DOI 10.37882/2223-2966.2025.05.07

ЭРГОНОМИЧЕСКИЙ АНАЛИЗ МЕТОДОВ РЕНДЕРИНГА ВЕБ-СТРАНИЦ НА ОСНОВЕ NEXT.JS ФРОНТЕНД ФРЕЙМВОРКА

ERGONOMIC ANALYSIS OF WEB PAGE RENDERING METHODS BASED ON NEXT.JS FRONTEND FRAMEWORK

B. Goryachkin
M. Dudnik

Summary. *Problem statement*. The article investigates the problem of choosing the optimal web page rendering method based on the Next.js framework. Three main methods are considered: Client-Side Rendering (CSR), Server-Side Rendering (SSR), and Static Site Generation (SSG). The relevance of the problem is driven by the need to ensure high loading speed, interactivity, and efficient resource utilization in the development of modern web applications.
Goal. To determine the most effective rendering method in terms of loading speed, interactivity, and resource utilization.
Results. The obtained data showed that SSG is the most effective method for static pages, while CSR and SSR demonstrate similar results for dynamic scenarios. The article provides recommendations for choosing a rendering method depending on the type of content and performance requirements. The analysis of Web Vitals metrics (FCP, TTI, LCP) and resource consumption (memory usage, number of network requests) confirmed the hypotheses about the impact of the rendering method choice on the performance of web applications.
Practical significance. The research results enable developers to make informed decisions when choosing a rendering method based on the type of content and performance requirements. The article provides practical recommendations for optimizing web applications based on Next.js, which contributes to improving user experience and development efficiency.

Keywords: frontend framework, Next.js, server-side rendering, client-side rendering, Web Vitals, CSR, SSR, SSG.

Горячкин Борис Сергеевич
кандидат технических наук, доцент,
Московский государственный технический университет им. Н.Э. Баумана
bsgor@mail.ru
Дудник Максим Валерьевич
Московский государственный технический университет им. Н.Э. Баумана
maxdudnik@mail.ru

Аннотация. *Постановка проблемы*. В статье исследуется проблема выбора оптимального метода рендеринга веб-страниц на основе фреймворка Next.js. Рассматриваются три основных метода: Client-Side Rendering (CSR), Server-Side Rendering (SSR) и Static Site Generation (SSG). Актуальность проблемы обусловлена необходимостью обеспечения высокой скорости загрузки, интерактивности и эффективного использования ресурсов при разработке современных веб-приложений.
Цель. Определить наиболее эффективный метод рендеринга с точки зрения скорости загрузки, интерактивности и использования ресурсов.
Результаты. Полученные данные показали, что SSG является наиболее эффективным методом для статических страниц, тогда как CSR и SSR демонстрируют схожие результаты для динамических сценариев. Статья содержит рекомендации по выбору метода рендеринга в зависимости от типа контента и требований к производительности. Анализ метрик Web Vitals (FCP, TTI, LCP) и ресурсопотребления (объем памяти, количество сетевых запросов) подтвердил гипотезы о влиянии выбора метода рендеринга на производительность веб-приложений.
Практическая значимость. Результаты исследования позволяют разработчикам принимать обоснованные решения при выборе метода рендеринга в зависимости от типа контента и требований к производительности. Статья содержит практические рекомендации по оптимизации веб-приложений на основе Next.js, что способствует улучшению пользовательского опыта и эффективности разработки.

Ключевые слова: фронтенд фреймворк, Next.Js, рендеринг на стороне сервера, рендеринг на стороне клиента, Web Vitals, CSR, SSR, SSG.

Введение

В современном цифровом мире веб-сайты играют важнейшую роль в глобальной коммуникации, распространении информации и деловых операциях. Веб-сайт состоит из статичных документов, созданных с использованием языка гипертекстовой разметки (HTML), который позволяет удобно обмениваться информацией при условии подключения к Интернету [1], [2]. При определении качества веб-сайта учитывается несколько факторов, включая скорость доступа, удобство чтения контента и согласованность макета или дизайна [3]. Сложность веб-сайта создает проблемы для разработчиков с точки зрения производительности. Чем больше объем отображаемых данных или контента, тем сильнее это влияет на скорость отображения в браузере и время загрузки страницы [2]. Разработчик веб-сайта должен уделять приоритетное внимание не только дизайну, но и повышению производительности и скорости веб-сайта. Скорость и отзывчивость веб-сайта оказывают существенное влияние на удержание пользователей и показатели конверсии, а также на имидж бренда [4].

Для повышения производительности веб-приложений выбор метода рендеринга имеет первостепенное значение [5].

Next.js — это фронтенд-фреймворк, созданный для расширения возможностей React и поддержки рендеринга на сервере. Он основан на React и добавляет функции для генерации статических страниц (SSG) и серверного рендеринга (SSR).

Next.js позволяет создавать как статические сайты, так и динамические приложения с минимальной конфигурацией, адаптируемой под конкретные задачи. Фреймворк включает маршрутизацию на основе файловой структуры, автоматическое разделение кода, поддержку CSS и Sass.

Next.js генерирует HTML на этапе сборки или на сервере, что ускоряет загрузку и улучшает SEO. Он поддерживает три основных метода рендеринга:

- SSG (Static Site Generation, статическая генерация сайта) — генерация страниц на этапе сборки.
- SSR (Server Side Rendering, рендеринг на стороне сервера) — рендеринг страниц на сервере при каждом запросе.
- CSR (Client Side Rendering, рендеринг на стороне клиента) — рендеринг страниц в браузере.

