УДК 005.591.6:681.327
DOI 10.26118/2782-4586.2025.26.66.002

Мыцык Ростислав Александрович
Кубанский Государственный Аграрный Университет им. Трубилина
Топчин Эдгар Гендрикович
Кубанский Государственный Аграрный Университет им. Трубилина
Ринас Николай Анатольевич
Кубанский Государственный Аграрный Университет им. Трубилина

Оптимизация клиент-серверного соединения

Аннотация. Данная статья написана для объяснения принципов работы клиент-серверного соединения, а также для решения проблемы и объяснения на работающем проекте.

Первоначальное исследование показало, что главная страница, оформленная с использованием тёмной цветовой гаммы и анимаций, страдала от длительных задержек при загрузке. Пользователи сталкивались с ситуацией, когда загружалась только HTML-разметка, но стили CSS применялись значительно позже. В качестве первой меры было проанализировано время загрузки из разных источников и сравнение с показателями оптимизации.

Для более глубокой оптимизации было принято решение — применить ряд подходов, направленных на улучшение производительности. В частности, одна из основных мер заключалась в улучшении обработки JavaScript и его интеграции с CSS, что позволило минимизировать время задержки между первичной загрузкой страницы и отображением стилей.

Кроме того, была внедрена система кэширования, что помогло сократить количество запросов к серверу и улучшить общую скорость загрузки.

В результате, было предложено уникальное решение, которые исправило проблему, и пользователи более не сталкивались с данной проблемой.

Ключевые слова: оптимизация, Клиент-серверное соединение, Динамический контент, Прелоадер, DevTools, JavaScript, Протокол.

Mycyk Rostislav Aleksandrovich
Kuban State Agrarian University. Trubilin
Topchin Edgar Gendrikovich
Kuban State Agrarian University. Trubilin
Rinas Nikolay Anatolevich
Kuban State Agrarian University. Trubilin

Optimization client-server connection

Abstract. This article is written to explain principles of client-server connection, as well as to solve the problem and to explanation on a working project.

Initial investigation showed that the home page, designed using a dark color scheme and animations, was suffering from long delays in loading times. animations, suffered from long loading delays. Users encountered a situation where only the HTML markup was loaded, but CSS styles were were applied much later. As a first measure, the download times from different sources were analyzed loading times from different sources and compared them to the performance of the optimization.

For more in-depth optimization, it was decided to decision - to apply a number of approaches aimed at improving performance. In particular, one of the main measures was to

improve the processing of JavaScript and its integration with CSS, which helped minimize the lag time between the initial page load and the display of styles.

In addition, a system of caching system, which helped to reduce the number of requests to the server and improve the overall loading speed.

As a result, a unique solution that fixed the problem, and users no longer encountered this problem.

Keywords: optimization, Client-server connection, Dynamic content, Preloader, DevTools, JavaScript, Protocol.

Введение

Актуальность

Актуальность темы «ОПТИМИЗАЦИЯ КЛИЕНТ-СЕРВЕРНОГО СОЕДИНЕНИЯ» обусловлена тем, что предприниматели и крупные компании стремятся переводить свои бизнес-процессы на онлайн-платформы, осуществляя это как частично, так и полностью.

Есть несколько способов это сделать:

- Веб-сайт, представляющий из себя:
- Сайт-визитку, где есть информация о компании с услугами и контактами
- Интернет-магазин с информацией о продукте и возможность его купить / заказать
- Сайт-навигатор, который перевод на другой сайт, продукт или тому подобное…
- Приложения для мобильных устройств или же для ПК, в котором есть все тоже самое, что и в веб-версии, только удобнее, а также занимает место в памяти на устройстве пользователя

Данная тема поможет ответить на сложные вопросы клиент-серверного соединения, так как эта технология используется не только в веб-приложениях, но и в обычных устанавливаемых приложений

В статье будет обсуждена важность серверного рендеринга (SSR) для масштабных проектов, а также для некоторых менее крупных решений в области веб-разработки.

