Программные системы и вычислительные методы
Правильная ссылка на статью:
Ратушняк Е.А. Исследование производительности современных клиентских веб-фреймворков // Программные системы и вычислительные методы. 2025. № 2. DOI: 10.7256/2454-0714.2025.2.74392 EDN: OYAYXV URL: https://nbpublish.com/library_read_article.php?id=74392

Исследование производительности современных клиентских веб-фреймворков

Ратушняк Евгений Алексеевич

ORCID: 0009-0006-3609-4194

студент; факультет программной инженерии и компьютерной техники; Национальный исследовательский университет ИТМО

197101, Россия, г. Санкт-Петербург, Петроградский р-н, Кронверкский пр-кт, д. 49

evgrat123@mail.ru

Статья из рубрики "Математическое и программное обеспечение новых информационных технологий"

DOI:

10.7256/2454-0714.2025.2.74392

EDN:

OYAYXV

Дата направления статьи в редакцию:

10-05-2025

Дата публикации:

26-05-2025

Аннотация: Предметом исследования является производительность рендеринга трёх современных фреймворков — React, Angular и Svelte — в типовых сценариях построения и обновления пользовательского интерфейса в веб приложениях. Объектом исследования являются сами фреймворки как комплексы технологических решений, включающие механизмы обнаружения изменений, виртуальные или компилируемые DOM структуры и сопутствующие оптимизации. Автор подробно рассматривает такие аспекты темы, как первичный и повторный рендеринг, операции обновления и удаления элементов, работа в линейных и глубоко вложенных структурах данных. Особое внимание уделяется практической значимости выбора фреймворка для коммерческих продуктов, где разница в производительности непосредственно влияет на конверсию, опыт пользователя и финансовую эффективность проекта. Описываются ключевые

внутренние механизмы — виртуальный DOM React, детектор Angular и компилируемый код Svelte, — которые определяют их поведение в разных нагрузочных сценариях. Методология основана на автоматизированном бенчмарке: унифицированный набор тестовых сценариев выполняется клиентскими приложениями на React, Angular и Svelte, эталонном JavaScript решении и сервере оркестраторе Express JS; время операций фиксируется через performance.now() в Chrome 126, критерий производительности – время до первой перерисовки. Новизна исследования заключается в комплексном лабораторном сопоставлении трёх фреймворков по четырём критически важным сценариям (первичный рендеринг, повторный рендеринг, обновление и удаление элементов) с учётом двух типов структур данных и привязкой к актуальным версиям 2025 года. Основными выводами проведённого исследования являются следующие: Svelte обеспечивает наибольшую производительность и лидирует при глубокой иерархии благодаря компиляции DOM операций; React показывает лучшие результаты при повторном обновлении длинных списков, используя оптимизированный алгоритм обнаружения изменений и ключи элементов; Angular гарантирует предсказуемость и архитектурную целостность, но увеличивает время перерисовки примерно на 60 % из-за детектора изменений. Универсального лидера не существует; рациональный выбор должен опираться на аналитический профиль операций конкретного приложения, что и подтверждают результаты представленного эксперимента.

Ключевые слова:

рендеринг, JavaScript фреймворки, React, Angular, Svelte, Core Web Vitals, виртуальный DOM, инкрементый DOM, производительность, веб-интерфейсы

Введение

Бурное развитие веб технологий преобразует ожидания пользователей: контент должен появляться практически мгновенно, а интерфейс – реагировать без ощутимых задержек. По оценкам Google, увеличение *Largest Contentful Paint* (LCP) всего на 100 мс приводит к снижению коэффициента конверсии интернет магазина на 1,3%. Следовательно, вопрос выбора фреймворка выходит за рамки вкусовых предпочтений и становится фактором финансовой эффективности продукта. Статистика показывает, что React, Angular и Svelte формируют самый «живой» круг разработчиков и наиболее высокую удовлетворённость [2]. В научном дискурсе приоритет отдается фреймворкам с активной экосистемой – React, Angular и, с недавнего времени, Svelte. Также эти фреймворки имеют разный подходы к обнаружению изменений: React – Virutal DOM [3], Angular – Incremental DOM [4], Svelte - переносит вычисления из времени выполнения в этап компиляции [5]. Использование фреймворков и библиотек значительно ускоряет разработку приложений и уменьшает их стоимость на 30% [13], а также производительность веб-фреймворков остается одним из важных параметров [14]. Все выше перечисленное приводит к необходимости сравнительного анализа фреймворков, а также обосновывает актуальность работы.

Целью статьи является эмпирическое сравнение производительности рендеринга React, Angular и Svelte в типовых сценариях построения и обновления пользовательского интерфейса. Для достижения цели решаются задача разработки унифицированного набора тестовых сценариев, автоматизация сбора и анализ экспериментальных данных для выявления недостатков и преимуществ фреймворков. Предметом исследования

является производительность рендеринга трёх фреймворков — React, Angular и Svelte — в типовых сценариях построения и обновления пользовательского интерфейса в веб-приложениях. Научная новизна исследования заключается в разработке и применении автоматизированной методики сравнительного анализа производительности фреймворков React, Angular и Svelte в различных эксплуатационных сценариях веб-приложений, что позволяет выявить их преимущества и недостатки.

Разработка тестового ПО

В рамках экспериментального исследования создаётся клиентское одностраничное [8] вебприложение, визуализирующее иерархическую выборку данных в табличной форме. Пример реализации строки таблицы представлен на рисунке 1. Каждая логическая запись отображается единообразным строковым шаблоном: контейнер строки таблицы разделяется на четыре ячейки, последовательно выводящие идентификатор элемента, его заголовок и текстовое описание, а также набор кнопок с управляющими действиями. Кнопки добавить и удалить являются декоративными, а для обновлений данных таблицы используется программные методы, а именно – обновление внутреннего представления данных, для тестирования производительности фреймворков. Спроектированный ряд, будучи атомарным визуальным модулем, упорядочивает потоки событий, изолирует рендеринг и служит универсальной единицей измерения производительности: фиксированные стили и одинаковый состав DOMузлов обеспечивают сопоставимость метрик времени построения, обновления и удаления для различных фреймворков. Также эксперимент имеет две структуры данных: бинарная – эмулирует реальные сайты с высокой вложенностью, линейная – эмулирует большие таблицы. Такая организация эксперимента даёт возможность строго регистрировать латентность, объём операций с DOM при масштабировании количества записей, что, в свою очередь, позволяет объективно оценить эффективность применяемых во фреймворках оптимизационных стратегий.

