

## TEXNİKA ELMLƏRİ TECHNICAL SCIENCES

# Comparative Analysis of React.js and Vue.js Technologies

Mahammad Nabiyeu 

**Abstract.** The selection of JavaScript-based libraries and frameworks for modern web application development has a direct impact on software performance, scalability, and long-term maintainability. This paper presents a comparative analysis of React.js and Vue.js, two widely used technologies in front-end development. The aim of the study is to evaluate these two approaches not only in terms of syntax and popularity but also with respect to architectural principles, component model, reactivity mechanisms, Virtual DOM implementation, performance, server-side rendering (SSR), ecosystem maturity, tooling support, and long-term maintainability. The research is based on a comparative analysis of the official technical documentation for React.js and Vue.js, as well as scholarly publications indexed in the Scopus and Web of Science databases. The findings indicate that React.js stands out due to its extensive ecosystem, architectural flexibility, and suitability for complex, large-scale applications. In contrast, Vue.js offers an integrated reactivity system, a gentler learning curve, comprehensive official tooling, and lower initial complexity, making it an effective solution for small- and medium-sized projects. The study concludes that the choice between React.js and Vue.js should be determined by the project's scale, functional requirements, the development team's technical expertise, and the product's long-term development strategy.

**Keywords:** React.js, Vue.js, JavaScript frameworks, front-end development, component-based architecture, reactivity, Virtual DOM, server-side rendering (SSR), performance, web application development

---

Odlar Yurdu University, Baku, Azerbaijan

E-mail: [nabiyevvm@gmail.com](mailto:nabiyevvm@gmail.com)

Received: 28 March 2026; Accepted: 29 May 2026; Published online: 30 July 2026

© The Author(s) 2026. This is an open access article distributed under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

## React.js və Vue.js texnologiyalarının müqayisəli təhlili

Məhəmməd Nəbiyev 

**Xülasə.** Müasir veb tətbiqlərinin hazırlanmasında JavaScript əsaslı kitabxana və çərçivələrin seçimi proqram təminatının performansına, genişlənmə qabiliyyətinə və uzunmüddətli müşayiətinə birbaşa təsir göstərir. Bu məqalədə front-end inkişafında geniş istifadə olunan React.js və Vue.js texnologiyalarının müqayisəli təhlili təqdim edilir.

*Tədqiqatın məqsədi bu iki yanaşmanı yalnız sintaktik xüsusiyyətləri və populyarlığı baxımından deyil, həm də memarlıq prinsipləri, komponent modeli, reaktivlik mexanizmləri, virtual DOM-un işləmə prinsipi, performans, server tərəfli renderləmə (SSR), ekosistemin yetkinliyi, alət dəstəyi və uzunmüddətli saxlanılma qabiliyyəti baxımından qiymətləndirməkdir. Araşdırma müqayisəli analiz metoduna əsaslanaraq React.js və Vue.js üzrə rəsmi texniki sənədlər, eləcə də Scopus və Web of Science bazalarında indekslənən elmi məqalələr əsasında həyata keçirilmişdir. Təhlilin nəticələri göstərir ki, React.js geniş ekosistemi, yüksək çevikliyi və mürəkkəb, iri miqyaslı layihələr üçün uyğun memarlıq imkanları ilə fərqlənir. Vue.js isə daxili reaktivlik sistemi, daha sadə öyrənmə prosesi, integrasiya olunmuş rəsmi alətləri və aşağı başlanğıc mürəkkəbliyi sayəsində kiçik və orta həcmli layihələrdə səmərəli həll kimi çıxış edir. Nəticə etibarilə müəyyən edilmişdir ki, React.js və Vue.js arasında seçim layihənin miqyası, funksional tələbləri, komandanın texniki hazırlıq səviyyəsi və məhsulun uzunmüddətli inkişaf strategiyası nəzərə alınmaqla həyata keçirilməlidir.*

**Açar sözlər:** *React.js, Vue.js, JavaScript çərçivələri, front-end proqramlaşdırma, komponent əsaslı arxitektura, reaktivlik, virtual DOM, server tərəfli renderləmə (SSR), performans, veb tətbiqlərinin hazırlanması*

Odlar Yurdu Universiteti, Bakı, Azərbaycan

E-poçt: nabiyevvm@gmail.com

Daxil oldu: 28 Mart 2026; Qəbul edildi: 29 May 2026; Onlayn dərc edildi: 30 İyul 2026

© Müəllif(lər) 2026. Bu açıq girişli məqalə Creative Commons Attribution–NonCommercial 4.0 Beynəlxalq Lisenziyasının (CC BY-NC 4.0) şərtlərinə uyğun olaraq yayımlanmışdır.

