The Comparison of Native Apps Performance on iOS (Swift) and Android with Cross-platform Application – Xamarin

Student project

Dawid Dobrzański, and Wojciech Zabierowski

***Abstract*—The article describes comparison of two technologies used for creating mobile applications – cross-platform Xamarin and native for Android and iOS. The base constitutes results of appropriate tests executed by application created for that purpose.**

***Index Terms*—Java, C#, Swift, Xamarin, Android, iOS, mobile apps, performance tests**

I. Introduction

Nowadays, there are more and more mobile devices and, consequently, mobile applications. There are many technologies which can be used for creating them. The most popular techniques are native technologies allowing making application on dedicated platform (for example Android, iOS, Windows 10 Mobile). Otherwise, there also exist two types of multiplatform solutions. First of them uses web technologies (HTML, CSS, JavaScript) but these applications have limited possibilities and their performance is much worse than the native ones. The second solution is Xamarin platform – it allows to create cross-platform applications which functionality and performance are comparable to native counterpart. We have to take into consideration fact, that Xamarin is intended for creating one, instead of two applications. It may be efficient solution for companies because making one application takes less time. Moreover, developers have to know only one technology while they use Xamarin what makes creating applications easier. These two facts imply cutting costs for company. This article is about comparing efficiency of Xamarin and native applications on Android and iOS operating systems. For that case, five tests which check app's performance by device components' high usage (for instance CPU, memory) were prepared.

II. Used Technologies

A. Java

Java is an object-oriented programming language and platform used for creating software. Oracle stands for Java's development. It is all-purpose technology which allows to make various types of application (for instance web application or mobile application). Java has been the most popular programming language for many years because Java applications may be run on many types of operating systems (for example: Windows, macOS, Linux distributions).

During compiling, the source code is being converted to the bytecode. Subsequently, Java virtual machine (JVM) runs the application.

Java and Android SDK enable to create native Android mobile applications. In particular, Android SDK contains:

- devices' emulator,
- Dalvik Debug Monitor System (debugger),
- Android Debug Bridge (adb),
- Traceview (application's logs).

B. Swift

Swift is a programming language created by Apple. By using it, we can make native applications for iOS, macOS and watchOS. Swift's main purpose is to trade up outdated Objective-C language which had been used for creating Apple's native applications up to year 2014. That technology is still developing – this year, Swift 4 was presented. Unfortunately, it sometimes comes across incompatibility with previous versions. However, Swift is the most promising tool for making iOS native applications and its popularity is increasing nowadays. Swift minimizes developer mistakes and allows to provide application stability and safety. These applications are built by using Low Level Virtual Machine (LLVM) compiler which was created in C++ programming language. Swift is available while using Xcode 6+ version – it coincides with date of iOS 8 premiere.

C. C#

C# is an object-oriented programming language, it is component of .NET platform. C# main purpose is to create applications collaborating with Microsoft platforms. Application written in this language is processing to Common Intermediate Language (CIL) during compilation and, during execution, it is being compiled to processor native language. In this article, C# found its usage in creating multiplatform application (Xamarin.Forms).

D. Xamarin

Xamarin is platform which allows to create multiplatform applications intended for a couple of mobile operating systems. Since 2016, Xamarin has belonged to Microsoft. Intellection of this platform is to make one application instead of two for each of platforms. The code written in C# is mapped for native platform. Due to that, cross-platform applications are quite

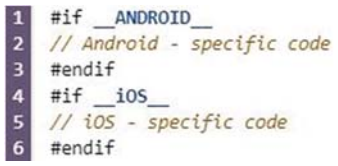
D. Dobrzański and W Zabierowski are with the Department of Microelectronics and Computer Science, Lodz University of Technology, Lodz, Poland (e-mail: wojtekz@dmcs.pl).

similar to dedicated ones. We can distinguish two types of multiplatform applications creating methods by using Xamarin:

- Xamarin.Native – in these applications logic layer is common but user interface (UI) is different for each of platforms;
- Xamarin.Forms – except for shared applications logic layer, interface is also common what allows to gain greater shared code rate.

