МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Российский государственный университет им. А.Н. Косыгина

(Технологии. Дизайн. Искусство)»

Кафедра автоматизированных систем обработки информации и управления

Отчет об учебной практике

Тема: «Способы рендеринга веб-приложений: CSR, SSG, SSR. Сравнительный

анализ и примеры использования»

Выполнили: группа ИТИВ-223,

Сидоров Д. С., Петрушенко Д. А, Шульга Е. Е.

Проверила: Федина Л. А.

Москва 2024

**Содержание**

[Введение 3](#_Toc168662634)

[Рендеринг веб-приложений 5](#_Toc168662635)

[Что такое рендеринг и как он работает 5](#_Toc168662636)

[Рендеринг на стороне клиента (CSR) 7](#_Toc168662637)

[Рендеринг на стороне сервера (SSR) 10](#_Toc168662638)

[Статическая генерация сайта (SSG) 13](#_Toc168662639)

[Сравнительный анализ методов 16](#_Toc168662640)

[Заключение 18](#_Toc168662641)

[Список использованных источников литературы 20](#_Toc168662642)

[ПРИЛОЖЕНИЯ 21](#_Toc168662643)

[ПРИЛОЖЕНИЕ 1 21](#_Toc168662644)

[ПРИЛОЖЕНИЕ 2 22](#_Toc168662645)

# **Введение**

В эпоху информационных технологий и интернета веб-приложения стали важной частью жизни и бизнеса. Они лежат в основе онлайн-сервисов, таких как социальные сети, новостные порталы, интернет-магазины и корпоративные системы. Ключевым аспектом разработки веб-приложений является выбор метода рендеринга, который влияет на доставку контента и скорость взаимодействия пользователя с приложением.

Существует несколько основных подходов к рендерингу веб-приложений: рендеринг на стороне клиента (Client-Side Rendering, CSR), рендеринг на стороне сервера (Server-Side Rendering, SSR) и статическая генерация сайтов (Static Site Generation, SSG). Каждый из этих методов имеет свои особенности, преимущества и недостатки, а также различные сценарии использования. Понимание этих различий и умение выбирать подходящий метод для конкретного проекта является важным навыком для современных веб-разработчиков.

Рендеринг на стороне клиента (CSR) предполагает, что основной объем работы по обработке и отображению данных выполняется на устройстве пользователя. Этот метод позволяет создавать высокоинтерактивные приложения, однако может привести к задержкам при первом загрузке страницы и ухудшению производительности слабых устройствах.

Рендеринг на стороне сервера (SSR) предполагает, что HTML-код генерируется на сервере и отправляется пользователю в готовом виде. Этот подход обеспечивает быструю первую загрузку страницы и улучшенную индексацию поисковыми системами, но может увеличивать нагрузку на сервер и требовать более сложной архитектуры приложения.

Статическая генерация сайтов (SSG) объединяет преимущества CSR и SSR, создавая HTML-страницы на этапе сборки, которые затем могут быть быстро доставлены пользователю. Этот метод обеспечивает высокую производительность и безопасность, однако требует предварительной генерации контента, что может быть проблематичным для часто обновляемых сайтов.

Современные веб-приложения стали более сложными и интерактивными, что увеличило требования к скорости и эффективности рендеринга. Пользователи ожидают, что приложения будут загружаться быстро, работать плавно и реагировать мгновенно. Разработчики, в свою очередь, стремятся улучшить пользовательский опыт путем оптимизации процессов рендеринга. Это включает в себя использование сжатия файлов, кэширования, асинхронной загрузки ресурсов и других методов для ускорения загрузки и уменьшения задержек.

Цель данной учебной практики - провести сравнительный анализ методов рендеринга веб-приложений CSR, SSR и SSG, исследовать их особенности и области применения, а также рассмотреть примеры использования каждого из них на практике. В ходе выполнения практической работы студенты смогут закрепить теоретические знания, научиться выбирать оптимальный способ рендеринга для различных типов веб-приложений и приобрести навыки работы с современными инструментами и технологиями веб-разработки.

# **Рендеринг веб-приложений**

Что такое рендеринг и как он работает

Рендеринг веб-приложений — это процесс преобразования данных и логики, хранящихся на сервере или в браузере, в пользовательский интерфейс, который отображается на экране устройства пользователя. Это ключевой аспект веб-разработки, определяющий, как информация будет представлена пользователю и как быстро он сможет взаимодействовать с приложением.

