**Министерство цифрового развития, связи и массовых  
коммуникаций Российской Федерации  
Ордена Трудового Красного Знамени  
федеральное государственное бюджетное  
учреждение высшего образования  
«Московский Технический Университет Связи и Информатики»**

**Реферат**по дисциплине «Технологии программирования Web-приложений»  
на тему «Переход на технологию SSR для типового React приложения: рекомендации и практические шаги»

Выполнил: студент гр. М092401(75) Савкин Д.И.  
Проверил: к.т.н., Гузеев А.В.

Москва 2025

**Содержание**

[Введение 3](#_Toc200241140)

[Теоретические основы 4](#_Toc200241141)

[Подготовка и миграция типового React-приложения 6](#_Toc200241142)

[Внесений изменений в исходный код 7](#_Toc200241143)

[Профилирование результатов 11](#_Toc200241144)

[Заключение 13](#_Toc200241145)

[Список использованных источников 14](#_Toc200241146)

# Введение

Современные веб-приложения, построенные на React, традиционно используют клиентский рендеринг (CSR), при котором браузер загружает минимальный HTML-файл, а затем динамически отрисовывает интерфейс с помощью JavaScript. Однако такой подход имеет ряд ограничений, особенно в вопросах SEO, скорости первоначальной загрузки и пользовательского опыта на слабых устройствах.

Server-Side Rendering (SSR) предлагает альтернативное решение: страницы генерируются на сервере и отправляются клиенту в готовом виде. Это позволяет поисковым системам корректно индексировать контент, сокращает время до появления первого контента и улучшает восприятие сайта пользователями. Для React-приложений SSR особенно актуален в проектах с высокими требованиями к SEO, таких как маркетплейсы, медиаплатформы и корпоративные сайты.

Переход на SSR требует пересмотра архитектуры приложения, настройки серверного рендеринга компонентов и обработки данных. Популярные фреймворки, такие как Next.js, значительно упрощают эту задачу, но даже при их использовании важно учитывать особенности гидратации, управления состоянием и производительности. В этой статье мы разберём ключевые преимущества SSR, основные сценарии его применения и практические аспекты внедрения в существующее React-приложение/

Целью данной работы является практическое внедрение поддержки SSR в существующее CSR веб-приложение.

# Теоретические основы

Server-Side Rendering — это метод рендеринга веб-приложений, при котором HTML-страницы формируются на сервере и передаются клиенту в готовом виде, в отличие от клиентского рендеринга, где основная работа выполняется в браузере с помощью JavaScript.

Когда пользователь запрашивает страницу, сервер выполняет следующие шаги:

1. обработка запроса — сервер получает URL и определяет, какой компонент или страницу нужно отрендерить;
2. загрузка данных — если страница требует данных (например, API-запросы), сервер выполняет их получение до рендеринга;
3. генерация HTML — React-компоненты рендерятся в строку HTML с помощью методов вроде renderToString();
4. отправка клиенту — браузер получает готовый HTML, что ускоряет первоначальную отрисовку; после загрузки JavaScript происходит гидратация, т.е. процесс «оживления» статичного HTML, добавления интерактивности.

Ключевые отличия SSR от CSR:

1. производительность — SSR сокращает время до первого контента (FCP), но может увеличить Time to Interactive (TTI) из-за гидратации;
2. SEO — поисковые системы лучше индексируют серверно-отрендеренный контент;
3. архитектура — SSR требует серверной инфраструктуры (Node.js, Deno, Bun), тогда как CSR работает на статическом хостинге.

SSR особенно полезен для контент-ориентированных сайтов, где важны скорость загрузки и SEO, но может быть избыточен для сложных SPA с интенсивной клиентской логикой.

Server-Side Rendering наиболее эффективен в проектах, где критически важны скорость первоначальной загрузки и поисковая оптимизация. Типичный пример — новостные порталы и медиаресурсы, где важно быстро показать контент пользователю и обеспечить его корректное индексирование поисковыми системами. В таких случаях SSR позволяет сразу отдавать готовый HTML с актуальным содержимым, не дожидаясь выполнения JavaScript в браузере.

Корпоративные сайты и лендинги также выигрывают от использования SSR, поскольку первое впечатление пользователя напрямую влияет на конверсию. Когда посетитель сразу видит контент, а не пустой экран с индикатором загрузки, вероятность его ухода существенно снижается. Особенно это актуально для пользователей с медленным интернет-соединением или слабыми мобильными устройствами.

Интернет-магазины представляют ещё один важный сценарий применения. Для них критична не только скорость загрузки товарных карточек, но и их корректное отображение в социальных сетях. SSR гарантирует, что превью-карточки будут содержать актуальные данные о товаре, так как они формируются на сервере в момент запроса.