In Xamarin.Forms code can be shared in two ways:

- Portable Class Library (PCL) – solutions made by it have two dedicated projects for each of platforms and one shared project – portable which is being included to dedicated projects while solution is being compiled.
- Shared Project – appropriate code blocks – platform-determined condition.



```
#if __ANDROID__
// Android - specific code
#endif
#if __iOS__
// iOS - specific code
#endif
```

Figure 1. Code sharing by using Shared Project (source: https://csharp-dev.pl/2017/03/31/xamarin-forms-okiem-poczatkujacego-wprowadzenie)

E. XML/XAML

XML is an universal tag language. It presents data as structure. It is not depended on any platform what implies that it can be used between different operating systems. XML is used in many technologic spaces. Except for presenting data it has usage in communication protocols which specialize in exchanging data (for example SOAP – Simple Object Access Protocol). In this article XML was used for creating interface in native application for Android.

XAML derives from XML. It is used for describing user interfaces on Microsoft platforms (for example: Windows Presentation Foundation (WPF). In this article XAML was used for creating user interface in cross-platform app in Xamarin.

III. Used Tools

A. Xcode

Xcode is integrated development environment (IDE) created by Apple. It contains everything what is necessary to develop applications intended for computers, smartphones, tablets and smartwatches with Apple operating systems. That tool is available for free from official app store. Except for applications source code, Xcode allows to create user interface and possibility of debugging or later testing created applications on virtual emulators. Xcode supports developing in Objective-C and Swift programming languages. In this article, it was used with dependency manager – CocoaPods.

B. Microsoft Visual Studio

Microsoft Visual Studio is an environment dedicated to Microsoft platforms. It was created also by Microsoft. It allows to creating console applications and applications having user interface. By using that tool, we can create applications intended for platforms like Windows, Windows 10 Mobile, Windows CE, XBOX console or Azure (Microsoft Cloud). Languages which we can use with that tool are, for instance, C#, Visual Basic, C++ and JavaScript. It is not difficult to configure that tool to work with Android and iOS SDK. Due to that it allows to use its embedded emulators. Otherwise, it has graphic manager – NuGet intended for managing extensions.

C. Android Studio

Android Studio is an environment which allows to create native applications intended for Android platform. That tool is based on IntelliJ and due to it we can write in Java. It is quite new tool because it was presented in 2013. It replaced Eclipse which had been default tool before. Now, Android Studio is a standard because its main purpose is to make faster and easier process of creating mobile applications. What is more, it is available at large scale – we can install it on operating systems like Windows, macOS or Linux. Applications made by using Android Studio can be debugged and their interface can be tested dependently on resolution and screen orientation. Applications are being built by usage of Gradle tool which also allows to add extensions.

IV. Mobile Applications and Tests Comparing Their Performance

For project needs, there were created three mobile applications which performance was comparing by five tests. The results of that tests will be presented in the next section. Below you can see short description of all used apps.

Native application dedicated to Android platform:

- Logic layer was created by Java language;
- User interface was created by XML;
- Android Studio was IDE used for programming;
- Target platform, on which application was prepared, is Android 7.1 Nougat (API 25);
- API 19 (4.4 KitKat) is minimal SDK necessary to run application;
- Gradle was used for extensions managing.

Native application dedicated to iOS platform:

- Xcode and Swift 3 were used for interface and logic layer creating;
- Application was made especially for iOS 10 but it can also work on previous versions and iOS 11 which, at this moment, is in beta tests.

Cross-platform application:

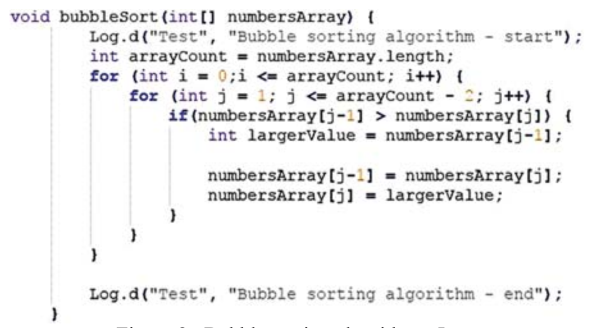
- It can be run on iOS and Android;
- Logic layer was created by object-oriented language – C#;
- User interfaces was created by XAML;
- Visual Studio (Community Edition) is tool which was used for programming in Xamarin.