## Introduction

Web applications have evolved from static pages into complex software systems that combine user interaction, state synchronization, data fetching, routing, and visual feedback in real time. As a result, the front-end layer has become a strategic part of software architecture rather than a simple presentation layer. In this environment, JavaScript-based component technologies play a central role because they determine how interfaces are structured, updated, tested, and maintained over time (Swacha & Kulpa, 2023).

React.js and Vue.js are among the most influential technologies in this area (Pano et al., 2018; Swacha & Kulpa, 2023). React is usually described as a library for building user interfaces, but in practice it functions as the foundation of a large ecosystem that includes routing, state management, server rendering, testing, and design-system tools (React, 2026a). Vue.js, by contrast, presents itself as a progressive framework: it can be used in a small part of an existing page or scaled into a complete single-page application with official routing and state-management solutions (Vue.js, 2026b). Both technologies are mature and production-ready, but they reflect different engineering philosophies (Rathinam, 2023).

The main research problem is therefore not to identify a universal winner. A framework that is suitable for a large enterprise product may be excessive for a small internal system, while a tool that is very easy to start with may require stronger architectural control when the product grows. For this reason, the present paper evaluates React and Vue through software-quality criteria: architecture, reactivity, component model, ecosystem, rendering behavior, server-side rendering, maintainability, scalability, and developer experience.

## Methods

The study uses a comparative analytical method based on secondary sources. Priority is given to peer-reviewed papers and conference proceedings that are visible in Scopus and/or Web of Science-related publication channels, including Springer series, ACM proceedings, and indexed journals. These academic sources are complemented by official React and Vue documentation because framework documentation is the most reliable source for implementation-level concepts such as hooks, refs, reactivity, rendering, and server APIs.

Recent literature supports a multidimensional comparison rather than a narrow benchmark-only approach. Loja and Maita evaluate Angular, React.js, and Vue.js through ISO/IEC 25010 performance-efficiency characteristics, emphasizing temporal behavior and resource utilization (Loja & Maita, 2024). Rathinam compares popular front-end frameworks and argues that the appropriate framework depends on project needs, team capability, and development context (Rathinam, 2023). Other studies focus on design-pattern performance in React and Vue (Gomes et al., 2023), framework popularity and ecosystem evolution (Swacha & Kulpa, 2023), adoption factors in JavaScript framework choice (Pano et al., 2018), and rendering strategy performance across modern web frameworks (Ollila et al., 2022). Together, these sources show that technology choice should be interpreted as an engineering decision shaped by both technical and organizational variables.

### *Architectural and Component Model*

React is built around reusable components and a declarative rendering model. Components receive data through props and manage local state through hooks such as `useState`, `useReducer` (React, 2026a), and `useEffect`. JSX allows developers to express interface structure and logic in JavaScript, which gives teams strong compositional power (React, 2026b). This approach is especially effective in large systems where shared components, design systems, and domain-specific abstractions are important. However, React deliberately leaves many architectural choices to developers. Routing, global state, data fetching, form management, and project structure are commonly solved through separate libraries or internal conventions.

This freedom is one of React's strongest advantages and one of its risks. In a mature team, architectural openness enables flexible integration with different back-end services, build pipelines, and deployment models. In a less experienced team, the same openness may lead to inconsistent folder structures, duplicated patterns, and unnecessary dependencies. Therefore, React scales best when the organization has clear standards for component boundaries, hook usage, state placement, testing, and code review.

