6. I.V. Kniazev. The advanced web applications caching and optimization using SWR // Sciences of Europe. – 2021. – No. 73(1). – P. 47-49. URL: https://cyberleninka.ru/article/n/prodvinutoe-keshirovanie-i-optimizatsiya-veb-prilozheniy-s-pomoschyu-tehnologii-swr (date of access: 11/17/2021).

7. I.V. Kniazev. Analyzing application performance using server-side rendering: migrating, configuring and deploying the Next.JS application // Sciences of Europe. – 2021. – No. 76(1). – P. 71-74. URL: https://cyberleninka.ru/article/n/analiz-raboty-prilozheniya-s-ispolzovaniem-server-side-rendering-migratsiya-nastroyka-i-razvertyvanie-prilozheniya-next-js (date of access: 11/25/2021).

8. Background processing using web workers. [Electronic resource] URL: https://angular.io/guide/web-worker (date of access: 11/29/2021).

9. O.V. Borodin, V.A. Egunov. Multi-threaded fronted image processing using Web-workers API // CASPIAN JOURNAL: Control and High Technologies. – 2021. – No. 3(55). – P. 33-46. URL: https://cyberleninka.ru/article/n/mnogopotochnaya-obrabotka-izobrazheniy-s-ispolzovaniem-api-web-workers (date of access: 12/3/2021).

10. Gowda V., Rangaswamy S. Addressing the Limitations of React JS // International Research Journal of Engineering and Technology (IRJET). – 2020. – No. 7(4). – P. 5065-5068. URL: https://www.irjet.net/archives/V7/i4/IRJET-V7I4966.pdf (date of access: 12/9/2021).

11. Mukhiya S.K., Hung H.K. An Architectural Style for Single Page Scalable Modern Web Application // International Journal of Recent Research Aspects. – 2018. – No. 5(4). – P. 6-13. URL: https://www.researchgate.net/publication/335259756_An_Architectural_Style_for_Single_Page_Scalable_Modern_Web_Application (date of access: 12/17/2021)

Статья поступила в редакцию 29.04.2023, одобрена после рецензирования 09.05.2023, принята к публикации 09.05.2023.
The article was submitted 29.04.2023; approved after reviewing 09.05.2023; accepted for publication 09.05.2023.

Научная статья
УДК 004

ОБЗОР ВЛИЯНИЯ РАЗЛИЧНЫХ БИБЛИОТЕК И РАЗЛИЧНОЙ РЕАЛИЗАЦИИ КОМПОНЕНТА НА ПРОИЗВОДИТЕЛЬНОСТЬ REACTJS ПРИЛОЖЕНИЯ

Марина Владимировна Агафонова
Университет ИТМО, Санкт-Петербург, Россия,
agafonova_mv@niuitmo.ru

Аннотация. В статье рассмотрены 4 варианта реализации проекта: от начального до конечного. В каждом из реализованных вариантов либо была применена иная библиотека по управлению состоянием приложения, либо собственный компонент графовидной структуры был реализован иным подходом. Сравниваются по производительности приложения библиотеки redux-symbiote и RTK Query и реализации графа как с помощью SVG-элементов, так и с помощью HTML-элементов. Описано, по какой причине случился переход от первоначального варианта реализации к конечного.

Ключевые слова: веб-разработка, производительность рендеринга, JavaScript, React JS, Redux, SVG, HTML

Для цитирования: Агафонова М. В. Обзор влияния различных библиотек и различной реализации компонента на производительность ReactJS приложения // Научно-технические инновации и веб-технологии. 2023. № 1. С. 52–57.

Original article

OVERVIEW OF THE INFLUENCE OF DIFFERENT LIBRARIES AND DIFFERENT COMPONENT IMPLEMENTATIONS AT REACTJS APPLICATION PERFORMANCE

Marina V. Agafonova
ITMO University, Saint Petersburg, Russia,
agafonova_mv@niuitmo.ru

Abstract. The article considers 4 options for the implementation of the project: from initial to final. In each of implemented options, either a different application state management library was used, or a custom graph-like structure component was implemented in a different way. The application performance of the redux-symbiote and RTK Query libraries and the implementation of the graph using both SVG elements and HTML elements are compared in terms of performance. It is described for what reason there was a transition from the initial implementation to the final one.

Keywords: web development, rendering performance, JavaScript, React JS, Redux, SVG, HTML

Уже не первый год разработка веб-приложений является самым популярным направлением разработки IT-продуктов. Компания Nexign в этом плане не является исключением, т.к. активно занимается импортозамещением и разработкой собственных продуктов, которые помогают в продвижении бизнеса в сфере телекома. Один из таких решений является продукт Revenue Management, представляющий собой клиент-серверное приложение. Его задачей является консолидация доходов по всем направлениям бизнеса и устранение разрыва между современными системами цифрового слоя и устаревшим монолитным биллингом [1].