Фреймворк позволяет комбинировать эти подходы в одном проекте, выбирая оптимальный метод для каждой страницы. По умолчанию Next.js выполняется предварительный рендеринг каждой страницы и сначала генерируется HTML для каждой страницы, так что не все делается с помощью JavaScript на стороне клиента [10]. Next.js предлагает методы предварительного рендеринга SSR (рендеринг на стороне сервера) и SSG (статическая генерация сайта), которые оптимизируют производительность и SEO [11]. Кроме того, Next.js предоставляет альтернативу CSR (рендеринг на стороне клиента).

Будет проведен сравнительный анализ эргономичности методов рендеринга в рамках Next.js фреймворка. Это исследование будет полезно разработчикам и исследователям при выборе подходящего метода рендеринга для разработки веб-приложений.

Схема алгоритма рендеринга веб-страницы

Главная цель процесса рендеринга заключается в преобразовании HTML, CSS и JavaScript в веб-страницу для обеспечения возможности взаимодействия с ней со стороны пользователя [2]. Схема алгоритма рендеринга веб-страницы представлена на рис. 1.

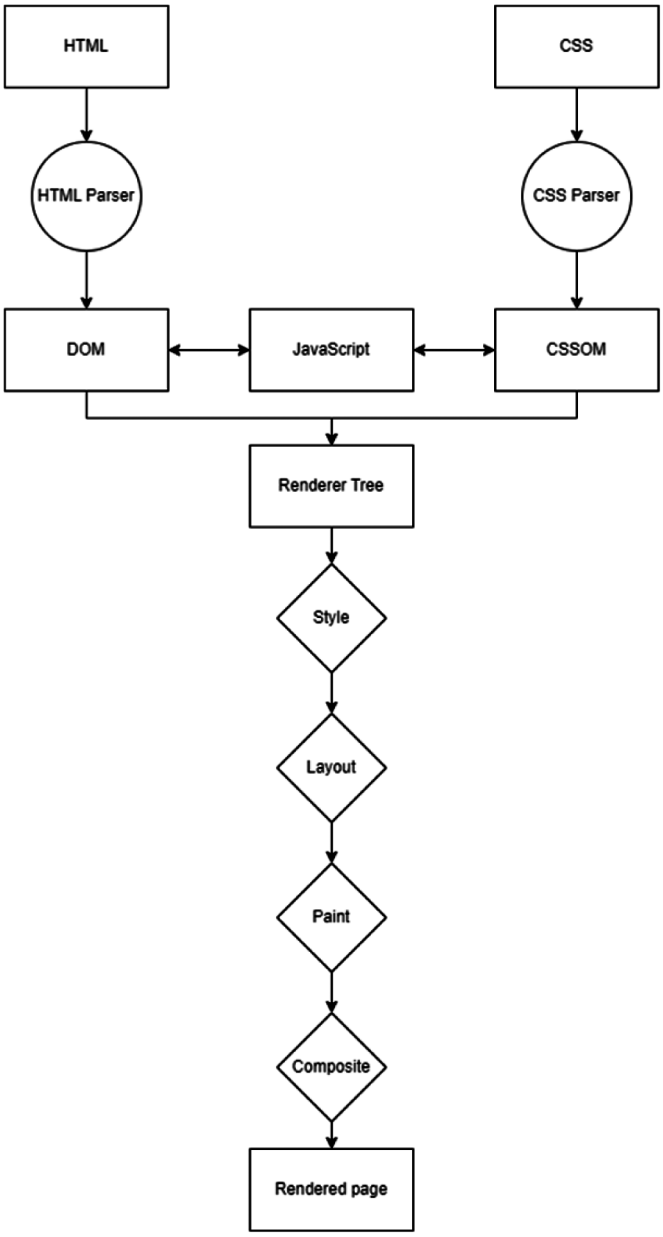

Рис. 1. Схема алгоритма рендеринга веб-страницы

Рендеринг страницы делится на 3 большие части:

1. Получение исходного HTML файла
2. Парсинг HTML файла с помощью HTML Parser и поэтапное получение указанных внутри него CSS и JS файлов. При этом парсинг HTML останавливается и возобновляется в том случае, если текущий CSS или JS файл полностью обработан CSS Parser или выполнен в JS Runtime соответственно. Исключение составляют скрипты с атрибутами defer и async, загружаемые параллельно ходу парсинга HTML.
3. Составление дерева рендеринга с уже включенными в него HTML элементами и CSS стилями для каждого из них, и поэтапная отрисовка этих элементов на странице с помощью 4 этапов:

— Style — применение CSS селекторов к элементам
— Layout — расчет положения элементов на странице и создание каркаса страницы
— Paint — отрисовка пикселей для каждого элемента
— Composite — применение слоев-уровней для элементов на странице

Методы рендеринга

Client-side Rendering

Client-Side Rendering (CSR) — это подход к разработке веб-приложений, при котором весь процесс рендеринга интерфейса происходит в браузере с помощью JavaScript. Сервер отправляет минимальный HTML-документ, а весь контент динамически создаётся и обновляется на клиенте после загрузки JS [8]. Этапы CSR с точки зрения браузера приведены на рис. 2.

Порядок работы CSR (Client-Side Rendering):

1. Запрос к серверу: браузер отправляет HTTP-запрос и получает минимальный HTML-файл. Обычно это пустая страница с контейнером для приложения (например,).
2. Загрузка JavaScript: браузер загружает JavaScript-файл, который содержит логику приложения и отвечает за создание интерфейса.
3. Выполнение JavaScript: после загрузки JS начинается процесс рендеринга интерфейса. JS создаёт DOM-структуру и вставляет элементы в контейнер.
4. Запрос данных с API: приложение отправляет запросы к серверу или сторонним API, чтобы получить необходимые данные для отображения.
5. Отрисовка контента: когда данные получены, приложение обновляет DOM и браузер отображает контент на странице.
6. Интерактивность и обновления: после первого рендеринга приложение становится интерактивным. Пользователь может взаимодействовать с компонентами, и изменения интерфейса происходят без перезагрузки страницы.