Проблема

В ходе доработки веб-сайта была выявлена проблема: на одном из высоконагруженных проектов пользователи начали жаловаться на странное поведение интерфейса при загрузке страниц. Контент периодически мерцал, на доли секунды появлялся без стилей, а некоторые элементы "прыгали" по странице во время загрузки.

Данный проект – это веб-сайт, изначально созданный с использованием стандартных средств разработки, который не предусматривал использование Server Side Rendering (SSR). Однако после добавления этой технологии возникла проблема.

Главная страница оформлена в темной цветовой гамме, присутствует красивые и гармоничные анимации, фоновые видео. Одним словом – современно.

У сайта есть красивый «preloader», представляющий собой красивую анимацию перед появлением всего контента. Однако возникает момент перед появлением этого прелоадера: HTML-разметка загружается до появления прелоадера и стилей, задерживаясь на примерно одну секунду. Это заметная задержка, которая явно неприятна для пользователей.

Задача на первый взгляд не сложная, но для её решить необходимо углубиться в понятие SSR (https://education.yandex.ru/journal/server-side-rendering), а точнее в клиент-серверное соединения, ведь до добавления этой технологии, проблемы не возникало…

Анализ предшествующих работ

Любая разработка сайтов начинается с классической верстки, про основы которой рассказывает Гречкин В.А. – «В современном мире верстка сайтов играет ключевую роль в создании эффективных и удобных пользовательских интерфейсов. Выбор подходящего метода верстки зависит от потребностей проекта, требований к адаптивности и целевой

аудитории. Следует помнить, что эффективная верстка сайта – это не только красивый дизайн, но и удобство использования для всех пользователей.». [1]

CSS не используется без HTML, это неразрывные элементы, об этом немного рассказал автор Тонхоноева А.А. в своей статье – «Для качественного дизайна приложения применяются стили CSS. С помощью каскадных таблиц стилей можно достаточно легко внести изменения в оформление сайта, изменять внешний вид отдельных элементов. CSS имеет более широкий набор атрибутов по сравнению с HTML, файлы со стилями можно использовать в разных HTML-файлах.» [2]

И третьим основным средством Веб-разработчика на клиентской части является JavaScript, о котором рассказывают авторы Анарбеков Т.Р., Ачекеев К.С., Сейтказиева Н.С., Мамажунус Кызы А. – «JavaScript – это язык программирования, который используют разработчики для создания интерактивных веб-страниц. Функции JavaScript могут улучшить удобство взаимодействия пользователя с веб-сайтом. Например: от обновления ленты новостей в социальных сетях и до отображения анимации и интерактивных карт.» [12]

Предшествующие исследования в данной области предоставили ценное понимание терминологии и концепций. Однако практическое применение этих знаний может значительно варьироваться в зависимости от конкретных обстоятельств и возникающих проблем.

В данной статье рассматривается уникальный практический случай, который может послужить поучительным примером для будущих исследователей и практиков в этой области.

Методы и материалы

- **Методы и подходы к решению проблемы**:
- В статье указан сам фрагмент кода, представляющий огромную ценность, ведь именно изменив код, разработчик может повлиять на всю работу сайта;
- Также в исследовании описывается опыт работы с панелью разработчика (DevTools) и сервисом «Google PageSpeed», как с основными инструментами, которые помогут выявить проблемы в работе сайта;
- **Объекты исследования**:

Данный сайт написан при помощи следующих средств разработки:

- Фреймворк – VUE;
- Различные дополнительные плагины и фреймворки для комфортной работы и дополнительных функций;
- Классический набор разработчика веб-приложений – JS, CSS (точнее SCSS, ключевой препроцессор, упрощающий работу) HTML (в данном фреймворке файлов HTML не предвидится, вместо них файлы с расширением «.vue»);
- Веб-браузер (Brave-browser) с удобными средствами разработчика;
- И главное средство – SSR, где и кроется данная проблема…

И самый главный объект исследования: конкретная проблема, обнаруженная на высоконагруженном проекте, такая как мерцание интерфейса и неправильная загрузка контента;

- **Алгоритмы и методы оптимизации**:
- В статье описываются конкретные шаги оптимизации, такие как сокращение количества итераций, удаление неиспользуемых функций и глобальных переменных, а также реструктуризация DOM-дерева
- **Результаты и обсуждения**

Как уже было сказано, ошибка не фатальная для человека, но фатальная для компании, ведь веб-сайт – это лицо их компании, их показатель для новых клиентов, но ошибка кривого отображения может отпугнуть пользователей, а это уже потеря прибыли, что естественно критично для компании.

