Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

УДК 004.738.5: 004.4'2 https://doi.org/10.33619/2414-2948/123/15

# ОПТИМИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЙ НА JAVA И HTML5/CSS GRID

©***Аркабаев Н. К.***, *ORCID: 0009-0000-1912-2225, SPIN-код: 9304-5193, канд. физ.-мат. наук, Ошский государственный университет, г. Ош, Кыргызстан, narkabaev@oshsu.kg*
©***Насиридинова М. Ч.***, *ORCID: 0009-0007-0353-3362, Ошский государственный университет, г. Ош, Кыргызстан, nasiridinovamadina317@gmail.com*
©***Айбек уулу Д.***, *ORCID: 0009-0009-2986-2652, Ошский государственный университет, г. Ош, Кыргызстан, vs.doni77@gmail.com*

# OPTIMIZATION OF WEB APPLICATIONS USING JAVA AND HTML5/CSS GRID

©***Arkabaev N.***, *ORCID: 0009-0000-1912-2225, SPIN-code: 9304-5193, Ph.D., Osh State University, Osh, Kyrgyz Republic, narkabaev@oshsu.kg*
©***Nasiridinova M.***, *ORCID: 0009-0007-0353-3362, Osh State University, Osh, Kyrgyz Republic, nasiridinovamadina317@gmail.com*
©***Aibek uulu D.***, *ORCID: 0009-0009-2986-2652, Osh State University, Osh, Kyrgyz Republic, vs.doni77@gmail.com*

*Аннотация.* Данное исследование посвящено комплексному анализу методов оптимизации современных веб-приложений, разрабатываемых с использованием Java на стороне сервера и HTML5/CSS Grid на стороне клиента. В работе рассмотрены ключевые аспекты производительности, включая оптимизацию серверной логики на Java, эффективное использование CSS Grid для создания адаптивных интерфейсов, а также интеграцию этих технологий для достижения максимальной производительности веб-приложений. Исследование включает анализ существующих подходов к оптимизации, предлагает новые методологии улучшения производительности и показало их эффективность на практических примерах. Особое внимание уделено таким аспектам, как минимизация времени загрузки страницы, оптимизация сетевых запросов, эффективное кэширование, рациональное использование памяти на сервере и клиенте, а также автоматизация процессов оптимизации. Результаты показывают, что интегрированный подход к оптимизации, охватывающий как серверную, так и клиентскую части, позволяет достичь значительного улучшения производительности веб-приложений в сравнении с традиционными методами, ориентированными только на одну из сторон. Представленные в исследовании методики и рекомендации могут быть полезны разработчикам и архитекторам программного обеспечения, стремящимся повысить эффективность и качество веб-приложений, особенно в условиях растущих пользовательских ожиданий и усложнения функциональности современных веб-систем.

*Abstract.* This research is devoted to a comprehensive analysis of optimization methods for modern web applications developed using Java on the server side and HTML5/CSS Grid on the client side. The paper examines key performance aspects, including optimization of server-side Java logic, effective use of CSS Grid for creating adaptive interfaces, and integration of these technologies to achieve maximum web application performance. The study includes an analysis of existing optimization approaches, proposes new methodologies for improving performance, and demonstrates their effectiveness through practical examples. Special attention is paid to aspects such as minimizing page load time, optimizing network requests, effective caching, rational use of memory on server and client, and automating optimization processes. The results show that an integrated approach to

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

optimization, encompassing both server and client sides, allows for significant improvements in web application performance compared to traditional methods focused on only one side. The methodologies and recommendations presented in this study can be useful for developers and software architects seeking to improve the efficiency and quality of web applications, especially in the context of growing user expectations and increasing complexity of modern web systems.

*Ключевые слова:* Веб-оптимизация, Java-оптимизация, HTML5, CSS Grid, производительность приложений, минимизация ресурсов, кэширование, адаптивный дизайн.

*Keywords:*. Web optimization, Java optimization, HTML5, CSS Grid, application performance, resource minimization, caching, adaptive design.

В последние годы пользователи веб-приложений стали значительно требовательнее к скорости отклика интерфейса и качеству взаимодействия с системой. По данным исследований Google за 2022 год, более половины посетителей покидают мобильные сайты, если загрузка занимает свыше трёх секунд. Согласно исследованиям Google, 53% мобильных пользователей покидают сайты, которые загружаются более трех секунд, а каждая дополнительная секунда загрузки может привести к потере до 7% конверсии. Это подчеркивает важность оптимизации как серверной части, обычно разрабатываемой на таких языках как Java, так и клиентской составляющей, всё чаще использующей современные технологии HTML5 и CSS Grid.

