sciendo

Műszaki Tudományos Közlemények vol. 15. (2021) 61–64.
DOI: Hungarian: https://doi.org/10.33895/mtk-2021.15.12
English: https://doi.org/10.33894/mtk-2021.15.12

EME MTSZ MTK

Comparative Analysis of Native and Cross-Platform iOS Application Development

Márk KOVÁCS,[1] Zsolt Csaba JOHANYÁK [2]

Neumann János Egyetem, GAMF Műszaki és Informatikai Kar, Informatika Tanszék, Kecskemét, Magyarország

[1] *kovacs.mark@gamf.uni-neumann.hu*

[2] *johanyak.csaba@gamf.uni-neumann.hu*

Abstract

Nowadays, mobile applications are developed for more and more areas, providing great help for our everyday lives. When designing a mobile application, the first important decision to make is to choose the targeted platform. Is it only phone or tablet as well? Should the app run on Android or iOS, or should it be available on both mobile operating systems? In the latter case, besides the native development environments, it is worth considering a cross-platform development environment to write the software. This study investigates both the development and performance aspects of some possibilities for iOS application development, namely, native iOS development in Xcode, Xamarin.iOS, and Xamarin.Forms frameworks.

Keywords*: iOS, native, cross platform, Xamarin, Xamarin.iOS, Xamarin.Forms.*

1. Introduction

There is an increasing demand for more complex applications in the dynamically expanding technology market. Developers face several challenges that can be met with different solutions. If an application needs to run on both the most popular mobile operating systems, namely, Android and iOS, one might want to consider some different options available for development frameworks. Additionally, one needs to be aware of the advantages and disadvantages of the chosen environment and framework.

This paper investigates some differences between the functioning of the apps created by native and cross-platform iOS application development using Xcode and Xamarin development systems.

In the case of Xamarin, two options are examined. The first is Xamarin.iOS, which can be used only for the development of applications targeting a device running the iOS operating system. Its interface editor is very similar to the Storyboard editor interface used in Xcode. The other option is Xamarin.Forms, with which can be used for the development of applications running on both iOS and Android operating systems.

1.1. Development environments

Smartphone applications can be divided into three major categories based on the technologies and environments used in development, namely, native applications, web applications, and hybrid or cross-platform applications. Native applications are developed in specific environments and frameworks for a particular group of operating systems. For example, Android Studio and Xcode IDE are such software offering the necessary tools for developing apps for iOS and Android, respectively. In contrast, applications targeting more than one operating system are developed using so-called cross-platform development tools.[1]

1.1.1. Xcode

Xcode is an IDE developed by Apple that allows building native applications for any Apple-related operating system, like iOS, iPadOS, watchOS, macOS, or even tvOS. The most common scenario is developing applications for iOS or iPadOS. When working in Xcode, Interface Builder is used for designing the interface that includes a Story-

board. The interface is built up using visual tools. Apple previously supported the Objective-C programming language, but since 2014, it has developed its own programming language called Swift. [2]

1.1.2. Visual Studio 2019 and Xamarin

Nowadays, Visual Studio is one of the most popular development environments. It facilitates the creation of multi-layered software targeting multiple platforms. Its basic programming languages are C-type languages, including C#, which is based on .NET.

Microsoft provides Xamarin for the mobile application development workload targeting multiple platforms. Xamarin is an open-source tool developed by the Xamarin company acquired by Microsoft in 2016. It provides two options for the creation of iOS- and Android-related cross-platform applications. [3]

The first one is Xamarin Native, which supports developing applications in C# using Android and iOS SDKs. One of our test programs was also developed in one of its subsystems (Xamarin.iOS). Here one is able to create the user interface in a very similar way to the Interface Builder used in Xcode. This option can be an alternative for those who for some reason do not want to develop it in Xcode in Swift. In fact, when working on a Mac, Visual Studio also offers the option of designing the user interface by Xcode Interface Builder. [4, 5]

The second option is Xamarin.Forms. This allows for a platform-independent solution that allows us to develop on both platforms, usually with less programming. Here the development environment generates a separate project for each targeted platform as well as a project for the shared codebase. Usually, most of the coding can be done working only in the shared codebase. Unlike in Android Studio and Xcode, here there are no visual tools supporting the user interface design. This task has to be done manually using the XAML language. [6, 7]