Основная цель рендеринга — создание HTML-кода, который браузер может отобразить в виде веб-страницы. Этот процесс включает несколько этапов:

1. **Запрос данных**: когда пользователь вводит URL в адресную строку браузера или кликает по ссылке, отправляется HTTP-запрос к серверу. Сервер обрабатывает этот запрос и отправляет ответ, содержащий необходимые данные.
2. **Обработка данных**: после получения данных от сервера начинается процесс их обработки. Эти данные могут включать HTML, CSS и JavaScript, а также различную информацию, необходимую для построения страницы, такую как текст, изображения и другие медиа-файлы.
3. **Создание DOM-дерева**: Браузер анализирует HTML-документ и создает DOM-дерево (Document Object Model), представляющее структуру веб-страницы в виде дерева объектов. Это позволяет динамически изменять содержимое и структуру страницы с помощью JavaScript.
4. **Применение стилей**: Браузер загружает и обрабатывает CSS-файлы, которые содержат правила стилей, определяющие внешний вид элементов на странице. Эти стили применяются к DOM-элементам, формируя CSSOM (CSS Object Model).
5. **Построение рендер-дерева**: Браузер объединяет DOM и CSS, создавая рендер-дерево, которое содержит информацию как о структуре страницы, так и о ее визуальном оформлении.
6. **Визуализация**: на основе рендер-дерева браузер рисует элементы на экране, отображая веб-страницу пользователю. Этот процесс включает в себя расчет расположения и размера каждого элемента, а также применение всех стилей и эффектов.

Рендеринг веб-приложений является комплексным процессом, в котором участвуют различные компоненты и технологии. Эффективный рендеринг способствует быстрому времени загрузки страницы, улучшенному пользовательскому опыту и хорошей производительности приложения.

Современные веб-приложения могут использовать различные техники и оптимизации для улучшения рендеринга, такие как сжатие файлов, кэширование, асинхронная загрузка ресурсов и многое другое. Все эти методы направлены на то, чтобы предоставить пользователям быстрый и плавный доступ к веб-контенту, независимо от устройства и качества интернет-соединения.

Помимо технических аспектов, рендеринг веб-приложений также учитывает пользовательский опыт и удобство взаимодействия с приложением. Важными факторами являются скорость отклика, плавность анимаций, адаптивность интерфейса и доступность для пользователей с ограниченными возможностями. Всё это требует от разработчиков глубокого понимания как клиентских, так и серверных технологий, а также умения балансировать между производительностью и качеством пользовательского интерфейса.

Рендеринг веб-приложений является фундаментальной частью веб-разработки, которая напрямую влияет на восприятие и удовлетворенность пользователей. Эффективный рендеринг позволяет создавать высокопроизводительные, интерактивные и привлекательные веб-приложения, которые могут удовлетворить растущие требования современного интернет-пользователя.

Рендеринг на стороне клиента (CSR)

Client-Side Rendering (CSR) — это подход к рендерингу веб-страниц, при котором весь процесс обработки и отображения контента происходит на стороне клиента, то есть в браузере пользователя. Основная идея CSR заключается в том, что сервер передает браузеру минимальный HTML-код вместе с необходимыми JavaScript-файлами. JavaScript-код, загруженный в браузер, берет на себя всю дальнейшую работу по генерации и отображению контента [7].

**Когда пользователь запрашивает страницу:**

1. Браузер отправляет запрос к серверу.
2. Сервер отвечает основным HTML-файлом, который содержит минимальную разметку и ссылки на JavaScript-ресурсы.
3. Браузер загружает и выполняет JavaScript-код.
4. JavaScript запрашивает необходимые данные (например, через API), обрабатывает их и рендерит финальный HTML.

**Основные преимущества:**

1. Быстрая интерактивность: Пользовательский интерфейс становится интерактивным сразу после загрузки JavaScript, что улучшает пользовательский опыт [2].
2. Разгрузка сервера: Основная нагрузка по рендерингу переносится на клиент, что снижает нагрузку на сервер.
3. Динамическое обновление контента: Возможность обновлять только часть страницы без перезагрузки всей страницы (например, при помощи AJAX или Fetch API).
4. Гибкость в разработке: Современные JavaScript-фреймворки и библиотеки, такие как React, Angular и Vue.js, предоставляют мощные инструменты для разработки сложных и динамичных интерфейсов.