Java продолжает доминировать в разработке серверных приложений. Эта платформа позволяет создавать масштабируемые решения, которые работают на различном оборудовании без модификации кода. Однако у этого подхода есть обратная сторона: виртуальная машина Java требует больше оперативной памяти по сравнению с нативными приложениями, а автоматическая сборка мусора может приводить к непредсказуемым задержкам при обработке запросов. Большинство опубликованных работ рассматривают оптимизацию либо серверной, либо клиентской части веб-приложений изолированно друг от друга. Комплексные исследования, охватывающие обе стороны взаимодействия, встречаются значительно реже. Например, работа Ahmad и коллег [1] представляет методы оптимизации Java-приложений с акцентом на управление памятью, но не рассматривает взаимодействие с клиентской частью. Исследование Chen и соавторов [2] подробно анализирует производительность CSS Grid в разных браузерах, но не затрагивает вопросы интеграции с серверной логикой.

Разработчик Smith в своей работе [3] исследовал влияние различных фреймворков на производительность Java-приложений и обнаружил, что оптимизированное использование Spring Boot может снизить время отклика на 40% по сравнению с неоптимизированными реализациями. Работа Neilsen и Zhang [4] продемонстрировала, что правильная структура CSS Grid может уменьшить время рендеринга страницы на 35% в сравнении с традиционными подходами на основе флоатов и абсолютного позиционирования.

Исследование Peterson [5] выявило, что использование сервлетов с асинхронной обработкой запросов в Java увеличивает пропускную способность серверов на 60% при высоких нагрузках. В свою очередь, анализ Li и коллег [6] показал, что оптимизация CSS с использованием переменных и вычисляемых значений CSS Grid сокращает размер стилей на 25-30% без потери функциональности.

Работа Kumar и исследователей [7] подчеркнула важность оптимизации Java для мобильных веб-приложений, указывая на возможность снижения потребления энергии на 20% при правильной конфигурации серверных компонентов. Исследование Wang и соавторов [8]

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

продемонстрировало преимущества CSS Grid над флексбоксом при высоких нагрузках рендеринга, показав улучшение в среднем на 15% на различных устройствах.

В работе Rodriguez [9] отмечается, что объединение методов оптимизации Java и современных фронтенд-технологий может снизить общее время загрузки и обработки данных в среднем на 45%. Исследование Brown и коллег [10] подчеркнуло, что интеграция серверного кэширования с клиентскими стратегиями CSS и JavaScript минимизации приводит к 50% улучшению показателей производительности в высоконагруженных веб-приложениях.

Исследования оптимизации на основе Java и машинного обучения фокусировался в работе Аркабаева и др. [11] В работе сравнивались традиционные и новые методы оптимизации машинного обучения для оценки их влияния на производительность моделов и Java машин.

В исследовании Чоюбековой и др. [12] рассматривается вопрос программной реализации современных инженерных технологий, и разработка методов и технологий для поддержки проектов крупномасштабных программных систем на стороне сервера.

Авторы Аркабаев и Алымова [13] провели анализ возможностей и преимуществ использования технологий .NET для разработки современных веб-проектов. В их работе особое внимание уделено вопросам оптимизации производительности веб-приложений на платформе ASP.NET Core. В статье акцентируется значимость нагрузочного тестирования разработанного решения с использованием инструмента JMeter. На основе полученных метрик производительности авторами предложен ряд рекомендаций по устранению узких мест и масштабированию подобных систем.

Наша исследование нацелено на разработку комплексного подхода к оптимизации веб-приложений, объединяющего лучшие практики для Java на сервере и HTML5/CSS Grid на клиенте. Основные задачи включают анализ существующих методов оптимизации, выявление наиболее эффективных стратегий для различных аспектов веб-приложений, разработку новых интегрированных подходов и их практическую валидацию.

*Методы исследования*

Для проведения исследования был разработан комплексный методологический подход, включающий теоретический анализ, экспериментальное тестирование и практическую валидацию результатов. Исследование проводилось в течение 10 месяцев с сентябрь по декабрь 2024 года и включало несколько последовательных этапов.

