionType.ADD:
      return { ...state, balance: state.balance + action.payload };
    case BalanceActionType.WITHDRAW:
      return { ...state, balance: state.balance - action.payload };
    case BalanceActionType.CLEAR:
      return { ...state, balance: state.balance - action.payload };
    default:
      return state;
  }
};

```

Рисунок 23 – Код для reducers

Код для store представлен на рисунке 24.

```
import { combineReducers, createStore } from 'redux'; 2.7k (gzipped: 1.2k)
import { balanceReducer } from './reducers';
import { BalanceState } from './types';

const rootReducer = combineReducers({
  balance: balanceReducer,
});

You, 5 месяцев назад | 1 author (You)
export interface State {
  balance: BalanceState;
}

export const store = createStore(rootReducer, undefined);
```

Рисунок 24 – Код для store

3.2.4 Управление состояниями компонентов в MobX

Управление состояниями компонентов в MobX не требует столь много кода и уместается в один файл (рисунок 25).

```
import { makeAutoObservable } from 'mobx'; 50.3k

You, 5 месяцев назад | 1 author (You)
export class BalanceState {
  balance = 1000;

  constructor() {
    makeAutoObservable(this);
  }

  add(value: number) {
    this.balance = this.balance + value;
  }

  withdraw(value: number) {
    this.balance = this.balance - value;
  }

  clear() {
    this.balance = 0;
  }
}
```

Рисунок 25 – Класс для управления состоянием в MobX

3.2.5 Управление состояниями компонентов в Recoil

В Recoil для организации состояния компонентов используется структура атомов и селекторов.

Код атома для хранения состояния с текущим балансом представлен на рисунке 26.

```
import { atom } from 'recoil'; 77.2

const BalanceState = atom<number>({
  key: 'BalanceState',
  default: 1000,
});

export default BalanceState;
```

Рисунок 26 – Атом для хранения состояния текущего баланса

Код атома для хранения состояния о количествах прогонов представлен на рисунке 27.

```
import { atom } from 'recoil'; 77.2

const CountState = atom({
  key: 'CountState',
  default: 100_000,
});

export default CountState;
```

Рисунок 27 – Атом для хранения состояния о количестве прогонов

Код атома для хранения промежуточного значения для вычислений представлен на рисунке 28.

```
import { atom } from 'recoil'; 77.2

const ValueState = atom({
  key: 'ValueState',
  default: 100,
});

export default ValueState;
```

Рисунок 28 – Атом для хранения промежуточного значения

Код атома для хранения времени замеров представлен на рисунке 29.

```
import { atom } from 'recoil'; 77.2

const Result = atom({
  key: 'Result',
  default: 0,
});

export default Result;
```

Рисунок 29 – Атом для хранения времени замеров

Код для селектора сложения представлен на рисунке 30.

```
import { selector } from 'recoil'; 77.2k (gzipped)
import ValueState from '../atoms/ValueState';
import BalanceState from '../atoms/BalanceState';

const Add = selector({
  key: 'Add',
  get: ({ get }) => {
    let value = get(ValueState);
    let balance = get(BalanceState);
    return (balance += value);
  },
});

export default Add;
```

Рисунок 30 – Селектор сложения

Код для селектора вычитания представлен на рисунке 31.

```
import { selector } from 'recoil'; 77.2k (gzipped)
import ValueState from '../atoms/ValueState';
import BalanceState from '../atoms/BalanceState';

const Withdraw = selector({
  key: 'Withdraw',
  get: ({ get }) => {
    let value = get(ValueState);
    let balance = get(BalanceState);
    return (balance = balance - value);
  },
});

export default Withdraw;
```

Рисунок 31 – Селектор вычитания

Код для селектора очистки поля представлен на рисунке 32.

```
import { selector } from 'recoil'; 77.2k (gzipped)
import BalanceState from '../atoms/BalanceState';

const Clear = selector({
  key: 'Clear',
  get: ({ get }) => {
    let balance = get(BalanceState);
    return (balance = 0);
  },
});

export default Clear;
```

Рисунок 32 – Селектор очистки поля

Код для селектора тестирования представлен на рисунке 33.

```
import { selector } from 'recoil'; 77.2k (gzipped)
import BalanceState from '../atoms/BalanceState';

const Test = selector({
  key: 'AddTest',
  get: ({ get }) => {
    let balance = get(BalanceState);
    balance += 100;

    return balance;
  },
});

