**Дата поступления рукописи в редакцию: 10.06.2026 г.**
**Дата принятия рукописи в печать: 30.06.2026 г.**
DOI: 10.24412/3034-4271-2026-3-79-85

***ХАНОВ Искандер Маратович,***
*студент,*
*Казанский государственный энергетический университет,*
*e-mail: iskanderhanov@yandex.ru*

# СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПОДХОДОВ К РЕАКТИВНОМУ РЕНДЕРИНГУ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ: ПРОПРИЕТАРНЫЕ МЕХАНИЗМЫ И СТАНДАРТ TC39 SIGNALS

**Аннотация.** *Реактивный рендеринг в современных JavaScript-фреймворках реализован через собственные, несовместимые между собой механизмы. В статье проверяется, способны ли стандартизируемые примитивы TC39 Signals стать основой конкурентоспособного фреймворка. На базе полифилла signal-polyfill разработан экспериментальный микро-фреймворк и сопоставлен с пятью распространенными решениями по трём показателям: размер дистрибутива, производительность DOM-операций и объем прикладного кода. По всем трём исследуемый фреймворк показал результаты на уровне аналогов или выше в целевом классе задач - встроенные виджеты и статические сайты без этапа сборки. Результаты подтверждают достаточность примитивов TC39 Signals для построения фреймворка рендеринга.*

**Ключевые слова:** *реактивный рендеринг, веб-фреймворк, TC39 Signals, виртуальный DOM, fine-grained реактивность, производительность DOM, размер дистрибутива, сравнительный анализ.*

***KHANOV Iskander Maratovich,***
*4th year Bachelor's student,*
*Kazan State Power Engineering University*

# COMPARATIVE ANALYSIS OF APPROACHES TO REACTIVE RENDERING OF USER INTERFACES: PROPRIETARY MECHANISMS AND THE TC39 SIGNALS STANDARD

**Annotation.** *Reactive rendering in modern JavaScript frameworks is implemented through proprietary, mutually incompatible mechanisms. This paper examines whether the standardized TC39 Signals primitives can serve as a foundation for a competitive framework. An experimental micro-framework was built on the signal-polyfill reference implementation and benchmarked against five established solutions across three metrics: bundle size, DOM performance, and application code volume. Across all three metrics, the framework proved comparable to or better than its peers in the target use case - embedded widgets and static sites without a build step. The results confirm that TC39 Signals primitives are sufficient for building a rendering framework.*

**Key words:** *reactive rendering, web framework, TC39 Signals, virtual DOM, fine-grained reactivity, DOM performance, bundle size, comparative analysis.*

## Введение

За последнее десятилетие клиентские веб-приложения заметно усложнились. Современные интерфейсы обязаны мгновенно реагировать на действия пользователя, синхронизировать состояние между множеством компонентов и эффективно управлять обновлениями DOM. Индустрия ответила появлением реактивных JavaScript-фреймворков - React, Vue, Angular, Svelte, Solid.js - каждый из которых реализует собственный механизм отслеживания изменений и обновления интерфейса [1; 2].

Широкое распространение этих инструментов выявило структурный изъян экосистемы. Реактивность каждого фреймворка закрыта и несовместима с остальными: реализации одной и той же задачи дублируются, компоненты не переносимы между экосистемами, а зависимость от инструментальной цепочки сборки нередко оказывается чрезмерной. Для крупных продуктов подобные издержки терпимы. Но для проектов, где критичны минимальный размер дистрибутива и работа без этапа компиляции - встроенных виджетов, статических сайтов с интерактивными элементами - существующие решения либо избыточны по объёму, либо по-прежнему навязывают собственный закрытый механизм [3; 4].

В 2024 году комитет по стандартизации JavaScript (Ecma TC39) вынес на рассмотрение предложение Signals - набор реактивных прими-

тивов, встраиваемых непосредственно в спецификацию языка [5]. Р. Айзенберг, один из авторов предложения, подчеркивает, что стандартизация позволит фреймворкам совместно использовать единый реактивный слой вместо того, чтобы каждый поставлял собственный - что сделает их компактнее и быстрее [6]. Уже сейчас доступна эталонная программная реализация - полифилл - позволяющая использовать предложенный API до его появления в браузерах. Отсюда вопрос, которому посвящена настоящая работа: достаточны ли стандартизированные примитивы для построения конкурентоспособного фреймворка рендеринга? Эмпирического ответа на него в научной литературе пока нет.

