

РАЗРАБОТКА МЕТОДА ОПТИМИЗАЦИИ ВЕБ-ПРИЛОЖЕНИЙ С ЦЕЛЬЮ ПОВЫШЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ

Д. Рахмани, старший преподаватель

М.Д. Баранов, студент

Д.А. Кузьмин, студент

Московский технический университет связи и информатики
(Россия, г. Москва)

DOI:10.24412/2500-1000-2025-3-1-237-241

Аннотация. В статье подробно рассматриваются методы оптимизации веб-приложений с использованием как стандартных подходов, так и технологии WebAssembly. Произведено комплексное сравнение производительности WebAssembly и JS, а также выполнено тестирование всех методов с использованием инструментов Lighthouse на двух веб-приложениях. Определено влияние различных методов оптимизации на основные метрики производительности, такие как FCP, LCP и TTI.

Ключевые слова: веб-приложения, Frontend, WebAssembly, JavaScript, Lighthouse, оптимизация.

В современном цифровом мире веб-приложения играют ключевую роль в обеспечении быстрого и удобного доступа к информации и сервисам. Требования к качеству непрерывно растут, поэтому низкая производительность [1] веб-приложения может привести к негативному пользовательскому опыту. По данным Google 53% пользователей отказываются от посещения сайта, если его загрузка занимает более 3 секунд. Основная проблема – оптимизация вычислений в веб-приложениях. JavaScript не всегда эффективен, особенно это касается веб-приложений, которые выполняют сложные алгоритмы и обрабатывают большие объемы данных непосредственно в браузере пользователя.

В данной статье будет представлен комплексный метод оптимизации только frontend-части веб-приложений, содержащий стандартные практики и современную технологию, такую как WebAssembly [2, 3]. Все подходы будут проанализированы на основе метрик производительности, предоставленные инструментом Lighthouse.

Таким образом, цель данной работы – разработать метод оптимизации frontend-части веб-приложений путем объединения классических методов и технологии WebAssembly.

Способы оптимизации:

1. Сжатие изображения. Сжатие представляет собой уменьшение размера исходного изображения до максимального размера,

который возможен для отображения в данный момент. В дополнение к этому, на практике применяется хранение изображения в разном качестве для отображения на разных устройствах.

2. Выбор формата. Основные форматы, используемые на данный момент: WebP, Jpg, Png, Svg.

- WebP: Средняя нагрузка на CPU, низкая – на GPU, средняя скорость рендеринга. Эффективен для современных устройств благодаря высокой степени сжатия.

- JPG: Низкая нагрузка на оба процессора, высокая скорость рендеринга, оптимален для фотографий.

- PNG: Высокая нагрузка на CPU, средняя – на GPU, подходит для использования изображений с прозрачностью.

- SVG: Формат векторной графики, где нагрузка зависит от сложности изображения. Идеален для логотипов и простых иконок.

3. Вырезание неиспользуемой части изображения. Обычно используется для мобильных устройств: из исходного изображения вырезают центр в n пикселей, где n – размер основного контента страницы.

4. Предзагрузка определенного контента. Предзагрузка (preload) позволяет заранее загрузить критически важные ресурсы до начала рендеринга основной страницы. Использование тегов `<link rel="preload">` помогает браузеру правильно распределить приоритеты

загрузки, что ведёт к ускорению отображения ключевого контента.

5. Lazy load. Lazy load позволяет прогружать только тот контент, который необходим пользователю, а не сразу при загрузке страницы. Для оптимизации можно добавить параметр для декодирования изображения. Он говорит браузеру, что можно декодировать изображение асинхронно, не дожидаясь полной загрузки изображения.

6. Использование SVG формата. Для логотипов, иконок и других элементов графики предпочтительно использовать векторные изображения в формате SVG. SVG-файлы обладают малыми размерами и масштабируются без потери качества.

7. Lazy background. В некоторых случаях изображения, установленные в качестве фона через CSS, могут негативно сказываться на производительности. Одним из решений является замена фоновых изображений на элементы `` с последующим применением lazy loading.

8. Использование шрифтов

Подключение шрифтов с внешних ресурсов (например, Google Fonts) может замедлять загрузку страницы. Рекомендуется:

- Загрузить шрифты локально с помощью инструментов вроде Google Fonts Helper;
- Ограничить количество используемых шрифтов до двух и начертания;
- Добавить предварительную загрузку шрифтов через `<link rel="preload">`, чтобы обеспечить быстрое отображение текста.