Рисунок 1 – реализация строки тестовой таблицы

В браузерной модели событийный цикл является единственным надёжным способом зафиксировать момент, когда первый визуальный кадр уже отрисован, является последовательное использование requestAnimationFrame и затем задачитаймера с нулевой задержкой. Следовательно, метка, поставленная внутри такого вызова, фиксирует время до появления кадра, обходя весь критический путь рендеринга [1]. Если внутри обработчика запроса кадра анимации поместить вызов таймаута с задержкой 0, создаётся отдельная задача. Она будет обработана в следующей итерации, то есть уже после того, как браузер осуществил отрисовку завершившегося кадра. Тем самым интервал между метками, снятыми непосредственно после изменения состояния и в начале таймерзадачи, охватывает полный путь сигнала — работу фреймворка, генерацию патчопераций, вычисление стилей, вычисление разметки, отрисовка — и

потому служит корректным измерением времени «данные → первый кадр» для любых современных JavaScript-фреймворков. Это и есть критерий наибольшей перерисовки LCP [7].

Экспериментальная установка состоит из многокомпонентной архитектуры (представленной на рисунке 2), включающей клиентский слой на каждом из трёх фреймворков и эталонное приложение на «чистом» JavaScript, сервер на базе Express JS и модуль автоматизированного сбора статистики. Каждое клиентское приложение запрашивает входные данные по пути /testcase/next, выполняет требуемую операцию и передаёт измеренное время выполнения обратно на сервер через /testcase/benchmark. Такой цикл повторяется для всех сценариев без участия человека, что обеспечивает полную воспроизводимость и исключает влияния сторонних задержек, связанных с ручным управлением браузером [6]. На сервере ведутся счётчики прогресса, позволяющие последовательно пройти весь объём тестов; по завершении генерируются файлы CSV и адаптивный HTML-отчёт, где результаты собираются в сводные таблицы с автоматически рассчитанными средними значениями и стандартными отклонениями. Автоматический «водитель» браузера, реализованный на Puppeteer, запускает браузер в интерактивном режиме, дожидается момента, когда выставиться флаг завершения эксперимента, и при необходимости повторяет прогон при возникновении исключений, тем самым гарантируя стабильность даже при случайных сбоях сети или браузера.

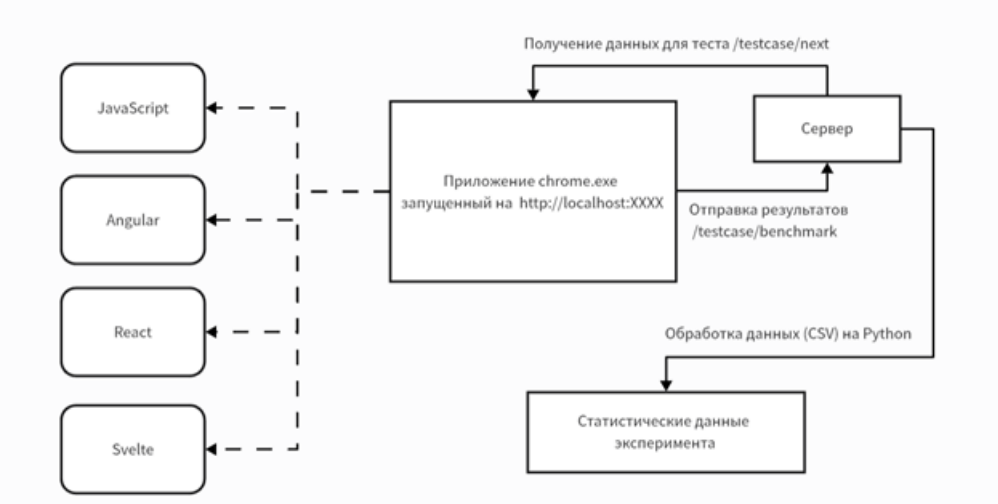

Рисунок 2 – архитектура тестового окружения

Исходный код экспериментального ПО выложен на GitHub и доступен по ссылке https://github.com/TimeToStop/benchmark.

Реализация тестового приложения с использованием фреймворков

В цикле HTML разметки используется функция определяющая уникальный идентификатор элемента; оно заставляет движок обнаружения изменений сравнивать элементы по возвращаемому ключу, поэтому при изменении порядка строк Angular переиспользует существующие компонентные экземпляры и обновляет лишь изменённые ячейки, а не всю коллекцию. Такая комбинация сокращает как количество пересчётов шаблонов, так и частоту «дорогих» операций перерасчёта компоновки. Код представлен на рисунке 3.

Для измерения производительности начальная метка ставится в момент программного изменения входного свойства компонента с немедленным вызовом обнаружения изменений, который принудительно проводит обнаружение только в текущем поддереве.

```
<div *ngIf="data">
  <app-row [data]="data"></app-row>
  <div>
    <app-child
      *ngFor="let child of data.children; trackBy: id"
      [data]="child"
    >
    </app-child>
  </div>
</div>
```

Рисунок 3 – реализация узлового компонента на Angular

Компоненты, представленные на рисунке 4 заданы как чистые функции, поэтому при каждом обновлении React сравнивает только результат их вызова, а не состояние экземпляров классов. Главная оптимизация проявляется в атрибуте ключа при обходе потомков: ключи дают алгоритму согласования устойчивую идентичность элементов, что позволяет ограничить перерасчёт теми узлами, для которых действительно изменились данные. Таким образом, при добавлении или удалении строк React выполняет точечные операции вместо полного пересоздания поддерева. Моментом фиксации начала отрисовки является вызов setState, который помещает обновление в очередь и запускает фазу согласования виртуального дерева.

```
function Child({ data }: { data: INode }) {
  return (
    <div>
      <Row data={data}></Row>
      <div>
        {data.children.map(child => <Child key={child.id} data={child}></Child>)}
      </div>
    </div>
  );
}
```

Рисунок 4 – реализация узлового компонента на React

Компонент объявляется один раз, но разворачивается компилятором в императивный цикл, который на этапе сборки генерирует минимальный набор вызовов вставки и удаления элементов — виртуальный DOM вовсе не участвует[19]. Поэтому изменение массива потомков приводит только к тем манипуляциям с DOM, которые действительно нужны, без промежуточного вычисления изменений [20]. Рекурсивное использование компонента сохраняет единый шаблон и позволяет компилятору повторно применять сгенерированный код на каждом уровне дерева, что минимизирует размер кода и накладные расходы во время исполнения. Начало измерения скорости отрисовки является обращение к tick(): эта функция добавить вызов функции, который выполняется, когда компилятор уже обновил реальные DOMузлы. Код представлен на рисунке 5.

```
<div>
    <Row {data}></Row>
    <div>
        {#each data?.children as child}
            <svelte:component this={self} data={child} />
        {/each}
    </div>
</div>
```

Рисунок 5 – реализация узлового компонента на Svelte

Анализ экспериментальных данных

Аппаратная платформа включает процессор Intel Core i7 8750H, 16 гигабайт оперативной памяти и 64 битную операционную систему; браузер - Google Chrome версии 126.0.6478.116. Фреймворки используются в актуальных стабильных релизах: Angular 16.2.12, React 18.3.1, Svelte 4.2.18. Такой подбор гарантирует актуальность выводов для практики 2025 года и снимает вопросы совместимости с современными браузерными API. Все измерения выполняются функцией performance.now(), чья точность порядка пяти сотых миллисекунды считается достаточной для фиксации производительности.