***Цель исследования*** - сравнить существующие механизмы реактивного обновления интерфейсов и экспериментально проверить, могут ли примитивы TC39 Signals служить надежной основой минималистичного фреймворка.

Для достижения этой цели решаются следующие задачи:

1. Классифицировать существующие подходы к реактивному обновлению DOM и определить, какие архитектурные ограничения возникают при проприетарной реализации реактивности.

2. Реализовать экспериментальный микро-фреймворк на основе TC39 signal-polyfill и сравнить его с пятью распространёнными решениями (React, Vue 3, Svelte, Solid.js, Alpine.js) по трём метрикам: размер дистрибутива, производительность DOM-операций, объем прикладного кода;

3. На основании полученных данных оценить, способны ли примитивы TC39 Signals стать основой фреймворка, конкурентного существующим решениям и не требующего инструментов сборки.

***Методология.*** В работе применены методы анализа и синтеза научной литературы, сравнительного анализа программных решений, а также экспериментальный метод с количественными измерениями.

## Анализ существующих подходов к реактивному рендерингу

Реактивный рендеринг - автоматическое обновление DOM при изменении данных приложения [1]. По типу реактивного механизма существующие решения делятся на два класса: проприетарные, реализованные в каждом фреймворке по-своему, и стандартизируемый - основанный на предложении TC39 Signals. Первый класс представлен пятью категориями подходов, различающимися способом отслеживания изменений и обновления DOM (от виртуального дерева до fine-grained сигналов). Второй на момент работы представлен экспериментальным микро-фреймворком на базе полифилла signal-polyfill. Общая классификация приведена на рис. 1.

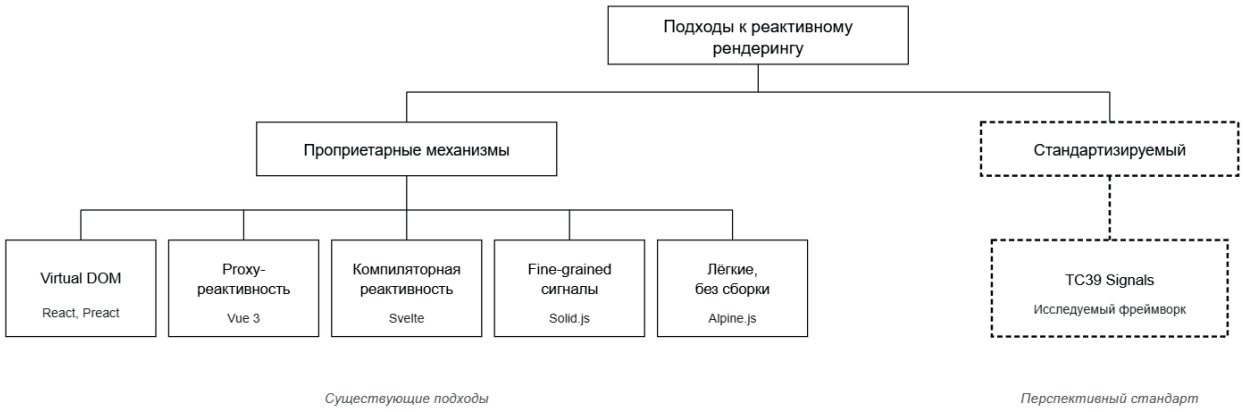



*Рисунок 1. Классификация подходов к реактивному рендерингу веб-интерфейсов*

***Virtual DOM (React, Preact).*** Разработчик декларативно описывает целевое состояние интерфейса, а фреймворк сам вычисляет минимальный набор DOM-изменений путем сравнения старого и нового дерева элементов [7]. Подход удобен, однако Ollila et al. [2] и Bai [8] показывают, что такое сравнение создает измеримые накладные расходы при работе с большими списками. Реактивность при этом встроена в ядро фреймворка и с другими решениями не совместима.

***Реактивность через Proxy (Vue 3).*** Изменения данных отслеживаются через стандартный Proxy API [9]. Реактивный модуль @vue/reactivity формально отделен от рендерера, но спецификации за ним нет - вне экосистемы Vue он остается специфичным для нее.

