DOI: 10.34185/1991-7848.itmm.2026.01.052

РОЛЬ JAVASCRIPT INTERFACE У МОДУЛЬНІЙ ОРГАНІЗАЦІЇ ТА РЕНДЕРИНГУ REACT NATIVE

Перемітько М.В.[1] [ORCID], Надригайло Т.Ж.[2] [ORCID]

[1]*Дніпровський державний технічний університет, PhD, Україна*

[2]*Дніпровський державний технічний університет, к.т.н., доцент, Україна*

Анотація. *У роботі розглянуто JavaScript Interface (JSI) як базовий механізм нової архітектури React Native та проаналізовано його роль у функціонуванні Turbo Modules і рендерингової підсистеми. Основну увагу зосереджено на принципі роботи JSI як інтерфейсного шару між JavaScript runtime і нативним середовищем, що забезпечує більш цілісну модель виконання та зменшує накладні витрати під час міжрівневої взаємодії. Показано, що Turbo Modules у цій архітектурі реалізують типізовану та відкладену модель підключення нативної функціональності, яка може спиратися як на спільний C++ рівень, так і на платформозалежний код Android та iOS. Також окреслено інфраструктурний вплив JSI на рендеринг у React Native через підвищення узгодженості обміну даними між JavaScript, C++-ядром і нативним представленням інтерфейсу.*

Ключові слова. *React Native, JavaScript Interface, JSI, Turbo Modules, Fabric, рендеринг, нативні модулі, C++.*

У сучасній архітектурі React Native підсистеми JSI, Turbo Modules і Fabric утворюють єдину модель виконання, у якій взаємодія JavaScript коду з платформеним середовищем, життєвий цикл модулів і побудова інтерфейсу розглядаються як взаємопов’язані процеси. У цій моделі JSI виконує роль інтерфейсного шару між JavaScript runtime та нативними абстракціями, Turbo Modules задають структуру й спосіб експонування прикладної функціональності, а Fabric забезпечує нову схему рендерингу. Такий підхід відображає еволюцію React Native у напрямі більш інтегрованої та продуктивної архітектури мобільних застосунків.

З функціонального погляду JSI є C++ орієнтованим інтерфейсом, який забезпечує уніфіковану модель взаємодії між JavaScript середовищем і нативним кодом незалежно від конкретного рушія. Його сутність полягає в інтеграції нативних функцій і об’єктів у єдиний контекст без використання

проміжних механізмів передачі повідомлень. Це дозволяє суттєво зменшити накладні витрати на перетворення даних і скоротити затримки при виконанні операцій. Усунення серіалізації та пряме зв'язування між JavaScript і нативним рівнем є ключовими чинниками зниження затримки та підвищення ефективності виконання [1].

У межах фреймворку JSI функціонує у зв'язці з типізованою моделлю опису нативної функціональності. Turbo Modules базуються на формалізованих специфікаціях, які визначають доступні операції та структури даних, після чого генеруються узгоджені інтерфейси для різних платформ. Такий підхід дозволяє уникнути неузгодженостей між JavaScript і нативною частиною та підвищує надійність взаємодії між шарами системи. Крім того, це відповідає загальній тенденції розвитку React Native, де значна увага приділяється не лише продуктивності, а й формалізації архітектури та зменшенню кількості ручного проміжного коду [2].

Turbo Modules визначають нову модель життєвого циклу нативних модулів, яка базується на принципі відкладеної ініціалізації. Модулі створюються лише в момент їх фактичного використання, що дозволяє зменшити початкове навантаження на систему та оптимізувати використання ресурсів. Така стратегія особливо важлива для великих застосунків, де значна частина функціональності використовується нерівномірно. Подібна організація сприяє зменшенню часу запуску та обсягу використаної пам'яті, що підтверджує ефективність даного підходу [2].

З погляду реалізації Turbo Modules підтримують різні рівні нативної інтеграції. Платформонезалежна логіка може бути реалізована у вигляді спільного C++ ядра, що дозволяє уникнути дублювання коду для різних операційних систем. Водночас функціональність, яка потребує доступу до специфічних можливостей платформи, реалізується на рівні нативного коду Android або iOS. Така багаторівнева модель забезпечує баланс між кросплатформністю та ефективною інтеграцією з платформеними API.

