Cross-Platform Mobile Development: Xamarin vs. React Native

Ziad Kadry & Sayeed Sajal

Department of Math and Computer Science

Minot State University

Minot, ND, 58703

ziad.kadry@ndus.edu

Abstract

Mobile application development is a relatively new technology, since the introduction of the Apple App Store in 2008, this market started to emerge with many different apps continually being posted as the value of the market was realized. Soon after, Google released the Play Store which is a marketplace for apps on the Android platform.

Development for these platforms is fundamentally similar yet inherently different, with IOS (Apple's mobile Operating System) using objective-c then moving on to currently using Swift as its primary developmental language. Google's Android used Java, then moved on to using Kotlin for its app development.

Mobile developers used to face a common problem since IOS and Android hold the majority of the market shares when it comes to active mobile devices on the market after its the development phase. How could it be brought on both major platforms efficiently? The simple solution was to develop the application twice, once using the IOS development environment and the other using the Android development environment

As we all know, the simplest solution isn't always optimal, and we need to hire extra developers to develop for each platform, the cost began to rise and came the question of how do we stay consistent across the platforms when they consist of two separate codebases? Here comes Cross-Platform Mobile Development! The idea was fundamental in its core, develop the application once in a language, let the framework take care of translating it to IOS and Android native code.

This paper will serve as an introduction to the techniques and frameworks used for Cross-Platform Mobile Development, namely two of the most popular frameworks in the space: Microsoft's Xamarin and Facebook's React Native. We will examine the fundamental differences of these platforms with a live example to demonstrate these differences in a practical way. The most common questions answered in this paper are: What are the pros and cons of using each platform? What are the costs associated with using them? Which platform is better optimized for performance? And finally, which of these platforms is better supported in the long run?

I. Background

Cross-platform mobile development tools are more popular than ever, with an influx of mobile applications on multiple platforms that emerged in the last decade, a cross-platform mobile development solution was deemed to be a step in the right direction. Fast forward to today, there are many different frameworks to get this job done, We have divided these solutions to two main categories based on the type of programming language the framework uses: Web languages based, Compiled languages based. With the existence of all these different frameworks, individual studies on them have been made in the past. At the same time, a comprehensive comparison between the two classifications would prove useful to prospect developers.

II. Introduction

The term cross-platform development refers to the process of developing an application(app) once that has the capability of being deployed on multiple platforms [1]. This isn't necessarily a new idea but didn't exist in early computers. The software was developed to target one platform primarily; IBM software was made for IBM hardware and like so for Apple hardware. This kind of restriction left software developers with a predicament, which platform do we focus on our resources? Developing for multiple platforms meant having to develop the same application multiple times with entirely separate projects that contained different code bases. This process introduced many disadvantages. Firstly, the cost of development instantly increased with the more platform the developer is targeting, different platforms meant needing developers with varying fields of expertise to work on developing these separate projects. Secondly, increased time of the developmental cycle, if the developers wanted to deploy their software on all the platforms at different times, that meant that they had to wait until the separate projects were completed before they could release it. Finally, inconsistency in the applications, this is widespread and, in a way, expected since these projects share the same core purpose but are developed independently for most, if not all, their codebases.

That's why cross-platform development comes in, write once, deploy anywhere [2] was the slogan that brought this idea to life. Using one codebase using on multiple platforms, inherently, provided a solution for the disadvantages of developing for different platforms separately. It also enabled uniformity between the different versions of the application compiled for different platforms, and it saved a lot of time and resources in the developmental cycle. Mobile development followed a similar trend with the mobile application market booming when the Apple app store was first introduced in July of 2008 [5]. Since then, there has been a prominent trend of increased popularity with 204 billion app downloads across platforms in 2019 [3]. This popularity sets any business in an advantageous position by having a mobile application that works across all major platforms, namely IOS and Android, which held 97%. Out of the 3.2 billion active devices in 2019 [4-6]. Knowing this information is enough of a factor to excite developers about the idea of cross-platform mobile development [7]. This paper will discuss the general methodology behind cross-platform mobile development and will compare two of the most popular frameworks: Xamarin and React Native.