Moreover:

- These applications are available on smartphones and tablets;
- Internet connection is essential to correct applications running;
- After tests, applications send request with results to appropriate server where data is stored in database and, furthermore, it is analyzed;
- Each of devices has its own unique id what allows to know what results we should compare with each other.

Tests prepared for applications performance comparing:

- Database operations. In that case, database based on SQLite engine was used. There was created 'Number' table which rows equal to values taken from mathematic operations (factorial for instance). Due to that, amount of inserting data to table operations is 3000. Afterwards, select operation is being performed. The same situation is with more than 2000 update and delete operations. At the end, the table is dropped and connection with database is made closed.
- Data sorting. For that test, bubble sort algorithm is used. It means that two consecutive items in sorting set are being compared. If appropriate items sequence is wrong, it is being changed. The algorithm ends when there is no items which sequence is needed to be changed. In this way, array with 50 000 items was being tested.

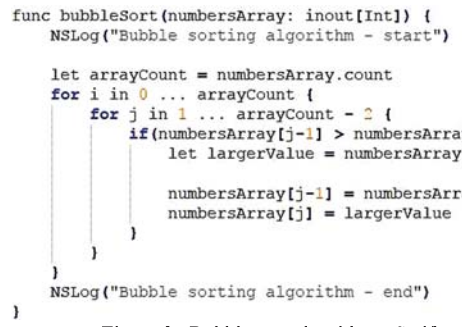


```java
void bubbleSort(int[] numbersArray) {
        Log.d("Test", "Bubble sorting algorithm - start");
        int arrayCount = numbersArray.length;
        for (int i = 0;i <= arrayCount; i++) {
            for (int j = 1; j <= arrayCount - 2; j++) {
                if(numbersArray[j-1] > numbersArray[j]) {
                    int largerValue = numbersArray[j-1];

                    numbersArray[j-1] = numbersArray[j];
                    numbersArray[j] = largerValue;
                }
            }
        }

        Log.d("Test", "Bubble sorting algorithm - end");
    }
```

Figure 2. Bubble sorting algorithm - Java.

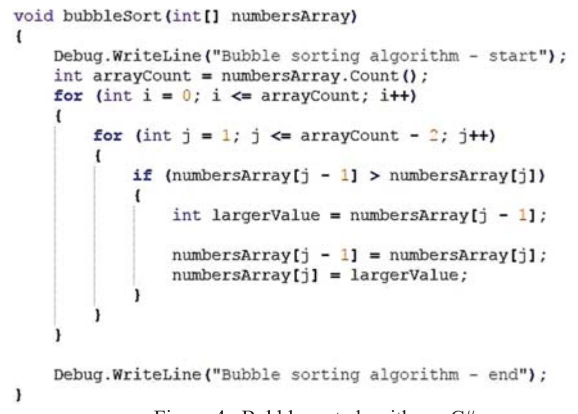


```swift
func bubbleSort(numbersArray: inout[Int]) {
    NSLog("Bubble sorting algorithm - start")

    let arrayCount = numbersArray.count
    for i in 0 ... arrayCount {
        for j in 1 ... arrayCount - 2 {
            if(numbersArray[j-1] > numbersArray[j]) {
                let largerValue = numbersArray[j-1]

                numbersArray[j-1] = numbersArray[j]
                numbersArray[j] = largerValue
            }
        }
    }
    NSLog("Bubble sorting algorithm - end")
}
```

Figure 3. Bubble sort algorithm - Swift.



```csharp
void bubbleSort(int[] numbersArray)
{
    Debug.WriteLine("Bubble sorting algorithm - start");
    int arrayCount = numbersArray.Count();
    for (int i = 0; i <= arrayCount; i++)
    {
        for (int j = 1; j <= arrayCount - 2; j++)
        {
            if (numbersArray[j - 1] > numbersArray[j])
            {
                int largerValue = numbersArray[j - 1];

                numbersArray[j - 1] = numbersArray[j];
                numbersArray[j] = largerValue;
            }
        }
    }

    Debug.WriteLine("Bubble sorting algorithm - end");
}
```

Figure 4. Bubble sort algorithm – C#.

