ВЫБОР АРХИТЕКТУРЫ ВЕБ-ПРИЛОЖЕНИЯ: ОТ SPA К ГИБРИДНЫМ РЕШЕНИЯМ

Перстнев Алексей

Студент, кафедра «Вычислительные системы и информационная безопасность», Донской государственный технический университет

Газизов Андрей Равильевич

Доцент, кандидат педагогических наук, заведующий кафедрой«Вычислительные системы и информационная безопасность», Донской государственный технический университет

Статья посвящена проблеме выбора архитектуры веб-приложения в условиях растущих требований к производительности и пользовательскому опыту. Рассмотрены подходы Client-Side Rendering (CSR), Server-Side Rendering (SSR) и статическая генерация (SSG), определены их практические ограничения. Особое внимание уделено гибридным моделям рендеринга, реализуемым в современных фреймворках. Проанализированы метрики производительности и SEO-доступности. Сделан вывод о преимуществах гибридного подхода для большинства классов веб-приложений.

Ключевые слова: веб-архитектура, SPA, SSR, SSG, гибридный рендеринг, производительность, SEO, клиентская гидратация, Next.js, фронтенд-разработка.

CHOICE OF WEB APPLICATION ARCHITECTURE: FROM SPA TO HYBRID SOLUTIONS

Perstnev Alexei

Student, Department of «Computer Systems and Information Security», Don State Technical University

Gazizov Andrey Ravilevich

Associate Professor, Candidate of Pedagogical Sciences, Head of the Department of «Computer Systems and Information Security», Don State Technical University

The paper addresses the problem of choosing a web application architecture under increasing performance and user experience requirements. Client-Side Rendering (CSR), Server-Side Rendering (SSR), and Static Site Generation (SSG) approaches are reviewed, and their practical limitations are identified. Special attention is given to hybrid rendering models implemented in modern frameworks. Performance and SEO accessibility metrics are analyzed. The conclusion establishes the advantages of the hybrid approach for most classes of web applications.

Key words: web architecture, SPA, SSR, SSG, hybrid rendering, performance, SEO, client-side hydration, Next.js, front-end development.

Проектирование современного веб-приложения начинается с вопроса, который кажется простым только на первый взгляд: где рендерить страницу — на сервере или в браузере? Ответ на этот вопрос определяет не только техническую архитектуру, но и пользовательский опыт, затраты на инфраструктуру и даже скорость вывода продукта на рынок.

За последние годы подходы к рендерингу претерпели значительную эволюцию. Период с 2015 по 2018 год ознаменовался бумом одностраничных приложений (SPA) с клиентским рендерингом (CSR) — разработчики стремились создать «нативные» по ощущениям веб-интерфейсы. Затем, с 2019 по 2021 год, набрала популярность статическая генерация (SSG), показавшая

свою эффективность для блогов и маркетинговых сайтов. Наконец, в 2022–2024 годах гибридный рендеринг стал доминирующей парадигмой, позволяющей смешивать статические, динамические и инкрементальные стратегии в рамках одного приложения.

Однако на практике многие команды продолжают делать этот выбор интуитивно — берут то, что работало на прошлом проекте, или то, о чём чаще говорят на профильных конференциях. Систематическое сравнение остаётся редкостью. Между тем, как отмечают исследователи, классический SPA, несмотря на преимущества в интерактивности после загрузки, проигрывает в критически важных для успеха приложений областях — первоначальной производительности и SEO. Это создаёт фундаментальные ограничения для его использования в публичных, контентно-ориентированных проектах.

Клиентский рендеринг: удобство разработки ценой первой загрузки

Одностраничное приложение загружает минимальный HTML-файл, после чего вся логика рендеринга выполняется в браузере. Скрипт загружает бандл, монтирует дерево компонентов, выполняет запросы к API и отрисовывает интерфейс. Переходы между страницами происходят мгновенно, без перезагрузки — пользователь получает опыт, близкий к нативному приложению.

Основная проблема SPA — первый визит. Браузер получает почти пустую разметку, загружает весь бандл JavaScript, парсит и исполняет его, и только после этого показывает пользователю что-либо осмысленное. Исследования показывают, что задержка в 100 миллисекунд снижает конверсию на 7%, а три секунды ожидания отсеивают более половины мобильных пользователей. Для внутренних корпоративных систем конверсия не критична, но накопленное раздражение сотрудников, вынужденных ежедневно ждать загрузки, снижает продуктивность и создаёт негативное восприятие продукта.

Вторая проблема — поисковая индексация. Поисковые роботы, получая пустой HTML, не видят контент и не индексируют данные. Это не баг

архитектуры, а её особенность. Для проектов, в которых даже часть маршрутов оказывается публичной — страница входа, справочник номенклатуры, публичные отчёты, — это становится серьёзным ограничением.

Третья, менее очевидная проблема — рост потребления памяти. По мере развития приложения бандл увеличивается. Браузер удерживает весь код в памяти. Клиентская гидратация требует ресурсов, которые конкурируют с другими вкладками, фоновыми задачами и системными процессами. Как отмечается в исследовании, избыточный клиентский код напрямую коррелирует с падением производительности на устаревшем оборудовании.

Серверный рендеринг: скорость отдачи ценой серверной нагрузки

При серверном рендеринге (SSR) сервер формирует готовый HTML по каждому запросу и отдаёт его клиенту. Браузер видит контент сразу, до загрузки JavaScript-скриптов. Это даёт существенное преимущество по метрике First Contentful Paint — в среднем на 40–60% лучше, чем при чистом SPA на аналогичном контенте. Кроме того, SSR обеспечивает превосходную SEO-доступность: поисковые роботы получают полностью готовую разметку.