При этом SSR может быть менее оправдан для сложных веб-приложений с интенсивной клиентской логикой, таких как административные панели или онлайн-редакторы. В таких случаях издержки серверного рендеринга могут перевесить преимущества, а клиентский рендеринг обеспечит более плавную работу интерактивных элементов. Выбор между SSR и CSR должен основываться на анализе конкретных требований проекта и поведения целевой аудитории.

# Подготовка и миграция типового React-приложения

Внедрение SSR в существующее React-приложение требует комплексного подхода и затрагивает различные уровни архитектуры проекта. Первостепенное значение имеет настройка серверной инфраструктуры, так как традиционное SPA, раздаваемое как статические файлы, теперь нуждается в Node.js сервере или альтернативном окружении для выполнения рендеринга. Это влечёт за собой необходимость обеспечения отказоустойчивости, балансировки нагрузки и грамотного кэширования.

Одной из ключевых технических задач становится организация универсального (изоморфного) кода, который может выполняться как на сервере, так и на клиенте. Особого внимания требует обработка данных - запросы к API должны выполняться на сервере во время рендеринга, но при этом избегать дублирования при последующей гидратации на клиенте. Для управления состоянием приложения приходится пересматривать подходы к хранению и синхронизации данных между сервером и клиентом.

Процесс гидратации представляет собой отдельную сложность - необходимо обеспечить точное соответствие между серверным HTML и клиентским React-деревом, иначе React будет вынужден полностью пересоздавать DOM, теряя преимущества SSR. Особенно проблематичными могут быть компоненты, зависящие от браузерного API или window-объекта, которые требуют специальной обработки или условного рендеринга.

Производительность серверного рендеринга требует особого контроля, так как каждый запрос приводит к выполнению JavaScript на сервере. Это делает необходимым внедрение стратегий кэширования, оптимизацию сборки и возможно, использование потокового рендеринга для крупных приложений. При этом важно соблюдать баланс между скоростью ответа сервера и актуальностью данных.

Интеграция с существующей инфраструктурой — ещё один важный аспект. Если приложение использует Redux, Router или другие популярные библиотеки, их необходимо адаптировать для работы в SSR-режиме. Также стоит учитывать особенности работы с аутентификацией, cookies и заголовками при серверном рендеринге.

Развёртывание SSR-приложения усложняется по сравнению с традиционным SPA — требуется настройка серверного окружения, мониторинг выполнения рендеринга, обработка ошибок и их корректное отображение как на сервере, так и на клиенте. Все эти факторы необходимо учитывать при планировании перехода на SSR.

В качестве объекта интеграции было выбрано существующее React решение с открытым исходным кодом в связке с системой сборки Bun, позволяющее вести электронный список задач.

Корневая директория проекта содержит множество файлов и дочерних директорий, однако лишь их некоторое подмножество требует изменений.

## Внесений изменений в исходный код

Входная точка сервера (это показано на рисунке ниже) представляет собой вызов функции serve, входящую в стандартный набор API используемой системы сборки Bun. Входная точка предустанавливает маршрутизацию, а также выставляет необходимые флаги, которые позволяют ускорить процесс разработки (возможность «горячей» замены файлов и перенаправление сообщений из консоли браузера в серверную консоль).

Точка маршрутизации «/» позволяет элегантно вернуть клиенту объект index, который был сформирован в процессе импортирования файла index.html. Процесс импортирования представляет собой комплексный процесс, который Bun производит автоматически, однако понимание ключевых шагов данного процесса критически необходимо для проведения дальнейшей интеграции.

Так, импортирование файла index.html подразумевает:

1. считывание и парсинг гипертекстового файла;
2. формирование таблицы ссылок на внешние файлы (теги <link>, <script> и др.);
3. парсинг и подготовка файлов-зависимостей согласно сформированной таблицы;
4. компиляция скриптовых файлов-зависимостей.



Рисунок 1 — Входная точка сервера

Поскольку SSR подразумевает снимает с клиента нагрузку наполнения страницы, то финальный гипертекст должен быть сформирован непосредственно на стороне сервера. React позволяет произвести рендеринг (формирование) гипертекста путём вызова функций renderToString() или renderToReadableStream(). Поскольку последняя позволяет сформировать поток символов, которые могут быть отправлены клиенту независимо друг от друга, что непосредственно влияет на производительность скорости загрузки страницы, следует использовать именно эту функцию, в честности, когда страница наполнена «тяжёлым» контентом и вынуждать клиент ожидать полного рендеринга страницы нецелесообразно.