Server-side Rendering

SSR (Server-Side Rendering) — это метод рендеринга веб-страниц, при котором HTML-код генерируется на сервере и отправляется в браузер в уже готовом виде. Это позволяет пользователю увидеть содержимое быстрее, по сравнению с клиентским рендерингом (CSR) [9]. Этапы SSR с точки зрения браузера приведены на рис. 3.

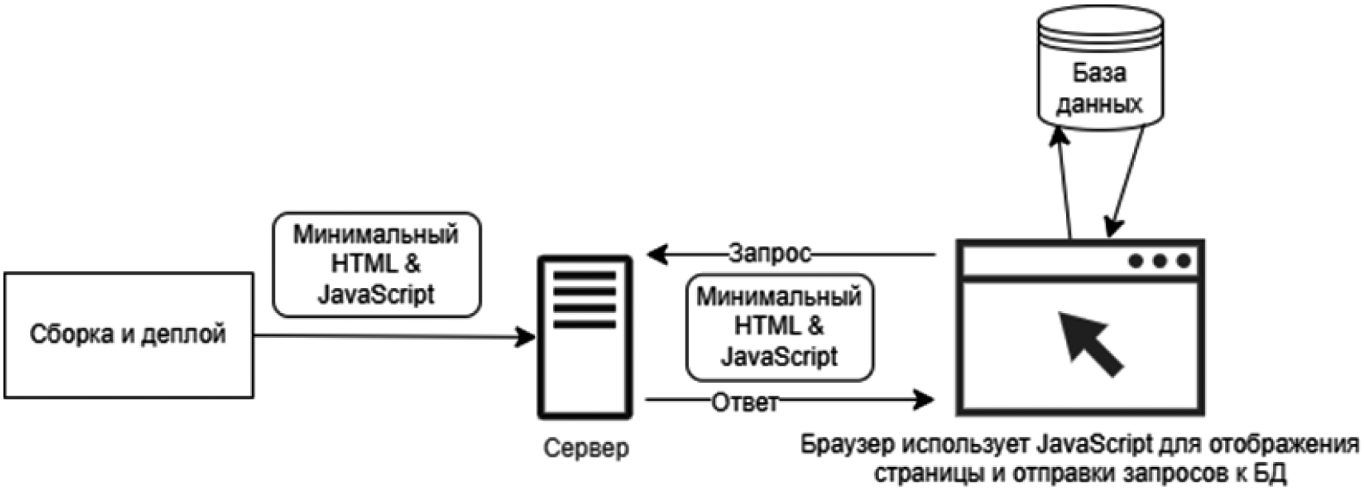

Рис. 2. Этапы CSR с точки зрения браузера

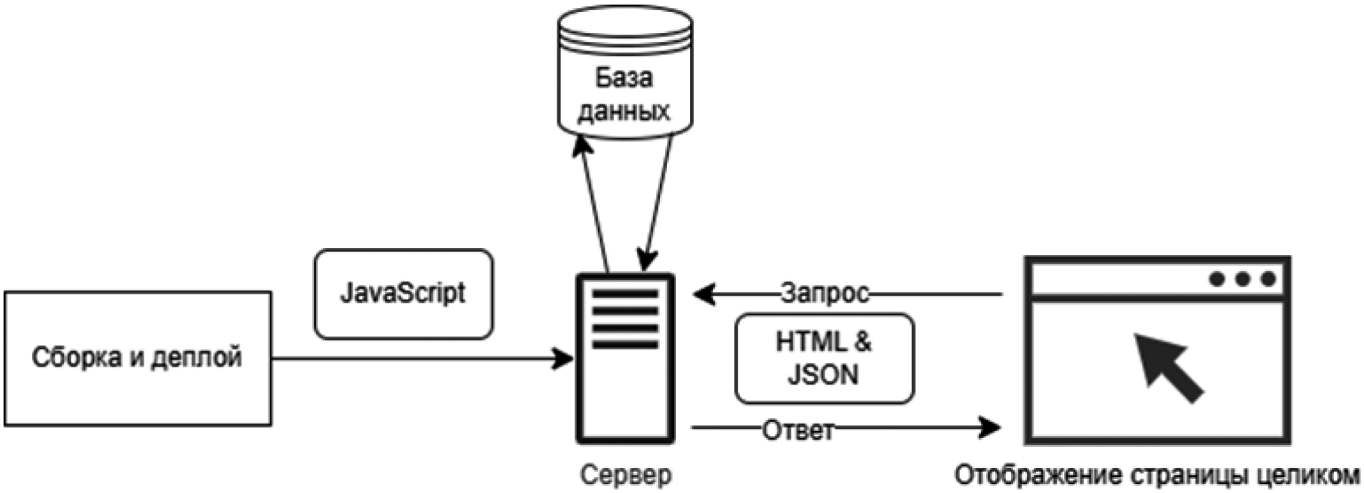

Рис. 3. Этапы SSR с точки зрения браузера

Порядок работы SSR (Server-Side Rendering):

1. Запрос от клиента: браузер отправляет http-запрос на сервер для загрузки веб-страницы.
2. Обработка запроса: сервер получает запрос, выполняет необходимую логику (например, запросы к базе данных или api).
3. Рендеринг html на сервере: сервер генерирует готовую html-страницу, выполняя шаблоны и вставляя в них данные.
4. Отправка html в браузер: сервер отправляет клиенту сформированный html с полным содержимым страницы.
5. Отображение страницы: браузер получает html и сразу отображает готовую страницу. контент становится доступным до завершения загрузки javascript.
6. Загрузка javascript и гидрация: после рендеринга html браузер загружает javascript, который добавляет интерактивность к странице. Процесс превращения статической страницы в интерактивную называется гидрацией [11].

