UDC 004.4’232:004.55

M. A. Tkachenko
V. V. Chopliak

THE REACT ECOSYSTEM AS A FOUNDATION FOR CROSS-PLATFORM APPLICATION DEVELOPMENT

Vinnytsia National Technical University

Анотація

У роботі досліджено архітектурні засади екосистеми React як єдиної платформи для кросплатформної розробки програмних застосунків. Проаналізовано механізм Fiber-рекомпозиції та модель подвійної буферизації рендерингу React, що забезпечує неблокувальне виконання на головному потоці. Розглянуто перехід React Native до New Architecture на базі JavaScript Interface (JSI) та прямих C++-прив'язок, що скорочує затримку виклику нативних методів з 5-10 мс до субмілісекундного рівня. Виконано порівняльний аналіз Electron та Tauri v2 як середовищ виконання для десктопних застосунків. Розглянуто React Server Components як механізм серверного рендерингу на рівні компонентів та пов’язані з ним ризики безпеки десеріалізації Flight-протоколу. Оцінено економічні переваги кросплатформного підходу: скорочення витрат на 30-41% та пришвидшення виходу на ринок на 40-60%.

Ключові слова: React, Fiber, JSI, React Native New Architecture, Electron, Tauri, React Server Components, кросплатформна розробка.

Abstract

This paper examines the architectural foundations of the React ecosystem as a unified platform for cross-platform application development. The Fiber reconciliation engine and its double-buffering rendering model are analyzed, along with React Native’s transition to the New Architecture based on JSI and direct C++ bindings, reducing native invocation latency from 5-10 ms to sub-millisecond. A comparative evaluation of Electron and Tauri v2 as desktop execution environments is presented, highlighting trade-offs in bundle size, memory footprint, and security model. React Server Components are examined as a server-side rendering primitive and the security implications of the Flight deserialization boundary are assessed. The economic advantages of the cross-platform approach are quantified: a 30-41% reduction in development costs and a 40-60% acceleration of time-to-market compared to maintaining separate native codebases.

Keywords: React, Fiber, JSI, React Native New Architecture, Electron, Tauri, React Server Components, cross-platform development.

Introduction

Building software that runs consistently across web browsers, mobile operating systems, and desktop environments has historically demanded separate engineering teams and codebases. React’s core design decision – decoupling the reconciliation engine from the rendering target – established a technical foundation for addressing this fragmentation. Where the legacy stack-based reconciler processed the component tree synchronously and blocked the main thread on deep rendering passes, the Fiber architecture introduced an iterative, pausable execution model constructed as a singly-linked list of work units [1]. This structural shift, combined with the progressive adoption of JSI-based native bindings and server-side rendering primitives, has positioned the React ecosystem as a technically and economically viable foundation for unified multi-platform development.

Research results

The Fiber engine restructures UI rendering as a traversable graph of fiber nodes, each carrying child, sibling, and return pointers that allow the runtime to traverse the component tree iteratively rather than recursively [1]. Rendering work is divided into execution slices of approximately 5-16 ms, yielding control to the host scheduler between slices to preserve responsiveness during complex UI updates [1]. React 19 formalizes concurrent rendering as the default execution mode, maintaining a double-buffered work-in-progress tree that is atomically committed to the live interface once rendering completes [1]. This model enables high-priority updates – such as keyboard or pointer interactions – to interrupt and preempt lower-priority background rendering passes without corrupting the displayed state [1].

On mobile targets, the React Native New Architecture replaces the asynchronous JSON bridge with the JavaScript Interface (JSI), a lightweight C++ API that allows the JavaScript engine to hold direct references to host C++ objects and invoke native methods synchronously [2]. This eliminates bridge serialization overhead: native invocation latency drops from 5-10 ms under the legacy model to sub-millisecond under JSI, a roughly 40× improvement in direct method execution [2]. The Fabric renderer executes layout calculations synchronously via the Yoga 3.0 engine, while TurboModules defer initialization until first access, reducing cold-start times by 25-43% [2]. Shopify's production migration of 300 screens to React Native – underpinned by the New Architecture – resulted in 86% code unification across platforms and the elimination of 1.8 million lines of redundant code, with Android launch times improving by 10% [3]. A complementary optimization developed during this migration, the FlashList component, achieves a 10× throughput improvement over the standard FlatList by recycling native cell views during scroll rather than allocating new objects [3].