Первичный рендеринг представляет собой формирование внутреннего представления компонентов, построение DOM и выполнение первой верстки. В линейной структуре тестовая таблица из N тысяч строк создаёт равномерную нагрузку по всей ширине окна; в бинарной структуре, напротив, рекурсивное ветвление формирует глубокое дерево с большим числом перекрёстных зависимостей стилей. В этих условиях эталонное приложение на чистом JavaScript закономерно фиксирует минимальное время из за отсутствия абстракций, однако React и Svelte оказываются близки к нему в бинарном случае: разница не превышает нескольких процентов благодаря тому, что виртуальный DOM React и компилируемые вызовы Svelte генерируют относительно компактные последовательности операций, что согласуется с выводами Makitalo et al. [10]. Angular демонстрирует увеличение LCP примерно на шестьдесят процентов вне зависимости от структуры, что согласуется с его более тяжёлым запуском времени выполнения и стратегией отслеживания изменений [9, 12]. При переходе к линейной таблице Svelte показывает просадку на двадцать процентов по сравнению с JavaScript, что объясняется увеличением объёма кода, необходимого для описания каждой строки, тогда как React сохраняет паритет благодаря оптимизированному преобразованию массивов в набор элементов. Результаты эксперимента при сценарии первичного рендеринга представлены на рисунках 6 и 7.

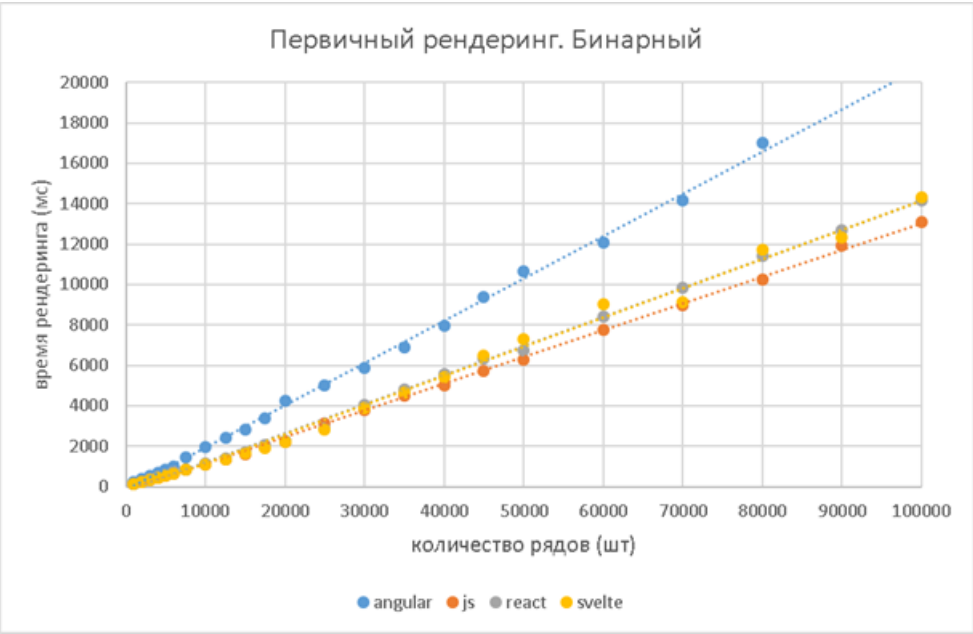

Рисунок 6 – результаты тестирования первичного рендеринга в бинарной структуре данных

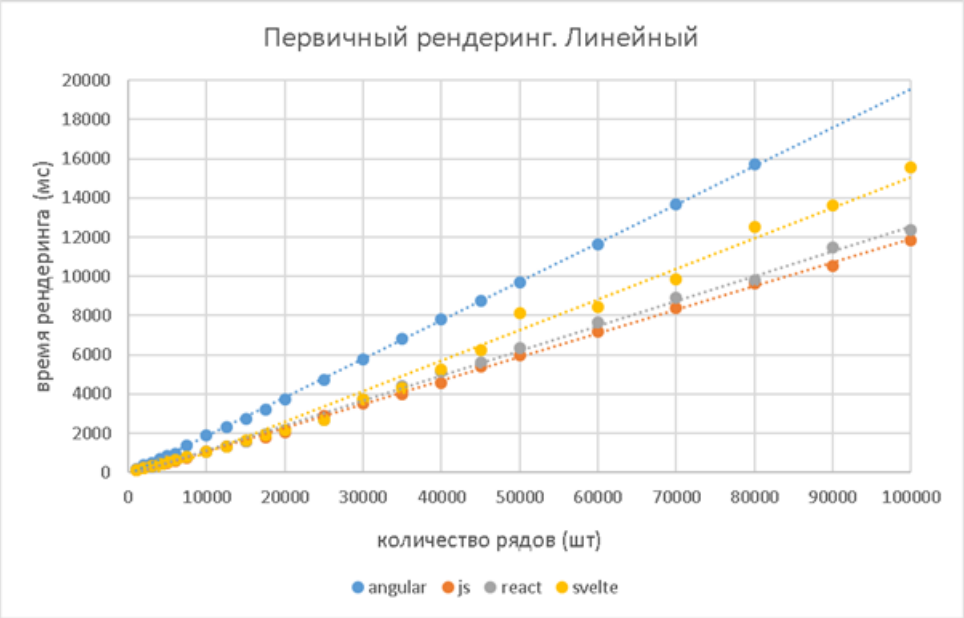

Рисунок 7 – результаты тестирования первичного рендеринга в линейной структуре данных

Повторный рендеринг, то есть полная перерисовка той же структуры данных, иллюстрирует, как фреймворки управляют уже построенным деревом. Здесь критически важны алгоритмы сравнения старого и нового состояния. В бинарной иерархии React оказывается самым медленным: вычисление изменений по глубокой структуре приводит к росту времени почти в полтора раза относительно Angular и вдвое относительно Svelte.