Если говорить о самой разработке клиентской части Revenue Management, то команде front-end разработки нужно было разработать единый интерфейс для многомодульной системы. Помимо базовых UI компонентов (таблицы, формы для заполнения и т.д.) на макетах присутствовали нестандартные решения, для разработки которых необходимо либо использовать дополнительные библиотеки, либо разрабатывать собственное решение с нуля. Таким нестандартным решением являлось представление списка объектов в виде графа. Особенность была в том, что каждый узел такого графа должен быть визуально соединен со следующим по порядку узлом, при этом у самих объектов могла быть зависимость по типу «родитель-потомок», которая должна была отображаться в виде бокового ребра. Примерный вид требуемого представления отображен на рисунке 1.

Первой задачей по разработке данного графа было исследование на тему того, какие библиотеки на данный момент позволяют отрисовать подобный граф и насколько они позволяют подстроить полученное решение под собственные стили и требования.

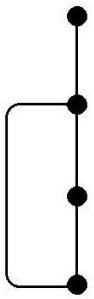

Рисунок 1 – Приблизительное представления требуемого графа

После попытки отрисовать граф с помощью библиотеки visx от компании Airbnb и адаптировать полученный от API массив объектов был сделан вывод, что все-таки данное решение не совсем подходило по бизнес-требованиям Revenue Management: для отрисовки графа средствами visx было необходимо преобразовать данные в иерархичный вид, т.е. в массиве первый элемент должен являться «корнем» дерева, а остальные элементы должны быть зависимы от данного элементы как «потомки». Если для случая, когда в полученном массиве нет своих зависимостей типа «родитель-потомок», ещё было удобно организовать иерархичное представление для отрисовки необходимого представления, то уже при наличии хотя бы одной связи возникали трудности в возможностях, которая предоставляла библиотека visx. А именно проблема возникла с плохо кастомизируемым компонентом ребра графа. Как было видно на рисунке 1, ребро зависимости «родитель-потомок» должно отображаться слева от общего графа, углы должны быть скругленными. В связи с тем, что других более подходящих библиотек не было найдено, было принято решение о разработке представления графа полностью с нуля при помощи базовых svg элементов (circle, path и другие).

В изначально утвержденном стеке технологий для разработки клиентского приложения была такая библиотека как Redux Symbiote, которая позволяла получать данные от сервера в зависимости от состояния приложения (загрузка и т.д.).

Если говорить о самом Redux, то это инструмент для управления состоянием данных и пользовательским интерфейсом в JavaScript приложениях с большим количеством сущностей. Обычно данная библиотека используется в связке с такими JavaScript фреймворками, как React, Vue, Angular. Redux решает следующие задачи.

1. Управление состоянием приложения, работающего с большим количеством данных.
2. Замена встроенных средств работы с состоянием в React.
3. Легкое масштабирование приложения.
4. Избавление от ошибок, связанных с объектом состояния.
5. Упрощенная отладка.
6. Повышенная производительность приложения.

Что касается библиотеки redux-symbiote, то она позволяет реализовать работу с редьюсерами (англ. reducers, функции, результат выполнения которых зависит только от входных данных) в более компактном и читабельном виде. Т.е. данный инструмент позволяет избавиться от написания шаблонного кода, который ещё называют «бойлерплейт». В API библиотеки представлена функция createSymbiote, которая создаёт «симбиойт», но как указано в примере документации, при деструктуризации возвращается действия (actions) и редьюсер. Когда происходит обращение к actions.setValue, то симбиойт вызывает обработчик действия с предыдущим состояниям и всеми переданными аргументами, распределенными после состояния.

Помимо двух обязательных аргументов (initialState и symbiotes) у функции createSymbiote есть третий опциональный параметр. В документации репозитория библиотеки он обозначен как namespace, значением по умолчанию которого является пустая строка. Но данный параметр может быть передан как в виде строки, так и в виде

объекта. Если будет передана только строка, то значение этой строки будет присвоено параметру namespace, который по сути является префиксом для каждого типа действия. Сам же объект может содержать следующие поля: namespace, defaultReducer, sepator.

У redux-symbiote был так же отмечен такой недостаток, как малое количество материалов для ознакомления с данной библиотекой. Т.е. на данный момент существует только краткое описание API с примерами на странице GitHub репозитория [2] и обзорная статья на Habr [3]. В связи с этим недостатком было принято решение о миграции архитектуры проекта с использования redux-symbiote на redux-toolkit.