***Компиляторная реактивность (Svelte).*** Вместо runtime-отслеживания компилятор генерирует императивный код обновления DOM напрямую [3; 4]. Дистрибутив получается минимальным, но ценой жёсткой зависимости от этапа сборки - CDN-подключение и встроенные виджеты оказываются недоступны.

***Fine-grained сигналы (Solid.js).*** Каждый DOM-узел подписан на конкретный сигнал и обновляется точечно [10]. По духу это близко к тому, что предлагает TC39 [5], однако API Solid.js собственный, а для JSX всё равно нужен ком-

пилятор.

***Легкие фреймворки без этапа сборки (Alpine.js).*** Реактивность объявляется прямо в HTML-атрибутах, подключение - одним тегом <script>. По сценарию использования ближе всего к задачам настоящей работы, но движок реактивности точно так же закрыт и несовместим с другими.

Итог для всех пяти категорий одинаков: реактивность остаётся проприетарной концепцией конкретного фреймворка, что ведет к фрагментации экосистемы и невозможности переноса компонентов. Инициатива TC39 Signals [5; 6] нацелена на устранение этой проблемы - стандартизацию реактивных примитивов на уровне спецификации языка.

## Описание исследуемого решения и методика сравнения

Для проверки того, покрывают ли стандартизированные реактивные примитивы потребности реального фреймворка, была разработана экспериментальная реализация. Представленная версия не является финальным продуктом - это минимально необходимый набор функций для проведения эксперимента.

Архитектура строится на трёх принципах, каждый из которых снимает одно из ограничений, выявленных при анализе существующих решений (рис. 2).

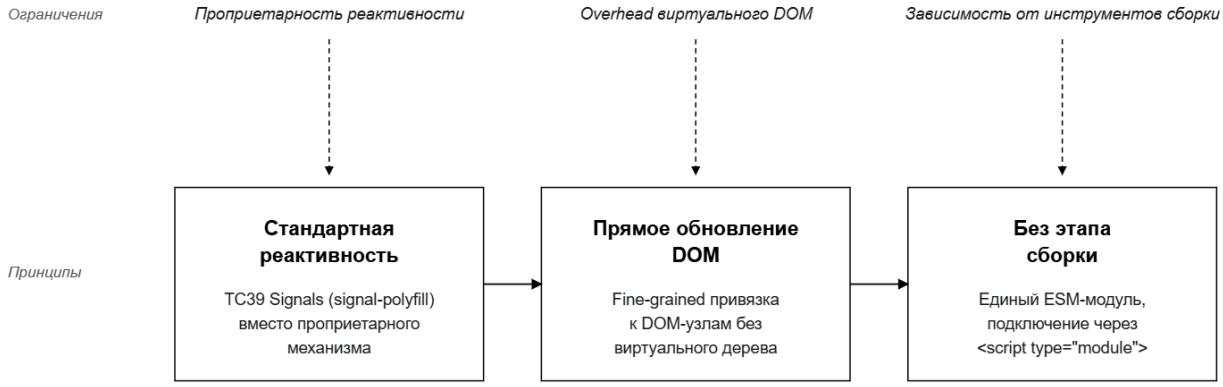



*Рисунок 2. Конструктивные принципы исследуемого фреймворка*

***Стандартная реактивность.*** Вместо собственного механизма отслеживания зависимостей фреймворк опирается на полифилл TC39 Signals - signal-polyfill [5], API которого следует черновику спецификации [11]. Когда браузеры реализуют Signals нативно, полифилл можно будет исключить без изменения прикладного кода [6].

***Прямое обновление DOM.*** Виртуальное дерево не строится - каждый реактивный узел шаблона при инициализации получает ссылку на конкретный DOM-элемент и обновляет его напрямую при изменении сигнала. По сути это fine-grained реактивность, но без компилятора.

***Отсутствие этапа сборки.*** Фреймворк распространяется как единственный ESM-модуль, подключаемый через *<script type=»module»>* или CDN. По этому признаку он близок к Alpine.js, но реактивность здесь опирается на стандартизируемый, а не собственный закрытый механизм.