Последний выигрывает за счёт прямых изменений конкретных узлов, поскольку компилятор заранее знает, какой участок DOM нужно затронуть. В линейной таблице картина меняется: React, использующий ключи элементов для быстрого обнаружения перемещений, выходит в явные лидеры, опережая Angular на треть, а Svelte – на пятнадцать процентов. Такое поведение подтверждает тезис о том, что эффективность виртуального DOM растёт вместе с долей одинаковых повторяющихся поддеревьев, тогда как при сложных вложенных структурах издержки сопоставления превышают выигрыш от выборочного обновления. Результаты эксперимента при сценарии повторного ренедринга представлены на рисунках 8 и 9.

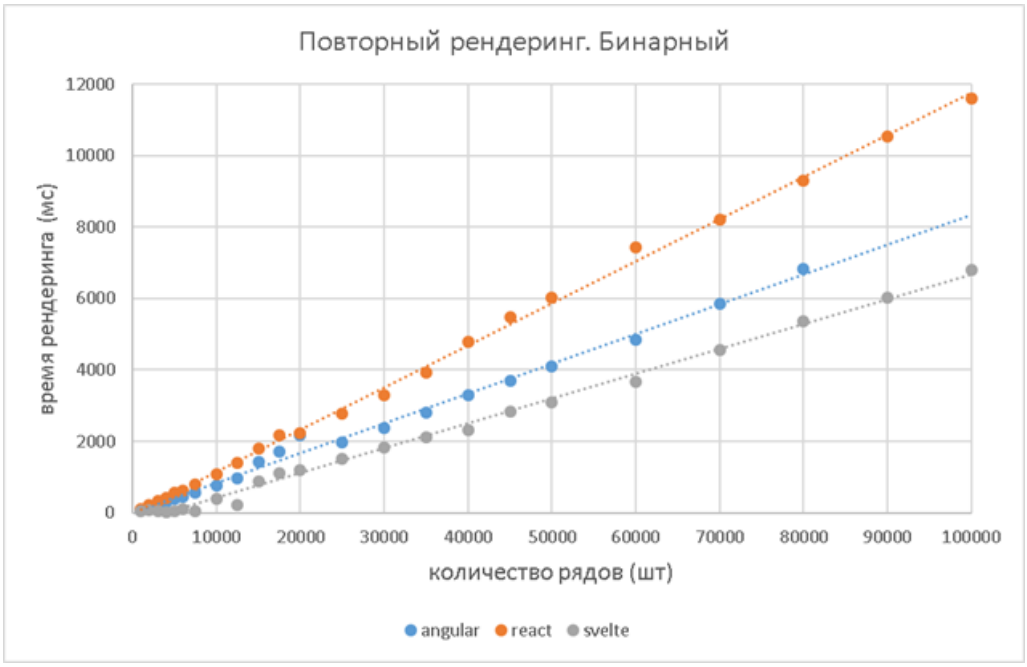

Рисунок 8 – результаты тестирования повторного рендеринга в бинарной структуре данных

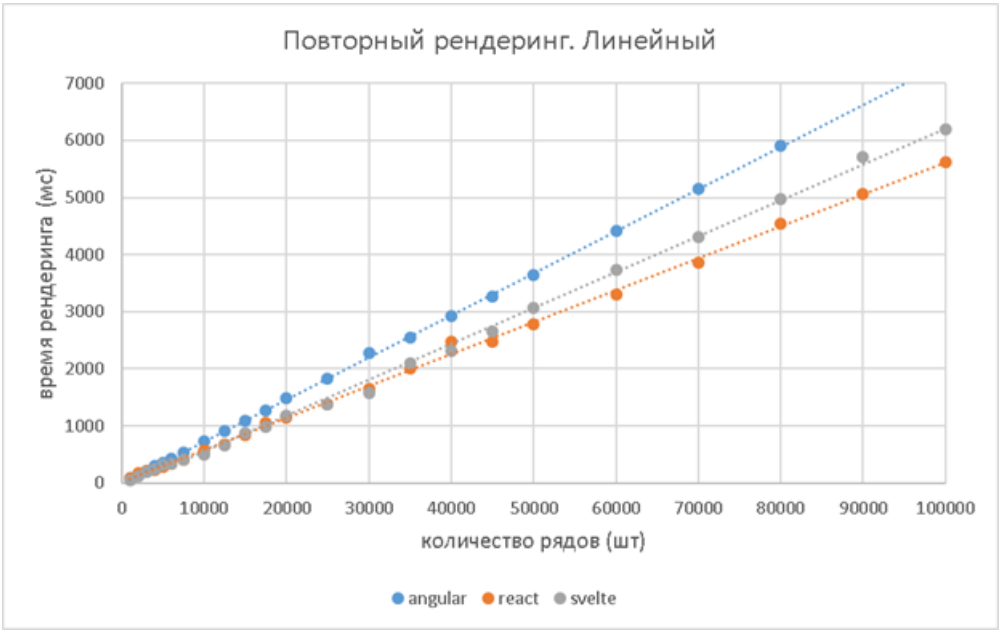

Рисунок 9 – результаты тестирования повторного рендеринга в линейной структуре данных

Операция обновления элементов моделирует небольшие изменения во время работы приложения. Производительность здесь определяется сочетанием скорости обновления узла, количества вызываемых перерисовок и тяжести синхронизации стилей. Полученные результаты демонстрируют явное преимущество Svelte: в глубоком бинарном дереве компилируемый код адресуется непосредственно к целевому узлу, изменяя лишь затронутые атрибуты и сводя задержку к минимуму. В линейной таблице React приближается к показателям Svelte благодаря линейному алгоритму обнаружения изменений и переиспользованию существующих элементов, однако всё-таки уступает в бинарном случае. Angular остаётся предсказуемо медленнее обоих конкурентов: детектор изменений проходит по всей иерархии компонентов независимо от масштаба изменения, что добавляет заметную задержку даже при модификации одной строки, что

согласуется с выводами в статье [11]. Результаты эксперимента при сценарии обновления ряда представлены на рисунках 10 и 11.