Static Site Generation

SSG (Static-Site Generation) — это метод генерации веб-страниц, при котором HTML-файлы создаются на этапе сборки (build time) и остаются неизменными до следующей сборки. Это позволяет раздавать готовые статические страницы напрямую с сервера или через CDN, что обеспечивает быструю загрузку и высокую производительность. [11]. Этапы SSG с точки зрения браузера приведены на рис. 4.

Порядок работы SSG (Static Site Generation):

1. Сборка проекта: при запуске команды сборки (npm run build или аналогичной) приложение проходит процесс генерации всех HTML-страниц.
2. Запрос данных: во время сборки приложение делает запросы к API, базе данных или файловой системе для получения необходимых данных (например, список постов или продуктов).
3. Генерация HTML: на основе полученных данных и шаблонов создаются статические HTML-страницы для каждого маршрута (URL). Эти страницы сохраняются как файлы на диске.
4. Развёртывание (Deployment): готовые HTML-файлы загружаются на сервер или CDN.
5. Запрос от клиента: при обращении к сайту браузер запрашивает уже сгенерированный HTML-файл, который мгновенно возвращается с сервера или из кэша CDN.
6. Отображение страницы: браузер отображает готовую HTML-страницу сразу после получения.
7. Загрузка и выполнение JavaScript: после рендеринга статической страницы загружается JavaScript, чтобы добавить интерактивность (гидрация).

Расчет времени рендеринга

Для SSR формула расчета приблизительного времени рендеринга будет иметь следующий вид:

$$Tssr = Tprocessing + Tdata + Tbuild + Tnetwork + Trender \quad (1)$$

- *Tprocessing* — время обработки запроса сервером
- *Tdata* — получение данных (запрос к базе данных или API)
- *Tbuild* — сборка HTML страницы на сервере
- *Tnetwork* — время на отправку ответа на клиент
- *Trender* — рендеринг страницы браузером

Для SSG формула расчета приблизительного времени рендеринга будет иметь следующий вид:

$$Tssg = Npages * Tbuild + Tnetwork + Trender \quad (2)$$

- *Npages* — количество страниц, которые необходимо собрать

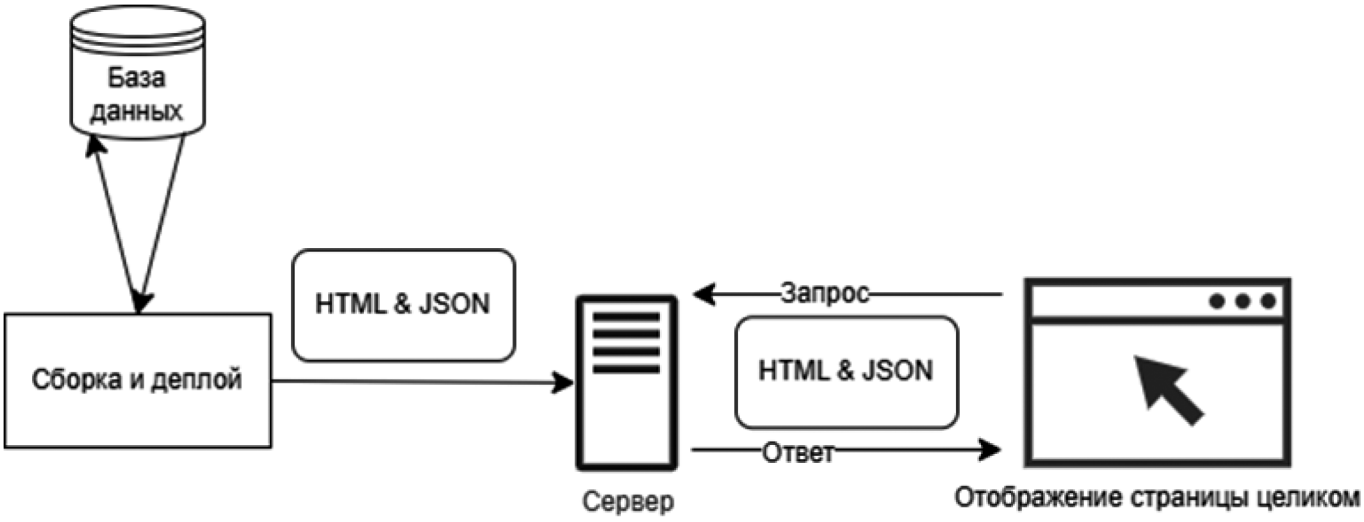

Рис. 4. Этапы SSG с точки зрения браузера

Стоит отметить, что операция по сборке страницы будет происходить на сервере всего лишь 1 раз, поэтому для клиентов такого сайта будет влиять только интернет-соединение и время браузерного рендеринга, что говорит о том, что SSG будет быстрее, чем SSR при прочих равных.

Для CSR формула расчета приблизительного времени рендеринга будет иметь следующий вид:

$$Tcsr = Tnetwork + Trender1 + Tjs + Tdata + Trender2 \quad (3)$$

- *Trender*1 — время рендеринга первой страницы (в основном пустая)
- *Tjs* — время полной обработки JS для получения данных и отрисовки контента
- *Trender*2 — время рендеринга итоговой страницы

В SSG и CSR *Tprocessing* не учитывается, так как данные представляют собой статичные CSS, JS и HTML файлы готовые к отправке клиенту.