- File reading and cryptographic algorithms. Tested file is built of more than 3000 rows, where each row is being encrypted by MD5, SHA1, SHA256, SHA512 algorithms. In case of used algorithm, encrypted sequence measure between 128 and 512 bits of length.
- Applying effects to picture. Downloaded image with 4000x4000 resolution is embedded in appropriate container (for example ImageView in Android). Then, three effects are applied on picture: gray scale, Gaussian blur and 180 degree rotation. After tests, modified image is embedded in container again.
- Interface test. It tests interface performance by refreshing data operation. There were used three items: label, text field and button. Afterwards, its values are being changed 10 000 times what finally equals to 30 000 operations.

The purpose of that tests was to judge if multiplatform solution – Xamarin.Forms – is effective against native solutions. It allows to answer question if that technology is worth the interest.

V. Application Comparison

A. Technical issues

Each of applications was created by different programming language and tools. Due to that fact, it was impossible to create them by the same way. Each of platforms has advantages and disadvantages.

Managing extended plugins enlarging functionality is appropriate example. During Android programming and creating application in Xamarin, developer may use built-in managers. In case of native application for iOS, Xcode allows to add libraries manually what is not comfortable at all. But there exists CocoaPods – external manager collaborating with Apple IDE. It means that we just have to configure dependencies – pods – in dedicated file called Podfile. Then, install them executing only one command.

Graphic interfaces was also created by another way. In case of native applications, developer could use graphic editors. Due to that, he could drag items on application screen. Live preview was also available, the same with editing by XML on Android. Both platforms has different view at items ordering. In Android, items are part of bigger layouts. In iOS, items are accommodating by constraints. In Xamarin it looks completely different than in previous ones. Visual Studio allows only to interfaces text editing what implies that process is much time-consuming. Developer can use just XAML language and view of application screen. It is also important to mention that Xamarin has many constraints when comparing with native solutions. An appropriate example constitutes round button with "START" caption – it requires custom renders to relevant displaying button on both platforms.

Multiplatform application purpose was to gain the highest rate of sharing code. In case of project application that rate is 60%. Microsoft declares that according to application functionality, it is possible to gain more than 90% of sharing code.

B. Application performance

- Database operations. In that situation significant time divergence on Android is seen. Native solution worked better with that test than cross-platform application. In case of iOS, results are close to each other with very little advantage of Xamarin app. Described discrepancy may be caused by libraries because it is hard to judge what mechanism were used. Native application high performance results from fact that system contains libraries collaborating with SQLite what makes them well optimized. It is worth mention that problem with testing in cross-platform application occurred on Android devices having version 7.0 minimum. Since that version, library files have not being added as default what means that multiplatform application has not worked correctly. The solution for that case is target change from API 25 version (Android 7.1) to API 23 (Android 6.1).

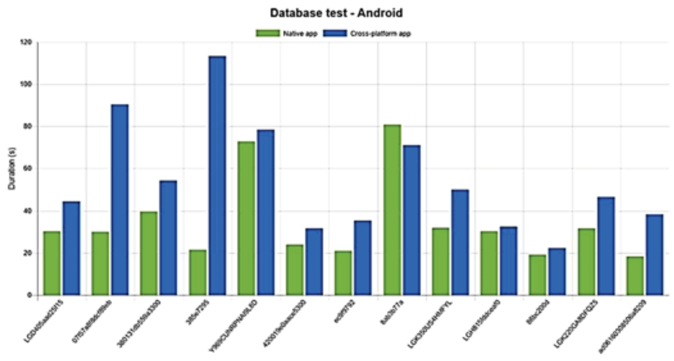


Figure 5. Database test - Android.