export default Test;
```

Рисунок 33 – Селектор тестирования

3.3 Результаты эксперимента

Результаты замеров для Redux при 100 000 рендерингов компонента представлены в таблице 1.

Таблица 1 – Результаты замеров для Redux при 100 000 рендерингов

100 000 рендерингов (мс)										
Десятки	Единицы									
	1	2	3	4	5	6	7	8	9	10
10	121	92	90	93	90	101	94	90	95	93
20	103	91	93	89	95	94	101	93	102	93
30	94	92	93	108	94	93	91	101	96	92

Среднее значение равно 95,56 мс.

Результаты замеров для MobX при 100 000 рендерингов компонента представлены в таблице 2.

Таблица 2 – Результаты замеров для MobX при 100 000 рендерингов

100 000 рендерингов (мс)										
Десятки	Единицы									
	1	2	3	4	5	6	7	8	9	10
10	51	41	40	36	40	37	36	48	38	35
20	40	35	37	36	45	39	43	35	40	37
30	38	35	36	38	40	48	35	36	37	35

Среднее значение равно 38,9 мс. Что при стандартных настройках почти в два раза быстрее по сравнению с Redux.

Результаты замеров для Recoil при 100 000 рендерингов компонента представлены в таблице 3.

Таблица 3 – Результаты замеров для Recoil при 100 000 рендерингов

100 000 рендерингов (мс)										
Десятки	Единицы									
	1	2	3	4	5	6	7	8	9	10
10	2295	2322	1800	1856	2500	1880	1861	2686	1979	1973
20	3067	1928	2005	1968	2274	1989	2007	2025	1683	1684
30	1736	1781	1816	1725	1710	1730	2232	1728	1718	1726

Среднее значение равно 1989,46 мс. Это значительно дольше чем у Redux и MobX вместе взятых. Эксперимент показывает, что при обработке состояния внутри цикла стандартными инструментами Recoil (хуком `useRecoilState` с последующим использованием функции `set`, именно такой метод по умолчанию рекомендуется в документации), при текущей версии библиотеки, Recoil показывает далеко не самые оптимальные показатели [44]. Особенно, если необходимо множественное выполнение цикла, в данном случае рендеринг выполняется крайне медленно. Не исключено, что путем оптимизации и использованием дополнительных сторонних библиотек можно добиться более быстрых результатов, однако в ситуации здесь и сейчас при

использовании инструментария по умолчанию библиотека работает с циклами крайне плохо в отличии от своих оппонентов.

Результаты данного эксперимента показывают, что MobX быстрее своих оппонентов в равных условиях, при настройке «по умолчанию», однако равнозначно говорить о том, что MobX может полностью заменить обе библиотеки нельзя. У них совершенно разный подход к хранению состояния, разное отношение к мутабельности состояния, к тому же нельзя исключить возможность дополнительной оптимизации. Выбор менеджера управления состояний должен обуславливаться прежде всего требованиями к проекту и личным предпочтениям разработчиков. И результаты данной работы должны помочь с этим выбором.

4 РАЗРАБОТКА РЕКОМЕНДАТЕЛЬНОГО ВЕБ-САЙТА

4.1 Требования к веб-сайту с рекомендациями по state-менеджерам

В ходе проведенных в данной работе исследования и эксперимента, было решено в качестве практической базы реализовать веб-сайт с рекомендациями по трем вышеописанным стейт-менеджерам.

Основное назначение рекомендательного веб-сайта по выбору менеджера состояний компонентов – дать пользователю подробную информацию о том, когда лучше использовать тот или иной менеджер управления состояний, показать их ключевые атрибуты, а также сильные и слабые стороны. Поскольку в настоящее время существует довольно большое количество вариантов ПО в области state-менеджеров, было решено выбрать три наиболее популярных и перспективных библиотеки: Redux, MobX и Recoil.

Оценка зарубежной и отечественной научной литературы дала понять, что на текущий момент, в мире науки четкое представление о работе библиотек управления состояниями компонентов имеется разве что у библиотеки Redux, в виду того, что она на данный момент является наиболее старой и популярной в мире веб-разработки. О других известных решениях таких как MobX и Recoil в зарубежной науке известно крайне мало, а в

отечественной в принципе практически ничего нет. И это несмотря на то, что на практике и в производстве данные библиотеки широко находят свое применение. Более того, зачастую разработчики предпочитают отойти от традиционного Redux к более новым решениям.

При выборе стейт-менеджера важно понимать основной механизм работы, их методологию, а также знать их достоинства и недостатки. В этом им поможет рекомендательный веб-сайт. Поскольку точного ответа на то, подходит ли библиотека всем или нет дать нельзя, так как ситуаций и проблем в управлении состояний бывает крайне много, такой сайт должен помочь начинающему веб-разработчику в понимании разницы между библиотеками и склонить его к выбору той или иной в зависимости от потребностей.

Предполагается, что веб-приложение будет иметь:

- краткое описание принципа работы трех наиболее востребованных библиотек управления состоянием, для более четкого понимания (Redux, MobX, Recoil),
- информацию относительно популярности библиотек (согласно источнику Github и Google Trends),
- общие рекомендации по выбору библиотеки управления состоянием (Redux, MobX, Recoil),
- возможность сравнения их производительности,
- информация об общем весе веб-приложения с использованием той или иной библиотеки,
- примеры кода для сравнения синтаксиса,
- ссылки на официальные документации библиотек для дальнейшего изучения.

4.2 Проектирование будущего интерфейса

Разработка вайрфреймов (wireframes) является важной составляющей при проектировании интерфейса информационной системы [45-46]. Макет позволяет еще до этапа разработки увидеть, как будет приблизительно

выглядеть конечный продукт. В нем не будет реализовано функционала, однако с помощью него можно увидеть, что именно задумано в конечной версии продукта и что предстоит реализовать на этапе разработки.

Для разработки вайрфреймов будущего интерфейса веб-сайта было решено использовать современный онлайн-инструмент для проектирования и прототипирования приложений Figma [47-48].

Figma представляет собой бесплатный для одиночного пользователя онлайн-сервис, с помощью которого можно быстро создавать вайрфреймы, макеты и прототипы для различных информационных продуктов [49-50].

Скриншоты вайрфреймов, разработанных в Figma, можно увидеть на рисунках 34-36.

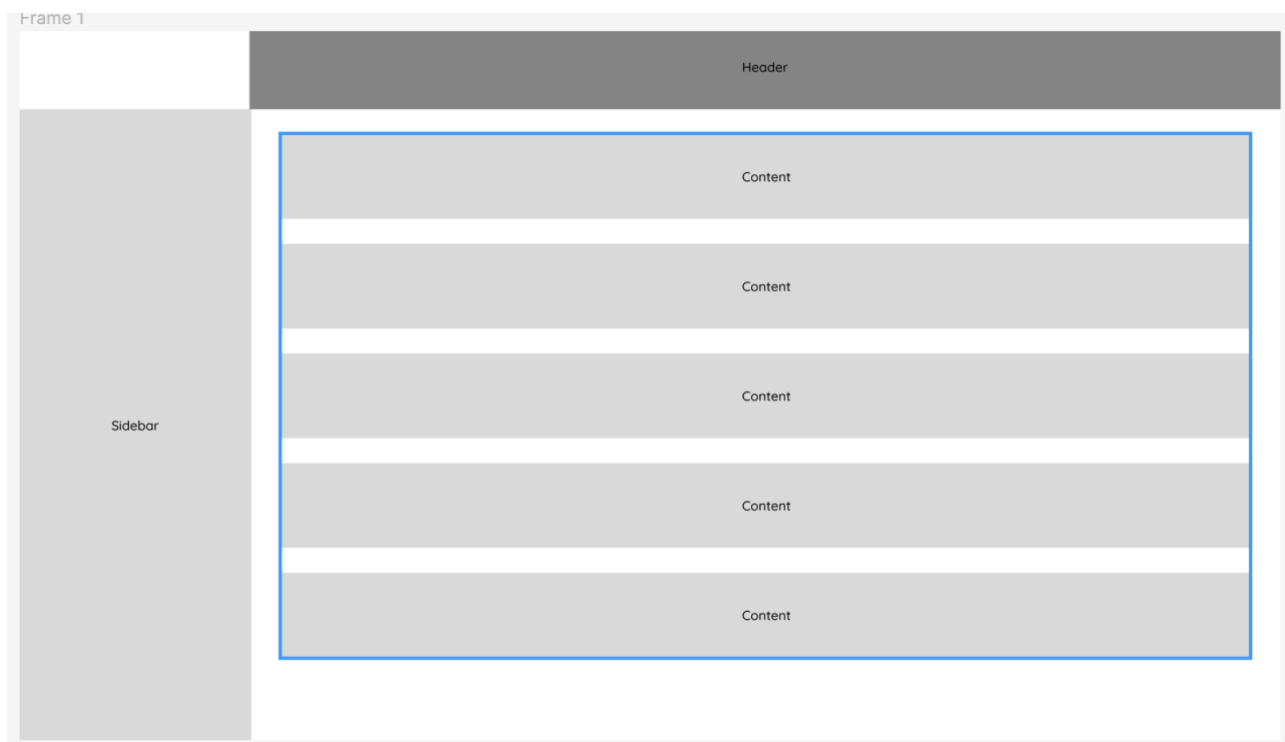


Рисунок 34 – Вайрфрейм версии сайта для ПК

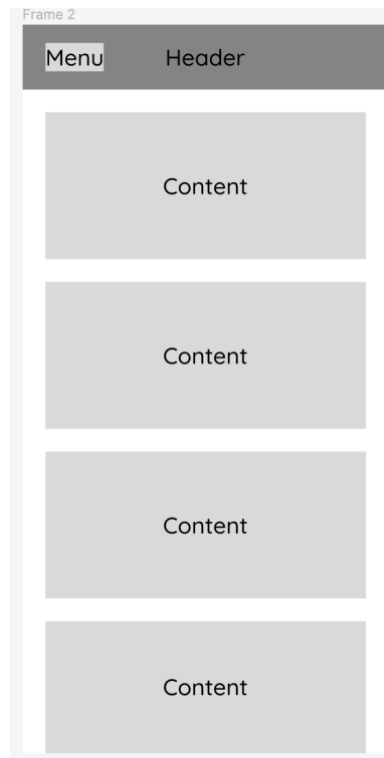


Рисунок 35 – Вайрфрейм мобильной версии сайта

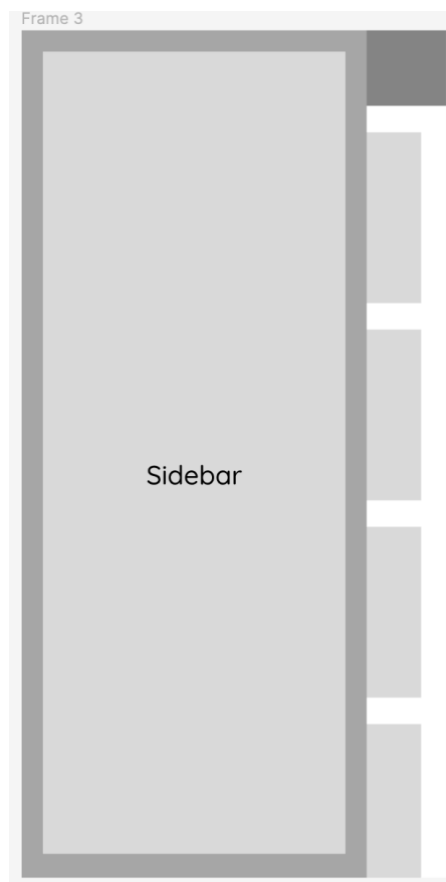


Рисунок 36 – Вайрфрейм мобильной версии сайта с раскрытым меню

4.3 Разработка проекта

4.3.1 Инициализация

В качестве SPA-фреймворка в соответствии с темой исследования используется фреймворк React [18]. Как уже говорилось ранее, на текущий момент он занимает лидирующие позиции в области фронтенд SPA-разработки и является наиболее актуальным и востребованным. React имеет огромную базу разработчиков и качественную документацию, что позволяет проще и быстрее найти решение возникших в ходе работы проблем.

Для того, чтобы отслеживать ошибки типов на этапе компиляции, а не во время выполнения кода вместо стандартного JavaScript, применяется типизированная надстройка TypeScript [51].

Для инициализации проекта используется пакетный менеджер yarn [52]. С помощью yarn через стандартный пакет инициализации проекта React в связке с TypeScript – Create-React-App. С помощью консольной команды «yarn create react-app my-app --template typescript» создается готовый для разработки шаблон с настроенным линтером для языка TypeScript. Дальнейшие добавления сторонних библиотек выполняется через «yarn add «имя библиотеки»».

4.3.2 Стилизация

Для стилизации компонентов решено выбрать библиотеку Styled-component [54]. Styled-component позволяет прямо внутри tsx файлов на языке TypeScript (и JavaScript) писать стили компонентов. Благодаря этому, классы стилей можно использовать как обычные TypeScript переменные, импортировать их, несколько раз использовать. Styled-component использует ограниченную область видимости, что означает, что стили действуют только внутри тех компонентов, внутри которых они определены или импортированы. Это помогает предотвратить конфликты стилей и упрощает поддержку кода. Пример использования Styled-component в текущем проекте представлен на рисунке 37.