III. Comparison Points

To compare Xamarin and React Native, points of comparison must be clearly defined and set as a guide to any measurements made on these frameworks. I've divided these points of measures into two main categories: 1- Points affecting user experience. 2- Points are affecting developer experience. (See Fig. 1). From a mobile application user's perspective, many factors come in play to determine whether they stay as an active user of our application, choosing the right tools for development proves crucial in maximizing user engagement. Most developers invest in high-end computers to develop their applications. This behavior could easily lead to a skewed perspective of the kind of computing devices their use will be mostly run on since low to mid-end devices hold the majority of shares in the global market [10]. Application size has a certain degree of expectations from users, and a graphics-intensive game is expected to be a few hundred megabytes. In contrast, a simple note-taking app is expected to be a fraction of that. On average, the size of an Android application is 11.5mb; for IOS, this number jumps to 34.3mb [11]. The responsiveness of our application could be seen in different areas, how long does our app take to startup? How responsive is our user interface? Is our application scalable and will perform well under large or unusual user inputs? These are all questions with answers that could make or break the user experience. On the other hand, from a developer's perspective, creating this kind of rigid user experience requires well-crafted tools to enable developers to sculpt great mobile applications and continue to refine them. If a framework uses a popular language that developers are already familiar with, it invites more developers to use the framework with ease without a steep learning curve. Each framework contains different features, comparing these features is not as straightforward as it seems; this is mostly a result of the lack of a defined standard for these frameworks and how they handle the implementation of different things. So instead, I will focus on the features that make the developmental cycle simpler without compromising on quality. The last two aspects that will be looked at to compare developers' experience are the frequency of updates on these frameworks to bring in the latest native features added by the mobile operating systems and availability of community support in whatever issues developers might run across during the developmental cycle.