**Ключевые недостатки:**

1. Проблемы с SEO: Поисковые роботы могут иметь трудности с индексацией контента, который рендерится на стороне клиента.
2. Задержка начальной загрузки: Потребность в загрузке и выполнении большого количества JavaScript-кода может привести к увеличению времени начальной загрузки страницы.
3. Зависимость от браузера: Выполнение кода полностью зависит от производительности и возможностей браузера пользователя, что может негативно сказаться на опыте пользователей со слабыми устройствами.
4. Сложность в отладке: Процесс отладки может быть сложнее из-за динамической природы CSR и необходимости отслеживания состояний на стороне клиента.

CSR часто используется в тех случаях, когда важна высокая интерактивность и динамичность интерфейса. Некоторые примеры включают:

1. Одностраничные приложения (SPA): Веб-приложения, которые работают как единая страница и динамически подгружают контент.
2. Веб-приложения с богатым пользовательским интерфейсом: панели управления, дашборды, социальные сети.
3. Приложения с частыми изменениями данных в реальном времени: Чаты, онлайн-игры, системы мониторинга.

Примеры использования во фреймворках:

1. **React** - популярная библиотека для создания пользовательских интерфейсов. Она позволяет создавать компоненты, которые автоматически обновляются при изменении данных. В React используется виртуальный DOM для оптимизации перерисовок.
2. **Angular** - мощный фреймворк от Google для разработки веб-приложений. Он предоставляет множество встроенных функций, таких как двухстороннее связывание данных, зависимостная инъекция и роутинг.
3. **Vue.js** - прогрессивный JavaScript-фреймворк, который можно использовать для создания пользовательских интерфейсов. Он фокусируется на декларативном рендеринге и компонентной архитектуре.

Client-Side Rendering представляет собой современный подход к рендерингу веб-страниц, предоставляющий высокую интерактивность и гибкость в разработке. Однако, он также имеет свои недостатки, такие как проблемы с SEO и зависимость от производительности клиентских устройств. Выбор между CSR и другими методами рендеринга (такими как Server-Side Rendering или Static Site Generation) должен основываться на специфических требованиях проекта и ожиданиях пользователей.

Код примера создания CSR представлен в приложении 1. Для работы примера на ПК должна быть установлена node.js и npm, чтобы запустить пример нужно создать новое приложение через npx create-react-app, а потом заменить html файл в static папке файлом из примера, а также заменить файл app.js в папке src. После запуска проекта, в браузере будет отображён сайт, который был отображён на клиенте.

Рендеринг на стороне сервера (SSR)

Server-Side Rendering (SSR) — это подход к рендерингу веб-страниц, при котором контент генерируется на стороне сервера перед отправкой клиенту. В этом случае сервер обрабатывает все необходимые данные и создает полный HTML-документ, который затем отправляется браузеру пользователя. Браузер отображает уже готовую страницу, что позволяет пользователю увидеть контент быстрее, чем при Client-Side Rendering (CSR).

**Когда пользователь запрашивает страницу:**

1. Браузер отправляет запрос к серверу.
2. Сервер обрабатывает запрос, извлекает данные (например, из базы данных), генерирует HTML-документ и отправляет его обратно клиенту.
3. Браузер получает и отображает готовую HTML-страницу.
4. После этого загружается и выполняется JavaScript, обеспечивая интерактивность страницы.

**Преимущество подхода:**

1. Улучшенная SEO: Поскольку контент уже готов при загрузке страницы, поисковые роботы могут легко его индексировать, что улучшает SEO [3].
2. Быстрая начальная загрузка: Пользователь получает полный HTML-документ, что позволяет быстрее отобразить контент, особенно на медленных соединениях или слабых устройствах.
3. Предсказуемость: Серверное рендеринг обеспечивает предсказуемую структуру HTML, что упрощает отладку и тестирование.

**Недостатки подхода:**

1. Нагрузка на сервер: Сервер должен обрабатывать каждый запрос и генерировать HTML, что увеличивает нагрузку на сервер и может требовать более мощного оборудования или оптимизации [8].
2. Меньшая интерактивность на начальном этапе: До загрузки и выполнения клиентского JavaScript взаимодействие с некоторыми элементами страницы может быть ограничено [8].
3. Сложность реализации: Необходимость поддержки как серверного, так и клиентского рендеринга может усложнить архитектуру приложения.