```

import { FC, useState } from 'react'; 4.1k (gzipped: 1.8k)
import styled from 'styled-components'; 33k (gzipped: 12.9
import { SidebarItem } from './SidebarItem';
import { HashLink as Link } from 'react-router-hash-link';

type SidebarLinkProps = {
  item: SidebarItem;
};

const SidebarLink = styled(Link)`
  display: flex;
  justify-content: space-between;
  align-items: center;
  height: 3.75rem;
  font-size: 1.125rem;
  padding: 2rem;
  text-decoration: none;
  color: black;

  &:hover {
    background-color: #d9d9d9;
  }
`;

```

Рисунок 37 – Пример использования Styled-component

Для использования Styled-component достаточно импортировать в проект метод styled, после чего поместить в него либо стандартный HTML-тег или тег кастомного компонента.

Кроме того, для использования готовых компонентов подключена библиотека Material UI [55]. Компоненты данной библиотеки выполнены в одноименном стиле Material Design, активно продвигаемом компанией Google [56]. Material UI значительно ускоряет процесс разработки компонентов, предоставляя разработчику готовые шаблоны компонентов, после чего остается лишь подстроить шаблон под свои нужды. Пример использования Material UI представлен на рисунке 38.

```

import Drawer from '@mui/material/Drawer'; 53.8k (gzipped: 18.4k)
import Box from '@mui/material/Box'; 23.3k (gzipped: 8.7k)
import Toolbar from '@mui/material/Toolbar'; 26.8k (gzipped: 9.9k)
import Divider from '@mui/material/Divider'; 29.5k (gzipped: 10.7k)

import React from 'react'; 6.9k (gzipped: 2.7k)

import leftsidemenu from '../store/leftsidemenu';
import Sidebar from '../components/Sidebar/Sidebar';
import { SidebarData } from '../components/Sidebar/SidebarData';
import { observer } from 'mobx-react-lite'; 5k (gzipped: 2k)

const drawerWidth = 240;

const LeftSideMenu: React.FC = observer(() => {
  return (
    <div>
      <Box
        component='nav'
        sx={{ width: { sm: drawerWidth }, flexShrink: { sm: 0 } }}
        aria-label='mailbox folders'>

```

Рисунок 38 – Использование готовых компонентов Material UI

4.3.3 Маршрутизация

Маршрутизацию сайта обеспечивает библиотека React Router, что позволяет сделать веб-сайт четко структурированным и логичным [57]. React Router позволяет воплотить идеи многостраничного подхода в SPA решении. Он обеспечивает навигацию между представлениями различных компонентов в приложении React, позволяет изменять URL-адрес браузера и использовать журнал браузера для осуществления переходов, поддерживает синхронизацию пользовательского интерфейса с URL-адресом.

Так как данный проект не имеет бекенд части, согласно официальной документации React Router тип маршрутизации следует использовать HashRouter [58]. HashRouter использует хеш-часть URL-адреса для отслеживания текущего маршрута. Хеш-часть URL-адреса – это часть URL-адреса после символа #, и она не отправляется на сервер, поэтому она не влияет на отрисовку вашего приложения на стороне сервера. HashRouter

обычно используется для приложений, которым не требуется поддержка рендеринга на стороне сервера или которые работают в среде, где сервер не может обрабатывать маршрутизацию. Код для обертки приложения внутри HashRouter представлен на рисунке 39.

```
import React from 'react'; 6.9k (gzipped: 2.7k)
import ReactDOM from 'react-dom/client'; 513 (gzipped: 319)
import './index.sass';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { HashRouter } from 'react-router-dom'; 4.6k (gzipped: 2.1k)

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <HashRouter basename="/">
      <App />
    </HashRouter>
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

Рисунок 39 – Обертка веб-приложения в HashRouter

Доступные с помощью React Router URL для данного проекта представлены на рисунке 40.

```
<Routes>
  <Route path="/" element={<Layout />}>
    <Route index element={<HomePage />}></Route>
    <Route path="/about" element={<AboutPage />}></Route>
    <Route path="/redux" element={<ReduxPage />}></Route>
    <Route path="/mobx" element={<MobXPage />}></Route>
    <Route path="/recoil" element={<RecoilPage />}></Route>
    <Route path="*" element={<NoFoundPage />}></Route>
  </Route>
</Routes>
```

Рисунок 40 – Маршрутизация сайта

Навигация по сайту осуществляется при помощи адаптивного бокового меню, превращающегося в меню бургер при просмотре на мобильных устройствах. Полный код бокового меню представлен на рисунке 41.

```
const Submenu: FC<SidebarLinkProps> = ({ item }) => {
  const [subnav, setSubnav] = useState(false);
  const showSubnav = () => setSubnav(!subnav);

  return (
    <>
      <SidebarLink to={item.path} onClick={showSubnav}>
        <div>
          <SidebarLabel>{item.title}</SidebarLabel>
        </div>
        <div>
          {item?.subnav && subnav ? item?.iconOpened : item?.iconClosed}
        </div>
      </SidebarLink>
      <div>
        {subnav &&
          item?.subnav?.map((subnavItem, index) => {
            return (
              <DropdownLink to={subnavItem.path} key={index}>
                <SidebarLabel>{subnavItem.title}</SidebarLabel>
              </DropdownLink>
            );
          })}
      </div>
    </>
  );
};

export default Submenu;
```

Рисунок 41 – Код для бокового меню

Информация по каждой библиотеке управления состоянием компонентов разделена на 6 разделов. Ими являются: общая информация о библиотеке, информация о документации, популярности, размеру, масштабируемости, а также эксперимент, где можно поиграться с количеством прогонов и посмотреть сколько времени займет рендеринг.

Боковое меню помимо основных URL содержит в себе элементы навигации к тем самым разделам по каждой библиотеке с доступом по якорным ссылкам. Полная информация по всем имеющимся URL хранится в формате JSON и представлена на рисунках 42-45 [59].

```
export const SidebarData: SidebarItem[] = [
  {
    title: 'Главная',
    path: '/#home',
  },
  {
    title: '0 проекте',
    path: '/about',
  },
]
```

Рисунок 42 – Информация о доступных URL (часть 1)

```
{
  title: 'Redux',
  path: '/redux#redux',
  iconClosed: <AiFillCaretDown />,
  iconOpened: <AiFillCaretUp />,
  subnav: [
    {
      title: '0 Redux',
      path: '/redux#redux-description',
    },
    {
      title: 'Документация',
      path: '/redux#redux-documentation',
    },
    {
      title: 'Популярность',
      path: '/redux#redux-popularity',
    },
    {
      title: 'Размер',
      path: '/redux#redux-size',
    },
    {
      title: 'Масштабируемость',
      path: '/redux#redux-scalability',
    },
    {
      title: 'Эксперимент',
      path: '/redux#redux-experiment',
    },
  ],
},
```

Рисунок 43 – Информация о доступных URL (часть 2)

```

{
  title: 'MobX',
  path: '/mobx#mobx',
  iconClosed: <AiFillCaretDown />,
  iconOpened: <AiFillCaretUp />,
  subnav: [
    {
      title: '0 MobX',
      path: '/mobx#mobx-description',
    },
    {
      title: 'Документация',
      path: '/mobx#mobx-documentation',
    },
    {
      title: 'Популярность',
      path: '/mobx#mobx-popularity',
    },
    {
      title: 'Размер',
      path: '/mobx#mobx-size',
    },
    {
      title: 'Масштабируемость',
      path: '/mobx#mobx-scalability',
    },
    {
      title: 'Эксперимент',
      path: '/mobx#mobx-experiment',
    },
  ],
},

```

Рисунок 44 – Информация о доступных URL (часть 3)

```

{
  title: 'Recoil',
  path: '/recoil#recoil',
  iconClosed: <AiFillCaretDown />,
  iconOpened: <AiFillCaretUp />,
  subnav: [
    {
      title: '0 Recoil',
      path: '/recoil#recoil-description',
    },
    {
      title: 'Документация',
      path: '/recoil#recoil-documentation',
    },
    {
      title: 'Популярность',
      path: '/recoil#recoil-popularity',
    },
    {
      title: 'Размер',
      path: '/recoil#recoil-size',
    },
    {
      title: 'Масштабируемость',
      path: '/recoil#recoil-scalability',
    },
    {
      title: 'Эксперимент',
      path: '/recoil#recoil-experiment',
    },
  ],
},