Помимо того, что на тему redux-toolkit имелась обширная документация и множество других ознакомительных материалов, при сравнении двух реализаций небольшого приложения с помощью redux-toolkit и redux-symbiote было выявлено, что вариант на redux-toolkit получился более компактным, интуитивно понятным. Отпала необходимость использовать «эффекты» для общения с API системы, как это было в варианте с redux-symbiote.

Redux Toolkit выполняет следующие функции:

1. Упрощает работу с типичными задачами и кодом Redux.
2. Позволяет использовать лучшие практики (best practices) Redux по умолчанию.
3. Предлагает решения, уменьшающие недоверие к бойлерплейтам.

В Redux Toolkit можно выделить следующие наиболее значимые функции.

1. Configurestore – создание и настройка хранилища.
2. Createreducer – описание и создание редьюсера.
3. Createaction – функция, возвращающая функцию создателя действия для заданной строки типа действия.
4. Createslice – объединение функционала createreducer и createaction.
5. Createselector – утилитита, которая была реэкспортирована из библиотеки Reselct для простоты использования [4].

Так же вместе с Redux Toolkit было принято решение об использовании RTK Query. Это инструмент для получения и кэширования данных, предназначенный для упрощения распространенных случаев загрузки данных в веб-приложении, избавляя от необходимости вручную писать логику загрузки и кэширования данных. По сути, RTK Query является дополнением, включенный в пакет Redux Toolkit, его функциональность построена поверх других API в Redux Toolkit.

Если применение Redux Toolkit и RTK Query коснулось всей архитектуры приложения, то следующее изменение было связано именно с методом отрисовки графа (см. рисунок 1). Был предложен совершенно иной подход – реализация элементов графа (узлов, ребер) только через стандартные html-элементы, а именно – div, универсальный контейнер для потокового контента. Такой подход позволял избавиться от лишних вычислений (пересчет координат в первой реализации) за счёт адаптивности стандартных html-элементов.

Чтобы проанализировать полученный результат в плане производительности, можно составить 4 варианта приложения, начиная с первоначальной реализации и заканчивая финальной версией. Данные варианты представлены в таблице 1.

Таблица 1 – Варианты комбинаций способа реализации графа и библиотеки для работы с Redux

	SVG	HTML
Symbiote	1 вариант	2 вариант
Toolkit	3 вариант	4 вариант

Первым вариантом для рассмотрения является первоначальная реализация поставленной задачи – отрисовка графа и иных элементов интерфейса для назначенной функциональности. В качестве второго варианта было реализовано сочетания библиотеки Redux-symbiote и реализация построения графа через стандартные HTML-элементы. Третий вариант являлся промежуточной стадией для перехода к окончательному варианту.

Приоритетной задачей на проекте была миграция уже сделанной функциональности (фичи) на новую архитектуру приложения, в котором получения данных с сервера осуществлялось теперь с помощью Redux-toolkit, а не redux-symbiote. Окончательный вариант представляет собой как применение инструмента Redux-toolkit для операций с данными, так и реализацию графа с помощью стандартных HTML-элементов.

Для оценки производительности клиентского приложения чаще всего используются средства, которые уже встроены во многих браузерах. Таким инструментом является «DevTools» (development tools) – это программа, позволяющая создавать, тестировать, отлаживать программное обеспечение. В данном инструменте имеется вкладка «Performance», в которой представлен интерфейс для оценки производительности сайта. Здесь можно посмотреть суммарную информацию о загрузке, выполнению скриптов JavaScript, рендерингу, отрисовке и другим процессам. Всё это представлено в виде кольцевой диаграммы, справа от которой находится сводное описание, сколько времени в миллисекундах занял тот или иной процесс.

Помимо вкладки «Performance» в DevTools присутствует такой инструмент как Lighthouse. Он не только тестирует сайт и показывает оценку производительности, но и даёт конкретные рекомендации: что можно улучшить, чтобы сделать загрузку сайта быстрее. Для запуска проверки достаточно выбрать необходимые параметры и нажать кнопку «Generate report». Lighthouse анализирует четыре показателя: производительность, доступность, SEO, Best Practices (лучшие практики) [5].

Касательно производительности, данный инструмент анализирует скорость загрузки сайта. На данную оценку влияют такие факторы как время блокировки, отрисовка стилей, загрузка интерактивных элементов, шрифтов и контента.