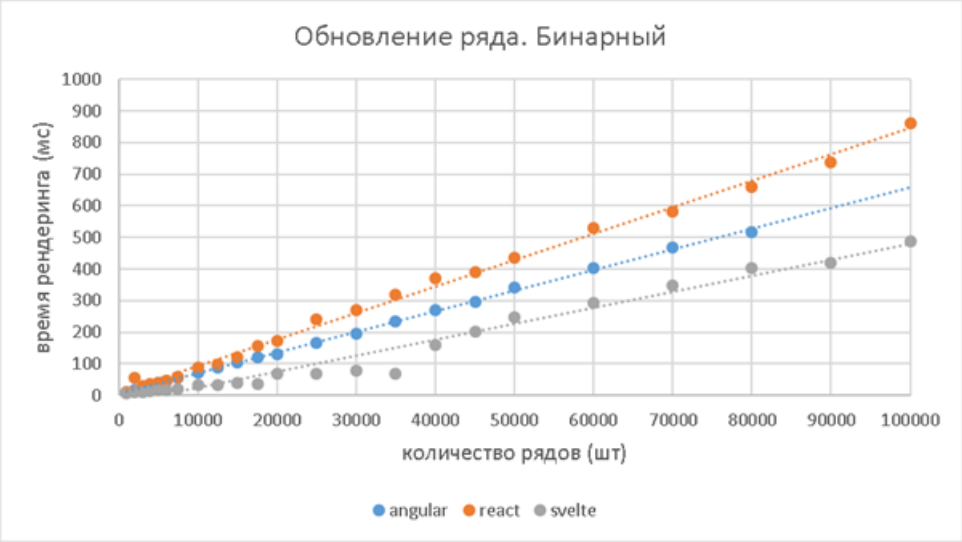

Рисунок 10 – результаты тестирования обновления ряда в бинарной структуре данных

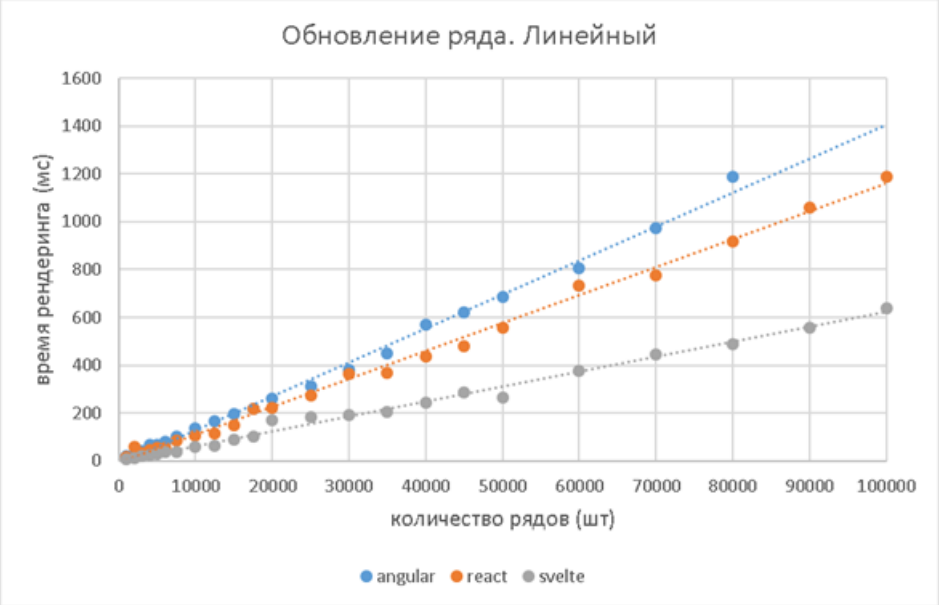

Рисунок 11 – результаты тестирования обновления ряда в линейной структуре данных

Удаление элементов раскрывает обратную сторону оптимизаций. В бинарной структуре Svelte демонстрирует минимальное время, так как точечно удаляет ветвь и немедленно высвобождает связанные обработчики событий. React и Angular, напротив, вынуждены пересчитывать влияние удаления на соседние узлы, что удлиняет операцию приблизительно на пятьдесят процентов. В линейной таблице React выходит вперёд: наличие ключей упрощает поиск удаляемого узла и сокращает обход коллекции, тогда как Svelte, не располагая встроенной виртуализацией больших списков [18], тратит время на обновление индексов, отчего отстаёт от React в среднем на сто сорок процентов. Этот результат подчёркивает необходимость сторонних библиотек виртуального прокручивания в проектах на Svelte, где наблюдается массивное удаление или добавление строк. Результаты эксперимента при сценарии удаления ряда представлены на рисунках 12 и 13.