```

Рисунок 45 – Информация о доступных URL (часть 4)

4.3.4 Менеджер управления состоянием

Для управления состоянием сложных компонентов решено взять MobX, поскольку веб-сайт будет содержать не так много данных для того, чтобы создавать единое централизованное хранилище. Как уже было описано в предыдущей главе, такое решение позволит сэкономить время на ререндеринге отдельных компонентов, а также уменьшить общее количество кода, в виду отсутствия шаблонности у данной библиотеки. В проекте MobX используется для управления состоянием компонента бокового меню, упомянутого ранее (рисунок 46).

```
import { makeAutoObservable } from 'mobx';

You, 5 месяцев назад | 1 author (You)
class LeftSideMenu {
  openMobile = false;

  constructor() {
    makeAutoObservable(this);
  }

  handleDrawerToggle = () => {
    this.openMobile = !this.openMobile;
  };
}

export default new LeftSideMenu();
```

Рисунок 46 – Управление состоянием компонента бокового меню

4.4 CI/CD и развертывание проекта

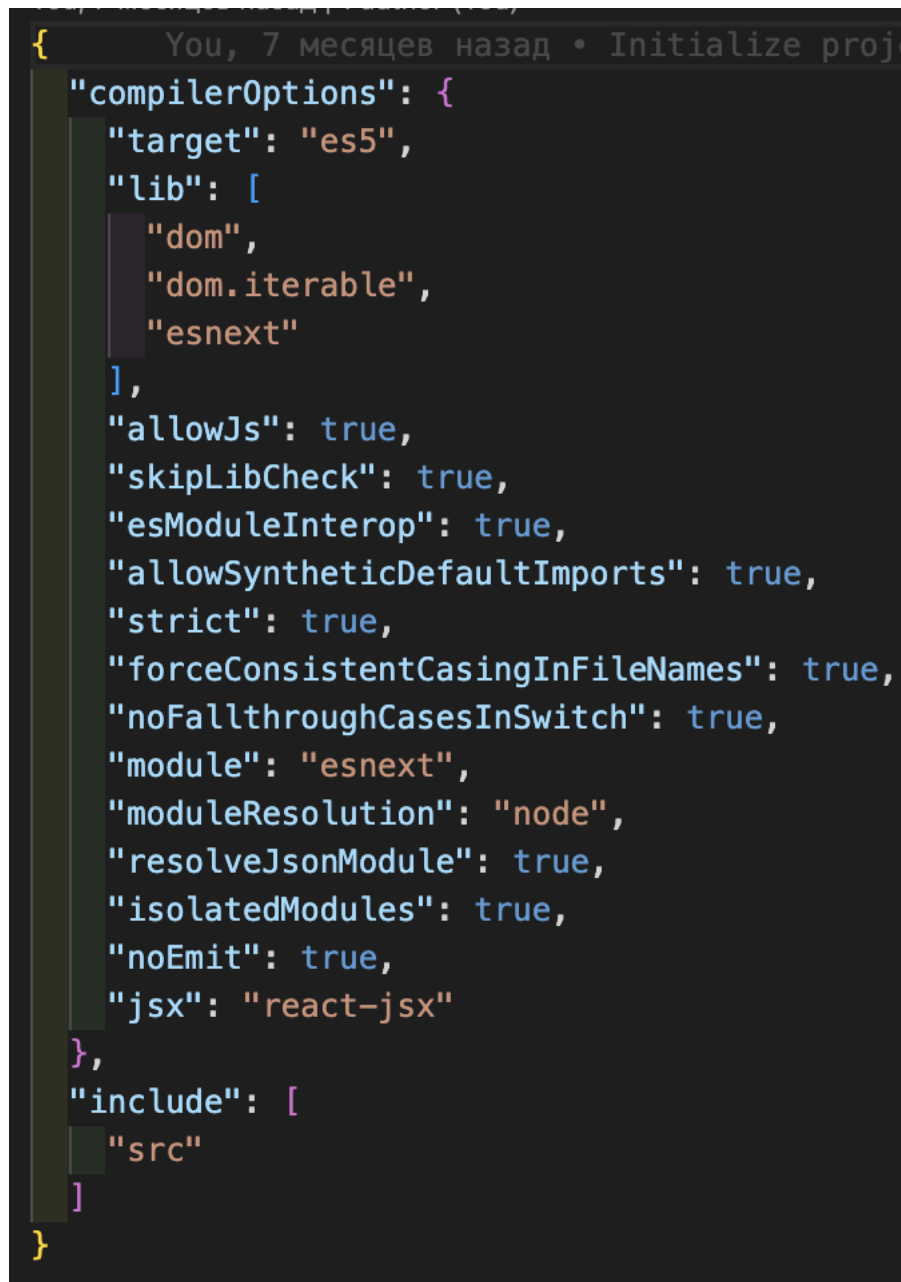
4.4.1 Тестирование

В качестве проверки кода на этапе кодирования используется Eslint. Конфиг Eslint представлен на рисунке 47.

```
module.exports = {
  extends: ['eslint:recommended', 'plugin:@typescript-eslint/recommended'],
  parser: '@typescript-eslint/parser',
  plugins: ['@typescript-eslint'],
  root: true,
  env: {
    node: true,
  },
};
```

Рисунок 47 – Файл .eslintrc.cjs

Проверка синтаксиса языка TS осуществляется на основе стандарта es5. Конфиг tsconfig.json представлен на рисунке 48.



```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": [
    "src"
  ]
}
```

Рисунок 48 – Конфиг tsconfig.json

Кроме того, в проекте используются юнит-тесты на важных компонентах, написанные с использованием Jest [60]. Примеры юнит-тестов компонентов представлены на рисунках 49-50.


```

afterEach(cleanup);

describe('PerfTest', () => {
  let balanceState: BalanceState;

  beforeEach(() => {
    balanceState = new BalanceState();
  });

  test('testing return of result for PerfTest', async () => {
    render(<PerfTest balanceState={balanceState} />);

    // Simulate user input
    const input = screen.getByTestId('input-perf-test');
    fireEvent.change(input, { target: { value: '1000' } });

    // Simulate button click
    const button = screen.getByText('Запуск теста производительности');
    fireEvent.click(button);

    // Wait for the result to be displayed
    await waitFor(() => {
      expect(screen.getByTestId('result')).not.toHaveTextContent(
        'Результат: -'
      );
    });
  });
});

```

Рисунок 49 – Юнит-тест компонента PerfTest

```

import { render, screen, cleanup } from '@testing-library/react';
import HomeInfo from './HomeInfo';
import { HomeInfoData } from '../../data/HomePage/HomeInfoData';

afterEach(cleanup);

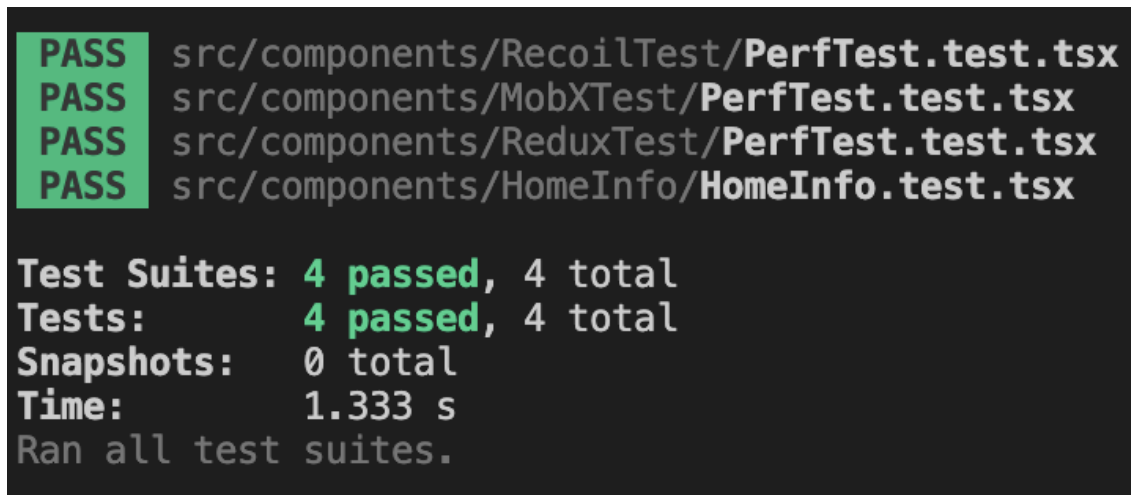
it('HomeInfo component displays data correctly', () => {
  render(<HomeInfo />);

  HomeInfoData.forEach(({ name, text }) => {
    expect(screen.getByText(name)).toBeInTheDocument();
    expect(screen.getByText(text)).toBeInTheDocument();
  });
});

```

Рисунок 50 – Юнит-тест компонента HomeInfo

Для проверки юнит-тестов достаточно ввести команду «yarn test». Пример выполнения команды для данного проекта можно увидеть на рисунке 51.



```
PASS src/components/RecoilTest/PerfTest.test.tsx
PASS src/components/MobXTest/PerfTest.test.tsx
PASS src/components/ReduxTest/PerfTest.test.tsx
PASS src/components/HomeInfo/HomeInfo.test.tsx

Test Suites: 4 passed, 4 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.333 s
Ran all test suites.
```

Рисунок 51 – Пример проверки юнит-тестов

Как можно увидеть на рисунке 51, все юнит-тесты прошли проверку, а значит можно выполнять деплой проекта.

