IJIRCCE

e-ISSN: 2320-9801 | p-ISSN: 2320-9798

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

Volume 11, Issue 3, March 2023

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.379

International Journal of Innovative Research in Computer and Communication Engineering

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 |

|| Volume 11, Issue 3, March 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1103002 |

Cross-Platform Mobile Development: Comparing React Native and Flutter, and Accessibility in React Native

Vishnuvardhan Reddy Goli

Lead Mobile Developer, ATT Inc (Innovative Intelligent Solutions LLC), Texas, USA

ABSTRACT: Modern application development requires developers to build multiple platform apps through cross-platform mobile development, which lets them create applications using the same codebase. This mobile application development area comprises two main frameworks, React Native and Flutter, which provide separate benefits and execution costs. As Facebook's framework, React Native connects JavaScript to native code structures, allowing developers to reuse web development expertise. Still, it is challenged due to speed degradation caused by delayed program exchange. The Skia graphics engine in Flutter by Google operates without a bridge for UI rendering, providing superior performance and seamless animations.

This paper compares React Native and Flutter regarding architecture, performance, and accessibility. Research shows that Flutter provides superior user interface rendering and memory management performance and has integrated accessibility functions that bolster accessibility standards and self-facilitated testing capabilities. The well-developed JavaScript ecosystem of React Native combined with its broad developer base makes it an ideal option for web developers starting mobile development work. The findings indicate React Native works better for JavaScript-based applications, yet Flutter performs outstandingly for high-performance and accessible mobile applications.

KEYWORDS: Cross-platform Mobile Development, Reactive Native, Flutter, Performance, Accessibility

I. INTRODUCTION

The development of mobile technology has proliferated rapidly, creating an upsurge in mobile frameworks necessary for developers to create apps for other operating systems using the same codebase. In the past, mobile applications were built natively in Swift for iOS and Java/Kotlin for Android, which guaranteed good performance but necessitated split code support. To overcome these challenges, cross-platform frameworks like React Native and Flutter have risen. These platforms provide code reusability and speed up the development time without compromising the near-native performance.

In 2015, Facebook introduced React Native, a JavaScript-based framework which enables developers to build mobile applications with the help of React.js principles. However, such a design limits the performance of native UI components because of its bridge-based architecture, and it cannot handle a complex update to the UI and animations accordingly [1]. In contrast, Google launched its Flutter in 2018, which does not require a bridge since it uses a Skia graphics engine to render directly and more efficiently [2].

Accessibility is a major factor in the development of mobile applications beyond performance. React Native covers screen readers, font scaling and high contrast modes, but manual configurations are more required than Flutter, which contains the accessibility properties integrated [2]. This paper undertakes a comparative analysis of React Native vs. Flutter, considering architecture, performance and accessibility to check out their feasibility of being employed for cross-platform mobile development.

II. COMPARATIVE ANALYSIS OF REACT NATIVE AND FLUTTER

Development Architecture

The mobile development strategies of React Native and Flutter differ from one another, thus affecting the speed of operation and user experience, as well as their sustainment capabilities. The development processes of the React Native function use bridge-based elements that split JavaScript execution into a distinct thread to manage native components

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 |

|| Volume 11, Issue 3, March 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1103002 |

through asynchronous functionalities efficiently [3]. The familiarity of JavaScript among web developers gives React Native broad acceptance because it enables them to use their existing React.js skills for mobile development. Latency problems occur from bridge-based communication in applications that need advanced UI rendering features with many state updates [3]. Each user interface operation in React Native requires processing through its JavaScript-to-native bridge, which causes performance degradation, thus resulting in delayed frame performance [4].

Flutter accomplishes this improvement by eliminating bridges from the development process. Flutter delivers UI elements using the Skia graphics engine; thus, it produces more consistent and smooth animations across multiple platforms [5]. The unified design of UI components leads to complete visual similarity between iOS and Android platforms, which minimizes differences between the operating systems. AOT compilation in Flutter translates Dart code to native machine code before execution, resulting in substantially better performance at start-up and while the application runs [6]. React Native employs just-in-time compilation for development while Ahead-of-time compilation is used only for production, which results in poorer performance than Flutter provides, according to [5].

Flutter delivers better performance than React Native, yet it produces sizeable applications that exceed React Native's. A direct outcome of Flutter's bundled rendering engine during compilation is that its applications become much heavier than React Native software, which impacts download times and storage capacity on limited devices [5]. The native platform UI capabilities of React Native applications operate dynamically, but Flutter needs developers to embed all graphical resources and components in application packages, which expands the program size [7].