Процесс работы

Первоначальное отображение представляло собою непонятные стрелочки и текст, который выглядел случайным образом. Белый фон, невнятный текст и лишние элементы создавали неприятное впечатление и вызывали нежелание продолжать использовать сайт.

Кроме того, первоначальная загрузка занимала чрезмерное количество времени и ресурсов клиентского устройства. Затем начинал работать уникальный прелоадер, разработанный специально для этого проекта.

Прелоадер (от англ."*preloader*") - предварительный загрузчик, особый индикатор, который информирует пользователя о том, что страница или контент находятся в процессе загрузки.

Он выполнял заложенный функционал, правильно отображался, но, как будто это происходило не вовремя. Впоследствии отображалась главная страница, но без динамической загрузки информации с сервера. После истечения 0.2 сек все становилось нормально. То есть, изначально загружается HTML, а не CSS, что должно быть сделано наоборот.

Начался процесс выявления проблем с оптимизацией сайта

Для начала следует открыть DevTools (панель разработчика) и начать анализировать происходящее. В панели «Network Panel» была зафиксирована интересная картина - HTML успевал загрузиться и «отрендериться» до применения общих стилей. В результате пользователь видел необработанную разметку, затем прелоадер, а после — резкий скачок к стилизованной версии.

Проблема усугублялась тем, что на сайте много динамического контента, который подгружается с сервера. Каждый такой запрос мог вызвать новый каскад визуальных артефактов. Сами запросы к серверу отрабатывались порядка 2-ух секунд.

Было принято решение зайти на сервис «PAGESREED». «Google PageSpeed» — это семейство инструментов Google, Inc., разработанное для оптимизации производительности веб-сайта. Он был представлен на конференции разработчиков в 2010 году.

Фактическая статистика отработки показана на скриншоте ниже [Рисунок 1] – составлено авторами.