4.4.2 Деплой проекта на Github

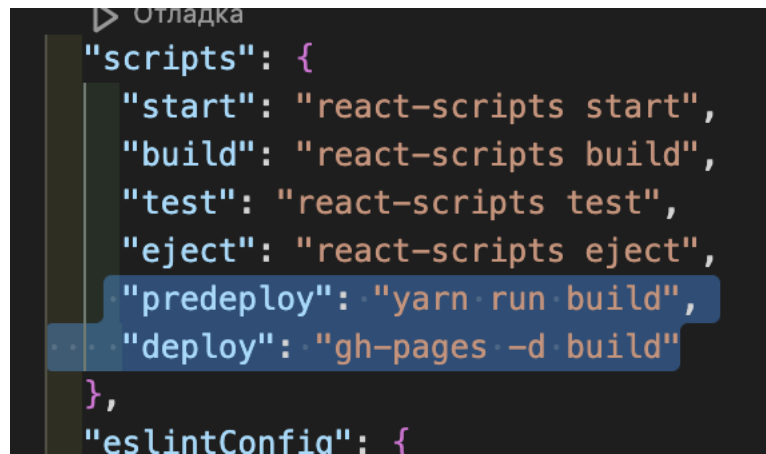
Для загрузки задеплойенной версии проекта (билда) в github была добавлена библиотека gh-pages как зависимость проекта командой «yarn add gh-pages --dev» (рисунок 52).



```
"devDependencies": {
  "@typescript-eslint/eslint-plugin": "^5.50.0",
  "@typescript-eslint/parser": "^5.50.0",
  "eslint": "^8.33.0",
  "gh-pages": "^5.0.0"
}
```

Рисунок 52 – Версия библиотеки gh-pages

Далее для упрощения работы с данной библиотекой, добавлены в файл package.json две команды (рисунок 53).



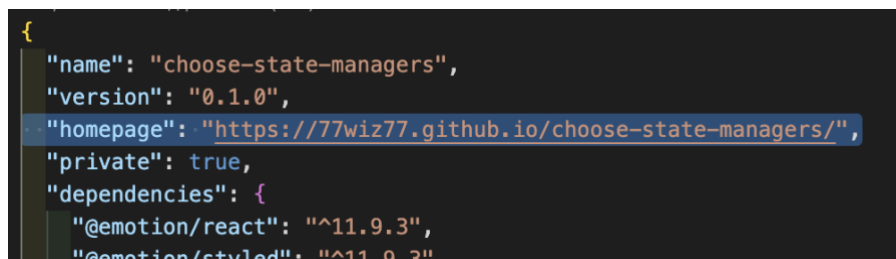
```

    "scripts": {
      "start": "react-scripts start",
      "build": "react-scripts build",
      "test": "react-scripts test",
      "eject": "react-scripts eject",
      "predeploy": "yarn run build",
      "deploy": "gh-pages -d build"
    },
    "eslintConfig": {

```

Рисунок 53 – Добавление команд

После в package.json указан URL главной страницы веб-сайта, именно она будет загружаться первой (рисунок 54).



```

{
  "name": "choose-state-managers",
  "version": "0.1.0",
  "homepage": "https://77wiz77.github.io/choose-state-managers/",
  "private": true,
  "dependencies": {
    "@emotion/react": "^11.9.3",
    "@emotion/styled": "^11.9.3"
  }
}

```

Рисунок 54 – Добавление домашней страницы

После в терминал введены указанные ранее команды «predeploy» и «deploy». В результате будет создана папка build, содержимое которой параллельно попадет на ветку gh-pages на GitHub. После чего можно будет просто перейти по URL указанном в package.json и задеплоенный веб-сайт откроется.

4.4.3 CI/CD скрипт сборки для Github Actions

Для упрощения процессов сборки и тестирования был создан скрипт в Github Actions для автоматического выполнения всех действий развертывания и тестирования, представленных в главах 4.4.1 и 4.4.2. Для этого в разделе Github Actions выбранного репозитория был создан новый workflow (рабочий процесс) (рисунок 55).

Choose a workflow

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and [set up a workflow yourself](#) →

🔍 Search workflows

Рисунок 55 – Создание конфигурации для Github Actions

Содержимое скрипта представлено на рисунке 56.

name: Deploy React Application

Controls when the action will run.

on:

Triggers the workflow on push or pull request events but only for the master branch

push:

branches: [master]

pull_request:

branches: [master]

A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:

build_test:

The type of runner that the job will run on

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [19.x] # We will deploy with only one version of node

Steps represent a sequence of tasks that will be executed as part of the job

steps:

Checks-out your repository under \$GITHUB_WORKSPACE, so your job can access it

- uses: actions/checkout@v2

- name: Use Node.js \${ matrix.node-version }

uses: actions/setup-node@v2

with:

node-version: \${ matrix.node-version }

- name: yarn ci, build and test

run: |

yarn install --frozen-lockfile

yarn run build --if-present

yarn test

- name: deploy to gh-pages

uses: peaceiris/actions-gh-pages@v3

with:

deploy_key: \${ secrets.ACTIONS_DEPLOY_KEY }

publish_dir: ./build

Рисунок 56 – Скрипт BuildTest.yml для Github Actions

После запуска action с представленным выше скриптом, сначала автоматически устанавливаются все необходимые библиотеки для работы самого action (рисунок 57).

```
✓ Set up job

1 Current runner version: '2.301.1'
2 ▶ Operating System
6 ▶ Runner Image
11 ▶ Runner Image Provisioner
13 ▶ GITHUB_TOKEN Permissions
27 Secret source: Actions
28 Prepare workflow directory
29 Prepare all required actions
30 Getting action download info
31 Download action repository 'actions/checkout@v2' (SHA:dc323e67f16fb5f7663d20ff7941f27f5809e9b6)
32 Download action repository 'actions/setup-node@v2' (SHA:1f8c6b94b26d0feae1e387ca63ccbd44d27b561)
33 Download action repository 'peaceiris/actions-gh-pages@v3' (SHA:bd8c6b06eba6b3d25d72b7a1767993c0aeee42e7)
34 Complete job name: build_test (19.x)
```

Рисунок 57 – Подготовительный этап

Далее идет этап извлечения кода репозитория в рабочую область Github Actions (рисунок 58).

```
✓ Run actions/checkout@v2

1 ▶ Run actions/checkout@v2
12 Syncing repository: 77wiz77/choose-state-managers
13 ▶ Getting Git version info
17 Temporarily overriding HOME='/home/runner/work/_temp/329ac75c-f6aa-400c-99e8-f5aabee1e61d' before making global git config changes
18 Adding repository directory to the temporary git global config as a safe directory
19 /usr/bin/git config --global --add safe.directory /home/runner/work/choose-state-managers/choose-state-managers
20 Deleting the contents of '/home/runner/work/choose-state-managers/choose-state-managers'
21 ▶ Initializing the repository
35 ▶ Disabling automatic garbage collection
37 ▶ Setting up auth
43 ▶ Fetching the repository
358 ▶ Determining the checkout info
359 ▶ Checking out the ref
363 /usr/bin/git log -1 --format='%H'
364 'a73c2fcel12a02cfff5234073f92c0777935fbbf13'
```