В таблице 2 представлена сводная информация по реализованным вариантам. В первой колонке обозначен номер варианта. Далее представлены четыре параметра измерений в Performance вкладке инструмента DevTools. Все эти значения относятся к временному значению в миллисекундах. И в последней колонке указана оценка из отчета Lighthouse из 100 возможных баллов.

Таблица 2 – Время загрузки, выполнения JavaScript, рендеринга и оценка производительность из Lighthouse

	Performance				**Lighthouse**
Версия	Loading, ms	Scripting, ms	Rendering, ms	Total, ms	Performance mark
Вариант 1	1	403	30	1929	49/100
Вариант 2	7	616	27	2120	68/100
Вариант 3	1	1481	35	4170	47/100
Вариант 4	0	2302	18	4202	41/100

По сводным данным в таблице видно, что производительность улучшилась в плане времени рендеринга страницы после замены всех svg-элементов на div-контейнеры в реализации компонентов графа. Разницу в производительности между вариантами с symbiote и toolkit можно объяснить размером используемых библиотек, что в итоге повлияло на итоговый файл сборки: redux-symbiote – 40.0 КБ, redux-toolkit – 12.5 МБ.

Заключение. Из полученных результатов можно увидеть, как выбор библиотеки или подход к реализации собственного компонента может повлиять на производительность.

Список источников

1. Гармонизация ядра BSS: единый подход к традиционным и инновационным источникам выручки [Электронный ресурс] URL: https://nexign.com/ru/blog/harnessing-traditional-and-new-revenue-streams-track-maturity-harmonization-bss-core (дата обращения: 30.09.2022)

2. Репозиторий sergeysova/redux-symbiote [Электронный ресурс] URL: https://github.com/sergeysova/redux-symbiote (дата обращения: 05.10.2022)
3. Redux-symbiote – пишем действия и редьюсеры почти без боли [Электронный ресурс] URL: https://habr.com/ru/post/442346/ (дата обращения: 09.10.2022)
4. Redux Toolkit как средство эффективной Redux-разработки [Электронный ресурс] URL: https://habr.com/ru/company/inobitec/blog/481288/ (дата обращения: 24.10.2022)
5. Как оптимизировать сайты с помощью Lighthouse [Электронный ресурс] URL: https://htmlacademy.ru/blog/articles/lighthouse (дата обращения 03.12.2022)

References

1. Harnessing Traditional and New Revenue Streams on Track to Maturity: Harmonization of the BSS Core [Electronic resource] URL: https://nexign.com/ru/blog/harnessing-traditional-and-new-revenue-streams-track-maturity-harmonization-bss-core (date of access: 30/09/2022)
2. Repository sergeysova/redux-symbiote [Electronic resource] URL: https://github.com/sergeysova/redux-symbiote (date of access: 05/10/2022)
3. Redux-symbiote – write actions and reducers almost without pain [Electronic resource] URL: https://habr.com/ru/post/442346/ (date of access: 09/10/2022)
4. Redux Toolkit as a tool for efficient Redux-development [Electronic resource] URL: https://habr.com/ru/company/inobitec/blog/481288/ (date of access: 24/10/2022)
5. How to optimize websites with Lighthouse [Electronic resource] URL: https://htmlacademy.ru/blog/articles/lighthouse (date of access: 03/12/2022)

Статья поступила в редакцию 29.04.2023, одобрена после рецензирования 09.05.2023, принята к публикации 09.05.2023.
The article was submitted 29.04.2023; approved after reviewing 09.05.2023; accepted for publication 09.05.2023.

Научная статья
УДК 004.4

НЕКОТОРЫЕ АСПЕКТЫ РЕАЛИЗАЦИИ HEADLESS CMS ДЛЯ ГЕНЕРАТОРА СТАТИЧЕСКИХ САЙТОВ HUGO

Павел Александрович Аксютин, Федор Владиславович Мельников
Российский государственный педагогический университет им. А. И. Герцена, Санкт-Петербург, Россия
pavel.aks@gmail.com, balrun.dev@gmail.com

Аннотация. В статье рассмотрены виды систем управления содержимым для статических сайтов. Представлена архитектура системы редактирования статического сайта, разработанного с использованием генератора статического сайта Hugo. Рассмотрен процесс обработки данных для взаимодействия между клиентской и серверной частями веб-приложения.

Ключевые слова: генератор статических сайтов, система управления содержимым, Headless CMS, инструменты разработки сайтов

Для цитирования: Аксютин П. А., Мельников Ф. В. Некоторые аспекты реализации Headless CMS для генератора статических сайтов Hugo // Научно-технические инновации и веб-технологии. 2023. № 1. С. 57–61.