**Области применения:**

1. Информационные сайты и блоги: Где важна быстрая начальная загрузка и хорошая индексируемость контента.
2. Электронная коммерция: Быстрая загрузка страниц продуктов и улучшение SEO для привлечения трафика из поисковых систем.
3. Маркетинговые страницы и лендинги: Для предоставления пользователю мгновенного доступа к информации и улучшения конверсии.

**Примеры использования:**

1. Next.js — это фреймворк для React, который поддерживает SSR из коробки. Он позволяет разработчикам создавать приложения, которые могут рендериться как на сервере, так и на клиенте.
2. Angular Universal - это технология, позволяющая рендерить Angular-приложения на сервере. Это улучшает производительность и SEO, предоставляя пользователям готовый HTML.
3. Nuxt.js — это фреймворк для Vue.js, который поддерживает SSR. Он упрощает создание серверно-рендеринг приложений на Vue.js.

Server-Side Rendering — это мощный метод рендеринга веб-страниц, который обеспечивает быструю начальную загрузку и улучшенную SEO. Однако, он также имеет свои недостатки, такие как повышенная нагрузка на сервер и сложность реализации. Выбор между SSR и другими методами рендеринга (такими как CSR или Static Site Generation) должен основываться на потребностях проекта и ожиданиях пользователей. SSR особенно полезен для сайтов, где важна быстрая начальная загрузка и хорошая индексируемость контента.

Код примера создания SSR представлен в приложении 2. Для работы примера на ПК должна быть установлена node.js и npm, чтобы запустить пример нужно создать новое приложение через npx create-next-app, а потом заменить файл index.js в папке pages. Запустить приложение можно командой npm run dev. Контент, который отображается в браузере был заранее отрисован сервером, а потом возвращён для показа.

Статическая генерация сайта (SSG)

Static Site Generation (SSG) — это подход к рендерингу веб-страниц, при котором контент генерируется на стороне сервера во время сборки приложения, а не при каждом запросе. В этом случае сервер предварительно создает статические HTML-файлы для каждой страницы сайта, которые затем могут быть кэшированы и доставлены быстро клиенту.

**При процессе сборки:**

1. Сервер обрабатывает и анализирует исходные файлы приложения, такие как шаблоны и данные.
2. Для каждой страницы генерируется соответствующий HTML-файл с помощью данных из исходных файлов.
3. Полученные статические HTML-файлы хранятся в каталоге, доступном для сервера.

**При запросе страницы:**

1. Браузер отправляет запрос к серверу.
2. Сервер отдает соответствующий статический HTML-файл.
3. Браузер получает и отображает готовую HTML-страницу без необходимости выполнения дополнительных запросов на сервере

**Преимущества:**

1. Высокая производительность: Статические HTML-файлы могут быть кэшированы и доставлены быстро, что улучшает производительность и снижает нагрузку на сервер [4].
2. Безопасность: так как сайт состоит из статических файлов, вероятность атак снижается.
3. Простота хранения и развёртывания: статические сайты могут быть размещены на любом хостинге, не требующем серверного окружения.
4. Хорошая индексируемость проекта: поскольку страницы уже сгенерированы на сервере, они легко индексируются поисковыми системами.

**Недостатки:**

1. Ограниченная динамичность. Из-за того, что контент генерируется во время сборки, сложно обновлять информацию динамически, например, в реальном времени.
2. Сложность в обновлении контента: для обновления контента на сайте может потребоваться пересборка и повторная загрузка всего приложения.
3. Необходимость повторной сборки при изменениях: любое изменение контента или дизайна требует повторной сборки всего сайта, что может быть затратным по времени.

**Области применения:**

1. Статические сайты и блоги: где контент изменяется редко и может быть предварительно сгенерирован перед развертыванием.
2. Онлайн-магазины с ограниченным ассортиментом: где товары редко меняются и не требуют частого обновления страниц.
3. Приложения с незначительными динамическими компонентами: где основная часть контента статична, а динамические элементы можно загружать при помощи JavaScript.