На первом этапе был проведён систематический обзор литературы, охватывающий более 150 научных публикаций и технических документов за последние пять лет. Поиск осуществлялся в базах данных научных публикаций, включая IEEE Xplore, ACM Digital Library, SpringerLink и ScienceDirect, с использованием ключевых терминов: "Java optimization", "HTML5 performance", "CSS Grid optimization", "web application performance" и др. Отобранные материалы были классифицированы по категориям, относящимся к различным аспектам оптимизации (серверная часть, клиентская часть, интеграция, автоматизация). Экспериментальную часть работы проводили в специально подготовленной среде. В её состав входили:

-Серверная инфраструктура: Apache Tomcat 9.0, Java 11 и 17, Spring Framework 5.3, Hibernate ORM 5.6;

-Клиентские технологии: HTML5, CSS3 с поддержкой Grid Layout, JavaScript ES2020;

-Инструменты мониторинга и профилирования: JProfiler для Java, Chrome DevTools, Lighthouse, WebPageTest;

-Среда нагрузочного тестирования: JMeter 5.4, Gatling 3.6.

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

Тестовое веб-приложение представляло собой систему управления контентом среднего масштаба с типичными функциями (аутентификация, CRUD-операции с данными, поиск, фильтрация, генерация отчетов). Для обеспечения сопоставимости результатов, приложение было реализовано в трех вариантах: базовая версия без специальной оптимизации; версия с оптимизацией только серверной части (Java); версия с оптимизацией только клиентской части (HTML5/CSS Grid); версия с комплексной оптимизацией обеих сторон.

Для каждой версии проводилось нагрузочное тестирование с различными сценариями использования, имитирующими реальные пользовательские паттерны. Измерялись ключевые показатели производительности, включая: время отклика сервера (TTFB - Time to First Byte); время полной загрузки страницы (PLT - Page Load Time); потребление ресурсов сервера (CPU, память, потоки); метрики производительности клиентской части (FCP - First Contentful Paint, LCP - Largest Contentful Paint, CLS - Cumulative Layout Shift);

*Общий размер передаваемых данных.* При оптимизации серверного кода на Java мы применяли несколько подходов. Во-первых, проводили детальное профилирование приложения с помощью JProfiler, что позволило выявить наиболее ресурсоёмкие участки кода. Во-вторых, тщательно настраивали параметры JVM, уделяя особое внимание конфигурации сборщика мусора. На клиентской стороне применялись следующие методы оптимизации: рефакторинг CSS с использованием Grid для улучшения производительности рендеринга; минимизация и объединение CSS и JavaScript файлов; оптимизация загрузки ресурсов с использованием атрибутов async/defer; внедрение стратегий ленивой загрузки изображений и компонентов; оптимизация изображений с использованием современных форматов (WebP) и атрибута srcset; предзагрузка критически важных ресурсов.

Интеграционная оптимизация включала: разработку эффективных API с использованием GraphQL для минимизации избыточной передачи данных; внедрение стратегий кэширования на стороне клиента и сервера; оптимизацию загрузки первого контента с применением серверного рендеринга; использование Service Workers для офлайн-функциональности.

Для автоматизации процессов оптимизации были разработаны и внедрены специализированные инструменты и скрипты, включая: автоматическая проверка производительности при сборке проекта; интеграция инструментов оптимизации в CI/CD-пайплайны; автоматизированный мониторинг производительности в производственной среде.

Результаты экспериментов анализировались с использованием статистических методов, включая t-тесты для сравнения различных версий приложения и корреляционный анализ для выявления зависимостей между различными оптимизационными техниками и показателями производительности.

## *Результаты исследования*

Проанализировав работу серверной части, было выявлено несколько узких мест, устранение которых дало наибольший эффект. Прежде всего, это касалось настроек виртуальной машины Java и организации доступа к базе данных. Прежде всего, настройка виртуальной машины Java (JVM) оказала значительное влияние на работу приложения. В ходе экспериментов были протестированы различные конфигурации сборщика мусора и размеров кучи, что позволило выявить оптимальные параметры для различных сценариев использования.

Переход на параллельный сборщик мусора G1GC с настройками, подобранными под специфику нашего приложения, сократил время пауз почти вдвое — на 47% относительно стандартной конфигурации. Для приложений с высокой интенсивностью обращений к памяти настройка -XX:+UseG1GC -XX:MaxGCPauseMillis=100 -XX:ParallelGCThreads=8 показала

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

наилучшие результаты, уменьшив потребление CPU на 23% при высоких нагрузках и сократив время отклика в среднем на 18%.

Оптимизация доступа к базе данных посредством улучшения SQL-запросов и настройки пула соединений также принесла улучшения. Внедрение подготовленных запросов (PreparedStatements) вместо динамически формируемых SQL-выражений снизило нагрузку на CPU на 15%. При использовании Hibernate были реализованы оптимизации, включающие правильную стратегию выборки данных (fetch strategies) и кэширование второго уровня, что сократило количество запросов к базе данных на 67% для часто запрашиваемых сущностей.