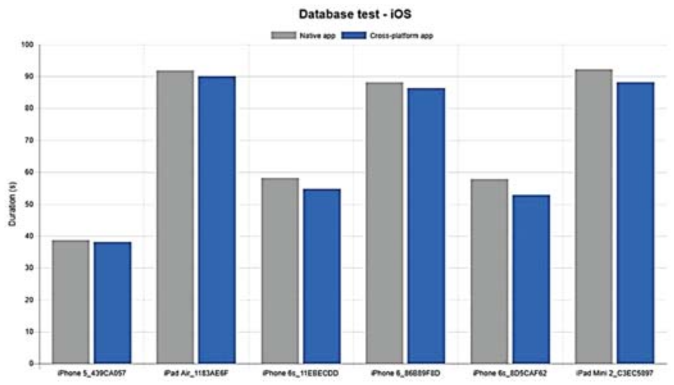


Figure 6. Database test - iOS.

- <u>Data sorting.</u> It is the most reliable test because there was not used any extensions. The only one used thing is basic languages functionality like conditions, loops and arrays. Time of data sorting is comparable in both platforms but the variances are lower on iOS platform. Android times differences (to native solution advantage) result from fact that it is necessary to run additional virtual machine (except JVM) for C# code.

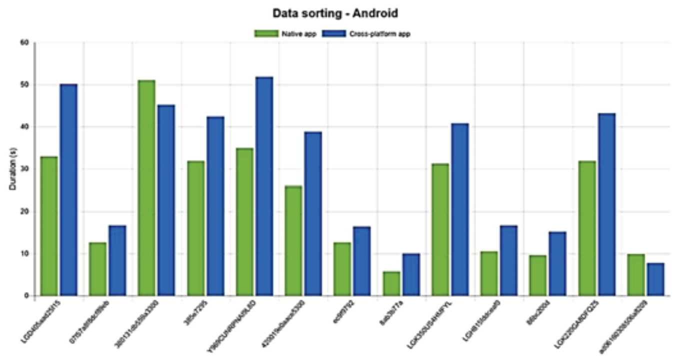


Figure 7. Data sorting - Android.

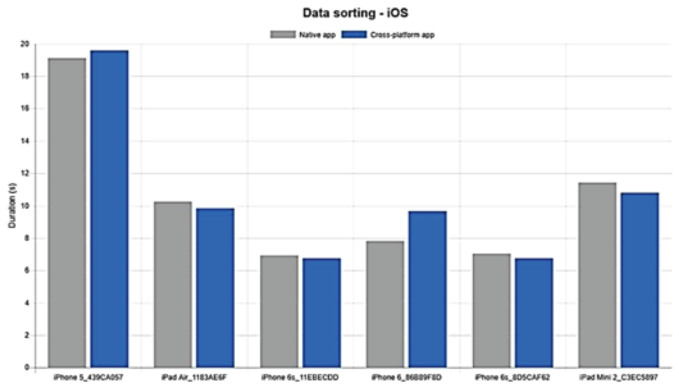


Figure 8. Data sorting - iOS.

- <u>File reading and cryptographic algorithms.</u> It characterizes variability of tests times on Android. It means that dependently on tested device, multiplatform application performs better or worse than native solution. It is seen that native solution worked better on LG smartphones. Devices manufacturers implement various solutions which main purpose is to improve CPU speed what impacts on performance and battery saving. In this case, it could blemish test result. On iOS platform, native solutions definitely beat cross-platform app. Differences can be caused by how cryptographic functions were implemented.