Однако у SSR есть обратная сторона. Серверный рендеринг переносит вычислительную нагрузку с клиента на сервер. Каждый запрос требует полного цикла рендеринга, что увеличивает нагрузку на серверную инфраструктуру и может оказаться дороже, чем ожидалось на старте проекта.

Кроме того, после получения HTML браузер всё равно загружает JavaScript для обеспечения интерактивности — этот процесс называется гидратацией. Страница выглядит готовой, но кнопки и элементы управления не работают до завершения гидратации. Это создаёт проблему рассинхронизации между видимым контентом и интерактивностью: пользователь видит интерфейс, пытается с ним взаимодействовать, но ничего не происходит. Исследования показывают, что такой опыт убивает доверие пользователей быстрее, чем медленная загрузка.

Гибридная модель: лучшее из двух миров

Современные фреймворки, такие как Next.js и Nuxt, предлагают гибридные подходы, позволяющие выбирать стратегию рендеринга на уровне отдельных маршрутов. Это принципиально меняет логику выбора архитектуры.

В рамках гибридной модели:

- **SSG (Static Site Generation)** — страницы генерируются на этапе сборки и отдаются с CDN. Идеально для контента, который меняется редко.
- **SSR (Server-Side Rendering)** — страницы рендерятся на сервере по запросу. Подходит для динамического контента, требующего актуальности.
- **ISR (Incremental Static Regeneration)** — страницы генерируются статически, но обновляются в фоне по расписанию или по событию без полной пересборки проекта.
- **CSR (Client-Side Rendering)** — рендеринг на клиенте для высокоинтерактивных разделов после авторизации.

По данным бенчмаркинга Next.js, статическая генерация доставляет контент на 68% быстрее SSR по метрике Time to First Byte и потребляет на 73% меньше процессорных ресурсов сервера. При этом ISR обеспечивает баланс между актуальностью данных и производительностью.

Особого внимания заслуживает Streaming SSR, появившийся в React 18 с поддержкой Suspense. Этот подход позволяет отправлять HTML частями до завершения полной гидратации, что улучшает Time-to-First-Byte и показатели Core Web Vitals. По данным исследований, SSR в сочетании со стримингом снижает показатель отказов (bounce rate) до 20% по сравнению с традиционным SSR.

Edge Rendering добавляет ещё один уровень оптимизации: развёртывание SSR на границах CDN (Vercel Edge, Cloudflare Workers) позволяет доставлять страницы с задержкой менее 50 мс TTFB для пользователей по всему миру, устраняя узкие места центральных серверов.

Почему гибридный подход выигрывает организационно

Ключевое преимущество гибридного подхода лежит не только в технической плоскости. Он выигрывает организационно.

Чистый SPA с отдельным API-сервером означает две кодовые базы: синхронизация типов данных, два деплоя, два набора конфигураций. Классический SSR требует ручной настройки кеширования, управления состоянием между запросами и отдельного серверного процесса.

Современный фреймворк с гибридным рендерингом даёт одну кодовую базу, один деплой, встроенную маршрутизацию, оптимизацию изображений и разбиение кода без ручной настройки. TypeScript в связке с такими фреймворками обеспечивает типобезопасность между серверным и клиентским кодом в рамках одного проекта, позволяя выявлять ошибки на этапе компиляции.

В 2025 году SSR перестал быть просто оптимизацией — он становится базовой архитектурой для самых быстрых и масштабируемых веб-приложений. Сочетание SSR с React Server Components позволяет сократить JavaScript-бандлы на 30–60%, ускорить гидратацию и улучшить пользовательский опыт на маломощных устройствах.

Разработка веб-приложения требует выбора архитектуры, которая будет работать не только в момент запуска, но и через год, когда нагрузка вырастет, а требования к производительности ужесточатся. Гибридный рендеринг даёт эту уверенность. SPA в чистом виде или классический SSR в отрыве от контекста проекта такой уверенности не дают.

Исследования показывают, что SSR и SSG становятся предпочтительными для проектов, где первостепенны быстрая первая загрузка и поисковое продвижение, в то время как SPA может оставаться выбором для сложных внутренних интерфейсов с высокими требованиями к интерактивности. Гибридный подход позволяет сочетать преимущества каждой стратегии, выбирая оптимальный вариант для каждого маршрута.

Ключевой вывод: выбор архитектуры рендеринга — это не техническое решение в вакууме. Это компромисс между скоростью первой загрузки, серверной нагрузкой, SEO-доступностью и сложностью разработки. Гибридная

модель предлагает наиболее сбалансированное решение для широкого спектра веб-приложений.

Список использованных источников

1. Романов М.А., Бадалин А.О., Рыбалка Д.В., Баженов А.Э. Анализ и сравнительное исследование архитектурных подходов к разработке одностраничных приложений (SPA, SSR, SSG) // Программные системы и вычислительные методы. – 2025. – № 4. – С. 108-117. DOI: 10.7256/2454-0714.2025.4.77190

2. Ramakrishnan G. Scaling Modern Frontend Development: Strategies and Methodologies // International Journal of Computer Applications. – 2025. – Vol. 186, No. 65. – P. 1-7.

3. Dudak A. Frontend application architecture // International Journal of Professional Science. – 2024. – № 11-2. – С. 32-39.

4. Patel R. SSR and SSG Performance Benchmarking of Next.js for Cloud-Deployed Applications. ResearchGate, 2025.

5. Kaushik K. How React.js SSR Redefines Performance, SEO, and Scalability in 2025. LinkedIn, 2025.