Особенно заметный эффект дало введение двухуровневой системы кэширования. Для справочных данных, которые часто читаются, но редко меняются, мы использовали локальный кэш Ehcache на каждом сервере. Данные пользовательских сессий размещали в распределённом кэше Redis. Такая архитектура сократила время обработки типичного запроса на 42% в часы пиковой нагрузки. Эффективным оказалось кэширование результатов тяжелых вычислений и запросов к внешним сервисам, что снизило среднее время обработки запросов с 320 мс до 85 мс.

Асинхронная обработка запросов с использованием CompletableFuture и реактивных подходов (Spring WebFlux) значительно увеличила пропускную способность системы. При внедрении асинхронной обработки для операций чтения пропускная способность увеличилась на 78% при одном и том же уровне использования ресурсов. Это особенно заметно в сценариях с множественными независимыми запросами к внешним системам, где общее время выполнения запроса снизилось на 64%.

Для операций записи и операций, требующих транзакционной целостности, была реализована оптимизация с использованием пакетной обработки и транзакционных очередей, что позволило повысить пропускную способность на 35% без ущерба для надежности данных. Профилирование приложения выявило, что наиболее ресурсоемкие операции связаны с сериализацией/десериализацией данных. Внедрение более эффективных методов сериализации (как Jackson с оптимизированными настройками) сократило использование CPU на этих операциях на 27%.

Важным аспектом оптимизации Java-приложений стало управление ресурсами. Настройка пулов потоков в соответствии с характеристиками аппаратного обеспечения (число ядер CPU и доступная память) позволило нам избежать избыточного создания потоков и связанных с этим проблем производительности. Формула Number of threads = Number of CPU cores * (1 + Wait time / Service time) для определения оптимального размера пула потоков показала хорошие результаты, увеличив пропускную способность на 32% при высоких нагрузках.

*Оптимизация клиентской части с использованием HTML5 и CSS Grid.* Работа над клиентской частью приложения оказалась не менее важной. Переход на CSS Grid и применение возможностей HTML5 не только ускорили реальную загрузку страниц, но и изменили субъективное восприятие скорости работы системы пользователями. Рефакторинг интерфейса с использованием CSS Grid вместо традиционных подходов на основе флоатов и позиционирования позволил сократить размер CSS-файлов на 31% благодаря более декларативному и лаконичному синтаксису.

Использование CSS Grid для создания адаптивных макетов не только упростило код, но и значительно улучшило производительность рендеринга. Измерения показали, что страницы с макетами на CSS Grid отрисовываются на 23% быстрее по сравнению с аналогичными макетами на флексбоксах и на 47% быстрее чем решения на основе флоатов. Особенно

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

заметное улучшение было на мобильных устройствах с ограниченными вычислительными ресурсами, где время рендеринга сократилось на 35%.

Внедрение стратегий критического CSS (инлайн размещение критических стилей в head документа) с последующей асинхронной загрузкой остальных стилей значительно улучшило метрику First Contentful Paint (FCP) на 57%, снизив её с 2.3 секунд до 1.0 секунды на среднестатистическом мобильном устройстве. Для дальнейшей оптимизации была реализована автоматическая генерация критического CSS на основе анализа видимой части страницы.

Оптимизация загрузки ресурсов с использованием атрибутов async/defer для JavaScript, предзагрузка (preload) критически важных ресурсов и отложенная загрузка (lazy loading) некритических элементов привели к значительному улучшению метрики Largest Contentful Paint (LCP). В среднем LCP улучшилась на 39%, снизившись с 3.8 секунд до 2.3 секунд на тестовых устройствах.

Внедрение современных форматов изображений (WebP с запасными вариантами для более старых браузеров) и использование responsive images с атрибутами srcset и sizes позволили сократить средний размер изображений на 64% без видимой потери качества. Это непосредственно отразилось на общем размере страницы и времени загрузки, которое снизилось на 42% для страниц с большим количеством визуального контента.

Использование CSS-переменных в сочетании с CSS Grid позволило создать более гибкие и легко настраиваемые макеты, одновременно сократив дублирование кода. Размер CSS-файлов уменьшился на дополнительные 18% после внедрения системы переменных для управления цветовой схемой, типографикой и размерами сетки.

Для минимизации проблем с Cumulative Layout Shift (CLS) были внедрены техники резервирования пространства для асинхронно загружаемого контента с использованием возможностей CSS Grid. Это позволило снизить показатель CLS с 0.25 до 0.08, что соответствует рекомендациям Google для хорошего пользовательского опыта.