Рисунок 58 – Извлечение кода репозитория

Затем идет настройка Node.js с указанной в скрипте версией (рисунок 59).

```
✓ Use Node.js 19.x

1 ▶ Run actions/setup-node@v2
7 Attempting to download 19.x...
8 Not found in manifest. Falling back to download directly from Node
9 Acquiring 19.6.0 - x64 from https://nodejs.org/dist/v19.6.0/node-v19.6.0-linux-x64.tar.gz
10 Extracting ...
11 /usr/bin/tar xz --strip 1 --warning=no-unknown-keyword -C /home/runner/work/_temp/c5fb32ff-e24d-4e38-b8eb-1f01cc66ba25 -f
/home/runner/work/_temp/4c301080-b74f-45c9-9556-a541b88672ea
12 Adding to the cache ...
13 Done
```

Рисунок 59 – Настройка Node.js

После идет процесс установки всех библиотек непосредственно для развертываемого проекта, проверка юнит-тестов, а также выполнение билда (развертываемой версии ПО) (рисунок 60-61).

```
✓ yarn ci, build and test 1m 36s
1 ▶Run yarn install --frozen-lockfile
6 yarn install v1.22.19
7 warning package-lock.json found. Your project contains lock files generated by tools other than Yarn. It is advised not to mix package
managers in order to avoid resolution inconsistencies caused by unsynchronized lock files. To clear this warning, remove package-lock.json.
8 [1/4] Resolving packages...
9 [2/4] Fetching packages...
10 [3/4] Linking dependencies...
11 warning "@emotion/react > @emotion/babel-plugin@11.9.2" has unmet peer dependency "@babel/core@^7.0.0".
12 warning "@emotion/react > @emotion/babel-plugin > @babel/plugin-syntax-jsx@7.18.6" has unmet peer dependency "@babel/core@^7.0.0-0".
13 warning " > @testing-library/user-event@13.5.0" has unmet peer dependency "@testing-library/dom@>=7.21.4".
14 warning " > react-chartjs-2@5.2.0" has unmet peer dependency "chart.js@^4.1.1".
15 warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has unmet peer dependency "@babel/plugin-syntax-
flow@^7.14.5".
16 warning "react-scripts > eslint-config-react-app > eslint-plugin-flowtype@8.0.3" has unmet peer dependency "@babel/plugin-transform-react-
jsx@^7.14.9".
17 warning " > styled-components@5.3.5" has unmet peer dependency "react-is@>= 16.8.0".
18 [4/4] Building fresh packages...
19 Done in 51.41s.
20 yarn run v1.22.19
21 $ react-scripts build --if-present
22 Creating an optimized production build...
23 Browserslist: caniuse-lite is outdated. Please run:
24 npx update-browserslist-db@latest
25 Why you should do it regularly: https://github.com/browserslist/update-db#readme
26 Browserslist: caniuse-lite is outdated. Please run:
27 npx update-browserslist-db@latest
28 Why you should do it regularly: https://github.com/browserslist/update-db#readme
29 Compiled successfully.
```

Рисунок 60 – Установка окружения и создание билда проекта

```
47 yarn run v1.22.19
48 $ react-scripts test
49 PASS src/components/ReduxTest/PerfTest.test.tsx
50 PASS src/components/MobXTest/PerfTest.test.tsx
51 PASS src/components/RecoilTest/PerfTest.test.tsx
52 PASS src/components/HomeInfo/HomeInfo.test.tsx
53
54 Test Suites: 4 passed, 4 total
55 Tests:      4 passed, 4 total
56 Snapshots:  0 total
57 Time:       2.747 s
58 Ran all test suites.
59 Done in 3.71s.
```

Рисунок 61 – Автоматическая проверка юнит-тестов

И, наконец, сам процесс деплоя билда на ветку gh-pages, а также сообщения, информирующие об успешном выполнении скрипта github actions и очистке ненужных данных (рисунок 62-63).

```
✓ ✓ deploy to gh-pages

1 ▶ Run peaceiris/actions-gh-pages@v3
13 [INFO] Usage https://github.com/peaceiris/actions-gh-pages#readme
14 ▶ Dump inputs
33 ▶ Setup auth token
44 ▶ Prepare publishing assets
75 ▶ Setup Git config
81 ▶ Create a commit
86 ▶ Push the commit or tag
89 [INFO] Action successfully completed
```

Рисунок 62 – Деплой билда

```
✓ ✓ Post Use Node.js 19.x 0s
1 Post job cleanup.

✓ ✓ Post Run actions/checkout@v2 0s
1 Post job cleanup.
2 /usr/bin/git version
3 git version 2.39.1
4 Temporarily overriding HOME='/home/runner/work/_temp/380ac557-bce9-4034-8189-dd528a5d518d' before making global git config changes
5 Adding repository directory to the temporary git global config as a safe directory
6 /usr/bin/git config --global --add safe.directory /home/runner/work/choose-state-managers/choose-state-managers
7 /usr/bin/git config --local --name-only --get-regexp core\.sshCommand
8 /usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'core\.sshCommand' && git config --local --unset-all 'core.sshCommand' || :"
9 /usr/bin/git config --local --name-only --get-regexp http\.https:\/\/github\.com\/\.extraheader
10 http.https://github.com/.extraheader
11 /usr/bin/git config --local --unset-all http.https://github.com/.extraheader
12 /usr/bin/git submodule foreach --recursive sh -c "git config --local --name-only --get-regexp 'http\.https:\/\/github\.com\/\.extraheader' && git config --local --unset-all 'http.https://github.com/.extraheader' || :"

✓ ✓ Complete job 0s
1 Cleaning up orphan processes
```

Рисунок 63 – Оповещение об успешном выполнении скрипта

Теперь можно перейти в раздел Actions и выбрать там поле «deployment». Далее, в появившемся окне нажать «view deployment» после чего можно будет увидеть готовый веб-сайт. Таким образом, результаты исследования по теме данной работы представлены по ссылке: <https://77wiz77.github.io/choose-state-managers/>.

4.4.4 Создание докер-контейнера на базе репозитория

Кроме загрузки веб-сайта в репозиторий github, также на основе репозитория был создан docker-образ, а после на основе этого образа docker-контейнер для того, чтобы каждый мог поэкспериментировать с собственной версией данного проекта. Для этого, была произведена авторизация в Docker Desktop, создан в корне проекта файл Dockerfile, в котором указаны инструкции по созданию образа (рисунок 64) [61].

```
# pull the base image
FROM node:lts-alpine

# set the working direction
WORKDIR /app

# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# install app dependencies
COPY package.json ./

COPY yarn.lock ./

RUN yarn install

# add app
COPY . ./

# start app
CMD ["yarn", "start"]
```

Рисунок 64 – Содержимое Dockerfile

Также в корне проекта создан файл .dockerignore, в котором указаны файлы и папки, которые не должны быть в docker-образе (рисунок 65).

```
node_modules
npm-debug.log
build
.dockerignore
**/.git
**/.DS_Store
**/node_modules
```

Рисунок 65 – Содержимое .dockerignore

Далее необходимо отправить созданные Dockerfile и .dockerignore в гит репозиторий.

После отправления всех изменений на github репозиторий, прописывается в терминале команда «docker build -t wizdock77/choose-state-managers github.com/77wiz77/choose-state-managers.git#master», где «master» означает ветку, на основе которой будет создан docker-образ [61]. После запускается на основе образа контейнер командой «docker run -p 3000:3000 wizdock77/choose-state-managers». Таким образом, перейдя в браузере по адресу на <http://localhost:3000/choose-state-managers>, можно увидеть задеплоенный в контейнере веб-сайт на React.

4.4.5 Загрузка докер образа в Docker Hub

Также, для демонстрации рабочей версии созданный образ был выложен в Docker Hub. Для этого были прописаны в корне проекта две команды: «docker tag wizdock77/choose-state-managers wizdock77/choose-state-managers» и «docker push wizdock77/choose-state-managers». Первая нужна чтобы создать тег образа, который ссылается сам образ, а вторая пушит образ в Docker Hub [62]. После зайдя на сайт Docker Hub и авторизовавшись с никнеймом использовавшимся в командах терминала ранее можно увидеть отправленный docker-образ (рисунок 66).