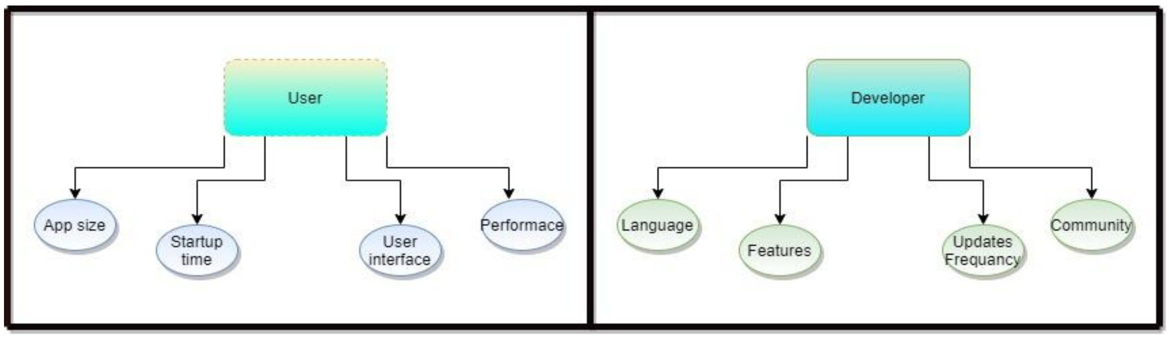


Fig. 1. Comparison points used to compare Xamarin and React native frameworks

IV. Xamarin

Xamarin is introduced in an article by Microsoft as “an open-source platform for building modern and performant applications for iOS, Android, and Windows with .NET” [11]. Xamarin studio and the complete development tools were released in 2013 [12]. This tool integrates very well into Visual studio, which makes installing it and using it an easy process. If we have Visual Studio, we might already have Xamarin installed on our computer! Xamarin uses C# as the primary language for developing all the business logic and XAML to design all the user interfaces [11]. Xamarin is split into four libraries: Xamarin.Android, Xamarin.IOS, Xamarin.Essentials, Xamarin.Forms. Most of the development will be done with Xamarin.Forms with Xamarin.Android and Xamarin.IOS having the ability to compile the application to their respective platforms natively, they also allow for the fine-tuning of features in either of the specific platforms to ensure the developed application delivers a tremendous native experience of the platform it’s deployed on. Xamarin works by translating the C# code to native instructions that will run on the deployed platform. (See Fig 2). Now that we’ve established what Xamarin is and what makes it great let’s talk about some of the shortcomings of the Xamarin tool. First, whenever an update is released for the Android or IOS platform, Microsoft has to implement these updates into Xamarin. This delay for the Xamarin to be updated could be detrimental for any developers trying to keep their applications sharp and equipped with the latest native features of these platforms. Unfortunately, there isn’t a workaround to this for developers as they have to wait for the update to be deployed for them to start updating their applications. The second disadvantage that Xamarin carries is that although most developers will be able to use Xamarin for free, professional, and enterprise edition of the software could be quite expensive. In 2020, it could cost up to $1199 per year. Fortunately, most individuals, small and mid-level businesses, will not be subject to these fees as the tool is free. The third disadvantage in Xamarin is the large app sizes as a simple Hello World application is 15.6 MB [13]. Lastly, the final con we believe is worth mentioning is that developing applications with heavy custom graphics are not easy to implement, going the native development route would make more sense in this case.

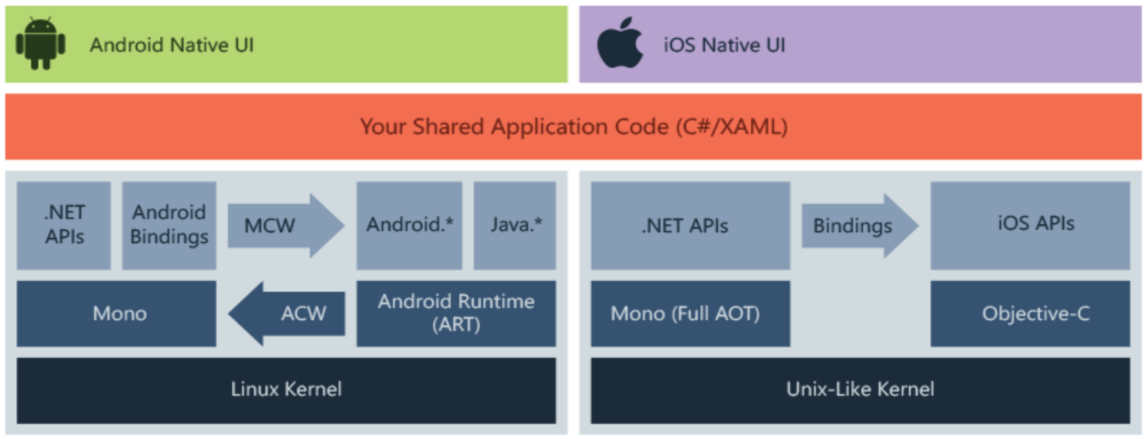


Fig. 2. An illustration of the cross-platform structure of a Xamarin application [11]

V. React Native

To define React Native, we first have to define React, React (also known as React.js or ReactJS) is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies [15]. It's a compelling framework that uses JavaScript to build cross-platform web interfaces. React Native is an open-source mobile application framework created by Facebook. It is used to develop applications for Android, iOS, Web, and UWP by enabling developers to use React along with native platform capabilities [14]. React Native is packed with features that make the development of cross-platform mobile applications a much simpler task! One of the significant features that this framework has is that all the development of business logic and the design of the user interface is done with one programming language: JavaScript, which reduces the learning curve associated with getting into and developing applications on this framework. React Native divides all the elements on the user interfaces to components and although the components included by default are rather simple, combining these components to create a much more complex component is made to be an easy task. The applications developed with React Native are compiled to Native instructions for the platform it's deployed on, hence the name. Another major feature React Native offers with ease is "Hot Reload" which enables the developers to see the changes to their application instantly when they save the changes, this the feature helps cut the development time since errors are caught sooner. When it comes to application size, React Native can produce minimal size applications, depending on the application, we can provide app sizes as small as 3.5 MB! [16]. React Native is packed with great features, and like any other framework, it has Its shortcomings. If a developer doesn't come from a web development background; this framework could deem to be a steep learning curve as the methodology of developing with React and web languages are quite different from most other types of development. Some applications that have many platform-specific features will require writing some native code, which makes it inconvenient at times for smaller teams with no access to platform-specific developers to complete their applications with ease. Another con for React Native is performance, while in the simplest applications the framework will produce high performing implementations, due to the shortcomings of JavaScript, any computationally heavy application will perform worse than its native counterpart [18].