Vue is also component-based, but it gives developers a more guided structure. Single-File Components place template, script, and style sections in one file, creating a clear and readable development unit (Vue.js, 2026a). Vue's template syntax is close to HTML, which makes the learning path smoother for developers who already know traditional front-end development. At the same time, Vue can still support advanced composition through the Composition API (Vue.js, 2026a). This combination makes Vue suitable for teams that want modern component architecture without too much initial architectural decision-making.

### *Reactivity and State Management*

State management is a central difference between the two technologies. React promotes explicit state flow (React, 2026a). Data moves from parent to child through props, and shared state is handled through Context API or external stores. This explicit model helps developers reason about ownership of data, but it also creates decisions about when to lift state, when to isolate state locally, and when to introduce a central store. In small applications this is simple; in large products it requires discipline.

Vue's reactivity is more integrated into the framework (Vue.js, 2026a). Reactive objects, refs, computed values, and watchers allow state changes to be tracked automatically and reflected in the interface. This creates a shorter path from data to UI behavior and can reduce boilerplate. For larger projects, Vue Router and Pinia provide official or officially recommended scaling paths. The result is that Vue often reduces uncertainty in early and medium-size projects, while React offers more freedom for teams that need a customized state architecture.

Neither approach is automatically better. React's explicitness is valuable when complex state transitions must be carefully controlled and tested. Vue's built-in reactivity is valuable when rapid development, readability, and predictable conventions are priorities. The choice depends on whether the project needs maximum architectural flexibility or a more integrated developer experience.

### ***Rendering, Performance, and SSR***

Performance comparisons should be interpreted carefully (Loja & Maita, 2024; Gomes et al., 2023; Ollila et al., 2022). Both React and Vue are capable of production-grade performance, and real results depend heavily on application architecture, bundle size, state locality, rendering frequency, network strategy, and developer practice. Empirical work shows that front-end frameworks may differ in temporal behavior and resource utilization, but it also confirms that performance is context-sensitive rather than absolute (React, 2026c). Rendering-focused research likewise indicates that framework strategies can scale differently as interface complexity grows (Ollila et al., 2022).

React uses a declarative model in which components describe the desired UI and the runtime reconciles changes. Optimization often involves memoization, stable references, state localization, lazy loading, and appropriate use of server rendering. Vue combines virtual DOM concepts with compiler-informed optimizations and dependency-tracked reactivity. In many ordinary applications, Vue can feel efficient with less manual tuning, whereas React can provide strong control when the team has enough performance-engineering expertise (Vue.js, 2026b).

Server-side rendering and hybrid delivery have become important because users expect fast first load, search engines require readable content, and businesses need scalable deployment. React supports server APIs and is strongly associated with meta-frameworks such as Next.js (React, 2026a). Vue supports SSR through official guidance and Vue-based meta-frameworks. React currently has broader market visibility in SSR-heavy products, while Vue offers a coherent path for teams that prefer a more unified documentation and tooling model.

### ***Developer Experience, Maintainability, and Scalability***

Developer experience directly influences project cost. React benefits from a very large community, strong labor-market familiarity, many third-party packages, and deep adoption in enterprise systems (Swacha & Kulpa, 2023). This makes recruitment and long-term ecosystem support easier. However, React's ecosystem breadth can increase onboarding complexity because developers must understand JSX, hooks, side effects, routing choices, state libraries, and data-fetching conventions before they can work confidently in a non-trivial product (React, 2026b).

Vue's main advantage is coherence. Its official documentation, template syntax, reactive model, router, and state-management guidance reduce early complexity (Vue.js, 2026a). This is useful in educational contexts, start-up, internal administrative systems, and small-to-medium products where delivery speed and readability matter. Vue can also scale to larger systems, but React is often preferred in organizations that already have platform teams, shared component libraries, and established engineering governance.

| Comparative Summary |                                              |                                                 |                                         |
|---------------------|----------------------------------------------|-------------------------------------------------|-----------------------------------------|
| Criterion           | React.js                                     | Vue.js                                          | Analytical note                         |
| Core model          | UI library with JSX and hooks                | Progressive framework with SFCs and templates   | React is more open; Vue is more guided. |
| Learning curve      | Moderate; ecosystem decisions add complexity | Smoother for many beginners and mixed teams     | Vue reduces initial overhead.           |
| State handling      | Hooks, Context, external stores              | Refs, reactive objects, Pinia                   | React is explicit; Vue is integrated.   |
| Performance         | Strong with disciplined optimization         | Efficient defaults and compiler support         | Architecture matters more than labels.  |
| SSR                 | Strong, especially through Next.js ecosystem | Strong through official SSR guidance and tools  | Both are production-capable.            |
| Best fit            | Large products needing flexible integration  | Small/medium projects needing clarity and speed | Context should drive the decision.      |

