24 июня 2022

React-ивное импортозамещение: как мы не побоялись сделать масштабный рефакторинг на уже запущенном проекте

Андрей Комаров,
фронтенд-разработчик ГК «КОРУС Консалтинг»

Со стартом активного импортозамещения еще несколько лет назад было трудно поверить, что оно сможет быть красивым, качественным и современным. Западные системы создавались и развивались много лет, а российское ПО создавалось с нуля, без такого обширного опыта (но, с другой стороны, и без легаси).

Вот уже как несколько лет наша команда успешно внедряет систему отчетности для одного из крупнейших транспортных холдингов нашей страны на базе российского ПО LuxMS. В этой статье расскажу про опыт нашего проекта.

НЕМНОГО ПРЕДЫСТОРИИ

Наш проект – это система отчетности, с ежедневным обновлением данных, созданная на базе импортозамещающей технологии LuxMS BI. Система

предназначена для разных групп пользователей, но ориентирована на топ-менеджмент заказчика. Географически она покрывает практически всю Россию.

Полностью готовая и работающая экосистема LuxMS BI позволила проводить полный цикл разработки системы отчетности: от экстракции данных до отображения их в виде различный форм визуализации. В отличие от аналогов, российская платформа дала быстро подстраивать визуальную часть под требования конкретного заказчика: достигается это путем фронтенд разработки в существующих визуальных сущностях BI-платформы – «дешах» (группах визуализаций).

НАСЛЕДИЕ РАЗРАБОТКИ: ДУБЛИРОВАНИЯ, ОБЪЕМНЫЙ ИМПОРТ, УСТАРЕВШИЙ API

Конечно, сейчас мы стремимся использовать только передовые технологии — но так было не всегда.

Изначально визуальная часть делалась на jQuery с использованием фреймов в одном деше, что рождало довольно большое количество проблем. Дублирование файлов, отсутствие «чистого» кода, большой объем файла для импорта, устаревший API и так далее. В целом было ясно, что jQuery не подходит для сложных пользовательских интерфейсов. Мы начали обсуждать рефакторинг.

Что для нас было важно:

1. Скорость работы. Мы обращали внимание на загрузку данных в приложение, на время отклика системы.
2. Повторное применение компонентов. Это сокращает время разработки.
3. Огромное сообщество. Это значит, есть, к которому можно обратиться за решением проблемы.
4. Браузерные инструменты, позволяющие выявлять баги и ошибки.
5. Легкий старт. Благодаря огромному количеству структурированной информации, обучение React не вызывает проблем.

После обсуждений для продолжения работы и рефакторинга существующих дешей мы выбрали React. Он полностью отвечал всем нашим критериям.

REACT: БЫСТРЕЕ, УДОБНЕЕ, МОЩНЕЕ

Для начала мы определили общие фрагменты нашего SPA.

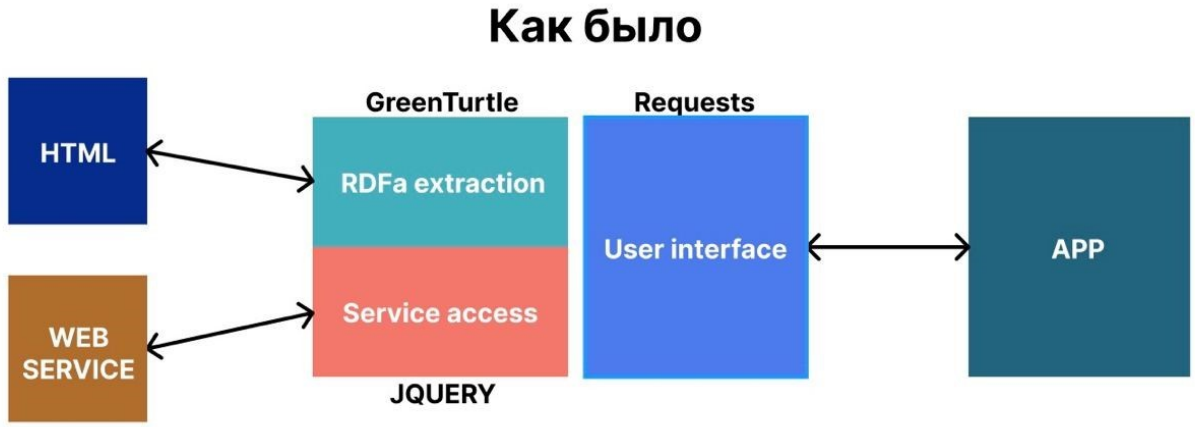
В отдельную верхнеуровневую среду мы написали общие компоненты, которые впоследствии переиспользовали.

Мы постарались разбить каждую визуализацию на множество мелких частей. Например прирост, линия, столбик — это отдельные компоненты, имеющие ряд внутренних настроек. Это было невозможно при старом подходе, так как

код становился нечитаемым.

