International Journal of Science and Engineering Applications
Volume 13-Issue 07, 48 - 51, 2024, ISSN:- 2319 - 7560
DOI:10.7753/IJSEA1307.1013

Comparative Functional and Performance Analysis of Apache Cordova and Android UI Development Tools

Anton Novikau
Head of Mobile Development
Talaera, 28 Liberty Street, 6th Floor
New York, USA

Abstract: mobile app usage is at an all-time high. According to some data, users spend an average of 2 hours and 20 minutes a day interacting with a mobile application. For the savvy business owner, this is a great opportunity to get in front of clients with entertaining, useful, high-quality applications. Given the wide range of available devices and operating systems on the market, it is not surprising that entrepreneurs and companies looking to enter the mobile app market are interested in creating apps that function on any device.
Since cross-platform development takes a long time and requires different knowledge (Swift/iOS or Java/Kotlin), cross-platform solutions have been implemented over the years. These include Unity, Ionic, Cordova and React Native. These solutions are ideal for developing and running applications on two platforms simultaneously.
In this article, we will see how to use Apache Cordova in the Android development environment compared to native tools.

Keywords: android programming, cross-platform efficiency, mobile development tests, apache cordova, java, kotlin.

1. INTRODUCTION

1.1 Apache Cordova

Mobile app development is rapidly gaining popularity, and nowadays, smartphone apps are often created before websites. With the improvement in mobile technology, developers now have access to a vast arsenal of app development tools tailored to a variety of usage scenarios.
Apache Cordova opens up new possibilities for mobile app developers, especially when working with the Android platform compared to traditional native tools. The main advantage of Cordova lies in its ability to simplify development across platforms, allowing the same code base to be used to create applications on different platforms, which significantly reduces development time and cost [3].
Apache Cordova is a robust open-source platform that enables developers to create cross-platform applications using widely adopted web technologies such as HTML5, CSS3, and JavaScript. A key feature of Cordova is its ability to transform standard web code into a format compatible with mobile devices. This transformation is achieved by providing access to device-specific functionalities, such as the camera or file system, through specialized Application Programming Interfaces (APIs).
Cordova-based applications are hybrid in nature, blending the elements of both web and mobile applications. They are packaged and distributed as native applications through app stores, while still maintaining a core web-based architecture. This hybrid model allows developers to leverage their existing web development skills to create mobile applications.

Beyond mobile platforms, Cordova also facilitates the development of desktop applications for Windows and macOS. This capability is achieved using web technologies akin to those employed by the Electron framework, thus extending the versatility of Cordova to desktop environments.
Despite its advantages, utilizing Cordova requires a strategic approach to tool selection and additional effort to optimize user interactions. This is crucial to ensure that the performance and user experience of Cordova-based applications meet the standards expected of native applications.
A significant feature of Cordova is its plugin architecture, which allows access to local device functions. Plugins offer JavaScript APIs that interface with native components, enabling developers to interact with device features directly from their web code. Developing with Cordova involves creating custom plugins and ensuring the appropriate Software Development Kits (SDKs) for the target platforms are pre-installed. Once set up, these plugins can be invoked through JavaScript to integrate native functionalities seamlessly into the web view.
Cordova presents an effective solution for developers aiming to extend their web development expertise to mobile and desktop application development. However, the successful implementation of Cordova projects hinges on careful planning and the proficient use of its plugin system to harness native device capabilities.

1.2 Native development

When high performance is a paramount requirement, native development remains the superior choice for application development. Native applications exploit the full capabilities of specific operating systems and device hardware, resulting in optimized performance and responsiveness. This is particularly critical in scenarios involving intensive data processing or complex graphics rendering, where frameworks like Apache Cordova might encounter performance bottlenecks and latency issues [4].
Native development environments, such as Android's View system and Jetpack Compose, provide direct and efficient access to system APIs. This direct access facilitates the creation of applications with superior performance and deeper integration with the underlying operating system.
The Android View system, a mature framework, employs XML for interface design and Java or Kotlin for application logic. This traditional approach is well-supported by a large developer community, making it a reliable and robust choice for many projects. The strong community support and extensive documentation available for Android View contribute to its stability and reliability, particularly in complex applications

requiring fine-grained control over the user interface and system interactions.