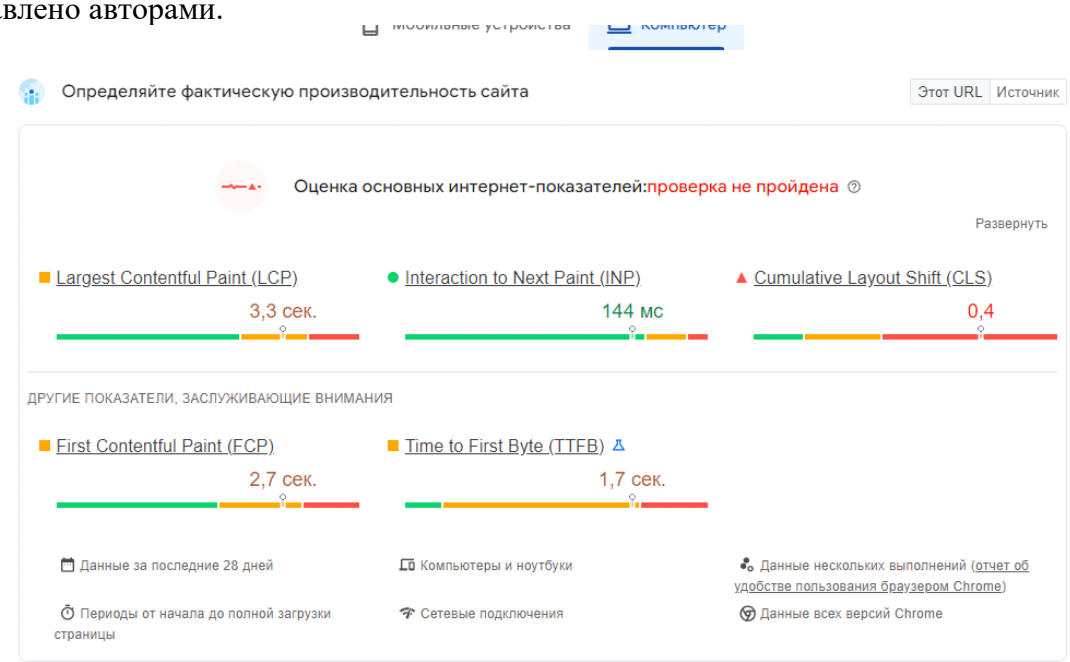


Рисунок 1 - Показания сайта в «PAGESPEED»

На данном рисунке можно отследить 5 показателей и вывод:

- *Link Control Protocol (LCP)* — протокол управления соединением;
- *Interaction to Next Paint (INP)* — это метрика производительности для измерения скорости отклика сайта после взаимодействия пользователя с веб-страницей;

• *Cumulative Layout Shift (CLS — Совокупный сдвиг вёрстки)* — метрика пользовательского опыта, измеряющая то, настолько нестабильный контент отображается пользователю;

• *First Contentful Paint (FCP)* — это время, которое проходит с момента открытия страницы и до момента, когда посетитель увидит какое-либо её содержимое (текст, изображение)

• *TTFB (Time To First Byte)* — это промежуток, прошедший с момента отправки запроса до получения первого байта информации;

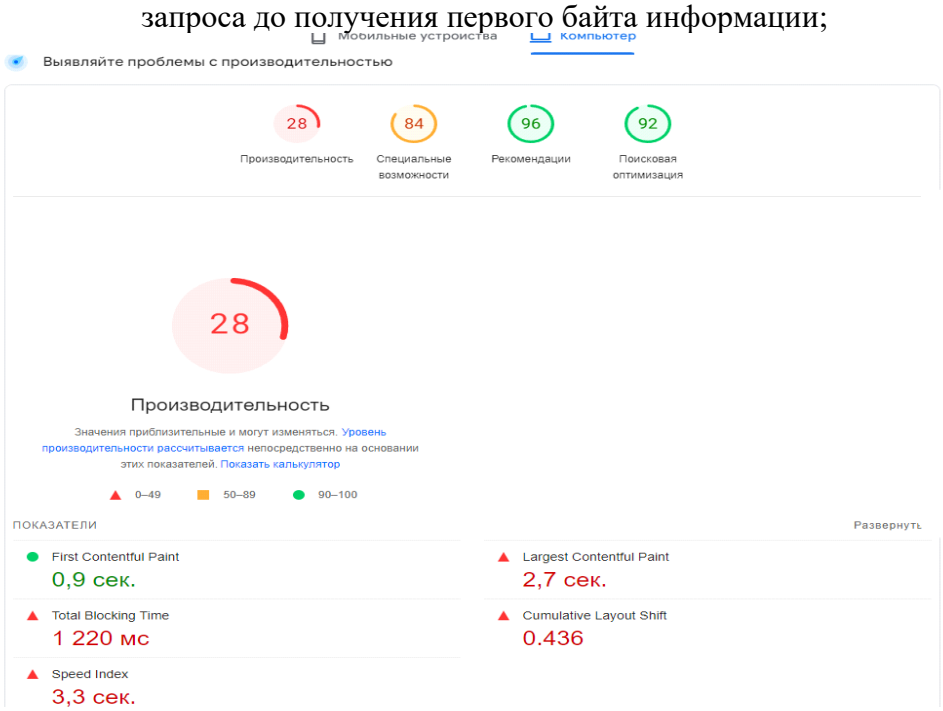


Рисунок 2 - Общий показатель производительности сайта

Не впечатляющие результаты, к сожалению. Общая производительно хуже некуда [Рисунок 2] – составлено авторами.

Но это может решить глобальную проблему, так как это возможное решение проблемы. По результатам анализа стало ясно, что скорость загрузки и время ответа сервера значительно ниже нормальных значений. Было принято решение исправить два пункта, отмеченных на скриншоте.