Стоит отметить, что в своем проекте мы используем хуки, а не классовый подход. Использование функциональных компонентов проще компонента, основанного на классах. Для него не нужно создавать экземпляр класса и вызывать render() — достаточно просто вызвать нужную функцию. Также хуки упрощают работу с методами жизненного цикла компонента. При их использовании в компонентах, основанных на классах, приходится по-отдельности работать с методами componentDidMount(), componentDidUpdate(), componentWillUnmount(), а при использовании хуков можно просто использовать useEffect.

В свою очередь это позволило еще упростить код и оптимизировать SPA.



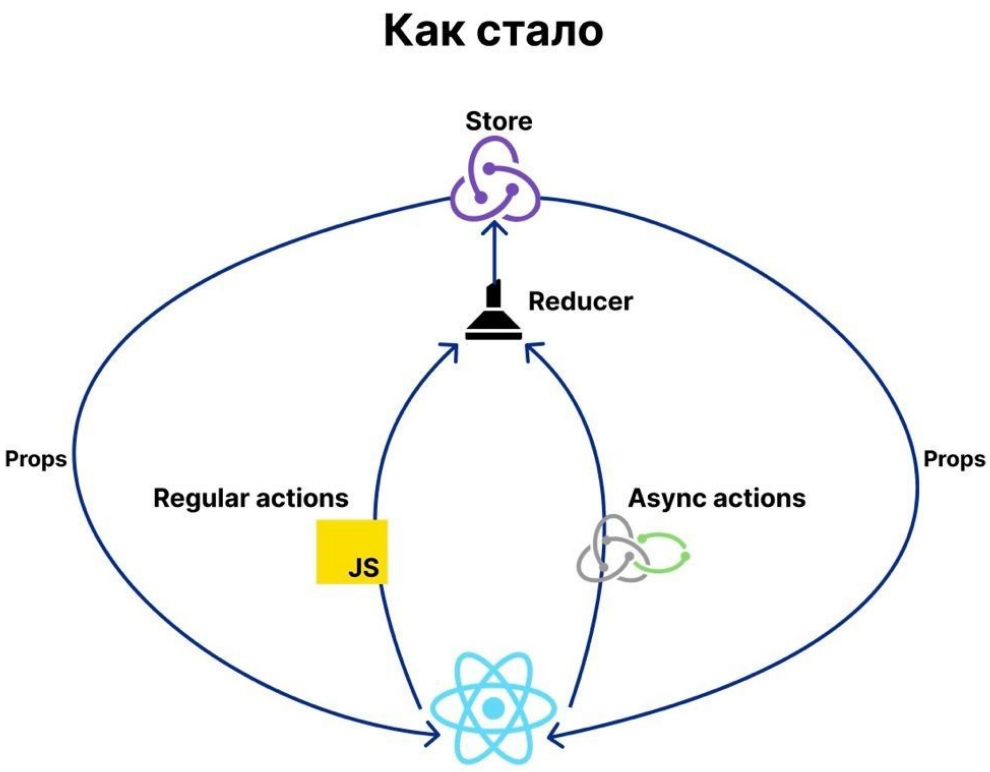
Мы также внедрили Recharts как наиболее подходящую библиотеку для построения классических графиков. Одно из основных преимуществ данной библиотеки – это возможность писать собственные компоненты к существующим компонентам верхнего порядка. Это позволило настраивать визуализацию, которую предлагает Recharts в точности как того требует

заказчик.

ПОЛНОЕ РЕВЬЮ ПРОЕКТА И РЕФАКТОРИНГ СТОИЛИ ТОГО

В целом использование React позволило уйти от концепции фреймов на одной странице. Также повысилась скорость работы BI приложения за счет Virtual DOM, который позволяет странице немедленно получать ответы от сервера и отображать обновления. Благодаря переиспользованию компонентов, мы смогли существенно сократить время разработки новых форм визуализации. А нисходящий поток данных существенно упростил отладку багов. Мы также добавили TypeScript для статической типизации и React-Redux для разделения логики и представления.

Мы не побоялись провести масштабное ревью всего проекта и потратить время на рефакторинг. Хотя в данной статье речь идет исключительно о frontend части, мы также обратили внимание на backend: переосмыслили недостатки, выявили особенности текущей реализации и оттолкнулись от потребностей проекта. Конечно, некоторые инструменты пришлось изучать — например Redux-Saga, где у нас не было опыта и экспертизы.



Благодаря масштабному рефакторингу мы добились реактивности и простоты нашего SPA и безусловно потраченное время стоило усилий. На текущем этапе развития мы имеем гораздо более удобный и современный продукт. Главный совет: не бойтесь рефакторинга. Это болезненный, но определенно оправданный опыт.