Возьмем скорость интернета за константу, тогда *TnetworkSSR ~ TnetworkSSG*, но при этом *TnetworkCSR* будет в несколько раз меньше, так как изначально в браузер приходит пустой HTML файл с подключенными скриптами.

Исходя из приведенных выше данных можно предположить, что SSG будет наиболее эффективной в рамках метрики TTI, но будет требовать наибольшего количества памяти для работы. CSR будет иметь наибольший показатель сетевых запросов, так как все запросы будут идти на стороне клиента и наименьший объем памяти, так как HTML генерируется на устройстве пользователя.

Эффективность CSR и SSR будет зависеть от параметров сервера и устройства клиента. Если количество подключений превысит ресурс мощностей сервера, тогда *Tbuild* будет занимать неограниченно большое время, при этом на CSR не будет влиять количество подключений, так как сервер при таком подходе всего лишь отдает статические файлы.

Выдвижение гипотез

На основании анализа предметной области, а также формул расчета времени рендеринга были выдвинуты следующие гипотезы:

Гипотеза 1: CSR будет иметь наибольший показатель сетевых клиентских запросов

Гипотеза 2: SSR будет быстрее CSR в рамках метрики FCP

Гипотеза 3: SSG будет наиболее эффективным в рамках метрики TTI

Гипотеза 4: CSR всегда будет требовать наименьший объем памяти для работы

Аппаратное обеспечение и процедура исследования

Аппаратное обеспечение представляет собой ноутбук под управлением Windows 11 с процессором Intel(R) Core(TM) i5-1135G7 11-го поколения с тактовой частотой 2,40 ГГц.

Характеристики 2,42 ГГц, 8 ГБ оперативной памяти DDR4 и 512 ГБ SSD. Node.js., в качестве программных средств используются Visual Studio Code и инструменты разработчика в браузерах Google Chrome, Mozilla Firefox и Opera. Предметом исследования является исходный код веб-системы «Film4u» в виде файла TypeScript, основанный на платформе Next.js framework. Веб-система «Film4u» имеет веб-структуру, показанную на рис. 5. Одна из страниц этой веб-системы (страница «Подборка») показана на рис. 6.

Процедура исследования показана на рис. 7 ниже.

Измеряемые величины

В качестве измеряемых параметров были выбраны характеристики основных переменных Web Vitals, а также объем памяти и количество сетевых запросов.

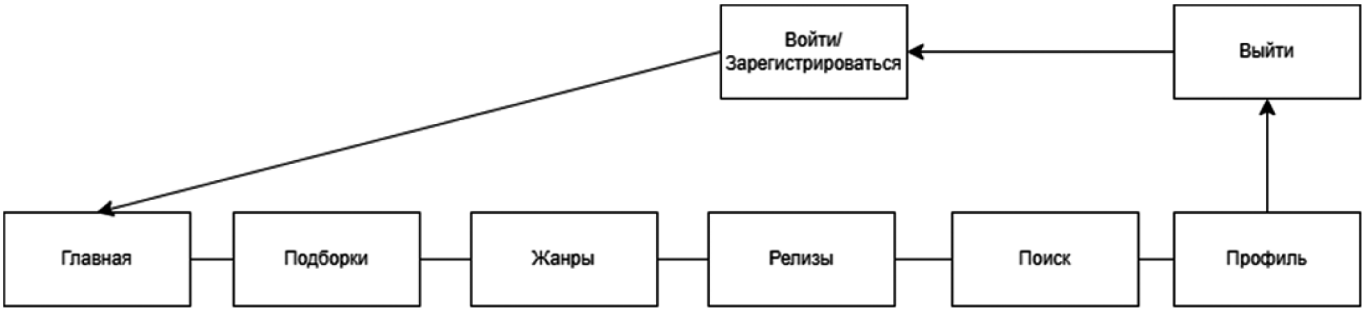

Рис. 5. Веб-структура веб-сайта «Film4u»