For desktop targets, Electron and Tauri v2 represent structurally opposed execution models. Electron bundles a full Chromium instance alongside a Node.js runtime inside every application package, producing installer sizes of 80–200 MB and idle RAM consumption of 150-400 MB, with cold-start latencies between 1.0 s and 5.0 s [4]. Tauri v2 delegates rendering to the OS-native WebView (WebKit on macOS, Edge WebView2 on Windows, WebKitGTK on Linux) and manages system interactions through a Rust backend, yielding installer sizes of 2-10 MB and idle RAM of 30-80 MB [4]. From a security standpoint, Electron's coexistence of Chromium and Node.js creates a path from XSS to remote code execution on the host machine if contextIsolation and nodeIntegration are not explicitly configured; Tauri v2 operates under a deny-by-default capability model in which all OS API access must be explicitly declared per window, minimizing the exploitable surface area of a frontend compromise [4]. Production binary comparisons reinforce the divergence: VS Code (Electron) ships at approximately 350 MB, while a comparable Tauri-based application reaches approximately 8 MB [4].

React Server Components (RSCs) extend the cross-platform model into full-stack execution by splitting rendering between server-only modules – which access databases and file systems without exposing credentials to the client bundle – and client components that carry interactive state and browser hooks [5]. The Flight protocol serializes server-rendered UI structures into a streamable intermediate format, which the client runtime consumes progressively, merging Webpack chunks and rendering the interface without full-page refreshes [5]. A critical engineering constraint of this model is that props crossing the server-to-client boundary are serialized into the Flight payload and are therefore visible in network traffic; sensitive data must be filtered at the boundary before serialization [5]. The deserialization complexity of the Flight engine has also surfaced severe security vulnerabilities: CVE-2025-55182 (CVSS 10.0) allowed unauthenticated remote code execution via the Flight parsing engine, patched in React 19.0.1, 19.1.2, and 19.2.1 [5]. From an economic perspective, shared React codebases across web, mobile, and desktop can reach 83% logic reuse, translating to a 30-41% reduction in upfront development costs and a 40-60% acceleration of time-to-market relative to maintaining separate native teams [6].

Conclusion

The React ecosystem has evolved from a UI library into a structurally coherent platform for multi-target software delivery. The Fiber engine's time-sliced, double-buffered execution model resolves the synchronous thread-blocking limitations of its predecessor, while the concurrent rendering default in React 19 formalizes priority-aware scheduling at the framework level. JSI-based native bindings bring React Native's performance within competitive range of fully native implementations, as evidenced by production results at Shopify. The Electron-Tauri v2 decision remains an active trade-off between rendering consistency and resource efficiency, with Tauri offering substantially lower overhead at the cost of cross-platform rendering uniformity. RSCs introduce server-side rendering composition at the component level, though the Flight deserialization boundary requires disciplined security practices and timely patch management. Across all targets, the shared business logic layer and standardized component model make the React ecosystem an economically and technically justified foundation for cross-platform application development.

REFERENCES

1. React Documentation. URL: https://react.dev/.
2. React Native – New Architecture. URL: https://reactnative.dev/docs/the-new-architecture/landing-page.
3. Shopify Engineering Blog. URL: https://shopify.engineering/shopify-s-new-react-native-architecture.
4. Tauri v2 Documentation. URL: https://v2.tauri.app/.
5. Next.js App Router – React Server Components. URL: https://nextjs.org/docs/app/building-your-application/rendering/server-components.
6. Statista – Most used cross-platform mobile frameworks. URL: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/.
7. Nykyporets S. S. , Kot S. O. , Hadaichuk N. M. , Ibrahimova L. V. , Chopliak V. V. Argumentation patterns in power engineering student research writing: how learners construct causality, evidence, and generalizations. *«Bulletin of Science and Education». Series "Philology".* 2026. № 2(44) 2026. P. 121-139. DOI: https://doi.org/10.52058/2786-6165-2026-2(44)-121-139.
8. Sachaniuk-Kavets'ka N. V., Nykyporets S. S. LLM-based automation for translating mathematical formulae and symbols: challenges and perspectives for technical communication. *Scientific innovations and advanced technologies. Series «Education/Pedagogy»*, 2026. № 3(55). P. 660-677. DOI: https://doi.org/10.52058/2786-5274-2026-3(55)-660-677.
9. Kravchenko K., Ketsyk-Zinchenko U., Suduk I., Nykyporets S., Cherednychenko V. Effectiveness of online platforms in developing language skills of higher education students. *Revista Eduweb*. 2025. 19(3). P. 303-314. DOI: https://doi.org/10.46502/issn.1856-7576/2025.19.03.19.

Ткаченко Максим Андрійович – студент групи 5ПІ-25б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: phexuss@gmail.com.

Науковий керівник: ***Чопляк Вікторія Володимирівна*** – викладач англійської мови, кафедра іноземних мов, Вінницький національний технічний університет, м. Вінниця, e-mail: nikavnuchkova@gmail.com.

Maksym A. Tkachenko – a student of 5SE-25b group, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: phexuss@gmail.com.

Scientific supervisor***: Victoria V. Chopliak*** – teacher of English, Foreign Languages Department, Vinnytsia National Technical University, Vinnytsia, e-mail: nikavnuchkova@gmail.com.