Comparison of cross-platform and platform-specific frameworks for mobile development

Lennert Van Looveren, Peter Karsmakers[1], Bram Schrijvers[2],
[1]KU Leuven, Faculty of Engineering Technology, Technology Campus Geel
[2] Zappware, Ilgatlaan 19, 3500 Hasselt, Belgium

Abstract Mobile phones are everywhere. They are used as cellphones, communication channels and as a personal computer. Therefore, they function more and more as a source of entertainment and information. This paper contains the process of the development of a tv guide application for mobile. Since there are many options on which framework an app can be developed, a research for which framework is the most compatible for such development could provide useful information. This year we have created this mobile application for two different frameworks, while Zappware provided the native iOS and Android version. Those applications were written in React Native and Xamarin. We document specific differences in various areas and weigh them against the native ones. This document also provides information about two other cross-platform frameworks: Apache Cordova and Qt.

***Index Terms*— Cross-platform frameworks, Android, iOS, Xamarin, React Native, Apache Cordova, Qt, Mobile development**

I. Introduction

Mobile applications can be created in various ways. The most natural way to develop them is by writing the iOS app through XCode in Swift, and the Android app through Android Studio in e.g. Java or Kotlin. This means there is a need for two separate fully developed applications. This research is dedicated to finding a good alternative for that scenario. The so-called cross-platform frameworks that are being researched: Qt, Apache Cordova, React Native and Xamarin. Their pros and cons get valued against each other, in both a theoretical and a practical way. The goal is to find the most practical framework, build a comparable application and compare those with the native applications Zappware have built. The parameters by which those will get evaluated are: speed, memory, development effort and user experience.

II. Cross-platform frameworks

Qt is a framework developed in C++ that among other things supports Android, iOS and Windows Phone. Applications are written in a combination of C++ and QML, which is a declarative scripting-language that uses JavaScript to build its logic with. Qt is a cross-platform framework, this means it can render one code on Android, iOS and Windows Phone. This cross-platform property exists since the C-preprocessor enables or disables chunks of code depending on the platform the application is running on. [1]

React Native is completely based on React, a JavaScript library built by Facebook to make big scalable web applications. React Native is a JavaScript framework build to render mobile application for iOS, Android and Universal Windows Programs (UWP). It is written in combination of JavaScript and JSX, which is a language comparable to XML. React Native uses the platform its own standard rendering API to create an application which will contain the same elements as a native designed app. Through this procedure the framework aims to possess the same look and feel as those native designed applications. [2]

Xamarin is a development framework which uses a .NET environment with iOS and Android C# libraries. While single-platform development is possible with Xamarin through those C# connections with native Android and iOS APIs, the cross-platform alternative is more interesting. Xamarin.Forms is a toolkit consisting of cross-platform UI-classes which can be considered as a bridge between the .NET environment and the single-platform libraries. This toolkit locates which of those libraries must be used depending on which platform its application is rendered on. The applications are coded in C# and XAML. [3]

Apache Cordova is an alternative which uses standard web technologies like HTML5, CSS3 and JavaScript for its cross-platform development. Applications run inside native application wrappers specifically build for each platform. A WebView renders onto that wrapper, which gets access to the device-level APIs like sensors, data, ... This WebView is basically a borderless browser where a UI gets rendered upon. This UI is made off native application components which the WebView can access through the OS of the device. [4]

III. Theoretical comparison

To distinguish the frameworks from each other a comparison is made of various properties. These include: The programming language, the way the framework renders the UI and the commercial use and its restrictions. After that a conclusion is made about the theoretical usefulness of those specific platforms. This conclusion gets used to determine which practical applications effectively get build.

React Native uses JavaScript and JSX for its code. Xamarin C# and XAML. Qt applications get developed in a combination of C++ and QML while Apache Cordova uses standard web application languages: HTML5, CSS3 and JavaScript. From this summary the conclusion can be made that only Apache Cordova exclusively uses languages which are already known and widely utilized. The rest of the frameworks has one language (JSX, XAML and QML) which is developed for the use in that specific framework only. Although it should be said those languages are all derived from a well-known alternative (respectively HTML, XML and CSS).