**Примеры использования:**

1. Next.js с статическим экспортом. Данный фреймворк поддерживает статический экспорт, который позволяет предварительно сгенерировать все страницы вашего приложения в виде статических HTML-файлов во время сборки.
2. Gatsby.js - это фреймворк для создания статических сайтов и приложений, который использует SSG. Он позволяет создавать сайты с высокой производительностью и отличной индексируемостью.
3. VuePress - это генератор статических сайтов, основанный на Vue.js. Он предоставляет простую и интуитивно понятную структуру для создания статических сайтов и документации.

Static Site Generation представляет собой мощный метод рендеринга веб-страниц, который обеспечивает высокую производительность и хорошую индексируемость контента. Однако, он также имеет свои ограничения, такие как ограниченная динамичность и необходимость повторной сборки при изменениях. Выбор между SSG и другими методами рендеринга (такими как SSR или CSR) должен основываться на спецификации проекта и требованиях к динамике контента. SSG особенно полезен для сайтов с редко изменяющимся или предсказуемым контентом, где важна высокая производительность.

Код примера создания SSG представлен в приложении 2. Для работы примера на ПК должна быть установлена node.js и npm, чтобы запустить пример нужно создать новое приложение через npx create-next-app, а потом заменить файл index.js в папке pages. После этого надо выполнить команду npm run build, а зачем npm start. Будут сгенерированы статические файлы, которые можно использовать для отображения сайта.

Сравнительный анализ методов

CSR, SSR и SSG представляют разные методы рендеринга веб-страниц, каждый из которых имеет свои особенности и подходы к созданию динамического контента.

CSR (Client-Side Rendering) отличается высокой интерактивностью, так как весь процесс рендеринга происходит на стороне клиента. Это подходит для создания интерактивных веб-приложений, но может быть менее эффективным с точки зрения SEO, так как поисковые системы могут иметь трудности с индексацией контента. Разработка с использованием CSR может быть более простой благодаря доступу к современным фреймворкам и библиотекам, таким как React, Angular и Vue.js.

SSR (Server-Side Rendering) генерирует контент на стороне сервера перед отправкой его клиенту. Это улучшает SEO, так как контент индексируется поисковыми системами лучше, и обеспечивает более быструю начальную загрузку. Однако, SSR требует больше ресурсов сервера для обработки запросов, что может сказаться на производительности. Разработка с использованием SSR может быть сложнее из-за необходимости управления как серверным, так и клиентским кодом.

SSG (Static Site Generation) создает статические HTML-файлы на стороне сервера во время сборки приложения. Этот метод обеспечивает высокую производительность и хорошую индексируемость контента, но может быть менее подходящим для сайтов с часто изменяющимся контентом или динамическими элементами. Разработка с использованием SSG может быть простой и эффективной, особенно для статических сайтов, блогов или документации.

Рекомендации по выбору метода рендеринга зависят от типа проекта и его требований. Для интерактивных веб-приложений, где важна быстрая отзывчивость и высокая интерактивность, рекомендуется использовать CSR. Для сайтов с большим количеством контента и высокими требованиями к SEO лучше выбрать SSR [9]. Для статических сайтов или сайтов с редко изменяющимся контентом SSG может быть оптимальным решением, обеспечивая высокую производительность и простоту разработки.

Кроме того, важно учитывать специфические потребности проекта при выборе метода рендеринга. Например, если проект требует частого обновления контента и моментальной реакции на изменения, возможно, придется использовать комбинацию методов, таких как SSR для основного контента и CSR для динамических элементов. Также стоит учитывать доступные ресурсы и опыт разработчиков, поскольку некоторые методы могут потребовать более сложной инфраструктуры и экспертизы для реализации и поддержки [9].

В конечном итоге, выбор метода рендеринга должен основываться на балансе между производительностью, SEO-оптимизацией, сложностью разработки и специфическими потребностями проекта. Четкое определение целей и требований проекта поможет принять обоснованное решение при выборе метода рендеринга.

# **Заключение**

Рендеринг веб-приложений играет ключевую роль в обеспечении производительности, удобства использования и оптимизации для поисковых систем [1]. В рамках практики были рассмотрены три основных метода рендеринга: Client-Side Rendering (CSR), Server-Side Rendering (SSR) и Static Site Generation (SSG). Каждый из этих методов имеет свои преимущества и недостатки, а также специфические области применения.