Вплив JSI на рендеринг має інфраструктурний характер і проявляється через взаємодію з новою системою рендерингу Fabric. У цій системі побудова

інтерфейсу здійснюється у вигляді багатофазного процесу, що включає формування структури компонентів, обчислення їхнього стану та застосування змін до нативного представлення. Використання спільного C++ ядра та потокобезпечних структур даних дозволяє виконувати частину цих операцій паралельно, що підвищує ефективність рендерингового конвеєра. У цьому контексті JSI зменшує витрати на взаємодію між JavaScript і C++ рівнем, що опосередковано впливає на швидкість обробки оновлень інтерфейсу та стабільність відображення.

Важливим аспектом є підтримка більш синхронної взаємодії між логікою застосунку та рендерингом, що дозволяє уникати проміжних неконсистентних станів інтерфейсу. Це особливо актуально для складних UI сценаріїв, де зміни в одному компоненті впливають на інші. Завдяки JSI забезпечується швидший обмін даними між рівнями системи, що сприяє більш передбачуваному та стабільному оновленню інтерфейсу.

Емпіричні результати підтверджують ефективність описаної архітектури. Досліджено, що використання нової архітектури React Native призводить до зменшення часу запуску застосунку, зниження використання оперативної пам’яті та скорочення затримки взаємодії між JavaScript і нативним рівнем . Також спостерігається підвищення плавності інтерфейсу, що проявляється у збільшенні частоти кадрів під час взаємодії з UI [3].

Разом із тим, продуктивність React Native застосунків залежить не лише від архітектури взаємодії, а й від вибору JavaScript рушія. Результати порівняння різних рушіїв за такими показниками, як час взаємодії, використання пам’яті, швидкість виконання та продуктивність рендерингу, свідчать, що найбільш ефективні конфігурації досягаються при поєднанні оптимізованого рушія з сучасною архітектурою, заснованою на JSI.

Висновки

Отже, впровадження JavaScript Interface у React Native забезпечило суттєві переваги на рівні архітектури та виконання, зокрема зменшення накладних витрат на взаємодію між JavaScript і нативним кодом, підвищення ефективності модульної системи, оптимізацію використання ресурсів і

створення передумов для більш стабільного рендерингу інтерфейсу. У поєднанні з Turbo Modules і Fabric JSI формує більш цілісну та продуктивну модель роботи фреймворку, тому важливим завданням залишається міграція на нову архітектуру вже існуючих застосунків і бібліотек, оскільки саме вона забезпечує повноцінне використання сучасних можливостей React Native та визначає подальшу технічну актуальність і ефективність екосистеми.

ЛІТЕРАТУРА / REFERENCE

1. Sayed S. A. N. A. Performance Optimization and Architectural Evolution in React Native Mobile Applications. – 2025. DOI: 10.5281/zenodo.17519381.
2. Garg P., Yadav B., Gupta S., Gupta B. Performance Analysis and Optimization of Cross Platform Application Development Using React Native. – 2023. P. 559–567. DOI: 10.1007/978-981-19-9304-6_51.
3. Kurant L., Bylina J. Impact of Selected JavaScript Engines on the Performance of Mobile Hybrid Applications. Journal of Software: Evolution and Process. – 2026. Vol. 38. DOI: 10.1002/smr.70086.

THE ROLE OF JAVASCRIPT INTERFACE IN THE MODULAR ORGANIZATION AND RENDERING OF REACT NATIVE

Mykhailo Peremitko, Tetiana Nadryhailo

Abstract. *The paper examines the JavaScript Interface (JSI) as a core mechanism of the new React Native architecture and analyzes its role in the operation of Turbo Modules and the rendering subsystem. The main focus is placed on the working principle of JSI as an interface layer between the JavaScript runtime and the native environment, providing a more coherent execution model and reducing overhead in cross-layer interaction. It is shown that Turbo Modules implement a typed and lazily initialized model of native functionality exposure, which may rely on a shared C++ layer as well as platform-specific Android and iOS code. The paper also outlines the infrastructural impact of JSI on rendering in React Native through improved consistency of data exchange between JavaScript, the C++ core, and the native UI representation. The role of JSI is generalized as a key instrument of contemporary architectural organization in React Native.*

Keywords. *React Native, JavaScript Interface, JSI, Turbo Modules, Fabric, rendering, native modules, C++.*