Jetpack Compose introduces a modern, declarative paradigm for user interface (UI) development. By utilizing Kotlin, it allows developers to construct UIs with more concise and expressive code. This declarative approach simplifies the development process, reduces boilerplate code, and enhances the maintainability of the application. Jetpack Compose's streamlined workflow not only accelerates development but also facilitates easier updates and modifications, which are crucial for maintaining long-term application quality [5].

In summary, while Apache Cordova is a viable option for many applications, especially those leveraging web technologies, native development tools like Android View and Jetpack Compose offer unparalleled performance and integration capabilities. These tools are essential for creating high-performance applications that demand rigorous interaction with the operating system and hardware resources.

2. METHODOLOGY

The aim of this research is to compare the capabilities of Apache Cordova and the Android Software Development Kit (SDK), focusing on both performance and functionality. This comparison seeks to highlight the strengths and limitations of each platform in real-world application development scenarios.

To evaluate performance, two identical applications were developed: one as a native Android application using the Jetpack Compose UI library, and the other as a Cordova-based application built with HTML, CSS, and JavaScript, compiled through Apache Cordova. Both applications were designed to render large datasets with asynchronous loading, ensuring that the user interface operations were carefully synchronized to maintain consistency between the two.

Performance testing involved a detailed and methodical approach using Android Studio's profiling tools. The applications were deployed on a Samsung Galaxy S21 mobile phone. The testing scenario included interactions that simulated typical user activities, such as browsing through extensive data lists and performing standard user actions. During these interactions, CPU resource utilization was meticulously monitored and recorded using Android Studio's built-in profiling features.

The recorded data provided a comprehensive view of CPU usage, capturing how each application managed system resources under similar operational conditions. This allowed for a direct comparison of their performance characteristics. The data collected were then subjected to a detailed analysis to identify trends and differences in CPU consumption between the two platforms. The analysis was complemented by graphical representations that visually depicted the CPU usage for both applications, facilitating a clear and comparative understanding of their performance profiles.

In addition to performance evaluation, this research includes a functional comparison between Apache Cordova and the Android Software Development Kit (SDK). The objective is to assess how each platform supports essential application functionalities and to understand the extent to which they enable or limit specific capabilities.

For the functional comparison, we developed two versions of a sample application. One was created as a native Android application using the Jetpack Compose UI framework, while the other was a Cordova application built with HTML, CSS, and JavaScript. Both applications were designed to perform the following tasks:

- Accessing device hardware features such as the camera and GPS.
- Managing local and remote data storage.
- Integrating with various external APIs.
- Handling user authentication and security features.
- Supporting offline functionality and synchronization.

The choice of these functionalities was guided by their common use in modern mobile applications, providing a robust basis for comparison.

The functional capabilities of both applications were evaluated on a Samsung Galaxy S21. The evaluation involved a series of tests designed to explore how each platform handles the aforementioned functionality. Results are evaluated using following metrics: complexity of working with device API, performance, platform independence, development complexity and development cost.

Each functionality was meticulously documented, noting the implementation process, any challenges encountered, and the overall user experience. Special attention was given to the following aspects:

- Ease of Development: How straightforward and developer-friendly the implementation process was on each platform.
- User Experience: The quality and responsiveness of the user experience, including any differences in how users interact with and perceive the application features.
- Platform Limitations: Any significant limitations or constraints imposed by either platform that affected the implementation or performance of the features.

The findings were systematically compared to highlight the strengths and weaknesses of Apache Cordova and the Android SDK. This comprehensive functional comparison aimed to provide actionable insights for developers choosing between these platforms for their mobile application projects.