9. Минификация кода. Минификация CSS и JavaScript подразумевает удаление всех избыточных символов, комментариев и пробелов, что приводит к уменьшению размера файлов

10. Оптимизация SVG файлов. Важно оптимизировать сами SVG-файлы, чтобы уменьшить их размер. Для этого используются специальные сервисы, которые удаляют избыточные данные из файла.

11. Отложенная загрузка

Если JavaScript-файлы не нужны для корректной загрузки страницы, их можно разместить в нижней части тега `<body>` добавив атрибут `defer`. Это позволит отложить их загрузку до тех пор, пока веб-сайт не будет полностью загружен, и они не будут "блоки-

ровать рендеринг", предотвращая загрузку веб-сайта.

Вышеперечисленные методы представляют собой "поверхностную" оптимизацию, необходимую для любого веб-приложения. Несмотря на то что JavaScript является основным языком веб-разработки, он имеет ограничения по производительности:

1. Многопоточность. JavaScript работает в однопоточном режиме, что ограничивает его возможности в многопоточных вычислениях.

2. Управление памятью. JavaScript имеет автоматическую сборку мусора, что может привести к непредсказуемым задержкам в работе приложений.

3. Оптимизация для сложных вычислений. JavaScript не всегда оптимально справляется с задачами, требующими интенсивных вычислений, такими как обработка изображений или сложные математические вычисления.

Данные ограничения могут быть решены с использованием более продвинутых технологий. Одной из них является WebAssembly.

WebAssembly(wasm) – это открытый формат байт-кода, который исполняется современными браузерами почти с нативной скоростью. Он позволяет запускать код, написанный на таких языках, как C, C++, Rust, и других, в браузере, предоставляя возможность переносить сложные вычислительные задачи в веб-среду. Основные характеристики:

- **Бинарный формат.** Компактное представление кода, что сокращает время загрузки и ускоряет компиляцию.

- **Кроссплатформенность.** Wasm работает на всех устройствах и в браузерах, что позволяет создавать универсальные решения.

- **Высокая производительность.** Благодаря почти нативной скорости выполнения, WebAssembly позволяет выполнять вычислительно сложные задачи, существенно превосходя JavaScript в ряде сценариев.

WebAssembly значительно ускорит выполнение таких операций, как обработка изображений, видео, аудио; математические вычисления или работа с 3D-графикой. Данная технология применяется в графических редакторах, эмуляторах, виртуальных машинах, веб-клиентах для финансовых торговых площадок, играх и базах данных.

Сравним JS и WebAssembly с помощью бенчмарка [4], использующий тесты, такие как:

- Сортировка массива методом QuickSort $O(n) = n \log(n)$ на объеме данных 10^4 .
- Числа Фибоначчи $O(n) = 2^n$ до числа не превосходящего 10^4 .
- Фильтра Гаусса $O(n) = n^2$ на фото с разрешением 640 x 400 px.

Время выполнения операций в зависимости от размера входных данных.

Формула вычислительной сложности: $T(n) = C * O(n)$, где:

- $T(n)$ – время выполнения,
 - C – коэффициент, зависящий от платформы,
 - $O(n)$ – сложность алгоритма.
- Сравнение представлено в таблице 1

Таблица 1. Сравнение эффективности JS И WebAssembly

Алгоритм	JavaScript (мс)	WebAssembly (мс)	Ускорение (раз)
Быстрая сортировка	280	85	3.3
Числа Фибоначчи	900	185	4.8
Фильтр Гаусса	36	24	1.4

На основе полученных данных можно сделать вывод, что WebAssembly ускоряет выполнение вычислительных задач до 5 раз.

Для объективной оценки эффективности предложенных методов оптимизации веб-приложений проведено комплексное тестирование с использованием инструмента Lighthouse [5] – открытого программного обеспечения от Google для аудита веб-страниц. Lighthouse позволяет оценивать производительность, доступность, SEO и другие аспекты веб-приложений, предоставляя детальные метрики и рекомендации по улучшению.

Ключевые метрики производительности:

- **Время загрузки страницы (Page Load Time)** – общее время, необходимое для полной загрузки страницы. Эта метрика критически важна для удержания пользователей.

- **Время до первого содержательного отображения (First Contentful Paint, FCP)** –

момент, когда браузер отображает первый визуальный контент. Оптимальное значение FCP – менее 1.8 секунды.

- **Наибольшая отрисовка контента (Largest Contentful Paint, LCP)** – время отображения самого крупного блока контента на странице. Оптимальное значение LCP – менее 2.5 секунды.

- **Время до интерактивности (Time to Interactive, TTI)** – время, через которое страница становится полностью интерактивной и отзывчивой на действия пользователя. Оптимальное значение TTI – менее 3.8 секунды.