Performance and Memory Utilization

Freedom of cross-platform frameworks from execution issues is the primary factor in choosing the optimal mobile development solution. Research demonstrates that Flutter delivers superior results to React Native regarding UI rendering speed and CPU efficiency. The experimental results presented by [4] demonstrated that when applications use continuous scrolling, React Native showed image pixel breakdowns that reduced frame rates, but Flutter maintained a constant frame rate of 60 FPS on various devices. The asynchronous bridge in React Native's JavaScript execution model creates processing delays that result in increased rendering latency.

Memory usage separates the two frameworks. [7] demonstrated that Ready Native applications used about 20% more memory than Flutter applications while operating at peak performance. JavaScript's garbage collection process creates occasional delays during memory management operations while simultaneously building a higher CPU load. Time-efficient memory administration in Flutter comes from its auto-garbage collection through Dart, which immediately releases unused memory space for improved performance and prolonged battery duration, according to [8].

The performance of applications particularly depends on how cameras and hardware function in the system. [9] performed CPU load tests on repeated camera usage, which showed that React Native used a higher CPU while Flutter needed only a lower CPU, so Flutter proved best at managing real-time hardware connections. The native compilation of Flutter eliminates processing overhead while directly executing system-level commands because it bypasses the intermediary bridge between the user interface and hardware components.

Ecosystem and Developer Adoption

The success rate of development structures depends on technical abilities combined with widespread user backing, mature ecosystem development, and available programmer support. React Native benefits from an extensive JavaScript ecosystem because it provides developers who understand web technologies easy access to the framework. Large-scale industrial activities at Facebook, Instagram, Airbnb, and Uber Eats rely on React Native to show commercial viability [4]. The extensive third-party resources and community plugins from JavaScript's widespread adoption enable developers to reduce development time and increase overall system flexibility [4].

The native functionality components in React Native require developers to depend on third-party modules, which introduces implementation hurdles. Platform-specific features within the framework need customized bridging solutions, which results in compatibility problems between different versions of Android and iOS [10]. Development complexities rise, and developers face reduced code maintainability because they need different configurations for operating systems [10].

The primarily newer development tool, Flutter, is now gaining significant market adoption among companies like Alibaba, Tencent, and BMW, which have implemented it in their development methodologies. Due to its extensive built-

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 |

|| Volume 11, Issue 3, March 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1103002 |

in widget library, Flutter enables developers to design user interfaces that need no additional third-party libraries. The Dart programming language adopted by Flutter faces challenges because it lacks an extensive developer base compared to JavaScript programs, thus increasing the difficulty for new developers [4].

Google has invested in developing Flutter to establish itself as a permanent solution for cross-platform development that now extends to web and desktop applications. Flutter gains momentum in the developer ecosystem through the combination of smaller community size and its integrated development environment and hot reload feature and performance optimizations [11].

III. ACCESSIBILITY IN REACT NATIVE

Screen Reader Support

Mobile applications must provide accessible features because this enables users with visual impairments to have inclusive interactions. Screen readers complete their essential task by using voice output to read aloud the different components of the user interface, which enables visually impaired users to understand applications through audio feedback. The screen readers VoiceOver for iOS alongside TalkBack for Android receive support from React Native for describing app components through text announcements and guidance navigation [12]. React Native provides screen reader functionality through manual developer setup of accessibility labels, roles, and hints to achieve proper screen reader results [12].

Developers must actively provide accessible user interface elements in React Native applications when dealing with custom components and third-party libraries. React Native lacks the default accessibility features provided by Flutter since the platform does not automatically assign accessibility roles for custom user interface components. Technical staff must write declarations explicitly describing button elements with accessible={true} alongside the description in accessibilityLabel="Button Name" to enable screen readers to detect all accessible materials.

The way React Native deals with changes that occur dynamically within the application poses a substantial challenge. Apps incorporating live notifications, chat updates, or regular UI changes need to activate screen reader notifications that provide users with real-time update announcements. The AccessibilityInfo API requires manual implementation for updating announcements as native environments provide this feature automatically, thus increasing development complexity [4]. Missed real-time information represents a barrier to accessibility in React Native applications when user implementation fails to meet proper standards [4].

Dynamic Font Scaling and High-Contrast Modes

User interfaces must provide text accessibility features to ensure legibility for people with vision impairments, dyslexics, and visual disabilities. The text elements in React Native automatically resize through dynamic font scaling based on system preferences. Text scaling support within React Native operates inconsistently among components made by developers and third-party vendors since developers must manually activate this feature.