VI. Comparison Results

To produce results based on actual experimental data, we have created two relatively identical applications, a version on Xamarin and another on React Native (See Fig 3.) The functionality of the app is simple; a search bar lies on top when the application is started which a name of a movie or a TV Show could be typed into, the app then connects to the OMDb database and searches for any matches for the typed name. The user could also touch any of the results and the app will display a page with more information about the selected content loaded from the API. The Xamarin application's APK file size was 28.3 MB while the React Native application's APK size was 6.5 MB, React Native has a clear advantage here since It's generated APK size is less than a 1/4th of Xamarin's APK. Xamarin's application size could be further reduced in size with some techniques, although I had trouble achieving the desired result [19]. Next, we set up an android emulator to be able to measure the performance of these applications. The settings of this virtual device were Illustrated (See Fig 4.) The primary test we ran was measuring the startup time of these applications, the Xamarin applications averaged at 2.76s for a startup, and the React Native application averaged at 1.65s (See Fig 5.) Both apps carried very intuitive user interfaces with native components and custom components created by combining native components, which ensured that users would have a native experience on their devices. Performance once inside the application seemed almost identical between the two implementations; no numerical measurements were done in this stage for the lack of the right measurement tool. Developing these applications was not a very time-consuming task. As a personal preference, a developer might be more comfortable with C# as a programming language, yet developing with React Native felt simple to pick up and get into. The fact that designing the user interface in Xamarin was done in Xaml gives it a slight disadvantage, using one language for the interface and business logic in React Native made it simpler to transition between the two tasks. The hot reload function in React Native made it easier to catch errors in the development cycle sooner as the errors could be detected and fixed instantly. Although currently, React Native is more popular (See Fig 6.) we found that the community for Xamarin was more helpful, which helps to Xamarin being around for longer.