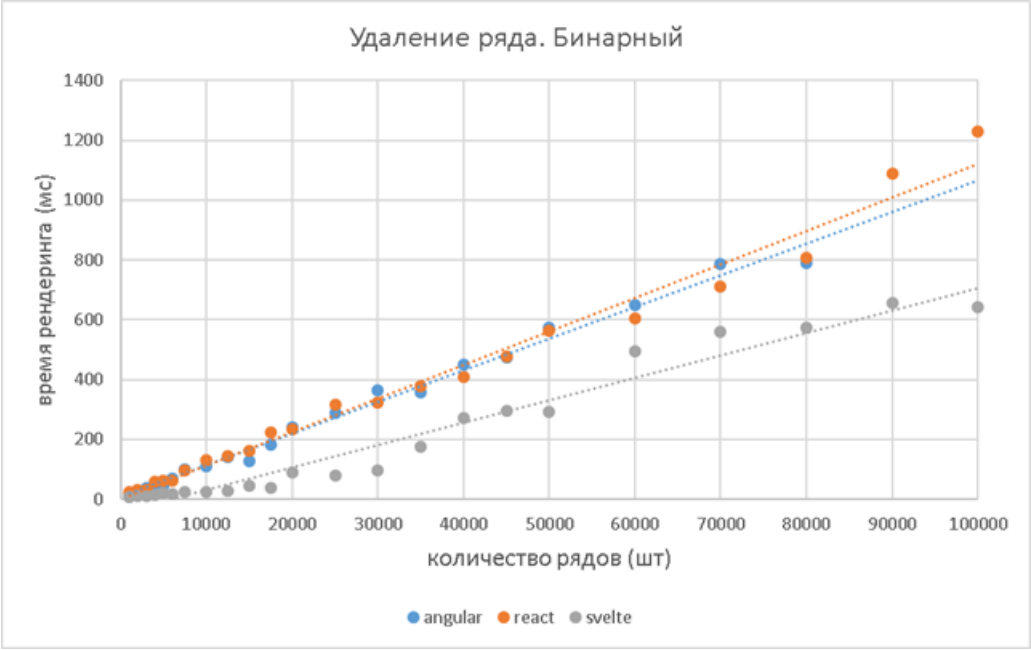

Рисунок 12 – результаты тестирования удаления ряда в бинарной структуре данных

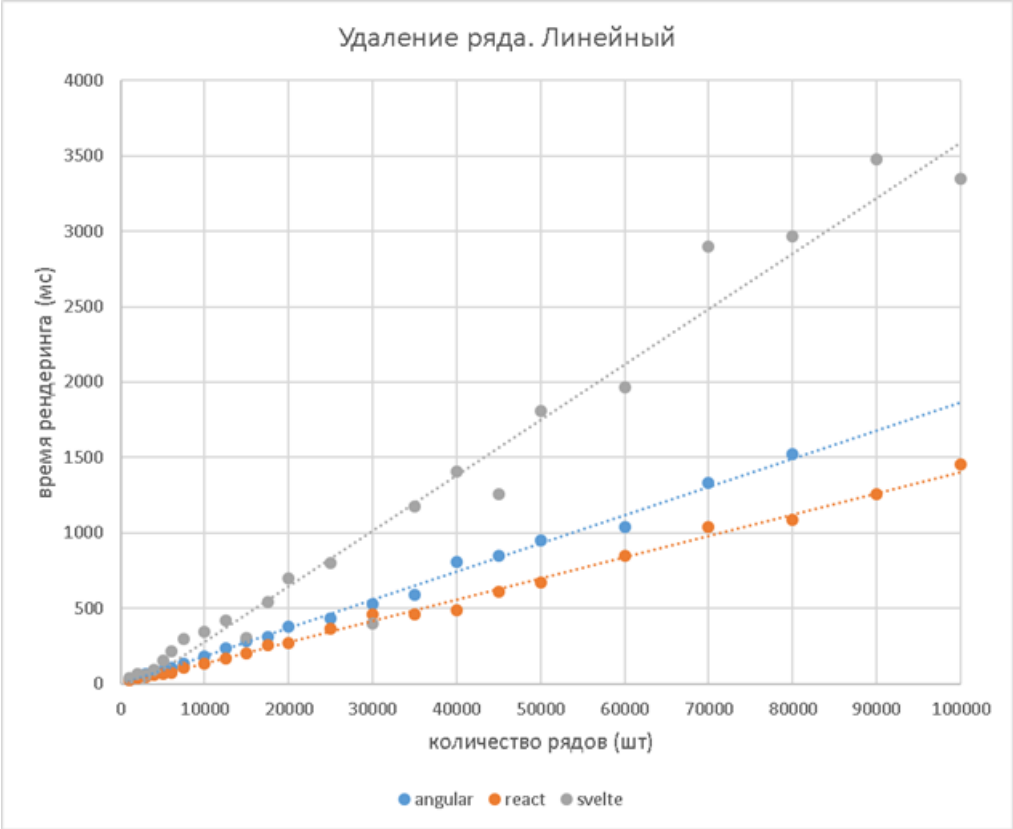

Рисунок 13 – результаты тестирования удаления ряда в линейной структуре данных

Качественная интерпретация выявленных закономерностей опирается на особенности внутренних механизмов. В React критическим фактором выступает стоимость вычисления различий в виртуальном DOM. При плоской коллекции уникальные ключи позволяют свести поиск изменений к линейному проходу; при глубокой иерархии, напротив, число сравнений растёт экспоненциально с глубиной, что делает алгоритм узким местом. Angular применяет детектор изменений, который гарантированно обходит все компоненты при любом событии, обеспечивая стабильность и предсказуемость, но взамен грузит процессор ненужными повторными вычислениями. Svelte избегает анализа во время выполнения: компилятор генерирует функции обновления, физически меняющие только

те свойства, которые могли измениться. Такой подход блестяще работает, пока объём кода остаётся умеренным; при длинных списках количество сгенерированных вызовов становится само по себе значительным, что объясняет потерю скорости при линейной таблице. Разброс данных и выводы о стабильности React и Svelte согласуются с выводами в статье [16].

Результаты эксперимента представлены в таблице 1.

Таблица 1 – результаты тестирования современных фреймворков

Сценарий	Структура данных	JavaScript	React	Angular	Svelte
Первичный рендеринг	Бинарная	Базовый ориентир	≈ то же, что JS	+ 60 % к JS	≈ то же, что JS
	Линейная	Базовый ориентир	≈ то же, что JS	+ 60 % к JS	+ 20 % к JS
Повторный рендеринг	Бинарная	—	+ 80 % к Svelte	+ 40 % к Svelte	Самый быстрый
	Линейная	—	Самый быстрый	+ 30 % к React	+ 15 % к React
Обновление ряда	Бинарная	—	+ 80 % к Svelte	+ 40 % к Svelte	Самый быстрый
	Линейная	—	+ 80 % к Svelte	+ 140 % к Svelte	Самый быстрый
Удаление ряда	Бинарная	—	+ 50 % к Svelte	+ 50 % к Svelte	Самый быстрый
	Линейная	—	Самый быстрый	+ 30 % к React	+ 140 % к React

Практическое применение результатов сводится к формулировке рекомендаций. Для систем, в которых доминируют длинные, однородные коллекции записей с частыми операциями вставки и удаления, целесообразно выбирать React, при этом обязательно включать мемоизацию компонентов и использовать виртуализацию списков, чтобы удерживать алгоритм определения различий в пределах линейного времени. В корпоративных решениях со сложной иерархической структурой, требующих строгой типизации и унифицированных паттернов, Angular остаётся удобным инструментом, но разработчикам следует неукоснительно использовать стратегию OnPush и функцию trackBy, чтобы снизить количество лишних проходов по дереву. Для проектов, ориентированных на мгновенную первую отрисовку и глубокие вложенные интерфейсы, оптимальным оказывается Svelte, однако эффективность сохраняется лишь при одновременном сокращении размера списков через виртуальное прокручивание и разбиение на страницы.

Заключение

Исследование демонстрирует, что ни один из популярных фреймворков не обладает универсальным превосходством. Svelte минимизирует время первой отрисовки и выигрывает в бинарных изменениях за счёт компиляции компонентов, React лидирует в повторном обновлении длинных списков благодаря оптимизации алгоритма обнаружения различий, Angular обеспечивает архитектурную целостность ценой более высокого LCP. Перспективы дальнейших исследований связаны с включением серверного рендеринга SvelteKit и React Server Components, а также анализом энергопотребления на мобильных

устройствах.

Рекомендации: ориентировать React на приложения с длинными динамическими списками, обязательно используя ключи, мемоизацию и виртуализацию; выбирать Svelte для интерфейсов с глубокой вложенностью и требованием сверхбыстрой первой отрисовки, дополняя его библиотеками виртуального прокручивания при больших наборах данных; применять Angular в корпоративных системах, где приоритетны типизация и встроенный стек, строго активируя trackBy, OnPush и дифференциальную загрузку. Универсального лидера нет; выбор технологии должен основываться на аналитическом профиле операций и топологии интерфейса. Представленная методика измерений обеспечивает воспроизводимую основу для дальнейшего сравнения новых версий библиотек. Перспективным выглядит использование технологии веб-ассемблера WebAssembly [15] для увеличения производительности, а также использования адаптивной гидратации [17] в дальнейших исследованиях.