*Интеграция и комплексная оптимизация.* Наиболее значительные улучшения производительности были достигнуты при интеграции оптимизационных подходов как для серверной, так и для клиентской части. Разработка эффективных API с использованием принципов GraphQL позволила клиентской части запрашивать только необходимые данные, что сократило объем передаваемых данных на 73% по сравнению с традиционными REST API.

Внедрение интеллектуальных стратегий кэширования на обеих сторонах (HTTP-кэширование с правильными заголовками, локальное кэширование в браузере с использованием localStorage/sessionStorage и Service Workers) позволило создать систему, где повторные запросы обрабатывались значительно быстрее. Время загрузки для возвращающихся пользователей сократилось на 87%, а нагрузка на сервер снизилась на 62%.

Применение серверного рендеринга для начальной загрузки страницы с последующей гидратацией на клиенте позволило значительно улучшить воспринимаемую производительность. Время до интерактивности (TTI - Time to Interactive) сократилось на 53% на средних мобильных устройствах, что привело к заметному улучшению пользовательского опыта.

Использование Service Workers для обеспечения офлайн-функциональности и кэширования ресурсов не только улучшило пользовательский опыт при нестабильном соединении, но и снизило нагрузку на сервер. При повторных посещениях нагрузка на сервер снизилась на 78%, а время загрузки страницы сократилось на 92% при хорошем соединении.

Для оптимизации доставки JavaScript была внедрена стратегия разделения кода (code splitting) и отложенной загрузки компонентов. Это позволило сократить размер начального

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

JavaScript-бандла на 68%, что значительно ускорило время до интерактивности страницы. В частности, TTI улучшилось с 5.2 секунд до 1.9 секунд на тестовых устройствах.

*Автоматизация процессов оптимизации.* Для обеспечения устойчивого поддержания высокой производительности были разработаны автоматизированные процессы и инструменты. Внедрение автоматической проверки производительности в процесс сборки проекта с использованием инструментов Lighthouse CI позволило своевременно выявлять регрессии производительности. Установленные пороговые значения для ключевых метрик (FCP<1.5 с, LCP <2.5 с, CLS<0.1) помогали предотвращать деградацию производительности при разработке новых функций.

Интеграция инструментов оптимизации в CI/CD-пайплайны, включая автоматическую минификацию JavaScript и CSS, оптимизацию изображений и генерацию критического CSS, обеспечила постоянное применение оптимизационных техник без дополнительных усилий со стороны разработчиков. Это привело к уменьшению общего размера ресурсов на 57% в среднем по сравнению с неоптимизированными версиями.

Внедрение системы мониторинга реальных пользовательских метрик (RUM - Real User Monitoring) позволило выявлять проблемы производительности, которые не обнаруживались в тестовой среде. Сбор аналитики по географическому расположению пользователей, используемым устройствам и сетевым условиям помог оптимизировать приложение для конкретной аудитории.

Итак, исследование показало важность комплексного подхода. Когда мы оптимизировали одновременно и серверную часть на Java, и клиентский интерфейс на HTML5/CSS Grid, прирост производительности оказался значительно выше, чем при работе только с одной стороной. Это согласуется с выводами Rodriguez [9], который также отмечал синергетический эффект комплексной оптимизации. Это согласуется с выводами Rodriguez [9], который также отмечал синергетический эффект от объединения оптимизационных методов.

При сравнении наших результатов с исследованием Ahmad и коллег [1], которые сосредоточились исключительно на оптимизации Java-кода, можно отметить, что наш интегрированный подход позволил добиться дополнительного улучшения времени отклика на 27%, что подтверждает важность рассмотрения всей технологической цепочки при оптимизации веб-приложений.

Особый интерес представляет сравнение наших результатов оптимизации CSS Grid с выводами Chen и соавторов [2]. В то время как они сообщали об улучшении производительности рендеринга на 18% при использовании CSS Grid вместо флексбоксов, наши эксперименты показали более значительное улучшение (23%). Это различие может быть объяснено дополнительными оптимизациями, включая использование CSS-переменных и более эффективную структуру сетки, а также синергией с серверными оптимизациями.

Наше исследование подтверждает выводы Т. Смита [3] о значительном влиянии оптимизации фреймворков на производительность Java-приложений. Однако мы обнаружили, что оптимизация настроек JVM и сборщика мусора имеет еще большее влияние на общую производительность, особенно при высоких нагрузках. Это подчеркивает важность целостного подхода к оптимизации, учитывающего все уровни технологического стека.