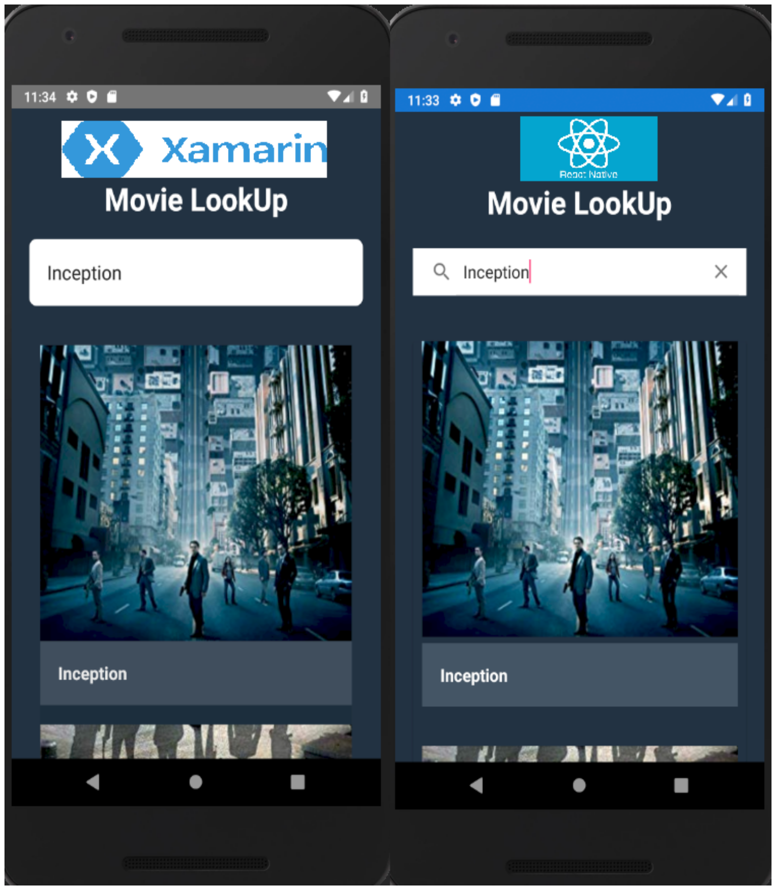
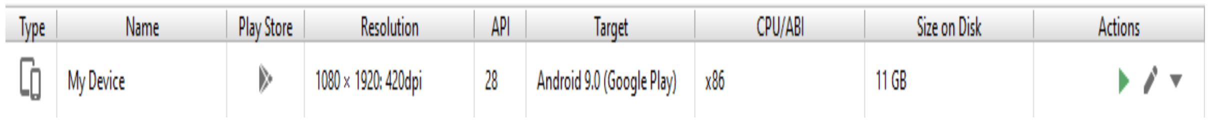


Fig. 3. On the left is the application developed with Xamarin and on the right is the application developed with React Native. NOTE: The logos are not a part of the applications, mainly to avoid any differences that could affect performance.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	My Device		1080 × 1920: 420dpi	28	Android 9.0 (Google Play)	x86	11 GB	

Fig. 4. The android virtual emulator used for the testing of the applications

Framework	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	Average
Xamarin	2.94s	2.76s	2.46s	2.84s	2.67s	2.80s	2.91s	2.77s	2.71s	2.82s	1.65s
React Native	1.78s	1.88s	1.82s	1.70s	1.52s	1.41s	1.50s	1.65s	1.73s	1.69s	2.76s

Fig. 5. A comparison between start up time of Xamarin and React Native applications, each application was tested ten times and the average time was calculated

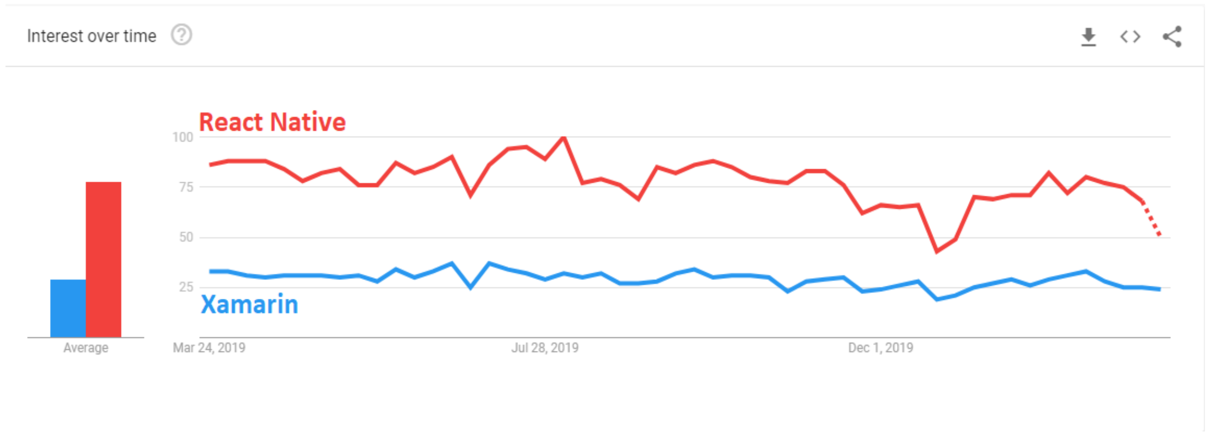


Fig. 6. A comparison between the search results for React Native and Xamarin on Google Trends