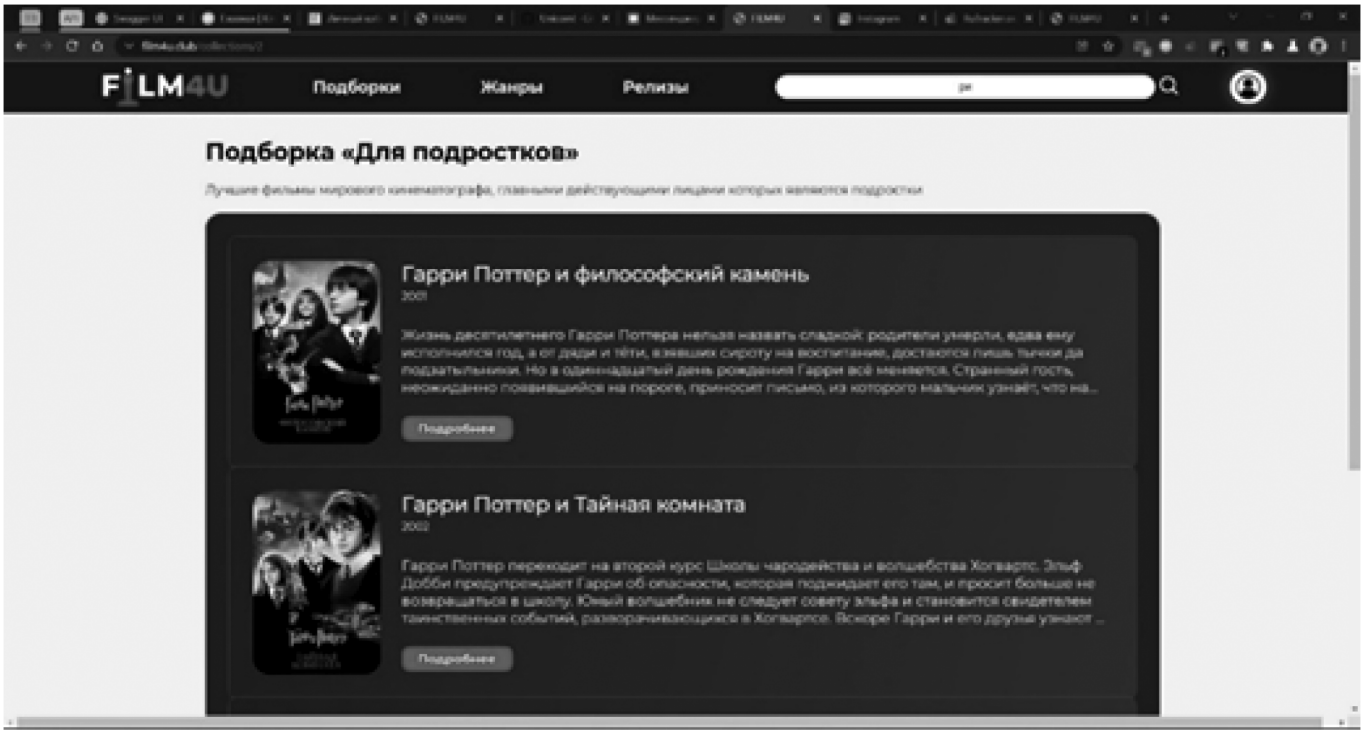

Рис. 6. Страница «Подборка»

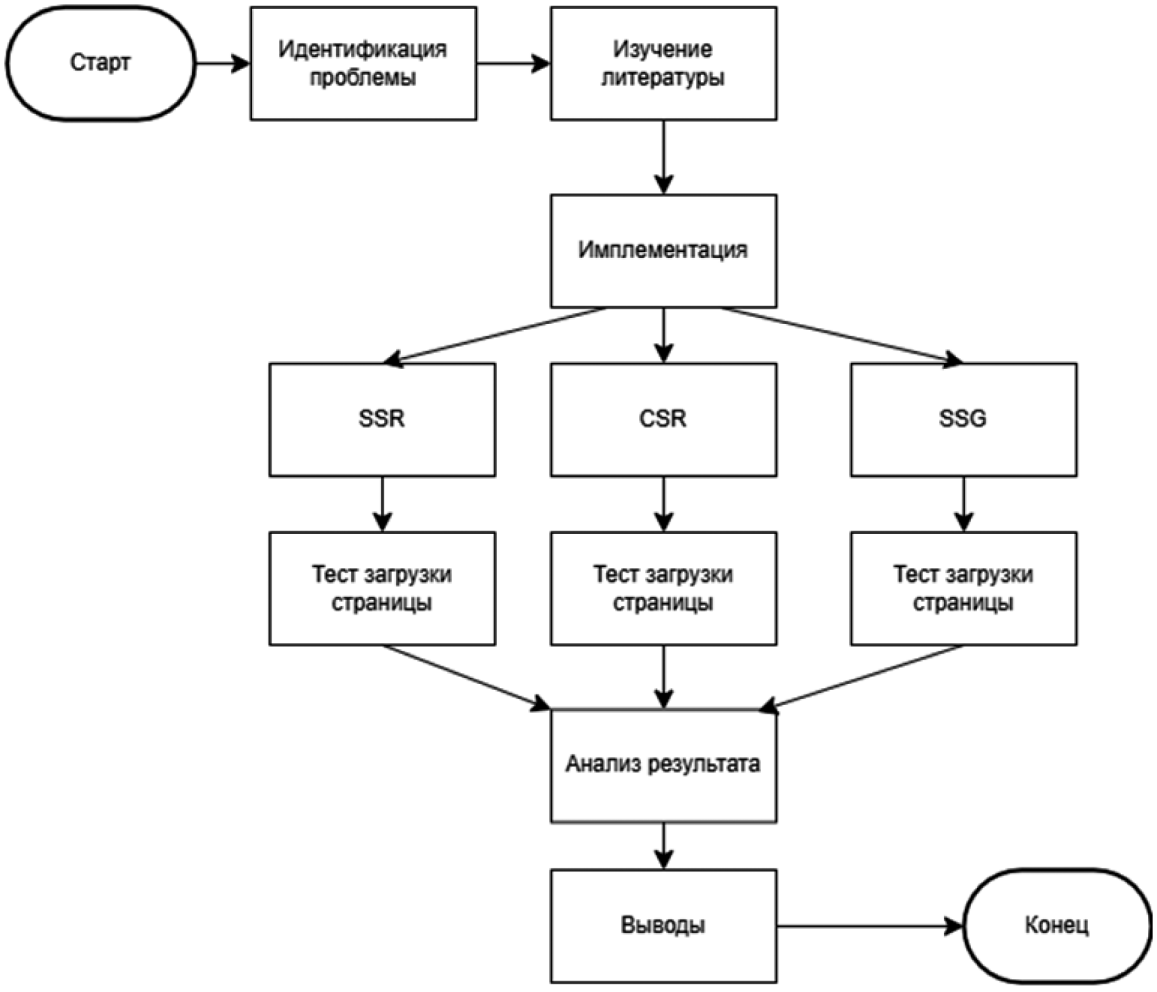

Рис. 7. Процедура исследования

Таблица 1.