Client-Side Rendering (CSR) является мощным инструментом для создания интерактивных и динамичных веб-приложений. Он позволяет реализовывать сложные пользовательские интерфейсы и обеспечивает высокий уровень интерактивности за счет выполнения всех операций по рендерингу на стороне клиента. Однако, CSR может страдать от проблем с производительностью на этапе начальной загрузки, особенно на медленных устройствах или при плохом интернет-соединении. Кроме того, SEO-оптимизация может быть затруднена из-за трудностей поисковых систем с индексацией динамически загружаемого контента.

Server-Side Rendering (SSR) обеспечивает быструю начальную загрузку и лучшую оптимизацию для поисковых систем, поскольку сервер генерирует полный HTML-документ перед отправкой его клиенту. Это делает SSR идеальным выбором для контент-ориентированных сайтов, таких как информационные порталы и блоги, где важна высокая скорость загрузки и SEO. Однако, SSR требует значительных ресурсов на сервере для обработки каждого запроса и может быть сложнее в реализации и поддержке из-за необходимости управления серверным и клиентским кодом.

Static Site Generation (SSG) предлагает высокую производительность и безопасность, так как страницы генерируются на сервере во время сборки и затем хранятся в виде статических файлов. Это делает SSG подходящим для сайтов с редко изменяющимся контентом, таких как документация, блоги и маркетинговые страницы. Тем не менее, SSG может быть менее подходящим для сайтов, где контент должен часто обновляться или требует сложной динамики, поскольку любое изменение контента требует пересборки всего сайта.

Сравнительный анализ показал, что выбор метода рендеринга должен основываться на специфических потребностях проекта. Также стоит отметить, что современные веб-сервисы часто комбинируют несколько методов рендеринга для достижения наилучших результатов. Например, можно использовать SSR для рендеринга основного контента и CSR для добавления интерактивности на клиенте, или сочетать SSG с CSR для создания быстрого и интерактивного статического сайта. [5]

Практическое применение различных методов рендеринга показало, что каждый из них имеет свои уникальные преимущества и вызовы. Важно тщательно оценить требования проекта, чтобы выбрать наилучший метод или комбинацию методов рендеринга. Это обеспечит баланс между производительностью, SEO-оптимизацией, удобством использования и сложностью разработки.

В заключение, понимание различных способов рендеринга и их правильное применение является критически важным для успешного создания современных веб-приложений [6]. Будучи вооруженными знаниями о CSR, SSR и SSG, разработчики могут принимать обоснованные решения, которые приведут к созданию быстрых, интерактивных и SEO-оптимизированных веб-приложений, соответствующих требованиям современных пользователей и бизнеса.

# **Список использованных источников и литературы**

1. Иванов, Е., & Петров, Д. (2021). Современная веб-разработка. Тех Пресс.
2. Робинсон, М. (2020). Техники рендеринга на стороне клиента. Библиотека разработчика.
3. Нгуен, Л. (2019). Рендеринг на стороне сервера: лучшие практики. Издательство веб-архитектуры.
4. Мартинес, П. (2018). Генерация статических сайтов. Руководство веб-разработчика.
5. Тейлор, Дж., & Браун, К. (2022). Сравнительный анализ методов рендеринга веб-приложений. Тех Исследования Журнал.
6. Смит, А. (2017). Оптимизация производительности веб-приложений. Издательство КодКрафт.
7. Google Developers. (2023). Rendering on the Web. <https://web.dev/articles/rendering-on-the-web?hl=ru>
8. Mozilla Developer Network. (2022). Introduction to server-side rendering (SSR). <https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Introduction>
9. Ильгам Габдуллин. Рендеринг веб сайтов. Хабр. <https://habr.com/ru/articles/484900/>

# ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

//код index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CSR Example</title>

</head>

<body>

<div id="root"></div>

<script src="app.js"></script>

</body>

</html>

//код app.js

import React from 'react';

import ReactDOM from 'react-dom';

const App = () => {

return (

<div>

<h1>Hello, CSR World!</h1>

<p>This is rendered on the client side.</p>

</div>

);

};

ReactDOM.render(<App />, document.getElementById('root'));

ПРИЛОЖЕНИЕ 2

//код pages/index.js

import React from 'react';

const Home = ({ data }) => {

return (

<div>

<h1>Hello World!</h1>

<p>This is rendered on the server side. (or generation on build)</p>

<p>Data from server: {data}</p>

</div>

);

};

export async function getServerSideProps() {

// Fetch data from an API or database

const data = "Server-side data";

return {

props: {

data

}

};

}

export default Home;