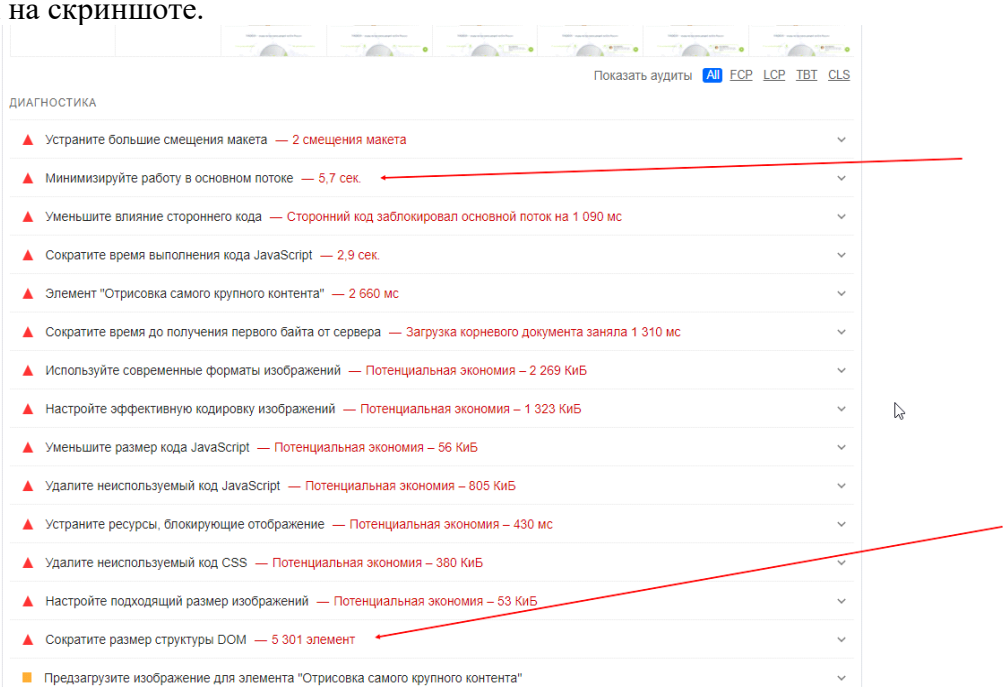


Рисунок 3 - Советы по оптимизации от Google

Благодаря современным средствам отладки можно посмотреть, из-за чего вызываются те или иные проблемы, воспользуемся этим решением [Рисунок 3] – составлено авторами.

Был проведён анализ информации по оптимизации кода на JavaScript, также с помощью различных ИИ для получения информации и советов касательно JS на проекте.

Согласно рекомендациям и изученной информации было принято несколько решений:

- сокращение количества итераций;
- чистка кода от неиспользуемых функций;
- чистка кода от глобальных переменных. (Переменные были перенесены в функции);
- реструктуризация структуры DOM дерева;

Выполнение первых трех пунктов не представляло особой сложности, однако пришлось потрудиться над DOM деревом. В данном проекте была автоматический вызов всех функций при первичной загрузке страницы на клиентском устройстве.