According to [5], consistent font scaling is an issue for React Native applications. This leads to debugging requirements to avoid text readability problems when users modify their device font size settings. Text scaling operates automatically for all widgets within Flutter, thus creating a solution that is both architect-friendly and access-ready from the basic installation [5].

In addition to high-contrast modes, the system is equally important because it helps users with color blindness or low vision view digital elements more effectively. The detection of high-contrast mode occurs automatically in React Native across iOS and Android devices, but the UI elements lack built-in features for accessibility contrast guideline compliance [1]. Developers need to use style adjustments for configuring color contrast settings, thus requiring them to perform an additional step to maintain Web Content Accessibility Guidelines (WCAG) compliance, according to [4]. The high-contrast mode features of Flutter operate throughout the framework core without needing any styling adjustments, according to [13].

Access barriers occur with non-text components, including graphical elements and buttons throughout React Native applications. The touch target sizing and contrast configuration for React Native components needs developer manual input because the framework does not perform automatic element scaling as Flutter does. The usability suffers when users with motor impairments or visual challenges try to interact with small touch targets since developers need to apply additional modifications [7].

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 |

|| Volume 11, Issue 3, March 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1103002 |

Gesture Navigation and Keyboard Accessibility

Users who depend on gestures, voice commands, and keyboard inputs need these methods to handle application operations. The gesture functionality in React Native supports navigation and keyboard accessibility through platform-based implementation that requires developer-operated navigation control configurations [1]. The implementation of touch gestures demands explicit definitions from React Native applications since gestures do not integrate deeply with operating systems at the native development level [1].

Keyboard accessibility is another concern [7] observed in focus management problems, specifically in React Native applications, when users use keyboards to navigate, leading to unexpected focus shifts or missing guidance about navigation. Developers must create custom focus management functions for React Native applications instead of using built-in Flutter features mainly because this development method consumes time and introduces potential errors [4]. Users with limited dexterity should have access to large touch targets that respond properly. Developers who use React Native need to handle button enlargement and interactive element modification themselves to improve user experience. Flutter's adaptive UI scaling makes automatic changes to buttons and UI components possible for full-text size accessibility [5].

Accessibility Testing and Compliance

Mobile applications need human and software-based accessibility testing protocols to comply with accessibility requirements. According to [6], the accessibility testing framework of React Native includes react-native-accessibility-library but does not include built-in automatic validation. Developers using React Native applications need third-party tools like Axe to spot accessibility issues, including poorly contrasting elements and labels that are absent or incorrectly ordered in focus [6]. The accessibility testing solution in Flutter features built-in audit tools, such as the Flutter accessibility inspector, which delivers immediate accessibility feedback [4]. The development process for Flutter supports automated accessibility checks directly in its environment, thus improving the detection of accessibility problems [2].

Summary of Accessibility Challenges

React Native is dependent on native modules and manual configurations, and hence, it affects the essential accessibility features of React Native through creating a platform-specific variation. Developers must manually define the instructions for accessibility roles, text features, contrast modes, and focus management, which requires extended development time and is extremely difficult. Due to natively accessible properties in Flutter, each widget system has such properties, making it easier for developers to implement them and create better conformity framework standards [2]. Flutter system provides built-in accessibility testing tools that allow developers to verify automatic compliance. Besides, React Native requires a team of developers working together to meet complete accessibility compliance or use third-party solutions and have access to manual bug debugging simultaneously [1].

IV. CONCLUSION

React Native and Flutter have become essential platforms in mobile app development. Research undertaken in this paper explored the differences and similarities between the platforms. The analysis reveals that React Native allows developers to deploy JavaScript when building native code through the JavaScript-to-naïve bridge principle embedded in the platform. Nonetheless, this approach affects smooth performance during intricate user interface interactions. In contrast, Flutter deploys a different approach, the Skia, which removes the need for a bridge, allowing smooth animations, low memory usage, and high execution speed. Nevertheless, the larger application size of Flutter becomes a potential issue when developers need to focus on creating applications for mobile devices with limited storage space

Accessibility is another crucial factor. Flutter provides native accessibility features through its framework, which automates many such adjustments, but React Native needs manual developer implementation for accessibility support. The development of accessibility features in React Native requires manual code definitions for specific roles and tags, which differs from Flutter because the framework integrates accessibility properties directly into its widgets. Flutter integrates built-in accessibility testing features, which differ from React Native since they use third-party detection tools called Axe for React Native.