Описанные выше функции принимают в качестве аргумента экземпляр JSX-объекта, что лишает возможности использовать автоматизацию разыменования и подготовки файлов-зависимостей, которые происходили в процессе импортирования. Таким образом, необходимо произвести трансляцию содержимого файла index.html в JSX-объект. Это показано на рисунке ниже.



Рисунок 2 — JSX-объект страницы

Основной издержкой JSX-объектов является невозможность использовать тег <script>, что подразумевает необходимость использовать дополнительные параметры при вызове функции renderToReadableStream(). Это показано на рисунке ниже.



Рисунок — Рендеринг страницы вручную

Указание конкретного файла JavaScript не позволяет решить возникшую проблему, поскольку данный файл формируется на этапе импортирования index.html автоматически. Так, необходимо выполнить этап подготовки файлов-зависимостей самостоятельно, путём вызова функции build, входящую в стандартный набор API используемой системы сборки Bun. Это показано на рисунке ниже.

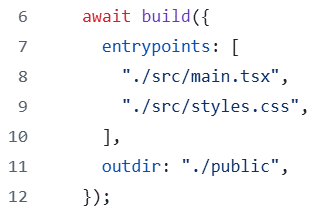


Рисунок — Подготовка файлов-зависимостей

Bun позволяет автоматически произвести этап компиляции файлов сценария (и любых других файлов) и сохранить полученный результат в указанную директорию. Чтобы клиент имел доступ к подготовленным файлом, необходимо выполнить соответствующую маршрутизацию к файлу на диске и вернуть его содержимое. Это показано на рисунке ниже.

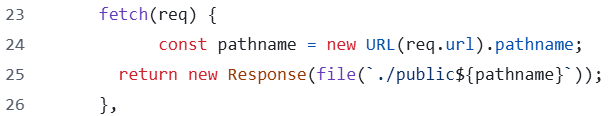


Рисунок — Маршрутизация файлов-зависимостей

## Профилирование результатов

Обращение к серверу до проведения миграции показано на рисунке ниже.



Рисунок 6 — Обращение к серверу (CSR)

Дерево файлов слева представляет собой индексный файл, а также файл стилей и файл сценария. Последние два файла находятся в логических дочерних директориях, который Bun создал автоматически. Имена файлов представляют собой шаблон, который был заполнен системой сборки случайным образом, чтобы исключить возможную коллизию имён. Гипертекстовый элемент root пуст, что означает необходимость клиента заполнить его содержимое собственными вычислительными средствами.

Обращение к серверу после проведения миграции показано на рисунке ниже.



Рисунок — Обращение к серверу (SSR)

Дерево файлов слева представляет также представляет собой индексный файл, а также файл стилей и файл сценария. Однако, все файлы находятся на одном уровне иерархии, что является результатом работы самописного маршрутизатора файлов. Имена файлов остались неизменными, что исключает любую автоматическую предварительной обработку за исключением тех, которые описаны в исходном коде. Гипертекстовый элемент root не пуст, что является признаком рендеринга индексного файла на стороне сервера.

# Заключение

Server-Side Rendering представляет собой мощный инструмент оптимизации React-приложений, особенно актуальный в условиях современных требований к веб-разработке. Его внедрение позволяет решить ключевые проблемы клиентского рендеринга, значительно улучшая показатели SEO и пользовательского опыта за счёт мгновенной отдачи контента. Однако переход на SSR требует тщательного анализа архитектурных решений и сопряжён с рядом технических сложностей, начиная от организации универсального кода и заканчивая оптимизацией процессов гидратации.

Ссылка на репозиторий: https://github.com/wkns37/todo-ssr

# Список использованных источников

1. React API Reference [Электронный ресурс] // React : [официальная документация]. URL: https://react.dev/reference/react (дата обращения: 01.06.2025).

2. Кейтс, А. Server-Side Rendering (SSR) with Bun and React [Электронный ресурс] // Блог Алекса Кейтса. URL: https://www.alexkates.dev/blog/server-side-rendering-ssr-with-bun-and-react (дата обращения: 01.06.2025).

3. Чейз, Р. Diving into SSR with Bun and React 19 [Электронный ресурс] // Блог Рейли Чейза. URL: https://blog.reilly.dev/diving-into-ssr-with-bun-and-react-19 (дата обращения: 01.06.2025).

4. Server-side rendering with React [Электронный ресурс] // Bun : [официальное руководство]. URL: https://bun.sh/guides/ecosystem/ssr-react (дата обращения: 01.06.2025).

5. SSR на Bun и React с нуля [Электронный ресурс] // Хабр. URL: https://habr.com/ru/articles/761756/ (дата обращения: 01.06.2025).