Для оценки эффективности стандартных методов оптимизации было создано тестовое веб-приложение, к которому последовательно применялись описанные ранее подходы. Результаты измерений представлены в таблицах 2 и 3.

Таблица 2. Результаты тестирования

Метрика	До оптимизации	После оптимизации
Page Load Time	4.1 с	1.5 с
First Contentful Paint (FCP)	2.6 с	0.8 с
Largest Contentful Paint (LCP)	3.9 с	1.2 с
Time to Interactive (TTI)	4 с	1.5 с

Таблица 3. Ранжирования методов

Методы оптимизации	Размер страницы	FCP	LCP	TTI	Сложность внедрения	Ранг
Сжатие изображений	7	6	6	5	2	5.75
Оптимальный формат изображения	7	6	6	5	2	5.75
Неиспользуемая часть изображения	3	5	4	1	1	3.1
Lazy Loading	0	0	4	5	3	3.75
Предзагрузка контента	0	5	4	5	2	4.3
FOUT	0	5	4	3	3	3.5
Minification (CSS/JS)	3	2	4	4	1	2.8
Оптимизация шрифтов	4	3	3	4	3	3.125
Отложенная загрузка JS	0	2	3	4	4	2.3
WebAssembly	0	2	3	2	6	1.5

Из данной таблицы видно, что по рангу стандартные методы превосходят WebAssembly, так как WebAssembly доказывает свою эффективность в специфических веб-приложениях с большим количеством вычислительных операций.

Рассмотрим пример веб-приложения со сложными вычислениями. Для анализа производительности JavaScript и WebAssembly была реализована транспортная задача – классический алгоритм линейного программирования

для оптимизации распределения ресурсов. Тестирование проводилось на различных размерах входных данных с многократными запусками для исключения случайных погрешностей. Результаты представлены в таблице 4.

Так как производительность такого веб-приложения значительно зависит от вычислений, то влияние стандартных методов минимально на приложение.

Таблица 4. Сравнение времени выполнения транспортной задачи

Размер задачи	JavaScript	WebAssembly	Соотношение
5x5	1370 мс	920 мс	1.49
7x7	5720 мс	3100 мс	1.84
10x10	28120 мс	12340 мс	2.28

Из результатов следует, что WebAssembly демонстрирует значительное преимущество в скорости выполнения вычислительно сложных задач. При этом относительное ускорение возрастает с увеличением размера входных данных

В заключение, стоит отметить, что комбинация стандартных методов и технологий WebAssembly представляет собой мощный инструмент для разработчиков, стремящихся оптимизировать свои веб-приложения.

Проведенное исследование продемонстрировало, что:

- WebAssembly позволяет ускорить выполнение вычислительно-интенсивных операций в 3-5 раз по сравнению с JavaScript.

- Классические методы оптимизации (сжатие изображений, применение lazy loading, минификация кода, оптимизация шрифтов) в совокупности способны улучшить ключевые метрики производительности до 2.5 раз (FCP, LCP, TTI).

- Особенно перспективно объединение WebAssembly с AI и машинным обучением. Эта совокупность может значительно улучшить веб-приложения, позволяя запускать сложные AI-алгоритмы прямо в браузере с минимальной задержкой и без постоянного подключения к серверу.

Библиографический список

1. Google Developers. "Web Performance Best Practices". – [Электронный ресурс]. – Режим доступа: <https://developers.google.com/web/fundamentals/performance>.
2. Веб-приложения будущего: что нужно знать о WebAssembly. – [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/selectel/articles/873662/>.
3. WebAssembly. – [Электронный ресурс]. – Режим доступа: <https://webassembly.org/>.
4. JS vs WASM. – [Электронный ресурс]. – Режим доступа: <https://takahirox.github.io/WebAssembly-benchmark/>.
5. Lighthouse Documentation. Google. – [Электронный ресурс]. – Режим доступа: <https://web.dev/lighthouse/>.

DEVELOPING A WEB APPLICATION OPTIMIZATION METHOD TO IMPROVE PERFORMANCE

J. Rahmani, *Senior Lecturer*

M.D. Baranov, *Student*

D.A. Kuzmin, *Student*

**Moscow Technical University of Communications and Informatics
(Russia, Moscow)**

***Abstract.** This article examines in detail methods for optimizing web applications using both standard approaches and WebAssembly technology. A comprehensive comparison of WebAssembly and JS performance is conducted, and all methods are tested using Lighthouse tools on two web applications. The impact of various optimization methods on key performance metrics such as FCP, LCP, and TTI is determined.*

***Keywords:** Web applications, Frontend, WebAssembly, JavaScript, Lighthouse, optimization.*