Измеряемые величины

№	Измеряемые величины (Measured values)	Описание (Description)
1	Количество запросов (Number of requests)	Количество запросов на отображаемой странице (The number of requests on the displayed page)
2	Объем памяти (Memory capacity)	Количество памяти необходимое для отображения страницы (The amount of memory required to display the page)
3	FCP (First Contentful Paint)	FCP измеряет время от начала загрузки страницы до момента, когда браузер отображает первый видимый контент (например, текст, изображение или SVG) (FCP measures the time from the beginning of page loading to the moment when the browser displays the first visible content (for example, text, image or SVG))
4	TTI (Time to Interactive)	TTI измеряет время, необходимое для того, чтобы страница стала полностью интерактивной, то есть, когда все скрипты завершили свою загрузку и обработку, и страница может реагировать на пользовательские действия (TTI measures the time it takes for a page to become fully interactive, that is, when all scripts have completed their loading and processing, and the page can respond to user actions.)
5	LCP (Largest Contentful Paint)	LCP измеряет время, когда наиболее крупный видимый элемент (например, изображение, текстовый блок или видео) на странице становится видимым на экране (LCP measures the time when the largest visible element (such as an image, text block, or video) on a page becomes visible on the screen.)

Проведение измерений

Для измерения метрик WebVitals использовался инструментарий Chrome DevTools [20]. Скорость загрузки страницы была протестирована на трех страницах: «Авторизация», «Профиль» и «Главная». Каждая страница была протестирована с использованием SSR, CSR или SSG на одном и том же устройстве и с использованием одних и тех же инструментов. Видеоролики, используемые в веб-системе «Film4u», представлены в формате MP4 и имеют размер от 160 до 200 Мб. Результаты измерений представлены в таблицах 2–4.

Таблица 2.

Измерения для страницы «Авторизация»

Параметр (Parametr)	SSR	SSG	CSR
Количество запросов (Number of requests)	10	10	10
Объем памяти (Мб) Memory capacity (MB)	710	710	710
FCP (Mc) (Ms)	156	117	108
TTI (Mc) (Ms)	274	180	422
LCP (Mc) (Ms)	251	170	387

Таблица 3.

Измерения для страницы «Профиль»

Параметр (Parametr)	SSR	SSG	CSR
Количество запросов (Number of requests)	9	10	12
Объем памяти (Мб) Memory capacity (MB)	554	549	539
FCP (Mc) (Ms)	253	136	248
TTI (Mc) (Ms)	706	524	608
LCP (Mc) (Ms)	642	493	497

Таблица 4.

Измерения для страницы «Главная»

Параметр (Parametr)	SSR	SSG	CSR
Количество запросов (Number of requests)	10	11	13
Объем памяти (Мб) Memory capacity (MB)	780	801	620
FCP (Mc) (Ms)	175	101	159
TTI (Mc) (Ms)	1525	1135	1350
LCP (Mc) (Ms)	925	980	901

Анализ полученных результатов

Исследование показало, что SSG (Static Site Generation) обеспечивает наивысшую скорость загрузки и отображения страниц по сравнению с CSR (Client-Side Rendering) и SSR (Server-Side Rendering).

Основные выводы:

1. 1.SSG — самый быстрый метод рендеринга.
   - Страницы заранее генерируются на этапе сборки (build time) и раздаются как готовые HTML-файлы.

- Это минимизирует задержку при загрузке и сокращает время до первой отрисовки (FCP), так как клиент сразу получает готовый контент.
- Однако SSG эффективен только для страниц с преимущественно статическим контентом. Если информация часто обновляется, потребуется частая пересборка, что может замедлить процесс разработки и доставки новых данных.

2. CSR и SSR показывают схожие результаты в динамических сценариях.
- CSR переносит рендеринг на клиент, что даёт высокую интерактивность, но увеличивает задержку перед первым отображением контента (FCP).
- SSR рендерит страницы на сервере, ускоряя первую отрисовку, но увеличивая нагрузку на сервер при каждом запросе.

3. Разница между CSR и SSR минимальна и зависит от:
- Мощности устройства клиента (для CSR): слабые устройства могут медленно обрабатывать JavaScript, что замедляет загрузку страницы.
- Производительности сервера (для SSR): сервер с высокой производительностью генерирует страницы быстрее, уменьшая задержку.

4. SSR является самым безопасным методом рендеринга, так как позволяет выполнить запросы на сервер еще до получения кода клиентом, тем самым минимизировав запросы из браузера.
5. CSR требует минимальное количество ресурсов для своей работы, так как вся нагрузка переносится на клиента. Это может помочь развивающимся компаниям, не имеющим средств на покупку мощных серверов.
6. На статических страницах (например, «Авторизация») SSG и SSR показывают себя значительно лучше, чем CSR по метрикам TTI и LCP. Это связано с тем, что SSG и SSR способны эффективно сгенерировать статический контент и сразу же отдать его клиенту, когда CSR будет необходимо время для инициализации скриптов JS.

Оценка выдвигаемых гипотез

Оценка выдвинутых гипотез приведена в таблице 5:

Заключение

В результате работы были исследованы 3 метода рендеринга веб-страниц SSG, SSR, CSR, выделены ключевые

Таблица 5.