3. EXPERIMENTAL RESULT

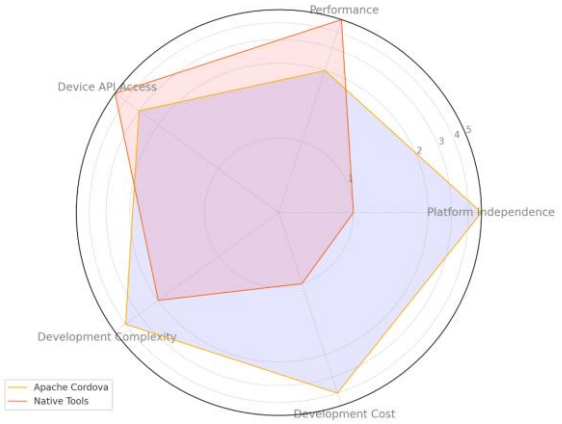

Figure 1 - Functional comparison between Apache Cordova and Android SDK

The functional comparison reveals that the choice between Apache Cordova and native development tools is significantly influenced by specific project requirements, especially when considering the trade-offs between performance, cost, and development flexibility. Apache Cordova excels in several functional areas:

- Development Cost: Cordova reduces costs by allowing developers to use existing web technologies (HTML, CSS, and JavaScript), streamlining the process of creating cross-platform applications.
- Platform Independence: Cordova's ability to compile code for multiple platforms from a single codebase offers significant advantages in terms of deployment across different operating systems.

International Journal of Science and Engineering Applications
Volume 13-Issue 07, 48 - 51, 2024, ISSN:- 2319 - 7560
DOI:10.7753/IJSEA1307.1013

- Complexity of Development: The simplicity and familiarity of web technologies make Cordova a less complex solution for developers, particularly those already proficient in these technologies.

Conversely, the Android SDK, with its native development tools, stands out in terms of:

- Device API Access: Native tools provide direct and efficient access to device-specific APIs, allowing deeper integration and use of hardware features.
- Rich Functionality: Native development frameworks like Jetpack Compose offer robust support for complex user interfaces and extensive platform-specific functionalities.

These functional differences are illustrated in Figure 1, which demonstrates the comparative strengths of Apache Cordova and the Android SDK based on the conducted experiment.

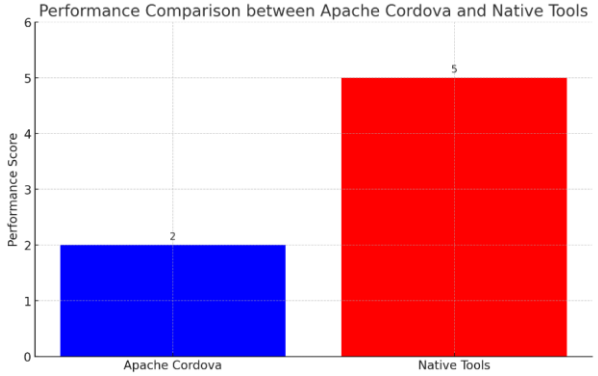

Figure 2 - performance comparison between Apache Cordova and native tools

The performance evaluation of the native Android application developed using the Android SDK and the Apache Cordova application revealed significant disparities favoring the native approach. The native application consistently outperformed the Cordova application in key performance metrics, highlighting its superior efficiency and responsiveness.

One of the most striking differences observed was in CPU resource utilization. The native Android application, benefiting from more straightforward access to the system's hardware and operating system APIs, managed to execute tasks with significantly lower CPU usage. This efficiency is largely due to the absence of additional abstraction layers, which are inherent in the Cordova framework. Cordova's reliance on web technologies (HTML, CSS, and JavaScript) introduces an extra layer that interprets and compiles these technologies into a format executable on mobile devices. This process inherently consumes more CPU resources.

In terms of user interface (UI) performance, the native Android application demonstrated superior responsiveness and fluidity. This was particularly evident during tasks involving complex UI rendering and asynchronous data loading. The native application maintained smooth interactions and quick response times, attributes critical for delivering a high-quality user experience.

Conversely, the Cordova application, due to its hybrid nature, struggled with latency issues during intensive UI operations. The additional processing overhead required to render web-based components within a native container resulted in noticeable lags and slower UI updates. This performance lag is especially problematic in applications that require real-time data processing and interactive graphics, where even slight delays can detract from user satisfaction.