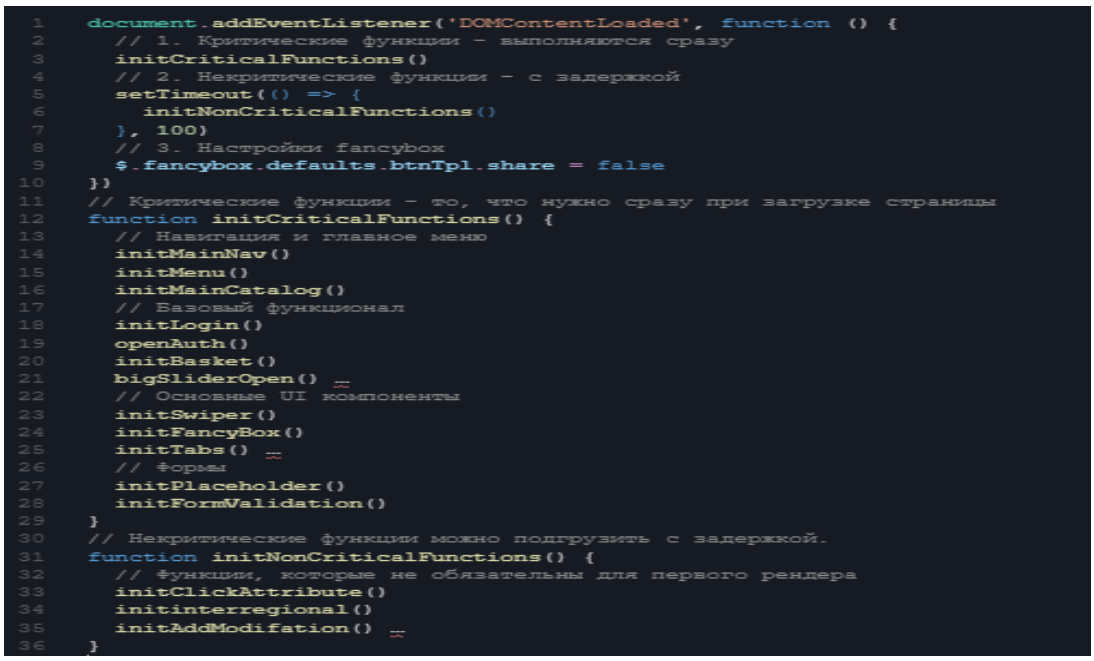
Первым делом я поставил условие на вызов функций внутри самих функций: [Рисунок 4] – составлено авторами.

```
document.addEventListener('DOMContentLoaded', () => {
  const popup_city = document.querySelector('#popup-city')
  const popup_contacts = document.querySelector('#popup-contacts-city')

  if (popup_city) {
    $.fancybox.open({
      src: `#${popup_city.id}`,
      type: 'inline',
    });
  }

  if (popup_contacts) {
    setTimeout(() => {
      const interval = setInterval(() => {
        if (document.querySelector('.popup-city._bs.fancybox-content')) return
        $.fancybox.open({
          src: `#${popup_contacts.id} `,
          type: 'inline'
        });
        clearInterval(interval)
      }, 1000)
    }, 5000)
  }
})

```

Рисунок 4 – Отрезок кода, отвечающий за условия вызова функций

Проделав данную работу с каждой функцией, было принято решение - переход к структуре DOM

Было принято решение проанализировать функции и разделить их на критически важные для первой загрузки и остальные, которые не влияют на отображение элементов. [Рисунок 5] – составлено авторами



```
document.addEventListener('DOMContentLoaded', function () {
  // 1. Критические функции - выполняются сразу
  initCriticalFunctions()
  // 2. Некритические функции - с задержкой
  setTimeout(() => {
    initNonCriticalFunctions()
  }, 100)
  // 3. Настройки fancybox
  $.fancybox.defaults.btnTpl.share = false
})
// Критические функции - то, что нужно сразу при загрузке страницы
function initCriticalFunctions() {
  // Навигация и главное меню
  initMainNav()
  initMenu()
  initMainCatalog()
  // Базовый функционал
  initLogin()
  openAuth()
  initBasket()
  bigSliderOpen() ...
  // Основные UI компоненты
  initSwiper()
  initFancyBox()
  initTabs() ...
  // Формы
  initPlaceholder()
  initFormValidation()
}
// Некритические функции можно подгрузить с задержкой.
function initNonCriticalFunctions() {
  // Функции, которые не обязательны для первого рендера
  initClickAttribute()
  initinterregional()
  initAddModifation() ...
}
```