Реализованного набора примитивов хватает для полноценного TodoMVC - стандартизированного тестового приложения-списка задач, принятого в сообществе как эталон для сравнения фреймворков [12]. Исходный код и все тестовые реализации опубликованы в открытом репозитории [13]. Ключевые характеристики фреймворка в сопоставлении с аналогами приведены в таблице 1.

*Таблица 1. Сравнение ключевых принципов рассматриваемых решений*

| Решение | Версия | Механизм реактивности | Виртуальный DOM | Требует сборку | Основа обновления DOM |
|---|---|---|---|---|---|
| React | 18.x | Проприетарный | Да | Да | Диффинг VDOM |
| Vue 3 | 3.x | Проприетарный (Proxy) | Нет | Да | Proxy + патч |
| Svelte | 4.x | Компилятор | Нет | Да | Генерируемый код |
| Solid.js | 1.x | Проприетарный | Нет | Да (JSX) | Fine-grained |

| | | | | | |
|---|---|---|---|---|---|
| Alpine.js | 3.x | Проприетарный | Нет | Нет | Proxy + патч |
| Исследуемый фреймворк | 1 | TC39 Signals (стандарт) | Нет | Нет | Fine-grained |

Сравнение ведётся по трём критериям, каждый из которых отвечает за практически значимый аспект.

***Размер дистрибутива*** - объем минифицированного бандла в сжатом виде (gzip, КБ). Для React, Vue 3, Solid.js и Alpine.js данные получены с bundlephobia.com [14]; для исследуемого фреймворка - измерены через сборку в Vite (ESM, production).

***Производительность DOM-операций*** - время (мс) типовых действий с динамическим списком из 1 000 строк: создание, замена, частичное обновление, выбор и удаление. В качестве инструмента выбран js-framework-benchmark [15]. Данные для пяти фреймворков взяты из опубликованных результатов этого репозитория; замеры исследуемого фреймворка выполнены по эквивалентной методике - Chrome, пятикратное повторение, исключение первого прогона как JIT-разогрева [13].

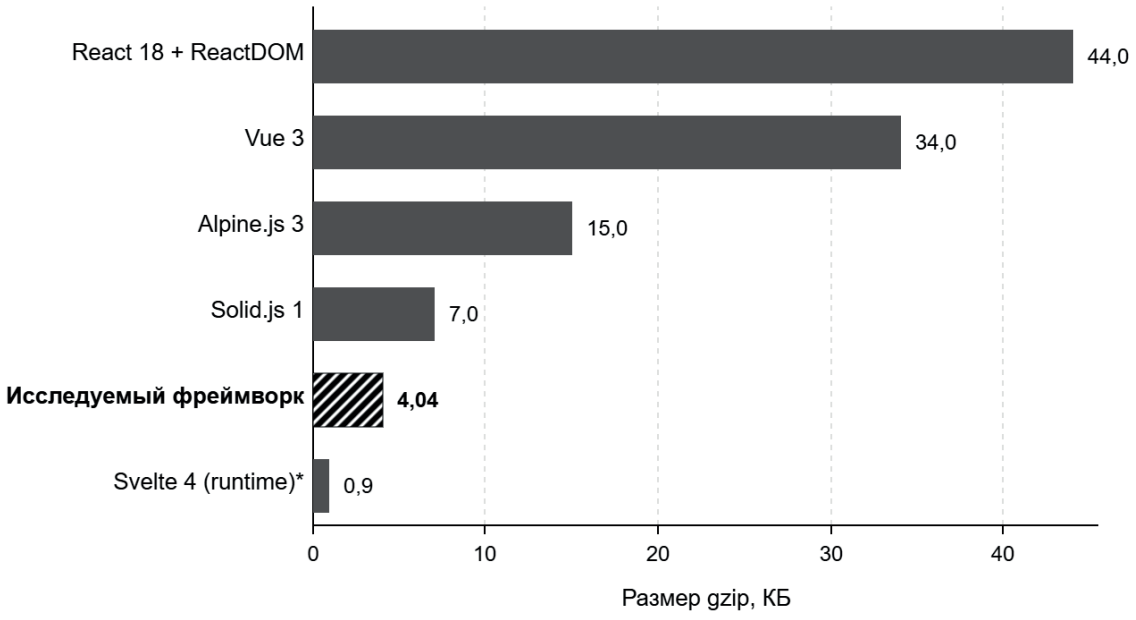
***Объем прикладного кода*** - содержательные строки (LOC) реализации TodoMVC [12]. Реализации React, Vue 3, Svelte и Solid.js взяты из tastejs/todomvc [12]; для Alpine.js официальная версия отсутствует. Реализация исследуемого фреймворка - файл *examples/todomvc/app.ts* [13].