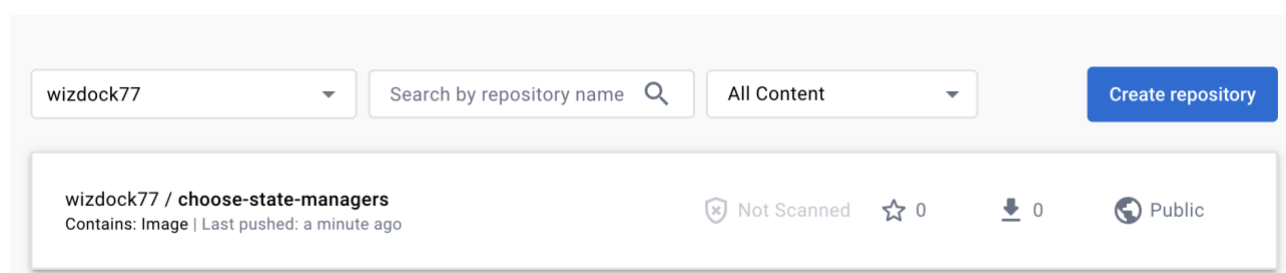


Рисунок 66 – Размещение docker-образа в Docker Hub

Также доступ к образу теперь можно получить по ссылке: <https://hub.docker.com/r/wizdock77/choose-state-managers>.

ЗАКЛЮЧЕНИЕ

В ходе выполнения ВКР была успешно достигнута поставленная цель и полностью решены задачи, способствующие ее достижению:

- 1) составлен обзор отечественных и зарубежных научных исследований, посвященных вопросам эффективности клиентской части веб-приложений с использованием актуальных библиотек управления состоянием в React,
- 2) определена методология компонентного подхода и причины необходимости менеджеров управления состояниями,
- 3) выделены основные метрики менеджеров управления событиями в React,
- 4) выполнено тестирование быстродействия менеджеров управления событиями в одинаковых условиях,
- 5) получены и описаны результаты, полученные в ходе исследования,
- 6) сформированы требования к будущему веб-сайту с информацией, полученной в ходе исследования,
- 7) разработан интерфейс веб-сайта с информацией по выбору библиотеки управления состояниями компонентов, полученной в ходе проведенного исследования.

В рамках первой главы произведен детальный анализ зарубежной и отечественной литературы, в рамках которого были рассмотрены научные труды по теме исследования.

Анализ отечественных научных источников литературы показал, что в настоящее время информация о библиотеках управления состоянием React.js ограничивается библиотекой Redux, другие же библиотеки либо освещены очень поверхностно, либо вообще не описаны. Это говорит о том, что актуальность изучения библиотек управления состоянием компонентов является высокой, а информация, полученная в ходе настоящего исследования, внесет свой вклад в развитие отечественных веб-разработок.

Проведенный анализ зарубежных научных источников по теме исследования «Эффективность и оптимизация клиентской части веб-приложений с использованием актуальных библиотек управления состоянием в React» позволил выявить актуальное состояние информации о текущем современном уровне развития библиотек управления состоянием, наиболее популярные и востребованные варианты и принципы их применения.

Во второй главе были описаны методология компонентного построения веб-приложений, пояснение необходимости сторонних библиотек управления состояниями компонентов в React, а также приведены принцип хранения состояния и механизм взаимодействия с ними. Исследования по данной главе показали общую картину проблемы, что позволило определить вводную информацию для следующей главы.

В третьей главе приведена сравнительная характеристика ключевых параметров библиотек, что дает детальное представление о каждой из выбранных библиотек. Выполнен эксперимент по определению наиболее быстрой библиотеки в равных условиях. Данный эксперимент показал, что в равных условиях библиотека MobX работает быстрее среди выбранных. Кроме того, в ходе эксперимента выяснилось, что на текущий момент библиотека Recoil крайне плохо адаптирована к работе множественного рендеринга компонентов с использованием циклов.

В четвертой главе описан процесс проектирования, разработки и DevOps веб-сайта, на котором опубликованы результаты проведенного исследования, а также реализована возможность проведения эксперимента с возможностью изменения количества прогонов рендеринга в случае, если необходимо узнать общее время рендеринга определенного количества компонент для каждой из выбранных библиотек.

Результаты, полученные в ходе выполнения ВКР могут служить в качестве опоры для разработчиков при выборе библиотеки управления состояниями компонентов в React. Кроме того, они дают возможность к дальнейшим исследованиям в области управления состояниями, в частности

усложнения представленного в работе эксперимента, актуализации информации в соответствии с будущими обновлениями и добавлении информации о других библиотеках управления состояниями.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Натроби́на, А.Э. Redux и новый react context API как инструменты управления состоянием веб-приложения / А.Э. Натроби́на, В.О. Федоров // Компьютерные технологии и моделирование в науке, технике, экономике, образовании и управлении: тенденции и развитие: Материалы международной научно-технической конференции, Махачкала, 16–18 октября 2019 года. – Махачкала: Дагестанский государственный технический университет, 2019. – С. 299-301. URL: <https://elibrary.ru/item.asp?id=43113854> (дата обращения: 24.11.2021).
2. Колышкина, Н.С. Сравнение библиотек Redux и mobx / Н.С. Колышкина, Л.Е. Малкова // Modern Science. – 2020. – № 6-2. – С. 267-270. URL: <https://elibrary.ru/item.asp?id=43005224&> (дата обращения: 25.11.2021).
3. Скороходов, И.С., Исследование и сравнение современных реализаций Flux-архитектур разработки веб-приложений / И.С. Скороходов, А.Н. Тихомирова // Наука, техника и образование. 2016. №6 (24). URL: <https://cyberleninka.ru/article/n/issledovanie-i-sravnenie-sovremennyh-realizatsiy-flux-arhitektur-razrabotki-veb-prilozheniy> (дата обращения: 27.11.2021).
4. Ким, А. Анализ и оптимизация ре-рендеринга компонентов / А. Ким // Современное образование: традиции и инновации. – 2021. – № 4. – С. 69-75. URL: <https://www.elibrary.ru/item.asp?id=47369456> (дата обращения: 28.11.2021).
5. Смекалин, Н.Н., Постановка и решение проблем с помощью Redux / Н.Н. Смекалин, С.Э. Шмаков, Е.В. Щенникова [и др.] // XLIX Огарёвские чтения: Материалы научной конференции: в 3 частях, Саранск, 07–13 декабря 2020 года. – Саранск: Национальный исследовательский Мордовский государственный университет им. Н.П. Огарёва, 2021. – С. 660-664. URL: <https://www.elibrary.ru/item.asp?id=46247432&> (дата обращения: 29.11.2021).
6. Троценко, Е.С. Технологии разработки мобильных приложений для занятия фитнесом / Е.С. Троценко // Евразийское Научное Объединение. – 2020. – № 6-2(64). – С. 152-154.

7. Podila, P. MobX Quick Start Guide: Supercharge the client state in your React apps with MobX / P. Podila, M. Weststrate – Packt Publishing Ltd (ISBN 978-1-78934-483-7), 2018. – P. 41-58.
8. Wieruch, R. Taming the State in React: Your journey to master Redux and Mobx / R. Wieruch – Leanpub (ISBN-13: 978-1720710769, ISBN-10: 1720710767), 2017. – P. 240-243.
9. Slepner, S. Recoil – Another React State Management Library? [Электронный ресурс] URL: <https://medium.com/swlh/recoil-another-react-state-management-library-97fc979a8d2b> (дата обращения: 30.11.2021).
10. Elrom, E. React and Libraries: Your Complete Guide to React Ecosystem / Apress (ISBN-13 (pbk): 978-1-4842-6695-3), 2021. – P. 150-152.
11. Официальный сайт Redux [Электронный ресурс] URL: <https://redux.js.org/> (дата обращения: 02.12.2021).
12. Официальный сайт MobX [Электронный ресурс] URL: <https://mobx.js.org/README.html> (дата обращения: 04.12.2021).
13. Официальный сайт Recoil [Электронный ресурс] URL: <https://recoiljs.org/> (дата обращения: 05.12.2021).
14. Казначеева, Е.О. Модульный и компонентный подходы в программировании / Е.О. Казначеева // Альманах научных работ молодых ученых университета ИТМО : XLVII научная и учебно-методическая конференция Университета ИТМО по тематикам: экономика; менеджмент, инноватика, Санкт-Петербург, 30 января – 02 2018 года. – Санкт-Петербург: Университет ИТМО, 2018. – С. 150-152. – EDN YXNFXF.
15. Шинкарев Александр Андреевич РЕТРОСПЕКТИВА РАЗВИТИЯ ВЕБ-ТЕХНОЛОГИЙ В СОЗДАНИИ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника. 2020. №4. URL: <https://cyberleninka.ru/article/n/retrospektiva-razvitiya-veb-tehnologiy-v-sozdanii-korporativnyh-informatsionnyh-sistem> (дата обращения: 25.05.2022).