VII. Conclusion

Choosing the right framework solely relies on preference. React Native seems to provide a better the experience overall with the price of having to learn React and JavaScript, while Xamarin also provides a great experience especially if the team already is accustomed to the .NET framework. Both frameworks are free for most developers which makes them a great starting point and gives developers the freedom of testing them without any overhead to get a feel of how the function and be able to make a further determination on which to use for projects. We started this comparison with a background in Xamarin and we like it, so we had to put any kind of bias aside and examine them both fairly, and that led me to realize how great React Native truly is primarily for experienced web developers who would find it easy to start developing Cross-Platform mobile applications. Further testing on IOS should be done on these platforms but due to not having access to a macOS device, we didn't have the option to run those tests like intended.

References

[1] What is cross-platform development? Definition & more| Sapho. (2020). Retrieved 26 February 2020, from https://www.sapho.com/glossary/cross-platform-development/

[2] Write once, run anywhere. (2020). Retrieved 26 February 2020, from https://en.wikipedia.org/wiki/Write_once,_run_anywhere

[3] Annual number of mobile app downloads worldwide 2019 | Statista. (2020). Retrieved 26 February 2020, from https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/

[4] Smartphone users worldwide 2020 | Statista. (2020). Retrieved 26 February 2020, from https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[5] 1983 to today: a history of mobile apps. (2015). Retrieved 26 February 2020, from https://www.theguardian.com/media-network/2015/feb/13/history-mobile-apps-future-interactive-timeline

[6] Casserly, M. (2020). Which is the more popular platform: iPhone or Android?. Retrieved 26 February 2020, from https://www.macworld.co.uk/feature/iphone/iphone-vs-android-market-share-3691861/

[7] Flutter vs. React Native vs. Xamarin - LogRocket Blog. (2019). Retrieved 26 February 2020, from https://blog.logrocket.com/flutter-vs-react-native-vs-xamarin/

[8] Prajapati, Mukesh (2016). STUDY ON XAMARIN CROSS-PLATFORM FRAMEWORK, International Journal of Technical Research and Applications e-ISSN: 2320-8163

[9] Kaushik, Vipul & Gupta, Kamali & Gupta, Deepali. (2019). React Native Application Development.

[10] Global smartphone market share 2019 | Statista. (2020). Retrieved 8 March 2020, from https://www.statista.com/statistics/271496/global-market-share-held-by-smartphone-vendors-since-4th-quarter-2009/

[11] What is Xamarin? - Xamarin. (2019). Retrieved 15 March 2020, from

https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

[12] Friedman, N. (2020). Announcing Xamarin 2.0 | Xamarin Blog. Retrieved 15 March 2020,https://web.archive.org/web/20130627074458/http://blog.xamarin.com/announcing-xamarin-2.0/

[13] Application Package Size - Xamarin. (2018). Retrieved 16 March 2020, from https://docs.microsoft.com/en-us/xamarin/android/deploy-test/app-package-size

[14] React Native. (2019). Retrieved 19 March 2020, from https://en.wikipedia.org/wiki/React_Native

[15] React (web framework). (2020). Retrieved 19 March 2020, from https://en.wikipedia.org/wiki/React_(web_fram

[16] How I Reduced the Size of My React Native App by 86%. (2018). Retrieved 19 March 2020, from https://medium.com/@aswinmohanme/how-i-reduced-the-size-of-my-react-native-app-by-86-27be72bba640

[17] A brief history of React Native. (2016). Retrieved 19 March 2020, from https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39

[18] Performance Issues and Optimizations in JavaScript: An Empirical Study Marija Selakovic and Michael Pradel

[19] Reducing iOS and Android App Size in Xamarin. (2020). Retrieved 19 March 2020, from https://heartbeat.fritz.ai/reducing-the-app-size-in-xamarin-deep-dive-7ddc9cb12688