Рисунок 5 – Отрезок кода, отвечающий за очередность вызова функций

Результаты анализа и распределения функций выявили некоторые проблемы с отображением CSS. Эти сложности решаются путем изменения стилей и переработки подхода к их вызову. После тщательного изучения информации и продолжительного мозгового штурма было найдено оптимальное решение, которое заключалось в изменении порядка вызова элементов в файле index.html. [Рисунок 6] – составлено авторами, [Рисунок 7] – составлено авторами,

```
<
<!doctype html>
<html lang="ru">

<head>
  <meta charset="UTF-8" />
  <link rel="icon" type="image/svg+xml" href="/vite.svg" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title></title>
  <! -- Предзагрузка основных стилей -->
    <link rel="preload" href="/src/assets/styles/main.css" as="style">
    <link rel="preload" href="/src/index.css" as="style">
    <! -- Базовые критические стили -->
      <style>
        body {
          margin: 0;
          padding: 0;
          opacity: 0;
          background-color: black;
        }

        body.loaded {
          opacity: 1;
          transition: opacity 0.2s ease;
        }

        #app {
          width: 100%;
          height: 100%;
        }
      </style>
</head>

```

Рисунок 6 – Отрезок кода главной страницы, где указаны мета-теги и главные стили

```
<body class="theme-dark">
  <div id="app"></div>
  <!-- Добавляем класс loaded когда страница загрузится -->
  <script>
    class ClassWatcher {
        constructor(targetNode, classToWatch, classAddedCallback, classRemovedCallback) {
            this.targetNode = targetNode
            this.classToWatch = classToWatch
            this.classAddedCallback = classAddedCallback
            this.classRemovedCallback = classRemovedCallback
            this.observer = null
            this.lastClassState = targetNode.classList.contains(this.classToWatch)
            this.init()
        }
        init() {
            this.observer = new MutationObserver(this.mutationCallback)
            this.observe()
        }
        observe() {
            this.observer.observe(this.targetNode, { attributes: true })
        }
        disconnect() {
            this.observer.disconnect()
        }
        mutationCallback = mutationsList => {
            for(let mutation of mutationsList) {
                if (mutation.type === 'attributes' && mutation.attributeName === 'class') {
                    let currentClassState = mutation.target.classList.contains(this.classToWatch)
                    if(this.lastClassState !== currentClassState) {
                        this.lastClassState = currentClassState
                        if(currentClassState) {
                            this.classAddedCallback()
                        }
                        else {
                            this.classRemovedCallback()
                        }
                    }
                }
            }
        }
    }
    let targetNode = document.body;
    function workOnClassAdd() {
        document.body.classList.add('loaded');
    }
    function workOnClassRemoval() {
        // alert("I'm triggered when the class is removed")
    }
    // watch for a specific class change
    let classWatcher = new ClassWatcher(targetNode, 'theme-white', workOnClassAdd, wor-kOnClassRemoval)
    let classWatcher2 = new ClassWatcher(targetNode, 'theme-dark', workOnClassAdd, wor-kOnClassRemoval)
    // targetNode.classList.add('trigger') // triggers workOnClassAdd callback

    window.addEventListener('load', () => {
        document.body.classList.add('loaded');
    });
  </script>
  <script type="module" src="/src/main.js"></script>
</body>
</html>»
```

Рисунок 7 – Отрезок кода главной страницы, где указаны мета-теги и главные стили

А вот оно и решение, последний элемент пазла, структура, которую необходимо было переделать. Новый порядок такой:

- Черный экран (до того, как не прогрузятся все стили)
- Прогрузка стилей
- Preloader
- Основная HTML разметка

Именно это нам и нужно, теперь нет проблемы с отображением стилей и отображением контента, порядок правильный

SSR хоть и полезная вещь, но при небольшой ошибки в проекте придётся довольно долго разбираться и искать проблему...

Заключение

Подводя итог, можно сказать, что проблема отображение динамического контента была решена. Была выявлена неполадка, которая заключалась в следующем. Первоначально загружалась HTML – разметка, но стили применялись позже. Это было неестественно, а также негативно сказывалось на восприятие веб-сайта пользователями.

Для решения этой проблемы был проведен анализ времени загрузки, проведен процесс оптимизации кода JavaScript и стилей CSS, а также был проведен рефакторинг главного компонента страницы – index.html. В коде главной страницы было добавлено свойство перезагрузки стилей до появления HTML структуры, что решило главную проблему.