1.2. Programming languages

1.2.1. Swift

The programming language Swift was developed by Apple and has gained popularity among application developers recently. Besides mobile application development, it also can be used for the creation of desktop programs and cloud-based services. Swift is a dynamic type language (the compiler has the ability to detect type automatically) and at the same time, it is a strongly typed language (type should be used strictly). These two features ensure that Swift is more secure and faster than many C-based languages. It also contains some characteristic features of C-type languages. The code written in it is easily readable and can be learned quickly. [4, 8]

1.2.2. C#

C# is a popular programming language developed by Microsoft for .NET. It is a general-purpose high-level object-oriented language that possesses several similar features to C++and Java. However, writing code in C# is more comfortable than in C++ as well as it supports rapid application development. C# is also a dynamic type and strongly typed language. Although currently it ranks only eighth on the IEEE ranking list of the top programming languages for mobile development [9], its clear advantage compared to all the languages preceding it on the list is that it supports application development for all the four main platform families (web, desktop, mobile and embedded).

2. Comparisons, tests

2.1. Demonstration of a test application

In the course of our investigation, we developed in all three environments (Xcode, Xamarin.iOS, and Xamarin.Forms) a relatively simple application. The program contains a list with 20 predefined items on it. The user can select an item followed by popping up an alert window. The selected item can be deleted as well. A new item can be added with the help of a textbox and a button. The goal of the investigation was the comparison of the three apps by means of loading (starting up) time and CPU usage in the course of the different operations.

The user interface (UI) (see **Figures 1** and **2)** could be created in a quite short time in all cases and the resulting appearance was similar as well. In this example, the list items were food names. In the case of native iOS and Xamarin.iOS one could easily create an almost identical UI while the usage of XAML in the case of Xamarin.Forms resulted in a slightly different appearance. **Figure 1** illustrates clearly the interface and operation of the application in Xcode. In our example, the list stores the names of foods.

2.2. Software and tools used for testing

Tests were carried out using the Xcode tester software called Instruments. It provides several

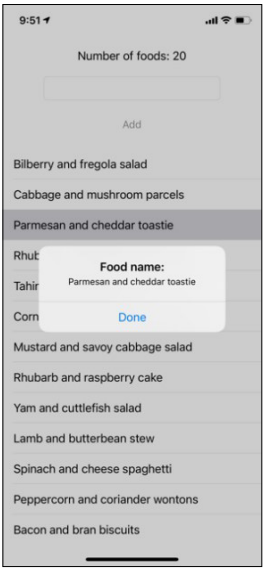

Figure 1. *Application developed in Xcode.*

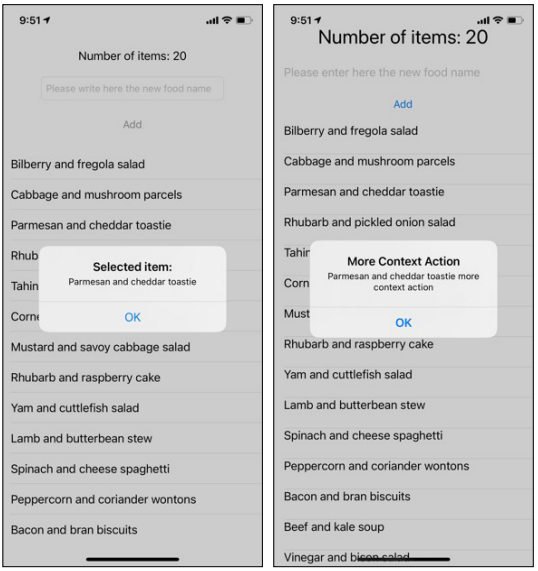

Figure 2. *Application interface written in Xamarin. iOS (left) and Xamarin.Forms (right).*

options for testing. One can measure the startup time of the application, and the hardware requirements of the developed application can be seen and recorded during usage. Processor requirements can be tracked, which greatly affects the speed of the application.

During testing, we examined five main functions of the app:

– starting
– scrolling
– row selection
– delete a row
– add a new item.

The test device was an iPhone XS 64GB a six-core device with 2×2.5 GHz and 4×1.59 GHz frequencies. All three applications were tested in the order listed above.

2.3. Test results

The results of the tests are presented in the tables below. The minimum and maximum processor requirements for each operation and the number of cores used are indicated.