**Результаты сравнительного анализа и их обсуждение**

Замеры размеров дистрибутивов собраны в табл. 2:

*Таблица 2. Размер дистрибутива фреймворков*

| Решение | Minified (КБ) | Gzip (КБ) |
|---|---|---|
| React 18 + ReactDOM | 144,0 | 44,0 |
| Vue 3 (runtime-only) | 96,0 | 34,0 |
| Alpine.js 3 | 44,0 | 15,0 |
| Solid.js 1 | 18,0 | 7,0 |
| Исследуемый фреймворк | 14,79 | 4,04 |
| Svelte 4 (runtime) | 1,7 | 0,9 |




** Runtime без учёта кода, генерируемого компилятором*

*Рисунок 3. Сравнение размеров библиотек по метрике gzip, КБ*

При 4,04 КБ gzip исследуемый фреймворк оказывается в 10,9 раза меньше React и в 3,7 раза меньше Alpine.js. Причина - отсутствие Virtual DOM и собственного механизма сравнения деревьев: логика отслеживания зависимостей целиком делегирована signal-polyfill, который при нативной поддержке Signals браузерами будет исключен полностью. Svelte формально показывает наименьший runtime (0,9 КБ), однако эта цифра обманчива - компилятор переносит логику реактивности непосредственно в код приложения, и реальная стоимость растёт вместе с его объемом.

Таблица 3 содержит данные по скорости DOM-операций:

*Таблица 3. Производительность DOM-операций, мс*

| Решение | Создание 1000 строк | Замена 1000 строк | Частичное обновление | Выбор строки | Удаление строки |
|---|---|---|---|---|---|
| React 18 | 43,0 | 44,0 | 56,0 | 12,0 | 36,0 |
| Vue 3.5 | 30,0 | 31,0 | 13,0 | 5,0 | 28,0 |
| Svelte 5 | 26,0 | 27,0 | 11,5 | 4,0 | 22,0 |
| Solid.js 1.9 | 25,0 | 26,0 | 11,2 | 4,0 | 18,0 |
| Alpine.js 3 | 190,0 | 191,0 | 60,0 | 16,0 | 70,0 |
| Исследуемый фреймворк | 43,6 | 44,1 | 3,7 | 3,0 | 6,8 |

Создание и замена списка целиком - на уровне React; оба уступают компиляторным решениям (Svelte, Solid.js) примерно в 1,7 раза. Точечные операции, однако, дали неожиданный для нас результат: частичное обновление заняло 3,7 мс - втрое быстрее Solid.js (11,2 мс) и в 15 раз быстрее React (56,0 мс). Мы объясняем это тем, что прямое обновление DOM-узлов без промежуточного сравнения деревьев устраняет тот самый overhead, который зафиксировали Ollila et al. [2] и Bai [8]. Для целевого сценария - виджеты с частыми точечными обновлениями - именно этот класс операций оказывается определяющим.

Объём кода TodoMVC для каждого фреймворка показан в табл. 4:

*Таблица 4. Объём кода TodoMVC (содержательных строк, без пустых и комментариев; источники: tastejs/todomvc [12], исследуемый фреймворк - [13])*

| Решение | Файлов | JS-логика | Шаблон/JSX | Итого |
|---|---|---|---|---|
| React 18 | 9 | 120 | 79 | 199 |
| Vue 3.5 | 5 | 117 | 50 | 167 |
| Svelte 5 | 4 | 85 | 46 | 131 |
| Solid.js 1.9 | 1 | 78 | 47 | 125 |
| Alpine.js 3 | - | - | - | н/д |
| Исследуемый фреймворк | 1 | 131 | 48 | 179 |

Реализация на исследуемом фреймворке уместилась в 179 строк в одном файле. Это на уровне Vue 3 (167 строк в 5 файлах) и компактнее React (199 строк в 9 файлах). Svelte и Solid.js лаконичнее - 131 и 125 строк - за счёт компиляции шаблонов. Но принципиальное отличие не в количестве строк, а в том, что вся логика умещается в одном файле без этапа сборки.