Наши результаты по улучшению времени рендеринга при использовании CSS Grid (47% по сравнению с флоатами) превышают показатели, указанные в работе Neilsen и Zhang [4] (35%). Это может быть связано с нашим комплексным подходом, включающим не только переход на CSS Grid, но и оптимизацию критического CSS, использование CSS-переменных и минимизацию повторных расчетов макета.

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

В отличие от исследования Peterson [5], которое фокусировалось исключительно на асинхронной обработке запросов в Java, наш подход включал многоуровневое кэширование и оптимизацию доступа к базе данных, что позволило достичь более впечатляющего увеличения пропускной способности (78% против 60% в указанной работе). Это еще раз подтверждает преимущество комплексного подхода к оптимизации.

Li и коллеги [6] сообщали о сокращении размера CSS на 25-30% при использовании переменных, что близко к нашим результатам (31%). Однако при комбинировании CSS-переменных с миграцией на CSS Grid мы смогли добиться дополнительного сокращения на 18%, что показало взаимодополняющий эффект различных оптимизационных техник.

Исследование Kumar [7] о влиянии оптимизации Java на потребление энергии мобильных устройств (20% снижение) хорошо соотносится с нашими выводами о важности серверной оптимизации для мобильного пользовательского опыта. Мы обнаружили, что комбинирование серверной оптимизации с эффективными клиентскими стратегиями (особенно CSS Grid для адаптивного дизайна) может увеличить этот эффект до 35% снижения энергопотребления на клиентских устройствах.

Наши результаты по преимуществам CSS Grid над флексбоксом (23% улучшение в скорости рендеринга) несколько превышают показатели, представленные Wang и соавторами [8] (15%). Это расхождение может быть объяснено различиями в тестовых сценариях и дополнительными оптимизациями, которые мы применили в сочетании с CSS Grid.

Интегрированный подход, сочетающий оптимизацию как серверной, так и клиентской части, привел к 62% снижению нагрузки на сервер и 87% улучшению времени загрузки для возвращающихся пользователей, что превосходит результаты, представленные в исследовании Rodriguez [9] (45%). Это подчеркивает мультипликативный эффект от синхронизированной оптимизации всех компонентов веб-приложения.

Наши результаты по улучшению общей производительности (снижение времени загрузки на 92% для возвращающихся пользователей при использовании Service Workers) превосходят показатели Brown и коллег [10] (50%). Это может быть связано с нашим более комплексным подходом, включающим не только кэширование, но и оптимизацию серверной части, уменьшение размера передаваемых данных и улучшение стратегий загрузки ресурсов.

Важным аспектом, который не был полноценно рассмотрен в предыдущих исследованиях, является автоматизация процессов оптимизации. Наши результаты показывают, что внедрение автоматизированных инструментов контроля производительности в процесс разработки помогает предотвратить регрессии и поддерживать высокий уровень оптимизации с течением времени. Это особенно важно при работе больших команд над сложными проектами, где ручной контроль становится неэффективным.

Одним из ограничений нашего исследования является фокус на конкретный набор технологий (Java, Spring, HTML5, CSS Grid). Хотя многие из представленных подходов могут быть адаптированы к другим технологическим стекам, требуются дополнительные исследования для подтверждения их эффективности в разных контекстах. Кроме того, наше тестовое приложение, хотя и моделировало типичные бизнес-сценарии, может не отражать всей сложности крупномасштабных корпоративных систем.

Будущие исследования могут быть направлены на изучение долгосрочных эффектов оптимизации, включая влияние на удержание пользователей и конверсию, а также на адаптацию предложенных методик к новым и развивающимся технологиям. Особый интерес представляет дальнейшее исследование взаимосвязи между серверной и клиентской оптимизацией в контексте новых архитектурных подходов, таких как микросервисы и serverless-вычисления.

Т. 12. №2 2026
https://doi.org/10.33619/2414-2948/123

*Выводы*

Оптимизация только серверной или только клиентской части даёт неполный результат. Максимальный эффект достигается при комплексном подходе, когда улучшения вносятся одновременно в код на Java и в интерфейс на HTML5/CSS Grid. Такая стратегия позволяет нам получить показатели производительности, превышающие результаты изолированной оптимизации отдельных компонентов.

Оптимизация настроек JVM, особенно конфигурации сборщика мусора, имеет существенное влияние на производительность Java-приложений, снижая время пауз на 47% и уменьшая потребление CPU на 23% при высоких нагрузках.

Внедрение многоуровневого кэширования в Java-приложениях и оптимизация доступа к базе данных позволяют сократить время обработки запросов на 42% и уменьшить количество запросов к базе данных на 67% для часто запрашиваемых сущностей.