2.3.1. Application developed in Xcode

The application started in 384 ms. Processor requirements are shown in **Table 1.**

While the application was running without user interaction (app was in foreground but none of its functions were used) it did not use any measurable processor resource.

2.3.2. Application developed in Xamarin.iOS

Processor requirements are shown in **Table 2.** The application started in 2.4 seconds, which is quite high for an application of this size. Like the native application, it required almost no measurable processor resources when it did not receive any user interaction.

2.3.3. Application developed in Xamarin.Forms

Processor requirements are shown in **Table 3.** The application started in 665 ms, which is good

Table 1. *Nativ iOS application processor requirements*

Function	Min %	Max %	CPU core
starting	10	170	6
scrolling	10	120	5
selecting	10	110	5
deleting	10	110	5
adding	10	120	6

Table 2. *Xamarin.iOS application processor requirements*

Function	Min %	Max %	CPU core
starting	10	120	6
scrolling	10	90	4
selecting	10	110	5
deleting	10	110	4
adding	10	130	6

Table 3. *Xamarin.Forms application processor requirements*

Function	Min %	Max %	CPU core
starting	50	130	6
scrolling	10	100	6
selecting	10	130	6
deleting	10	110	4
adding	10	120	6

compared to the native iOS application. However, even at rest (when it did not receive any user interaction), a constant 10% CPU usage was measured.

3. Summary and conclusions

In general, all three applications performed well; however, there are differences in some respects, which are summarized below.

In terms of development steps, native and Xamarin.iOS are very close to each other. Functions can be implemented similarly in both of them, and the UI design is facilitated by a WYSIWYG-like (What You See Is What You Get) visual tool. On the other hand, when developing the app in Xamarin.Forms, the interface must be written in XAML without a visual aid. Thus, one can see the appearance of the UI only at runtime. This increases the time demand of the development especially for those programmers who are used to the visual design.

At the launch, both the native and Xamarin. Forms applications provided similar good performance, namely, the shorter time necessary for startup, while Xamarin.iOS performed much worse. Although using the most resources, the native application was able to start the fastest. For all three applications, the biggest resource requirement appeared at startup. However, there was no significant memory consumption in any of the cases.

It can be considered a shortcoming that the Xamarin.Forms app ran at a minimum but constant 10 % CPU load even at rest. In terms of the functions of the test application, each performed with different values, but overall, they performed similarly.

Acknowledgement

This research is supported by EFOP-3.6.1-16-2016-00006 "The development and enhancement of the research potential at Pallas Athena University" project. The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

References

[1] Molenda, D., Skublewska-Paszkowska M.: *Analysis of the Possibility of Shortening the Time of Creating a Mobile Application for Android and iOS Systems Using Xamarin Technology*. Journal of Computer Sciences Institute, 12. (2019) 226–231. https://doi.org/10.35784/jcsi.493

[2] Swift, About Swift. (accessed on: 2021. 02. 26.) https://swift.org/about/

[3] Prajapati M., Phadake D., Poddar A.: *Study on Xamarin cross-platform framwork.* International Journal of Technical Research and Applications, 4/4. (2016) 13–18.

[4] Vishal K., Kushwaha A. S.: *Mobile Application Development Research Based on Xamarin Platform.* 4th International Conference on Computing Sciences (ICCS), Jalandhar, India, 2018, 115–118. https://doi.org/10.1109/ICCS.2018.00027

[5] Ebone A., Tan Y., Jia X.: *A Performance Evaluation of Cross-Platform Mobile Application Development Approaches*" IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Gothenburg, Sweden, 2018, 92–93.

[6] Pawel G., Maria S.-P., Edyta L., Jakab S.: *Performance Analysis of Native and Cross-Platforms Mobile Applications*, IAPGOŚ 2/2017, (2017) 50–53. https://doi.org/10.5604/01.3001.0010.4838

[7] Altersoft, The Good and The Bad of Xamarin Mobile Development, 2020. (letöltve: 2021. 02. 28.) https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/

[8] Bilberg D.: *Comparing Performance between React Native and Natively Developed Smartphone Applications in Swif., A Comparative Analysis and Evoluation of the React Native Framework.* https://www.diva-portal.org/smash/get/diva2:1215717/FULLTEXT01.pdf

[9] IEEE *Spectrum Interactive: The Top Programming Languages.* [Megtekintve: 2021.03.14.] https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2020