Библиография

1. Morgan P. JavaScript DOM Manipulation Performance: Comparing Vanilla JavaScript and Leading JavaScript Front-end Frameworks. 2020.
2. Levlin M. DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue and Svelte. LUT University, 2020. С. 58.
3. Aggarwal S. Modern WebDevelopment using ReactJS // International Journal of Recent Research Aspects. 2018. Т. 5, № 1. С. 133137.
4. Saks E. JavaScript frameworks: Angular vs React vs Vue. 2019.
5. Paakkanen J. Upcoming JavaScript Web Frameworks and Their Techniques. LUT University, 2022. 62 с.
6. Siahaan M., Kenidy R. Rendering performance comparison of React, Vue, Next and Nuxt // Jurnal Mantik. 2021. Т. 5, № 3. С. 18511860.
7. Iseal Sheed. Performance Benchmarking Techniques for React Applications. 2025.
8. Iseal Sheed. Edge Computing and React: Enhancing Performance at the Edge. 2025.
9. Białecki G., Pańczyk B. Performance analysis of Svelte and Angular applications // Journal of Computer Sciences Institute. 2021. Т. 19. С. 139-143. DOI: 10.35784/jcsi.2633. EDN: OYEGVL.
10. Piastou M. Comprehensive performance and scalability assessment of frontend frameworks: React, Angular, and Vue.js // World Journal of Advanced Engineering Technology and Sciences. 2023. Т. 9, № 2. С. 366376. DOI: 10.30574/wjaets.2023.9.2.0153. EDN: JNSAZF.
11. Маньшина Е. В., Ермолаева А. А. Сравнительный анализ производительности серверного и клиентского рендеринга вебприложений, созданных с помощью фреймворка Angular // Научнотехнические инновации и вебтехнологии. 2022. № 2. С. 44148.
12. Karić A., Durmić N. Comparison of JavaScript Frontend Frameworks - Angular, React, and Vue // International Journal of Innovative Science and Research Technology (IJISRT). 2024. С. 1383-1390. DOI: 10.38124/ijisrt/ijisrt.
13. Kaveri P. R. Framework-Agnostic JavaScript Component Libraries: Benefits, Implementation Strategies, and Commercialization Models // 2024 IEEE 16th International Conference on Computational Intelligence and Communication Networks (CICN). 2024. С. 1441-1446. EDN IGADIU.
14. Kasenda R., Tenda J., Iman E., Manantung J., Moekari Z., Pantas M. The Role and Evolution of Frontend Developers in the Software Development Industry // Jurnal Syntax Admiration. 2024. Т. 5. С. 5191-5196. DOI: 10.46799/jsa.v5i11.1852.

15. Акиева З. М., Мурзин Д. А., Эльтаев С. И. Применение WebAssembly для повышения производительности интерактивных веб-приложений // Экономика и управление: проблемы, решения. 2024. Т. 7, № 12. С. 168-174. DOI: 10.36871/ek.up.p.r. 2024.12.07.021. EDN: WJASJX.
16. Dubaj S., Pańczyk B. Comparative of React and Svelte programming frameworks for creating SPA web applications // Journal of Computer Sciences Institute. 2022. Т. 25. С. 345-349. DOI: 10.35784/jcsi.3020. EDN: AYUSVM.
17. Chen K. Improving Front-end Performance through Modular Rendering and Adaptive Hydration (MRAH) in React Applications. 2025. DOI: 10.48550/arXiv.2504.03884.
18. Кравцов Е. П. Разработка высокопроизводительных React-приложений: методы и практики оптимизации // European science. 2024. № 1 (69).
19. Дрогайцев И. А., Трамова А. М. Фронтенд-фреймворк Svelte как альтернатива популярным решениям в контексте создания систем поддержки принятия решений в сфере управления организацией с учетом трансформации экономики России // Известия КБНЦ РАН. 2024. № 4. DOI: 10.35330/1991-6639-2024-26-4-113-121. EDN: WRZTPH.
20. Яровая Е. В. Нестандартные архитектура в написание веб приложений // Столыпинский вестник. 2022.

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Предметом исследования в рецензируемой статье выступает производительность рендеринга в современных веб-фреймворках.
Методология исследования базируется на разработке тестового программного обеспечения путем создания клиентского веб приложения, визуализирующего иерархическую выборку данных в табличной форме для проведения эксперимента с двумя структурами данных: бинарная – эмулирует реальные сайты с высокой вложенностью, линейная – эмулирует большие таблицы.
Актуальность работы авторы связывают с тем, что вопрос выбора фронтенд фреймворка становится фактором финансовой эффективности продукта, поскольку использование фреймворков и библиотек значительно ускоряет разработку приложений и уменьшает их стоимость - всё это приводит к необходимости сравнительного анализа фреймворков.
Научная новизна исследования состоит в представленных результатах эмпирического сравнения производительности рендерингов фреймворков React, Angular и Svelte в типовых сценариях построения и обновления пользовательского интерфейса, разработке унифицированного набора тестовых сценариев, автоматизации сбора и анализа экспериментальных данных для выявления недостатков и преимуществ фреймворков.
Структурно в работе выделены разделы, озаглавленные следующим образом: Введение, Разработка тестового ПО, Реализация тестового приложения с использованием фреймворков, Анализ экспериментальных данных, Заключение, и Библиография.
В публикации отмечено, что при современном уровне развития средств визуализации, информационных технологий и компьютерной техники контент должен появляться на экране практически мгновенно, без ощутимых задержек. Авторами подробно, с иллюстрациями фрагментов программного кода отражена реализация тестового приложения, исходный код экспериментального ПО выложен в репозиторий на GitHub и доступен по приведенной в статье ссылке. Результаты эксперимента при сценарии первичного рендеринга представлены в графическом виде, текст статьи иллюстрирован

таблицей и 13 рисунками. По результатам исследования сформулированы рекомендации по использованию фреймворков React, Angular и Svelte для решения типичных классов задач, с указанием в каких случаях целесообразно отдавать предпочтение тем или иным программным платформам для получения двухмерного изображения.

Библиографический список включает 14 источников – интернет-ресурсы и научные публикации российских и зарубежных авторов по рассматриваемой теме на русском и иностранных языках. В тексте публикации имеются адресные отсылки к списку литературы, подтверждающие наличие апелляции к оппонентам.

