Оглавление

Глава 1

Отрисовка приложения на стороне сервера

Следующий шаг на пути изучения React - отрисовка на стороне сервера. **Изоморфные приложения (Universal applications)** хорошо в SEO (Search Engine Optimisations), а также позволяют обмениваться знаниями между фронтендом и бекендом.

Такие приложения могут уменьшить время между началом загрузки страницы и моментом, когда пользователь начинает воспринимать информацию на ней. Однако отрисовка на стороне сервера React приложений влечет за собой дополнительные расходы, поэтому мы должны понимать, когда нам действительно это необходимо.

В этой главе мы посмотрим, как настроить отрисовку на стороне сервера, а также разберемся в плюсах и минусах этого подхода.

В этой главе мы рассмотрим следующие вопросы:

- Что такое Изоморфное приложение
- Причины, по которым мы можем захотеть включить отрисовку на стороне сервера
- Создание простого React приложения с отрисовкой на стороне сервера
- Загрузка данных при отрисовке на стороне сервера и концепция dehydration/hydration

- Использование **Next.js** от Zeith для создания React приложения, которое будет запускаться и на сервере и на клиенте

1.1 Универсальные приложения

Когда мы говорим о JavaScript web приложениях, мы обычно думаем о коде, который выполняется в браузере пользователя.

Основной подход, по которому работают SPA приложения, это передача на клиент почти пустого HTML с тегом script, в котом будет загружен код приложения. Затем этот код уже напрямую взаимодействует с DOM в браузере для отображения UI пользователю. Таким образом приложения работали последние несколько лет, таким же образом множество из них продолжают работать сейчас.

В этой книге мы уже рассмотрели, как создавать приложения с помощью React компонент и как они работают в браузере. Но чего мы еще не касались, как React может создавать те же самые компоненты на стороне сервера, что называется **Отрисовкой на стороне сервера (Server-Side Rendering, SSR)**.

Перед тем, как мы углубимся в детали, давайте поймем, что же это в принципе означает, что приложение может отрисовываться и на сервере и на клиенте. Многие годы мы были вынуждены создавать отдельные приложения для сервера и клиента: например, Django приложение, которое создает страницы на сервере, и какие-то JavaScript фреймворки, такие как Backbone или jQuery, на клиенте.

В таком подходе требуется две команды разработчиков с различными наборами навыков. Если вам требовалось передавать данные между страницей, созданной на сервере, и JavaScript приложением, то вы могли вставлять специальные переменные в тэг *script*. При использовании двух различных языков программирования не было возможности делиться общей информацией, такой как модели или отображения, между различными частями приложения.

С момента релиза Node.js в 2009 было приложено немало усилий на укрепление позиций JavaScript в бекенд разработке, где немалую роль сыграли фреймоврки для создания web приложений, такие как, например, **Express**.

Использование одного и того же языка с обеих сторон позволяет не только упростить для разработчиков переиспользование их знаний, но

также открывает различные пути обмена кодом между клиентом и сервером.

В частности, с React концепция изоморфных приложений стала очень популярной в JavaScript сообществе.

Создание **изоморфного приложения** означает разработку приложения, которые будет выглядеть одинаково и на сервере и на клиенте.

Использование одного языка программирования открывает новые возможности для переиспользования логики. Также упрощается анализ кода и уменьшается количество дублирующегося кода.

React делает еще один шаг вперед, предоставляя удобное API для отрисовки компонент на сервере, а также сам выполняет все необходимые действия, чтобы сделать страницу интерактивной (например, обработчики событий) в браузере.

Термин изоморфный не очень хорошо подходит в данном случае, потому что в случае с React приложения именно одинаковые. По этой причине один из создателе React Router, *Майкл Джексон (Michael Jackson)*, предложил более значимое название для этого паттерна: *Универсальный (Universal)*

Универсальным приложением мы будем называть приложение, которое использует одинаковый код для запуска и на сервере и на клиенте.

В этой главе мы поговорим о том, почему мы должны рассматривать создание Универсальных приложений, и как React помогает отрисовывать компоненты на стороне сервера.

1.2 Мотивация к Отрисовке на стороне сервера

SSR - отличный инструмент, но не стоит использовать его только для того, чтобы использовать его, необходимо понимать, как этот инструмент может улучшить наше приложение и какие проблемы решить.

1.2.1 SEO

Одна из главных причин для отрисовки приложения на стороне сервера - **Поисковая оптимизация (Search Engine Optimization, SEO)**.

Проблема в том, что когда мы отдаем поисковому роботу (crawler) поисковой машины пустую HTML страницу, он не может вытащить из

нее значимую информацию.

На данный момент Google уже должен уметь запускать JavaScript при сканировании страниц, но все равно есть множество ограничений, а SEO является критическим фактором для множества бизнесов.

Последние годы мы были вынуждены писать два приложения: одно для поисковых роботов, которое отрисовывалось на сервере, а второе для клиента, которое работало в браузере.

Нам приходилось делать это, так как приложения, создаваемые на сервере не обладали достаточной интерактивностью, ожидаемой пользователями, а приложения, запускаемые в браузере, не индексировались поисковыми машинами.

Поддержка двух приложений вместо одного доставляет хлопот и делает код менее гибким и открытым к изменениям.