Maintainability depends less on the framework name and more on the quality of conventions. A React project with strong standards can remain clean for years, while a Vue project without modular discipline can still become difficult to maintain. Nevertheless, the two technologies distribute complexity differently: React places more responsibility on the team to design the surrounding architecture, whereas Vue provides more default structure and therefore lowers the cost of early standardization.

Results and Discussion

The comparative analysis indicates that neither React.js nor Vue.js demonstrates universal superiority across all software quality dimensions. Instead, the reviewed evidence consistently shows that framework effectiveness depends on the interaction between technical requirements, project scale, organizational maturity, and developer expertise. This finding is consistent with previous comparative studies emphasizing context-dependent framework selection rather than absolute performance rankings (Pano et al., 2018; Rathinam, 2023; Swacha & Kulpa, 2023).

From an architectural perspective, React provides greater flexibility by allowing developers to assemble application architecture from independent libraries and ecosystem tools. This flexibility is advantageous in enterprise environments where applications require custom state management, sophisticated routing strategies, micro-frontend architectures, or integration with heterogeneous back-end systems. However, the analysis also suggests that such flexibility increases the need for organizational standards, coding conventions, and architectural governance. Without these controls, projects may experience inconsistent implementation patterns and increased maintenance costs.

Vue.js demonstrates a contrasting pattern by offering a more integrated development model. Official support for routing, state management, and Single-File Components reduces architectural uncertainty and enables faster project standardization. The reviewed literature indicates that this integrated approach is particularly beneficial for educational environments, start-ups, administrative information systems, and small-to-medium software projects where rapid development and code readability are primary objectives.

The comparison of state-management models further illustrates the philosophical differences between the two technologies. React's explicit state flow encourages precise control over application logic and facilitates testing of complex state transitions. Conversely, Vue's reactive programming model simplifies implementation by automatically propagating state changes to the user interface. These approaches represent different trade-offs between architectural flexibility and development simplicity rather than differences in technical capability.

Performance analysis likewise reveals no definitive technological advantage. Existing empirical studies demonstrate that rendering efficiency, memory utilization, and application responsiveness are influenced more strongly by application architecture, rendering strategy, bundle optimization, and developer practices than by framework choice alone (Loja & Maita, 2024; Gomes et al., 2023; Ollila et al., 2022). Consequently, performance optimization should be viewed as an engineering discipline that extends beyond selecting a particular framework.

The findings also highlight important organizational considerations. React benefits from broader industrial adoption, extensive third-party ecosystems, and greater labor-market availability, making it attractive for long-term enterprise development. Vue, on the other hand, offers a more coherent learning experience and lower onboarding complexity, reducing development costs for teams with limited experience or restricted project timelines. Therefore, framework selection should balance technical characteristics with organizational resources, development processes, and long-term maintenance strategies.

Overall, the comparative findings support the view that framework selection represents a strategic software engineering decision rather than a purely technical comparison. Sustainable software quality depends not only on the characteristics of React.js or Vue.js but also on project governance, coding standards, testing practices, developer competence, and architectural consistency throughout the software life cycle.

### ***Practical recommendations***

For teams selecting a framework, the first practical step is to define the project profile before comparing syntax. If the product is expected to grow into a multi-team platform, React is often safer because its ecosystem supports many specialized architectural patterns. If the product is an administrative panel, educational system, prototype, or medium-scale business application, Vue can shorten delivery time because core decisions are already supported by official conventions.

The second recommendation is to evaluate the team rather than the framework alone. A team already experienced with hooks, component composition, testing libraries, and state-management patterns can benefit strongly from React. A team with mixed experience or limited time for architectural experimentation may benefit more from Vue because its templates, Single-File Components, and official scaling tools make the codebase easier to standardize. In both cases, maintainability requires written conventions for naming, folder structure, component reuse, state ownership, and performance review.