Figure 2 graph illustrating the performance comparison between the two approaches further underscores these findings. It vividly shows that the native Android application, developed using the Android SDK, consistently achieves better performance metrics across all tested scenarios. This includes lower CPU usage, faster UI rendering, and more efficient handling of complex tasks

The results of this performance comparison unequivocally suggest that for projects where high performance is crucial, the Android SDK is the superior choice over Apache Cordova. Native development provides unparalleled efficiency, responsiveness, and capability to exploit the full potential of device hardware and system resources. Apache Cordova, while beneficial for its cross-platform development ease and cost efficiency, cannot match the performance levels achievable through native development tools. Therefore, for applications demanding top-notch performance and minimal latency, especially those involving intensive data processing or intricate graphics, native development with the Android SDK is the recommended path.

4. CONCLUSION

Based on the experiment and analysis of the results, we can conclude that an application written using native development tools was less resource-intensive than a similar application developed using Cordova. However, in 2024, Apache Cordova continues to be an excellent choice for mobile application development, effectively linking web technologies with mobile platforms. The platform provides developers with convenient tools for cross-platform development, including code reusability, easy access to device functionality, and active community support. Cordova makes it easy to customize applications for different platforms to meet the needs of a wide range of users and remain competitive in the mobile development market.

Apache Cordova is particularly suited for projects where you need to quickly develop and launch applications across multiple platforms with limited financial resources, and where deep integration with operating systems and high performance are not key requirements. In contrast, native apps offer greater stability and speed by better utilizing built-in device features such as camera, geolocation, and accelerometer. This optimization makes native apps the ideal choice for tasks where maximum responsiveness and performance are critical.

Each of these approaches has advantages and disadvantages. Apache Cordova is ideal for projects requiring rapid development and multi-platform support, but its use can lead to performance compromises and limited access to native features. On the other hand, Android View and Jetpack Compose provide more powerful capabilities for building high-performance applications, but require more in-depth knowledge of specialized technologies and languages [6].

Ultimately, the choice between Apache Cordova and native tools will depend on the specifics of the project, the resources and time available, and the skill level of the developers. It is important to comprehensively evaluate project requirements and resources to make an informed decision regarding technologies for mobile application development.

5. REFERENCES

[1] Giacomo Balli - Technology Advisor. "A Comprehensive Comparison of Flutter, React Native, and Cordova." Big Balli, 2023. URL: https://bigballi.com/blog/flutter-vs-react-native-vs-cordova (date of reference: 21.05.2024).

[2] Apache Cordova - The Official Website [Electronic resource]. – Mode of access: https://cordova.apache.org/, free. - Extracted from the screen. - date of reference: 24.04.2024.

[3] Android Developers [Electronic resource]. – Mode of access: https://developer.android.com/, free. - Extracted from the screen. - date of reference: 24.04.2024.

[4] Appsmith. "Apache Cordova vs. Native Android Development: What You Should Know" [Electronic resource]. – Mode of access: https://community.appsmith.com/solution/create-android-app, free. - Extracted from the screen. - date of reference: 24.04.2024.

[5] Clockwise Software. "Cordova vs React Native: Frameworks Performance Comparison." Clockwise Software, 2023. URL: https://clockwise.software/blog/cordova-vs-react-native-frameworks-performance-comparison/ (дата обращения: 21.05.2024).

[6] Mobile App Development. "Apache Cordova vs. Native Android: Which One Should You Use?" [Electronic resource]. – Mode of access: https://enlear.academy/cordova-vs-react-native-for-mobile-development-what-to-choose-in-2022-ddc91143f7f4, free. - Extracted from the screen. - date of reference: 24.04.2024.

[7] 100apps. "The Pros and Cons of Using Apache Cordova for Mobile Development" [Electronic resource]. – Mode of access: https://www.trustradius.com/products/apache-cordova/reviews?qs=pros-and-cons, free. - Extracted from the screen. - date of reference: 24.04.2024.