Из замечаний стоит отметить следующие. Во-первых, в статье, подготовленной на русском языке имеется немало специальных терминов на английском языке, которые не сопровождаются их переводом – это может затруднить восприятие материала широким кругом читателей. Обращает на себя внимание и обилие заимствованных иностранных слов в тесте. Конечно, заимствованные слова в IT-сфере получили значительное распространение по вполне объяснимым причинам, но представляется уместным, по возможности, минимизировать использование иностранных слов, в том числе и в названии публикации. Во-вторых, авторами не выполнены рекомендации по оформлению списка литературы, принятые издательством: «Рекомендованный объем списка литературы для оригинальной научной статьи – не менее 20 источников, который должен содержать: не менее трети зарубежных источников; не менее половины работ, изданных в последние 3 года. В списке литературы не указываются … Интернет-источники, включая информацию с сайтов, а также статьи на сайтах и в блогах… Все вышеперечисленные источники упоминаются в тексте статьи в скобках, наряду с прочими комментариями и примечаниями авторов».

Тема статьи актуальна, соответствует тематике журнала «Программные системы и вычислительные методы», но материал нуждается в доработке в соответствии с высказанными замечаниями.

Результаты процедуры повторного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.

Со списком рецензентов издательства можно ознакомиться здесь.

Представленная статья на тему «Исследование производительности современных клиентских веб-фреймворков» и посвящена актуальному вопросу. Использование фреймворков и библиотек значительно ускоряет разработку приложений и уменьшает их стоимость на 30%, а также производительность веб-фреймворков остается одним из важных параметров. Все вышеперечисленное приводит к необходимости сравнительного анализа фреймворков, а также обосновывает актуальность работы.

В качестве цели статьи авторы указывают эмпирическое сравнение производительности рендеринга React, Angular и Svelte в типовых сценариях построения и обновления пользовательского интерфейса. Для достижения цели решаются задача разработки унифицированного набора тестовых сценариев, автоматизация сбора и анализ экспериментальных данных для выявления недостатков и преимуществ фреймворков.

В статье представлен достаточно широкий анализ литературных российских и зарубежных источников.

Стиль и язык изложения материала является достаточно доступным для широкого круга читателей. Практическая значимость статьи четко обоснована. Статья по объему соответствует рекомендуемому объему от 12 000 знаков.

Статья достаточно структурирована - в наличии введение, заключение, внутреннее членение основной части (разработка тестового программного обеспечения, реализация

тестового приложения с использованием фреймворков, анализ экспериментальных данных).

Авторами в рамках экспериментального исследования создано клиентское одностраничное веб приложение, визуализирующее иерархическую выборку данных в табличной форме. Результаты исследования представлены в табличном и графическом виде. Практическое применение результатов сводится к формулировке рекомендаций.

Авторами в заключительной части статьи представлены рекомендации, в том числе: ориентировать React на приложения с длинными динамическими списками, обязательно используя ключи, мемоизацию и виртуализацию; выбирать Svelte для интерфейсов с глубокой вложенностью и требованием сверхбыстрой первой отрисовки, дополняя его библиотеками виртуального прокручивания при больших наборах данных; применять Angular в корпоративных системах, где приоритетны типизация и встроенный стек, строго активируя trackBy, OnPush и дифференциальную загрузку. Универсального лидера нет; выбор технологии должен основываться на аналитическом профиле операций и топологии интерфейса.

По мнению авторов представленная методика измерений обеспечивает воспроизводимую основу для дальнейшего сравнения новых версий библиотек. В качестве перспектив исследования авторы выделяют использование технологии веб-ассемблера WebAssembly для увеличения производительности, а также использования адаптивной гидратации в дальнейших исследованиях.

К недостаткам можно отнести следующие моменты: из содержания статьи не прослеживается научная новизна. Отсутствует четкое выделение предмета исследования.

Рекомендуется четко обозначить научную новизну исследования, сформулировать предмет.

Статья «Исследование производительности современных клиентских веб-фреймворков» требует доработки по указанным выше замечаниям. После внесения поправок рекомендуется к повторному рассмотрению редакцией рецензируемого научного журнала.

Результаты процедуры окончательного рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.

Со списком рецензентов издательства можно ознакомиться здесь.

Статья посвящена сравнительному анализу производительности трех популярных клиентских веб-фреймворков – React, Angular и Svelte – в типовых сценариях построения и обновления пользовательского интерфейса. Автор исследует такие аспекты, как первичный и повторный рендеринг, обновление и удаление элементов, используя линейные и бинарные структуры данных.

Методология исследования включает разработку унифицированного набора тестовых сценариев, создание клиентского одностраничного веб-приложения и автоматизированный сбор данных. Для измерений использовались инструменты Puppeteer и браузер Google Chrome, что обеспечило высокую точность и воспроизводимость результатов. Автор также применил функцию `performance.now()` для фиксации времени выполнения операций, что соответствует современным стандартам тестирования производительности.

Актуальность исследования не вызывает сомнений, так как выбор веб-фреймворка напрямую влияет на пользовательский опыт и финансовые показатели продукта. Статья отвечает на запросы разработчиков и бизнеса, предоставляя эмпирические данные для

обоснованного выбора технологии. Упоминание таких метрик, как Largest Contentful Paint (LCP), подчеркивает практическую значимость работы.

Научная новизна исследования заключается в разработке автоматизированной методики сравнительного анализа производительности фреймворков, а также в детальном изучении их поведения в различных сценариях. Автор не только подтверждает известные выводы, но и выявляет новые закономерности, например, влияние структуры данных на эффективность React и Svelte.

Статья написана четким и логичным языком, с соблюдением академических норм. Структура работы включает введение, описание методологии, анализ результатов и выводы, что делает материал легко воспринимаемым. Использование таблиц и графиков (Рисунки 6–13) наглядно демонстрирует результаты, а ссылки на авторитетные источники (например, [10], [16]) укрепляют доверие к исследованию.

Автор приходит к обоснованному выводу, что ни один из фреймворков не является универсальным решением. React демонстрирует преимущества в работе с длинными списками, Angular — в корпоративных системах с типизацией, а Svelte — в сценариях с глубокой вложенностью и требованием к быстрой первой отрисовке. Эти выводы подкреплены экспериментальными данными и имеют практическую ценность для разработчиков.

Статья будет полезна широкому кругу читателей: от frontend-разработчиков, выбирающих инструменты для проектов, до исследователей в области веб-технологий. Рекомендации по оптимизации (например, использование виртуального прокручивания для Svelte) делают материал особенно ценным для практиков.

Статья представляет собой качественное исследование, сочетающее научную строгость с практической направленностью. Работа заслуживает высокой оценки и может быть рекомендована к публикации в топ-5 статей месяца издательства. Учитывая актуальность темы и глубину анализа, материал вызовет значительный интерес у читательской аудитории.

Рекомендация: принять статью к публикации без доработок.