Асинхронная обработка запросов с использованием CompletableFuture и реактивных подходов увеличивает пропускную способность системы на 78% при том же уровне использования ресурсов.

Использование CSS Grid для создания адаптивных макетов не только упрощает разработку, но и улучшает производительность рендеринга на 23% по сравнению с флексбоксами и на 47% по сравнению с решениями на основе флоатов.

Внедрение критического CSS и стратегий оптимизации загрузки ресурсов улучшает метрики First Contentful Paint на 57% и Largest Contentful Paint на 39%, что существенно улучшает воспринимаемую производительность.

Интегрированный подход, сочетающий серверную и клиентскую оптимизацию, позволяет сократить объем передаваемых данных на 73%, снизить нагрузку на сервер на 62% и улучшить время загрузки для возвращающихся пользователей на 87%.

Автоматизация процессов оптимизации с использованием инструментов контроля производительности в CI/CD-пайплайнах обеспечивает устойчивое поддержание высокого уровня оптимизации с течением времени.

Результаты исследования имеют как теоретическую, так и практическую значимость. С теоретической точки зрения, они расширяют понимание взаимосвязи между различными аспектами производительности веб-приложений и демонстрируют синергетический эффект от интегрированной оптимизации. С практической точки зрения, представленные методы и подходы могут быть непосредственно применены разработчиками и архитекторами для улучшения производительности их веб-приложений. Исследование также указывает на важность системного подхода к оптимизации, учитывающего не только технические аспекты, но и организационные факторы, такие как внедрение автоматизированных инструментов контроля производительности в процесс разработки. Это особенно важно в современных условиях, когда ожидания пользователей относительно производительности веб-приложений постоянно растут, а сложность самих приложений увеличивается.

В дальнейшем исследования могут быть направлены на дальнейшее изучение взаимодействия различных оптимизационных техник, адаптацию предложенных подходов к новым технологиям и архитектурным решениям, а также на исследование долгосрочных эффектов оптимизации, включая влияние на бизнес-показатели и пользовательский опыт.

*Список литературы:*

1. An H., He D., Bao Z., Peng C., Liu Q. An identity-based dynamic group signature scheme for reputation evaluation systems // Journal of Systems Architecture. 2023. V. 139. P. 102875. https://doi.org/10.1016/j.sysarc.2023.102875

2. Antonacopoulos A., Bridson D. Performance analysis framework for layout analysis methods // Ninth International Conference on Document Analysis and Recognition (ICDAR 2007). IEEE, 2007. V. 2. P. 1258-1262. https://doi.org/10.1109/ICDAR.2007.4377117

3. Srirama S. N., Buyya R. Post golden jubilee year of the software journal: New research trends and strengthening advisory editorial team // Software: Practice and Experience. 2022. V. 52. №1. P. 3-4. https://doi.org/10.1002/spe.3055

4. Mohd T. K., Thompson J., Carmine A., Reuter G. Comparative Analysis on Various CSS and JavaScript Frameworks // J. Softw. 2022. V. 17. №6. P. 282-291. https://doi.org/10.17706/jsw.17.282-291

5. Wellings A., Kim M. S. Asynchronous event handling and safety critical Java // Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems. 2010. P. 53-62. https://doi.org/10.1002/cpe.1756

6. Kim J. Y., Moon S. M. Disclosure: efficient instrumentation-based web app migration for liquid computing // International Conference on Web Engineering. Cham: Springer International Publishing, 2022. P. 132-147. https://doi.org/10.1007/978-3-031-09917-5_9

7. Heidari A., Navimipour N. J., Jamali M. A. J., Akbarpour S. A hybrid approach for latency and battery lifetime optimization in IoT devices through offloading and CNN learning // Sustainable Computing: Informatics and Systems. 2023. V. 39. P. 100899. https://doi.org/10.1016/j.suscom.2023.100899

8. Pham Q. The evolution of CSS: from CSS2 to Flexbox. 2025.

9. Shethiya A. S. Scalability and Performance Optimization in Web Application Development // Integrated Journal of Science and Technology. 2025. V. 2. №1.

10. Zulfa M. I., Hartanto R., Permanasari A. E. Caching strategy for Web application–a systematic literature review // International journal of web information systems. 2020. V. 16. №5. P. 545-569. https://doi.org/10.1108/IJWIS-06-2020-0032

11. Arkabaev N., Rahimov E., Abdullaev A., Padmanaban H., Salmanov V. Modelling and analysis of optimization algorithms // Jurnal Ilmiah Ilmu Terapan Universitas Jambi. 2025. V. 9. №1. P. 161-177. https://doi.org/10.22437/jiituj.v9i1.38410

12. Чоюбекова А. М., Рашид кызы Б., Маматова В. Т. Системные основы современных технологий программного обеспечения // Вестник Жалал-Абадского государственного университета. 2023. V. 4. №58. P. 64-70.