Оценка гипотез

№	Гипотеза (Hypothesis)	Результат (Result)	Возможные причины (Possible reasons)
1	CSR будет иметь наибольший показатель сетевых клиентских запросов (CSR will have the highest rate of network client requests.)	Истина (Truth)	Очевидно, что в рамках SSG и SSR большинство запросов будут идти на стороне сервера, когда в CSR абсолютно все сетевые запросы будут идти на стороне клиента. (Obviously, within SSG and SSR, most requests will go on the server side, while in CSR absolutely all network requests will go on the client side.)
2	SSR будет быстрее CSR в рамках метрики FCP (SSR will be faster than CSR within the FCP metric)	Ложь (False)	Ввиду маломощности используемого сервера и оптимизаций браузера и исполняемого кода JS CSR загружал первый контент быстрее, однако пользователь в SSR сразу же получал готовую страницу, а в CSR какое-то время видел белый экран, что является не лучшим пользовательским опытом. (Due to the low power of the server used and browser optimizations and executable code, JS CSR loaded the first content faster, but the user in SSR immediately received the finished page, and in CSR he saw a white screen for a while, which is not the best user experience.)
3	SSG будет наиболее эффективным в рамках метрики TTI (SSG will be most effective within the framework of the TTI metric.)	Истина (Truth)	Исходя из формул расчета SSG экономит время на повторной генерации HTML страницы, что благоприятно влияет на метрику TTI. Исследование это подтвердило. (Based on the calculation formulas, SSG saves time on re-generating the HTML page, which has a positive effect on the TTI metric. The study confirmed this.)
4	CSR всегда будет требовать наименьший объем памяти для работы (CSR will always require the least amount of memory to work with.)	Ложь (False)	На большинстве страниц CSR действительно требует меньше ресурсов памяти, однако на странице "Авторизация", представляющую из себя статическую форму для регистрации, он показывает примерно такой же результат, как и другие методы рендеринга, что говорит нам о том, что для страниц без динамического контента CSR менее эргономичен, чем SSR и SSG. (CSR does require less memory resources on most pages, but on the Authorization page, which is a static registration form, it shows about the same result as other rendering methods, which tells us that CSR is less ergonomic than SSR and SSG for pages without dynamic content.)

результаты, а также произведена оценка выдвинутых гипотез.

Все методы рендеринга используются для своих задач. SSG, хоть и показал себя самым эффективным, будет работать хуже на сайтах с динамическим, часто изменяющимся контентом. Поэтому SSG необходимо использовать только на статических сайтах с минимальной динамикой.

CSR наименее эффективен на статических сайтах и предназначен для отображения динамического контента с минимальными ресурсными затратами. Из недостатков можно отметить худшую поддержку SEO-оптимизаций, так как какое-то время пользователь видит пустой экран, а также нестабильность загрузки страницы на разных устройствах клиента.

SSR является оптимальным выбором и предоставляет самый широкий спектр возможностей: настройку SEO, скрытые запросы, балансировка нагрузки и т. д. Однако требует мощных и дорогостоящих серверов для своей работы. В моем исследовании использовался средний ноутбук и SSR показал себя хуже SSG и CSR.

ЛИТЕРАТУРА

1. Супрапто А., Сасонгко Д. Evaluasi Performa Website Berdasarkan Pengujian Beban Dan Stress Menggunakan Loadimpact (Studi Kasus Website Iain Salatiga) // Netw. Eng. Res. Oper. 2021. Т. 6, № 1. С. 31. DOI: 10.21107/nero.v6i1.198.
2. Сантосо М.Ф. Teknik Single Page Application (SPA) Layout Web Dengan menggunakan React Js Dan Bootstrap // J. Khatulistiwa Inform. 2021. Т. 9, № 2. С. 107–114. DOI: 10.31294/jki.v9i2.11357.
3. Сулиман. Analisis Performa Website Universitas Teuku Umar Dan Universitas Samudera Menggunakan Pingdom Tools Dan Gtmetrix // J. Sist. Inf. dan Sist. Komput. 2020. Т. 5, № 1. С. 24–32. DOI: 10.51717/simkom.v5i1.47.
4. Юсуф А., Нурясин И., Сари З. Optimasi Kecepatan Loading Time Web Template Dengan Implementasi Teknik Front-End // J. Repos. 2020. Т. 2, № 11. С. 1456. DOI: 10.22219/repositor.v2i11.746.
5. Оллила Р., Макиало Н., Микконен Т. Modern Web Frameworks: A Comparison of Rendering Performance // J. Web Eng. 2022. Т. 21, № 3. С. 789–813. DOI: 10.13052/jwe1540-9589.21311.
6. Буи Д. Next.Js for Front-End and Compatible Backend Solutions. South-Eastern Finland, University of Applied Science, 2023.
7. Йоханссон Й. Create React App vs NextJS, 2021.
8. Хаджин А. The Ultimate Next.js Ebook. JS Mastery, 2023.
9. Рива М. Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production. Packt Publishing, 2022.
10. Лазуарди М.Ф.С., Ангграини Д. Modern Front End Web Architectures with React.Js and Next.Js // Int. Res. J. Adv. Eng. Sci. 2022. Т. 7, № 1. С. 132–141.
11. Джартаргар Х.А., Далали С. React Apps with Server-Side Rendering: Next.js // J. Telecommun. Electron. Comput. Eng. 2022. Т. 14, № 4. С. 25–29.
12. Патель В. Analyzing the Impact of Next.JS on Site Performance and SEO // Int. J. Comput. Appl. Technol. Res. 2023. Ноябрь. DOI: 10.7753/ijcatr1210.1004.
13. Искандар Т.Ф., Лубис М., Кусумасари Т.Ф., Лубис А.Р. Comparison between client-side and server-side rendering in the web development // IOP Conf. Ser. Mater. Sci. Eng. 2020. Т. 801, № 1. DOI: 10.1088/1757-899X/801/1/012136.

Журнал «Современная наука: актуальные проблемы теории и практики»