**Применимость и ограничения**

Наибольший выигрыш от исследуемого подхода достигается там, где одновременно критичны малый размер дистрибутива и работа без инструментов сборки: встроенные виджеты, статические сайты с интерактивными элементами, подключение через CDN. Для крупных SPA с серверным рендерингом и развитой инструментальной экосистемой зрелые решения вроде React и Vue остаются более обоснованным выбором. Вместе с тем использованный бенчмарк js-framework-benchmark измеряет операции на одном плоском списке строк. Нам не удалось охватить такие характерные для реальных приложений сценарии,

как многоуровневая вложенность компонентов и обновления, приходящие асинхронно (по таймерам, сетевым запросам, пользовательским событиям) - это направление для дальнейшего исследования. Кроме того, TC39 Signals на момент работы находился на стадии Stage 1 [5], и вместо нативной реализации использовался полифилл.

**Заключение**

Проведенное исследование подтверждает: стандартизированных реактивных примитивов TC39 Signals достаточно для построения конкурентоспособного фреймворка рендеринга в классе задач, где критичны минимальный размер дистрибутива и отсутствие этапа сборки.

Анализ пяти категорий существующих решений показал, что реактивность во всех них реализована как проприетарная, несовместимая между собой концепция. Инициатива TC39 Signals предлагает выход из этой фрагментации - стандартизацию реактивных примитивов на уровне спецификации языка.

Экспериментальное сравнение по трём метрикам дало следующие результаты. По размеру исследуемый фреймворк (4,04 КБ gzip) меньше React в 10,9 раза и Alpine.js - в 3,7 раза. По производительности точечных операций он втрое быстрее Solid.js и в 15 раз быстрее React. По объёму кода - 179 строк в одном файле против 199 строк в девяти у React - при схожем функциональном охвате и без инструментов сборки.

Настоящая работа является первой эмпирической оценкой применимости полифилла TC39 Signals как основы фреймворка рендеринга. Дальнейшие исследования целесообразны после нативной реализации Signals в браузерах, а также в части серверного рендеринга, крупных приложений и возможности переноса компонентов между фреймворками, опирающимися на единый стандартизированный реактивный слой.

**Список литературы:**

[1] Bainomugisha E., Lombide Carreton A., Van Cutsem T., Mostinckx S., De Meuter W. A Survey on Reactive Programming // ACM Computing Surveys. — 2013. — Т. 45, № 4. — Article 52. — 34 с.

[2] Ollila R., Mäkitalo N., Mikkonen T. Modern Web Frameworks: A Comparison of Rendering Performance // Journal of Web Engineering. — 2022. — Т. 21, № 3. — С. 789–813.

[3] Harris R. Frameworks without the framework: why didn't we think of this sooner? // Svelte Blog. — 2016. — URL: https://svelte.dev/blog/frameworks-without-the-framework (дата обращения: 13.04.2026).

[4] Harris R. Virtual DOM is pure overhead // Svelte Blog. — 2018. — URL: https://svelte.dev/blog/virtual-dom-is-pure-overhead (дата обращения: 13.04.2026).

[5] Ehrenberg D., Eisenberg R. et al. TC39 Proposal: Signals // GitHub TC39. — 2024. — URL: https://github.com/tc39/proposal-signals (дата обращения: 13.04.2026).

[6] Eisenberg R. A TC39 Proposal for Signals // Medium (Better Programming). — 2024. — URL: https://eisenbergeffect.medium.com/a-tc39-proposal-for-signals-f0bedd37a335 (дата обращения: 13.04.2026).

[7] React Team. React — A JavaScript library for building user interfaces // Meta Open Source. — 2024. — URL: https://react.dev (дата обращения: 13.04.2026).

[8] Bai A. Million.js: A Fast Compiler-Augmented Virtual DOM for the Web // Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23). — New York : ACM, 2023. — С. 1813–1820.

[9] You E. Composition API RFC // GitHub. — 2019. — URL: https://github.com/vuejs/rfcs/blob/master/active-rfcs/0013-composition-api.md (дата обращения: 13.04.2026).