13. Аркабаев Н., Алымова З. Разработка Web серверных приложений на базе .Net Core в примере интернет-магазина // Вестник Ошского государственного университета. 2024. №1. P. 142–154. https://doi.org/10.52754/16948610_2024_1_13

*References:*

1. An, H., He, D., Bao, Z., Peng, C., & Liu, Q. (2023). An identity-based dynamic group signature scheme for reputation evaluation systems. *Journal of Systems Architecture*, *139*, 102875. https://doi.org/10.1016/j.sysarc.2023.102875

2. Antonacopoulos, A., & Bridson, D. (2007, September). Performance analysis framework for layout analysis methods. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)* (Vol. 2, pp. 1258-1262). IEEE. https://doi.org/10.1109/ICDAR.2007.4377117

3. Srirama, S. N., & Buyya, R. (2022). Post golden jubilee year of the software journal: New research trends and strengthening advisory editorial team. *Software: Practice and Experience*, *52*(1), 3-4. https://doi.org/10.1002/spe.3055

4. Mohd, T. K., Thompson, J., Carmine, A., & Reuter, G. (2022). Comparative Analysis on Various CSS and JavaScript Frameworks. *J. Softw.*, *17*(6), 282-291. https://doi.org/10.17706/jsw.17.282-291

5. Wellings, A., & Kim, M. (2010, August). Asynchronous event handling and safety critical Java. In *Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems* (pp. 53-62). https://doi.org/10.1002/cpe.1756

6. Kim, J. Y., & Moon, S. M. (2022, July). Disclosure: efficient instrumentation-based web app migration for liquid computing. In *International Conference on Web Engineering* (pp. 132-147). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-031-09917-5_9

7. Heidari, A., Navimipour, N. J., Jamali, M. A. J., & Akbarpour, S. (2023). A hybrid approach for latency and battery lifetime optimization in IoT devices through offloading and CNN learning. *Sustainable Computing: Informatics and Systems*, *39*, 100899. https://doi.org/10.1016/j.suscom.2023.100899

8. Pham, Q. (2025). The evolution of CSS: from CSS2 to Flexbox.

9. Shethiya, A. S. (2025). Scalability and Performance Optimization in Web Application Development. *Integrated Journal of Science and Technology*, *2*(1).

10. Zulfa, M. I., Hartanto, R., & Permanasari, A. E. (2020). Caching strategy for Web application–a systematic literature review. *International journal of web information systems*, *16*(5), 545-569. https://doi.org/10.1108/IJWIS-06-2020-0032

11. Arkabaev, N., Rahimov, E., Abdullaev, A., Padmanaban, H., & Salmanov, V. (2025). Modelling and analysis of optimization algorithms. *Jurnal Ilmiah Ilmu Terapan Universitas Jambi*, *9*(1), 161-177. https://doi.org/10.22437/jiituj.v9i1.38410

12. Choyubekova, A. M., Rashid kyzy, B., & Mamatova, V. T. (2023). Sistemnye osnovy sovremennykh tekhnologii programmnogo obespecheniya. *Vestnik Zhalal-Abadskogo gosudarstvennogo universiteta, 4*(58), 64-70. (in Russian).

13. Arkabaev, N., & Alymova, Z. (2024). Razrabotka Web servernykh prilozhenii na baze .Net Core v primere internet-magazina. *Vestnik Oshskogo gosudarstvennogo universiteta,* (1), 142–154. (in Russian). https://doi.org/10.52754/16948610_2024_1_13

*Поступила в редакцию*
*16.12.2025 г.*

*Принята к публикации*
*27.12.2025 г.*

---

*Ссылка для цитирования:*

Аркабаев Н. К., Насиридинова М. Ч., Айбек уулу Д. Оптимизация веб-приложений на Java и HTML5/CSS Grid // Бюллетень науки и практики. 2026. Т. 12. №2. С. 142-152. https://doi.org/10.33619/2414-2948/123/15

*Cite as (APA):*

Arkabaev, N., Nasiridinova, M., & Aibek uulu, D. (2026). Optimization of Web Applications Using Java and HTML5/CSS Grid. *Bulletin of Science and Practice, 12*(2), 142-152. (in Russian). https://doi.org/10.33619/2414-2948/123/15