К счастью, с React мы можем отрисовывать приложения на стороне сервера и отдавать их поисковым роботам в удобном для индексации виде.

Также это влияет на отображение ссылок на наши страницы в социальных сетях, где как правило показывается превью страниц, которыми делятся пользователи.

Например, с помощью Open Graph мы можем сказать Facebook, что для определенной страницы должны быть показаны определенные изображение и заголовок.

Но сделать это без SSR невозможно, так как поисковые машины извлекают информацию из страниц, полученных от сервера. А если наш сервер присылает по всем URL пустую страницу, то и превью всех наших страниц будет пустым.

1.2.2 Общая кодовая база

У нас не много вариантов при разработке web приложений: мы можем использовать JavaScript, или вместо него можем использовать JavaScript. Конечно, есть языки программирования, которые компилируются в JavaScript, но принципиально картину это не меняет.

Возможность использования такого же языка программирования на сервере открывает новые возможности поддержки приложений и обмена знаниями внутри компании.

Реализация общем логики для клиента и сервера на одном языке программирования упрощает внесение изменений в эту логику, так как не

придется реализовывать их дважды разными командами, что ведет к уменьшению количества ошибок в коде.

Усилия, которые необходимо затратить для поддержки одного проекта, будут значительно меньше усилий по актуализации двух различных приложений.

Возможность использования одного языка упрощает сотрудничество между командами, так как они находятся в одном информационном пространстве, что ускоряет принятие решений и внесение изменений.

1.2.3 Повышение производительности

Мы любим SPA приложения за то, что они быстры и отзывчивы, но есть одна проблема: перед тем как пользователь сможет сделать что-либо на сайте должен быть загружен и запущен бандл с приложением.

Это может не быть проблемой для пользователей с быстром интернет соединением, но для пользователей с 3G соединением время ожидания станет ощутимым. В целом это не проблема UX, но это также влияет на конверсию. Крупными e-commerce сайтами было доказано, что изменение времени загрузки в большую или меньшую сторону даже на несколько миллисекунд может значительно влиять на прибыль.

Например, если мы отдаем с сервера пустую HTML страницу с тегом *script*, которая показывает индикатор загрузки до момента, когда пользователь сможет полноценно взаимодействовать с сайтом, то восприятие производительности сайта пользователем будет искажено не в лучшую для нас сторону.

Если же мы отрисовываем приложение на стороне сервера, и пользователь видит часть контента сразу после загрузки страницы, то он с большой вероятностью останется на странице, даже если для полноценной работы ему придется ждать то же самое время, так как загрузку бандла с приложенем никто не отменял.

Таким образом мы можем значительно улучшить восприятие пользователем производительности сайта за счет того, что будем часть контента отдавать вместе со страницей.

1.2.4 Не все так просто

Очевидно, что даже с удобным API для создания Универсальных приложений, создание таких приложений будет требовать дополнительных

усилий. Таким образом, перед тем, как включать отображение на сервере, нужно убедиться, что у нас есть причины для этого, а наша команда готова это сделать.

Как мы увидим дальше, отрисовка React компонент - не единственная проблема, которую нужно решить для полноценной отрисовки на стороне сервера.

Нас ожидает настройка и поддержка сервера с роутингом, управление потоком данных, кеширование контента, чтобы быстрее отдавать данные, и множество других интересных задач на пути создания полноценного Универсального приложения.

По этим причинам я советую начинать с приложения, которое полностью работоспособно без поддержки сервера, а затем оценивать плюсы и минусы перехода на SSR.

Включать отрисовку на стороне сервера стоит при жесткой необходимости в этом. Например, если вам нужна оптимизация для поисковых машин или кастомизация отображения для социальных сетей.

Если вы осознали, что ваше приложение загружается слишком долго, а все оптимизации (подробнее об оптимизациях мы поговорим в следующей главе) уже применены, вы можете решить использовать SSR для уменьшения времени между началом загрузки страницы и первого взаимодействия пользователя с контентом.

Кристофер Пожер(Christopher Pojer), инженер Facebook, сказал в Twitter, что в приложении Instagram они используют SSR только в целях SEO. Он сказал, что для приложений с динамическим контентом, как например Instagram, использовать SSR в целях улучшения восприятия пользователя не эффективно:

$$https://twitter.com/cpojer/status/711729444323332096$$

1.3 Простой пример

Пришло время создать простое приложение с отрисовкой на стороне сервера и посмотреть, какие шаги нужно выполнить для создания Универсального приложения.

Мы создадим минимальное и простое приложение, так как наша цель - посмотреть, как работает SSR, а не предоставить комплексное решение или шаблон для проекта.

В данном разделе мы предполагаем, что вы знакомы с концепциями сборки JavaScript приложения и такими инструментами как Webpack. Также будет полезен опыт работы с Node.js, но даже без него, со знанием JavaScript, вы сможете разобраться в этой главе.

Приложение будет состоять из двух частей:

- Серверную часть, в которой мы будем использовать *Express* для создания сервера, который будет отдавать HTML страницу с отрисованным React приложением
- Клиентскую часть, где мы будем отображать приложение привычным для нас образом с помощью $react - dom$.

1.4 A data fetching example

1.5 Next.js

1.6 Заключение