The selection between React Native and Flutter depends on the development needs of the developers' projects. Mobile developers moving from web applications will find a suitable transition with React Native because of its established

| e-ISSN: 2320-9801, p-ISSN: 2320-9798| www.ijircce.com | |Impact Factor: 8.379 |

|| Volume 11, Issue 3, March 2023 ||

| DOI: 10.15680/IJIRCCE.2023.1103002 |

JavaScript framework, but Flutter delivers enhanced interface consistency along with speed enhancements and inclusion features for applications needing increased touch sensitivity and easy access.

REFERENCES

[1] M. Singh and S. G, "Comparative Analysis of Hybrid Mobile App Development Frameworks," *International Journal of Soft Computing and Engineering*, vol. 10, no. 6, pp. 21–26, Jul. 2021, doi: https://doi.org/10.35940/ijsce.f3518.0710621.
[2] S. A. Bhagat, "Review on Mobile Application Development Based on Flutter Platform," *International Journal for Research in Applied Science and Engineering Technology*, vol. 10, no. 1, pp. 803–809, Jan. 2022, doi: https://doi.org/10.22214/ijraset.2022.39920.
[3] N. Shevtsiv, D. Shvets, and N. Karabut, "Prospects for Using React Native for Developing Cross-platform Mobile Applications," *Central Ukrainian Scientific Bulletin. Technical Sciences*, no. 2(33), pp. 208–213, Dec. 2019, doi: https://doi.org/10.32515/2664-262x.2019.2(33).208-213.
[4] M. Mahendra and B. Anggorojati, "Evaluating the performance of Android based Cross-Platform App Development Frameworks," *2020 the 6th International Conference on Communication and Information Processing*, Nov. 2020, doi: https://doi.org/10.1145/3442555.3442561.
[5] E. GÜLCÜOĞLU, A. B. USTUN, and N. SEYHAN, "Comparison of Flutter and React Native Platforms," *Journal of Internet Applications and Management*, Dec. 2021, doi: https://doi.org/10.34231/iuyd.888243.
[6] A. Biørn-Hansen, T.-M. Grønli, G. Ghinea, and S. Alouneh, "An Empirical Study of Cross-Platform Mobile Development in Industry," *Wireless Communications and Mobile Computing*, vol. 2019, pp. 1–12, Jan. 2019, doi: https://doi.org/10.1155/2019/5743892.
[7] A. Fentaw, "Cross platform mobile application development: a comparison study of React Native Vs Flutter," 2020. Available: https://jyx.jyu.fi/bitstream/handle/123456789/70969/1/URN%3ANBN%3Afi%3Ajyu-202006295155.pdf
[8] P. Rao, B. Pavan, A. Srivastava, K. Venkata Amani, and A. Sharma, "Distinction of Mobile Frameworks: Flutter Vs Native Apps," Peer-Reviewed, Open Access, 2022. Available: https://www.irjmets.com/uploadedfiles/paper/issue_6_june_2022/26317/final/fin_irjmets1655531771.pdf
[9] D. Białkowski and J. Smołka, "Evaluation of Flutter framework time efficiency in context of user interface tasks," *Journal of Computer Sciences Institute*, vol. 25, pp. 309–314, Dec. 2022, doi: https://doi.org/10.35784/jcsi.3007.
[10] M. Singh and S. G, "Comparative Analysis of Hybrid Mobile App Development Frameworks," *International Journal of Soft Computing and Engineering*, vol. 10, no. 6, pp. 21–26, Jul. 2021, doi: https://doi.org/10.35940/ijsce.f3518.0710621.
[11] D. Mota and R. Martinho, "An Approach to Assess the Performance of Mobile Applications: A Case Study of Multiplatform Development Frameworks," *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2021, doi: https://doi.org/10.5220/0010497401500157.
[12] S. Mascetti, M. Ducci, N. Cantù, P. Pecis, and D. Ahmetovic, "Developing Accessible Mobile Applications with Cross-Platform Development Frameworks," *arXiv (Cornell University)*, Jan. 2020, doi: https://doi.org/10.48550/arxiv.2005.06875.
[13] I. Malnati, F. Fabio, D. Candidate, and Palumbo, "POLITECNICO DI TORINO The Flutter Framework: Analysis in a Mobile Enterprise Environment," 2021. Available: https://webthesis.biblio.polito.it/19111/1/tesi.pdf

INNO SPACE
SJIF Scientific Journal Impact Factor

Impact Factor: 8.379

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details