[10] Carniato R. A Hands-on Introduction to Fine-Grained Reactivity // DEV Community. — 2021. — URL: https://dev.to/ryansolid/a-hands-on-introduction-to-fine-grained-reactivity-3ndf (дата обращения: 13.04.2026).

[11] ECMA International. ECMAScript 2024 Language Specification (ECMA-262). — 2024. — URL: https://tc39.es/ecma262/ (дата обращения: 13.04.2026).

[12] TasteJS. TodoMVC — Helping you select an MV* framework // GitHub. — 2012–2026. — URL: https://github.com/tastejs/todomvc (дата обращения: 13.04.2026).

[13] Автор. micro-reactive — микро-фреймворк для реактивного рендеринга интерфейсов // GitLab. — 2025–2026. — URL: https://gitlab.com/eners/micro-reactive-framework (дата обращения: 13.04.2026).

[14] Bundlephobia — online tool for finding the cost of adding a npm package to your bundle. — URL: https://bundlephobia.com (дата обращения: 13.04.2026).

[15] Krause S. js-framework-benchmark // GitHub. — 2015–2026. — URL: https://github.com/krausest/js-framework-benchmark (дата обращения: 13.04.2026).

**References:**

[1] Bainomugisha E., Lombide Carreton A., Van Cutsem T., Mostinckx S., De Meuter W. A Survey on Reactive Programming // ACM Computing Surveys. - 2013. - Т. 45, No. 4. - Article 52. - 34 p.

[2] Ollila R., Mäkitalo N., Mikkonen T. Modern Web Frameworks: A Comparison of Rendering Performance // Journal of Web Engineering. - 2022. - Т. 21, No. 3. - P. 789–813.

[3] Harris R. Frameworks without the framework: why didn't we think of this sooner? // Svelte Blog. — 2016. — URL: https://svelte.dev/blog/frameworks-without-the-framework (accessed: 13.04.2026).

[4] Harris R. Virtual DOM is pure overhead // Svelte Blog. — 2018. — URL: https://svelte.dev/blog/virtual-dom-is-pure-overhead (accessed: 13.04.2026).

[5] Ehrenberg D., Eisenberg R. et al. TC39 Proposal: Signals // GitHub TC39. — 2024. — URL: https://github.com/tc39/proposal-signals (accessed: 13.04.2026).

[6] Eisenberg R. A TC39 Proposal for Signals // Medium (Better Programming). - 2024. - URL: https://eisenbergeffect.medium.com/a-tc39-proposal-for-signals-f0bedd37a335 (access date: 04/13/2026).

[7] React Team. React — A JavaScript library for building user interfaces // Meta Open Source. - 2024. - URL: https://react.dev (access date: 04/13/2026).

[8] Bai A. Million.js: A Fast Compiler-Augmented Virtual DOM for the Web // Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23). - New York: ACM, 2023. - pp. 1813–1820.

[9] You E. Composition API RFC // GitHub. — 2019. — URL: https://github.com/vuejs/rfcs/blob/master/active-rfcs/0013-composition-api.md (date of access: 13.04.2026).

[10] Carniato R. A Hands-on Introduction to Fine-Grained Reactivity // DEV Community. — 2021. — URL: https://dev.to/ryansolid/a-hands-on-introduction-to-fine-grained-reactivity-3ndf (date of access: 13.04.2026).

[11] ECMA International. ECMAScript 2024 Language Specification (ECMA-262). — 2024. — URL: https://tc39.es/ecma262/ (date of access: 13.04.2026).

[12] TasteJS. TodoMVC — Helping you choose an MV* framework // GitHub. — 2012–2026. — URL: https://github.com/tastejs/todomvc (accessed: 13.04.2026).

[13] Author. micro-reactive — A micro-framework for reactive rendering of interfaces // GitLab. — 2025–2026. — URL: https://gitlab.com/eners/micro-reactive-framework (accessed: 13.04.2026).

[14] Bundlephobia — An online tool for finding the cost of adding an npm package to your bundle. — URL: https://bundlephobia.com (accessed: 13.04.2026).

[15] Krause S. js-framework-benchmark // GitHub. — 2015–2026. — URL: https://github.com/krausest/js-framework-benchmark (accessed: 13.04.2026).