Finally, framework choice should be reviewed together with deployment needs. Content-heavy and SEO-sensitive systems should consider SSR or hybrid rendering from the beginning. Highly interactive dashboards should pay more attention to state locality, rendering frequency, and bundle optimization. These concerns are solvable in both ecosystems, but the cost of solving them differs according to the chosen architecture and the team's experience (Rathinam, 2023; Swacha & Kulpa, 2023).

## Conclusion

React.js and Vue.js are both mature front-end technologies, but they represent different balances between flexibility and convention. React is better understood as a flexible component foundation supported by a very broad ecosystem. It is especially strong when teams need custom architecture, deep integration options, enterprise-scale patterns, and long-term ecosystem availability. Vue is better understood as a progressive but more integrated framework. It is especially strong when teams need fast onboarding, readable structure, official guidance, and reduced architectural ambiguity (Swacha & Kulpa, 2023).

The comparison therefore leads to a contextual conclusion. React should be preferred when a project requires extensive customization, strong hiring availability, or integration with a large product ecosystem. Vue should be preferred when clarity, development speed, coherent tooling, and lower initial complexity are more important. The most rational decision is not based on popularity alone, but on project scale, team experience, maintainability requirements, performance expectations, and long-term product strategy (Rathinam, 2023; Pano et al., 2018).

## References

- Gomes, F., Soares, P., & Araújo, A. A. (2023). *Examining the performance implications of design patterns in front-end web development: A preliminary comparative study with React and Vue.js*. In *Proceedings of the 29th Brazilian Symposium on Multimedia and the Web* (pp. 1–10). ACM. <https://doi.org/10.1145/3617023.3617057>
- Loja, A., & Maita, T. (2024). Comparative evaluation of performance efficiency in terms of temporal behavior and resource utilization, according to the ISO/IEC 25010 model, in a web application developed with Angular, React.js, and Vue.js. In G. F. Olmedo Cifuentes, D. G. Arcos Avilés, & H. V. Lara Padilla (Eds.), *Emerging research in intelligent systems* (Lecture Notes in Networks and Systems, Vol. 902, pp. 293–308). Springer. [https://doi.org/10.1007/978-3-031-52255-0\\_21](https://doi.org/10.1007/978-3-031-52255-0_21)
- Ollila, R., Mäkitalo, N., & Mikkonen, T. (2022). Modern web frameworks: A comparison of rendering performance. *Journal of Web Engineering*, 21(3), 789–813. <https://doi.org/10.13052/jwe1540-9589.21311>
- Pano, A., Graziotin, D., & Abrahamsson, P. (2018). Factors and actors leading to the adoption of a JavaScript framework. *Empirical Software Engineering*, 23, 3503–3534. <https://doi.org/10.1007/s10664-018-9613-x>
- Rathinam, S. (2023). Analysis and comparison of different frontend frameworks. In *Applications and Techniques in Information Security* (Communications in Computer and Information Science, Vol. 1804, pp. 243–257). Springer. [https://doi.org/10.1007/978-981-99-2264-2\\_19](https://doi.org/10.1007/978-981-99-2264-2_19)
- React. (2026a). *Built-in React hooks*. <https://react.dev/reference/react/hooks>
- React. (2026b). *Quick start*. <https://react.dev/learn>
- React. (2026c). *Server APIs*. <https://react.dev/reference/react-dom/server>
- Swacha, J., & Kulpa, A. (2023). Evolution of popularity and multiaspectual comparison of widely used web development frameworks. *Electronics*, 12(17), Article 3563. <https://doi.org/10.3390/electronics12173563>
- Vue.js. (2026a). *Reactivity fundamentals*. <https://vuejs.org/guide/essentials/reactivity-fundamentals.html>
- Vue.js. (2026b). *Rendering mechanism*. <https://vuejs.org/guide/extras/rendering-mechanism.html>
- Vue.js. (2026c). *Server-side rendering (SSR)*. <https://vuejs.org/guide/scaling-up/ssr.html>