Looking at the way those frameworks render their UI, React Native and Xamarin both use native UI-controllers. Qt uses a mix of native components and the OpenGL API, this is a library that renders 2D and 3D vectors. Apache Cordova is the only framework which operates using a WebView.

A company is also interested in the financial side of such development platforms. All frameworks have an open-source version for developers to use. React Native and Xamarin have one under the MIT-license. [5] [6]

But while Xamarin has a paid Visual Studio IDE version, React Native does not. This means all features are enabled for React Native. Qt has an open-source LGPL v3-license or a GPLV2/GPLV3-licence, with an option for a commercial license. [7] Apache Cordova us an Apache licence version 2.0. [8] Some platforms build upon Cordova like Adobe PhoneGap have paid versions though.

All things considered is React Native the framework with the most potential. Its combination of free-to-use, native UI-controllers and with JavaScript a very known, widely used language potentially makes it a very viable framework. Since the look and feel of an application is one of the most important features, the Xamarin app is considered the second-best option since it also utilizes the native UI-controllers. This feature trumps the other options because it does not use a form of bridge between the device and application, this would automatically mean some performance-issues e.g. delays, connection problems, ...

Because of time-related issues not all frameworks could be tested, and thus only two implementations were made. Due to this theoretical comparison the options who were chosen are React Native and Xamarin.

IV. Application

The application that shall be built through the chosen frameworks is a basic tv-guide application as seen in Figure 1. This is the app that Zappware has built for Android and iOS use. Although the layout will be different on the two frameworks, since the frameworks will sometimes use different components, the overall look stays the same. The goal will be to mimic this example as good as possible and after that conclude the limits and possibilities of the used cross-platform frameworks. After the applications are finished developing, those will get tested on various elements including: development effort, speed, memory and user experience.

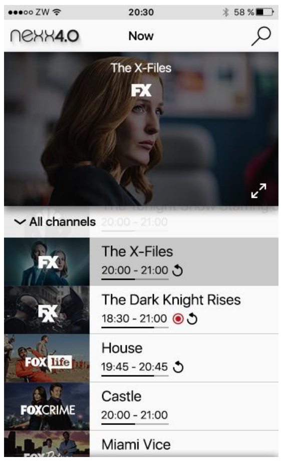

Figure 1 Target application

V. Tools

To test the created applications, some tools were used to determine particular variables.

Testobject.com is an online test-API where applications can run through a cloud to determine their use on variant devices. The feature has a paywall for extended use, but gives a trial to new users. To use Testobject the applications their .ipa or .apk file(respectively for iOS and Android) get uploaded unto the cloud. These files undergo an automatic test procedure in an environment which depends on the devices the user chooses. After the calculations are done, Testobject provides among other things crash reports, quality reports and the option to manually test your application online. These reports are provided in the shape of logs, these logs get used to determine the installation time and the startup time of the applications.

To monitor the memory usage of the Android application, the OS provides information on the device itself. iOS on the other hand needs a tool to monitor such things. XCode offers an instrument which can analyze the apps on memory usage. The instrument is called Activity monitor and provides information about leaked memory, abandoned memory and general memory usage including cache- and RAM-memory.

VI. Practical comparison

As stated before, the application will undergo testing to determine various important elements in the making-of and results of the process. This paper reviews the most important conclusions which have been made about those elements for the two used frameworks.

The development effort which the framework needs to release an application from start to finish, is the first important element. React Native does not use an IDE like Xamarin(Visual Studio) does. This means a lot of tools meant for debugging are absent, which Visual Studio does provide though. In React Native there are tools for debugging with online sources, but are not as extensive as those inside Visual Studio. The coding languages in which the applications are written both feel familiar. Although the JavaScript that React Native uses does feel less complicated and more straight-forward than the combination of XAML and C# that Xamarin uses. This feeling gets amplified because the document around Xamarin are not as obvious as those around React Native. While for React Native a lot of answers can be found online, Xamarin does not have the same active community. The processes for the installation and use for Xamarin itself was well-documented though. The structures in React Native and Xamarin are different but can both be used for big complex applications. Also, the deployment of the finished application on iOS and Android for both frameworks was a clear process without problems.