Применения этих методов значительно улучшило пользовательский опыт и ускорило всю работы сайта. В результате пользователи более не сталкивались с проблемами отображения компонентов на страницах сайта. Работа над оптимизацией клиент-серверного соединения показала, что даже небольшие изменения загрузки компонентов существенно влияют на общий результат работы всего проекта.

Список источников

1. Гречкин В. А. Инструменты верстки сайтов // Национальный исследовательский университет «МЭИ». 2024. С. 91–95.
2. Тонхоноева А. А. Средства разработки web-приложений // Бурятский государственный университет имени Доржи Банзарова. 2019. С. 36–39.
3. Кузьмичев И. П. Профилирование и оптимизация Web Приложений // Псковский государственный университет. 2023. С. 197–200.
4. Берьянов М. С. Сравнение отрисовки сайта на стороне клиента (CSR) и на стороне сервера (SSR) // Университет ИТМО. 2023. С. 39–41.
5. Агафонова М. В. Оптимизация рендеринга веб-приложения на основе frontend-фреймворка // Университет ИТМО. 2023. С. 47–52.
6. Филисов Д. А. Высвобождение производительности: глубокое погружение в приложения с высокой нагрузкой // Инновационная Наука. 2023. С. 37–47.
7. Кравцов Е. П. Разработка высокопроизводительных react-приложений: методы и практики оптимизации // Sberdevices. 2024. С. 53–58.
8. Azhariyah S., Muhammad Mukhlis Framework CSS: tailwind css untuk front-end website store Pt. XYZ // Politeknik Negeri Subang. 2024. С. 30–36.
9. Перфильев Н. В., Макаров Д. А. Оптимизация сборки frontend-приложений // Забайкальский государственный университет. 2019. С. 44–46.
10. Насиров Э. Ф., Кириллов Д. С., Чернова М. В. Фреймворк javascript VueJS // ФГБОУ ВО «Казанский национальный исследовательский технологический университет». 2020. С. 65–67.
11. Nugroho M. A., Wuryanto E., Faqih K. Optimizing uncapacitated facility location problem with cuckoo search algorithm based on gauss distribution // Airlangga University. 2023. С. 361.
12. Анарбеков Т. Р., Ачекеев К. С., Сейтказиева Н. С., Мамажунус Кызы А. Разработка компьютерной игры "2024 " на языке программирование java script и CSS // Кыргызский государственный университет имени И. Арабаева. 2023. С. 24–32.
13. A Jartarghar H., Rao Salanke G., A.R A.K., G.S SH., Dalali SH. React apps with server-side rendering: Next.JS // Department of Computer Science and Engineering, R.V College of Engineering, Bengaluru, India. Don Bosco Institute of Technology, Bengaluru, India. 2022. С. 25–29.
14. Кирилов Р. Ф., Кудряшова А. А., Пузанков К. Р., Смолин А. А. Оптимизация производительности веб-приложений на Vue.js и React: лучшие практики и инструменты // ОГАПОУ «Ульяновский авиационный колледж - Межрегиональный центр компетенций». 2024. С. 31–35.

Сведения об авторах

Мыцык Ростислав Александрович, студент 3-го курса кафедры «Информационных Систем и технологий», Кубанский государственный аграрный университет им. Трубилина, г. Краснодар, Россия

Топчин Эдгар Гендрикови, студент 3-го курса кафедры «Информационных Систем и технологий», Кубанский государственный аграрный университет им. Трубилина, г. Краснодар, Россия

Ринас Николай Анатольевич, доцент, кандидат технических наук Кубанский государственный аграрный университет им. Трубилина, г. Краснодар, Россия

Information about the authors

Mycyk Rostislav Aleksandrovich, 3rd year student of the Department of Information Systems and Technologies, Kuban State Agrarian University named after Trublina, Krasnodar, Russia

Topchin Edgar Gendrikovich, 3rd year student of the Department of Information Systems and Technologies, Kuban State Agrarian University named after Trublina, Krasnodar, Russia

Rinas Nikolay Anatolyevich, Associate Professor, Candidate of Technical Sciences, Kuban State Agrarian University named after Trublina, Krasnodar, Russia