16. Mikowski M. Single Page Web Applications: JavaScript end-to-end. Shelter Island, Manning Publications, 2014. 432 p
17. Байдыбеков А.А., Гильванов Р.Г., Молодкин И.А. СОВРЕМЕННЫЕ ФРЕЙМВОРКИ ДЛЯ РАЗРАБОТКИ WEB-ПРИЛОЖЕНИЙ // Интеллектуальные технологии на транспорте. 2020. №4 (24). URL: <https://cyberleninka.ru/article/n/sovremennye-freymvorki-dlya-razrabotki-web-prilozheniy> (дата обращения: 25.05.2022).
18. Официальный сайт ReactJS [Электронный ресурс] URL: <https://reactjs.org/> (дата обращения: 25.05.2022).
19. Новиков Сергей. Angular vs. React vs. Vue: Сравнение, 2017. [Электронный ресурс], 2018. Режим доступа: <https://habr.com/post/338068/> (дата обращения: 26.05.2022).
20. Сервис поиска работы по России hh.ru [Электронный ресурс] URL: https://hh.ru/?hhtmFrom=vacancy_search_list (дата обращения: 29.05.2022).
21. Официальный сайт Vue [Электронный ресурс] URL: <https://vuejs.org/> (дата обращения: 28.05.2022).
22. Сергачева Марина Александровна, Михалевская Карина Анатольевна АНАЛИЗ ФРЕЙМВОРКОВ ДЛЯ РАЗРАБОТКИ СОВРЕМЕННЫХ ВЕБ-ПРИЛОЖЕНИЙ // Кронос: естественные и технические науки. 2020. №2 (30). URL: <https://cyberleninka.ru/article/n/analiz-freymvorkov-dlya-razrabotki-sovremennyh-veb-prilozheniy> (дата обращения: 02.06.2022).
23. Байнов Артем Маратович, Кривоногова Анастасия Евгеньевна, Николаев Алексей Сергеевич, Богомолова Ольга Игоревна Обзор современных фреймворков и инструментов, используемых для разработки web-приложений // Наука без границ. 2020. №1 (41). URL: <https://cyberleninka.ru/article/n/obzor-sovremennyh-freymvorkov-i-instrumentov-ispolzuemyh-dlya-razrabotki-web-prilozheniy> (дата обращения: 02.06.2022).
24. Официальный сайт Angular [Электронный ресурс] URL: <https://angular.io/> (дата обращения: 28.05.2022).

25. Сулыз А.В., Панфилов А.Н. Технология разработки одностраничного веб-приложения на платформе Angular 8 // Молодой исследователь Дона. 2020. №2 (23). URL: <https://cyberleninka.ru/article/n/tehnologiya-razrabotki-odnostranichnogo-veb-prilozheniya-na-platforme-angular-8> (дата обращения: 02.06.2022).
26. Сучков А.А., Гек Д.К., Багаева А.П. ИСПОЛЬЗОВАНИЕ REACTJS В СОВРЕМЕННОЙ WEB-РАЗРАБОТКЕ // Актуальные проблемы авиации и космонавтики. 2019. №. URL: <https://cyberleninka.ru/article/n/ispolzovanie-reactjs-v-sovremennoy-web-razarabotke> (дата обращения: 30.05.2022).
27. Banks A., Porcello E. Learning React: functional web development with React and Redux. – " O'Reilly Media, Inc.", 2017.
28. Mezzalira L. Mobx: Simple state management //Front-End Reactive Architectures. – Apress, Berkeley, CA, 2018. – С. 129-158.
29. Tharaka R. Recoil the next state management library for React [Электронный ресурс] URL: <https://medium.com/swlh/intro-to-recoil-d689a77c5f04> (дата обращения: 29.05.2022).
30. Блог команды разработчиков Recoil [Электронный ресурс] URL: <https://recoiljs.org/blog/2022/10/11/recoil-0.7.6-release/> (дата обращения: 05.11.2022).
31. Что такое NPM Trend [Электронный ресурс] URL: <https://medium.com/codex/how-developers-use-npm-trends-to-select-javascript-dependencies-c23a1a1f8f89> (дата обращения: 10.12.2022).
32. Аналитическая поисковая система на основе использования пакетов npm [Электронный ресурс] URL: <https://npm trends.com/mobx-vs-recoil-vs-redux> (дата обращения: 10.12.2022).
33. Chinnathambi K. Learning React: A Hands-on Guide to Building Web Applications Using React and Redux. – Addison-Wesley Professional, 2018.
34. McFarlane T. Managing State in React Applications with Redux. – 2019.
35. Caspers M.K. React and redux //Rich Internet Applications w/HTML and Javascript. – 2017. – Т. 11.

36. Рекомендация от разработчиков redux [Электронный ресурс] URL: <https://redux.js.org/style-guide/#only-one-redux-store-per-app> (дата обращения: 12.12.2022).
37. Freeman A. Using a Redux Data Store //Pro React 16. – Apress, Berkeley, CA, 2019. – С. 531-559.
38. Ventura L. Analysis of Redux, MobX and BLoC and how they solve the state management problem. – 2022.
39. Bugl D. Learn React Hooks: Build and refactor modern React. js applications using Hooks. – Packt Publishing Ltd, 2019.
40. Elrom E. Optimize Your React App //React and Libraries. – Apress, Berkeley, CA, 2021. – С. 371-402.
41. Elrom E. Integrating State Management //Integrating D3. js with React. – Apress, Berkeley, CA, 2021. – С. 125-145.
42. Документация для интерфейса Performance [Электронный ресурс] URL: <https://developer.mozilla.org/ru/docs/Web/API/Performance> (дата обращения: 05.12.2022).
43. Примеры использования интерфейса Performance [Электронный ресурс] URL: <https://habr.com/ru/company/skillfactory/blog/538334/> (дата обращения: 20.12.2022).
44. Информация об хуке useRecoilState из официальной документации Recoil [Электронный ресурс] URL: <https://recoiljs.org/docs/api-reference/core/useRecoilState/> (дата обращения: 20.12.2022).
45. Arnowitz J., Arent M., Berger N. Effective prototyping for software makers. – Elsevier, 2010.
46. Roth R. E. et al. Wireframing for interactive & web-based geographic visualization: designing the NOAA Lake Level Viewer //Cartography and Geographic Information Science. – 2017. – Т. 44. – №. 4. – С. 338-357.
47. Официальный сайт Figma // [Электронный ресурс] URL: <https://www.figma.com> (дата обращения: 10.10.2022).

48. Putra Z.E.F.F., Ajie H., Safitri I.A. Designing a user interface and user experience from Piring Makanku application by using Figma application for teens //IJISTECH (International Journal of Information System and Technology). – 2021. – Т. 5. – №. 3. – С. 308-315.
49. Sharma V., Tiwari A.K. A Study on User Interface and User Experience Designs and its Tools //World Journal of Research and Review (WJRR). – 2021. – Т. 12. – №. 6.
50. Bilousova L.I., Gryzun L.E., Zhytienova N.V. Fundamentals of UI/UX design as a component of the pre-service specialist's curriculum. – 2021.
51. Официальная документация языка TypeScript [Электронный ресурс] URL: <https://www.typescriptlang.org/docs/> (дата обращения: 30.05.2022).
52. Официальный сайт Yarn [Электронный ресурс] URL: <https://yarnpkg.com/> (дата обращения: 30.05.2022).
53. Официальная документация пакета Create-React-App [Электронный ресурс] URL: <https://create-react-app.dev/docs/adding-typescript> (дата обращения: 30.05.2022).
54. Официальная документация библиотеки Styled-component [Электронный ресурс] URL: <https://styled-components.com/docs> (дата обращения: 30.05.2022).
55. Официальный сайт Material UI [Электронный ресурс] URL: <https://mui.com/> (дата обращения: 30.05.2022).
56. Официальный гайдбук стилей Material Design [Электронный ресурс] URL: <https://m3.material.io/> (дата обращения: 30.05.2022).
57. Официальный сайт React Router [Электронный ресурс] URL: <https://reactrouter.com/> (дата обращения: 30.05.2022).
58. Информация о HashRouter из официальной документации React Router [Электронный ресурс] URL: <https://reactrouter.com/en/main/router-components/hash-router> (дата обращения: 30.05.2022).
59. Pezoa F. et al. Foundations of JSON schema //Proceedings of the 25th international conference on World Wide Web. – 2016. – С. 263-273.

60. Moroz B. Unit Test Automation of a React-Redux Application with Jest and Enzyme. – 2019.
61. Официальная документация Docker // [Электронный ресурс] URL: <https://docs.docker.com/> (дата обращения: 02.12.2022).
62. Официальная документация Docker-Hub // [Электронный ресурс] URL: <https://hub.docker.com/> (дата обращения: 02.12.2022).