If we conclude the recently formed statements, there is a case to be made that React Native thrives in an environment where there is a quick, simple prototype needed for an application. Because of how the setup works without an IDE, the ready-to-use simplicity, a programming language that requires less experience and less insight and an active community that solves problems with extensively documented solutions. Xamarin is more the framework that needs a longer learning curve and is harder to use. That way it would probably serve better as a stand-alone platform where a lot of experience and insight is needed to make it work. As a reference to the theoretical comparison, all features are available for React Native. This in contrast with Xamarin which has a paid version of Visual Studio, React Native is more suitable as a prototype platform than Xamarin is.

Analyzing the memory usage of the applications is next. The result of the test can be found inside Table 1 and Table 2 for iOS and Android respectively. The most notable thing is that the React Native application took significantly more RAM-memory than its Xamarin counterpart. These differences could be allocated on the fact that React Native version had more features since the Xamarin application was not finished. This is a reason why a conclusion could not be made.

In terms of speed (Table 3 and 4), the installation time was immediately declared a useless variable, since this is proportional with the file-size of the application. The startup time had some interesting insight though. On Android the Xamarin app performed significantly worse than the React Native alternative. But on iOS the React Native app performed worse than its counterpart. An explanation could be the difference in hardware inside the used devices and the structure of the application. If the reason was the framework, those problems would be consistent on Android and iOS. The significant startup time of the Xamarin Android app gets explained by the way it setups its list inside the application. It is a static reference, which Android had a hard time handling.

Because of these nuances there has been judged that there is no significant difference in the startup time between the cross-platform framework itself, this conclusion gets derived off the fact that the iOS-timings are only a little different from each other. This discrepancy could be explained due to the difference in features.

Table 1 Android memory results

	Native (Android)	React Native	Xamarin
Internal memory $[MB]$	49,45	17,86	26,84
RAM-memory $[MB]$	343	252	169
Cache-memory $[kB]$	492	3200	20

Table 2 iOS memory results

	Native (iOS)	React Native	Xamarin
Internal memory $[MB]$	94,5	24,6	24,6
RAM-memory $[MB]$	288,1	126,3	97,8
Cache-memory $[kB]$	115	4	3000

Table 3 Android speed results

	Native (Android)	React Native	Xamarin
Installation time [s]	26,59	9	10,75
Startup time [s]	1,193	0,348	6,601

Table 4 iOS speed results

	Native (iOS)	React Native	Xamarin
Installation time [s]	15,13	7,46	7,46
Startup time [s]	0,3916	0,4639	0,3404

The last and most important feature of a mobile application is its user experience. Previous elements like memory and speed all add to the user experience, but cannot be finally judged by numbers alone. Since this is a subjective matter, this element will be tested by a survey. This survey will consist of parameters which the participant must give a score within the range of one and five. The participant will rate these parameters for each application in iOS and Android, including the native apps made by Zappware's development team. The survey has a sample size of ten participants with different backgrounds and various expertise levels. The results of the survey are shown in Table 5 and Table 6 with the first result being the mean and the second being the median of each column.

Looking at the results, it is obvious the native applications gets (almost) perfect scores the whole survey. This makes sense since this application is developed by a whole team of developers who realized this after a lot of hours and effort. The only thing that is noticeable: the startup time does not score as good in Android. This can be explained, like previously stated, by the difference in hardware. But the startup time of this application cannot be compared with the other applications considering the native app is more complex and comprehensive.

The questions about the components, like the video player and date picker, were answered in positive fashion. Although the date picker in the iOS applications of the cross-platform frameworks gets worse results. This is because in the iOS application the date picker gets blocked by the iOS standard clock. This is a small inconvenience which results in the fact that both React Native and Xamarin applications both need some form of platform-specific code to solve such issues.

The results in startup time are consistent with the startup times tested in Table 3 and Table 4. The explanation for these results have already been discussed.

Scrolling through a list on an application should run as smooth as possible. The scores on this part are based on the number of jitters occur. Those are creaks that happen when scrolling through such a list. There is a difference between the first time a list gets scrolled and the second time. This phenomenon occurs when the data is loaded in the cache-memory of the device the first time. The second time the list gets scrolled through the data will be obtained from that cache-memory, this results in a smoother experience. The results confirm that this phenomenon happens in the applications, with better results the second time opposite to the first time on Android. The scores show that the React Native version runs smoother on Android and that they both score the highest on iOS.

The general look and feel is also judged by the participants. This results in a conclusion that the Xamarin and React Native applications feel quite similar and do not look different. Making a judgment about this element is hard, since they are so similar. The React Native application gets an average score 0.2 higher than its Xamarin counterpart on Android. The reverse is true for iOS with a difference of 0.4. This concludes that a React Native app has a slight preference in Android, while Xamarin is more so preferred in the OS of Apple.

All this information accumulates to the fact that determining which application is the best option is hard. It is obvious that the native application is the best option on all accounts, which makes sense. On the native platforms every little detail can be adapted and everything surrounding their components is possible. On a hybrid framework the application gets limited since not all components are available and not every single variable can be changed. But the options to build a good working application in both frameworks exist. To use such a framework to build a prototype for an application sure is possible. For this purpose, React Native seems the better choice of the two discussed frameworks. As stated earlier, React Native has the financial and simplicity advantage over Xamarin. For these reasons the conclusion is made that the React Native is the preferred framework to create hybrid applications. Although it is not good enough to replace native designed applications if user experience is the most important element of the application.

Table 5 Android results survey user experience (mean / median)

	React Native	Native Android	Xamarin
Startup Speed	4,1 / 4	3,4 / 3	1,9 / 2
Scroll 1	3,3 / 3	5 / 5	2,7 / 2,5
Video	5 / 5	5 / 5	5 / 5
Date	4,3 / 4,5	5 / 5	4,5 / 4,5
Scroll 2	4,1 / 4	5 / 5	3 / 3
Score	3,7 / 4	5 / 5	3,5 / 3,5

Table 6 iOS results survey user experience (mean / median)

	React Native	Native iOS	Xamarin
Startup Speed	2,8 / 3	4,7 / 5	5 / 5
Scroll 1	5 / 5	5 / 5	4,7 / 5
Video	5 / 5	5 / 5	5 / 5
Date	3,6 / 3,5	4,6 / 5	3,2 / 3,5
Scroll 2	5 / 5	5 / 5	4,3 / 4,5
Score	3,5 / 4	5 / 5	3,9 / 4

VII. Conclusion

A cross-platform framework is an alluring idea in theory. In practice it will not satisfy all requirements to fully replace native development. Too many factors ensure that the general performance and possibilities of the hybrid alternatives perform inadequate opposite to the native option. This does not mean there is no place for cross-platform development in the world of Information Technology. It could function perfectly as a prototype-framework to build demos for various purposes. The recommendation of this paper consists of using React Native for this exact purpose. Although this choice is still a personal one, depending upon which specific factors the developer attaches importance to.

VIII. References

[1] B. Starynkevitch, "How is QT cross-platform," [Online]. Available: [Online]. Available: https://www.quora.com/How-is-QT-cross-platform. [Accessed 2018 Maart].

[2] B. Eisenman, Learning React Native, O'Reilly Media, Incc., 2016.

[3] D. Hermes, Xamarin Mobile Application Development - Cross-platform C# and Xamarin.Forms Fundamentals, Apress, 2015.

[4] Apache Cordova, "Overview Apache Cordova," [Online]. Available: https://cordova.apache.org/docs/en/latest/guide/overview/. [Accessed April 2018].

[5] "License React Native," [Online]. Available: https://github.com/facebook/react-native/blob/master/LICENSE. [Accessed April 2018].

[6] [Online]. Available: https://blog.xamarin.com/xamarin-for-all/. [Accessed April 2018].

[7] "Open source licenseQt," [Online]. Available: http://doc.qt.io/qt-5/opensourcelicense.html. [Accessed April 2018].

[8] "License Apache Cordova," [Online]. Available: https://github.com/apache/cordova-android/blob/master/LICENSE. [Accessed April 2018].

[9] B. Starynkevitch, "How is QT cross-platform," [Online]. Available: https://www.quora.com/How-is-QT-cross-platform. [Accessed Maart 2018].