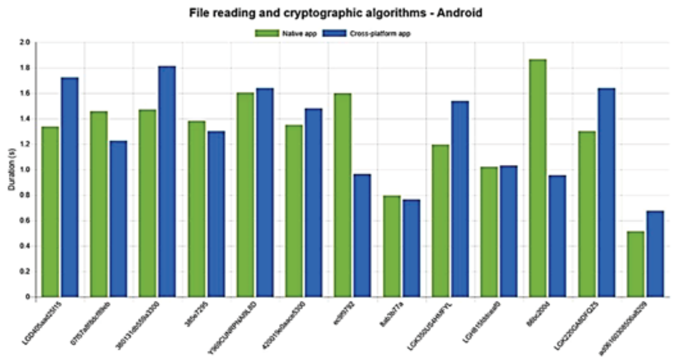


Figure 9. File reading and cryptographic algorithms - Android.

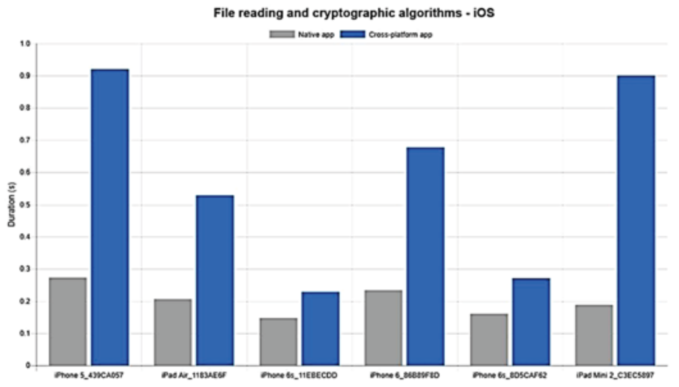


Figure 10. File reading and cryptographic algorithms - iOS.

- <u>Applying effects to picture.</u> In this case, differences are significant. Multiplatform solution is definitely more efficient that in native ones. It is caused by used libraries which mechanisms are completely distinct. In Xamarin, effects are used on containers displaying picture instead of just on picture. Due to that, operations perform so fast.

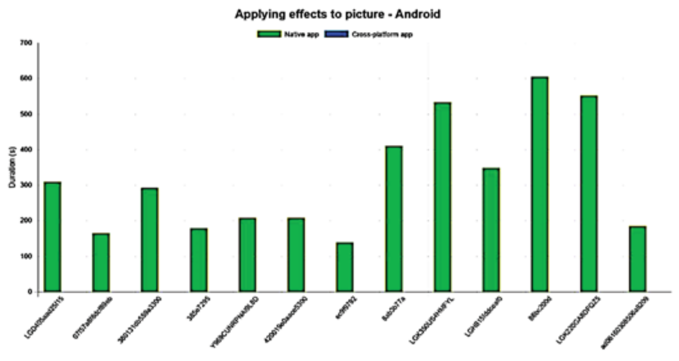


Figure 11. Applying effects to picture - Android.

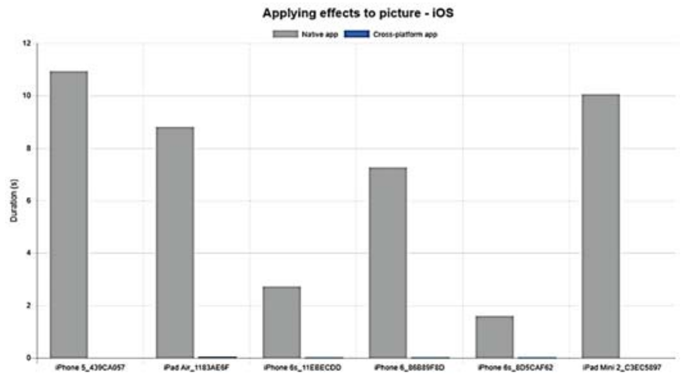


Figure 12. Applying effects to picture - iOS.

- Interface test. It is the most important test – it determines interface working, i.e. how application will be judged by potential user. There was used only interface elements, not libraries. Results are definitely better on native solutions where everything emerged the most optimized. Multiplatform application has to transform whole interface for being compatible with native determinants. For that reason, time of operation executing is longer. Worth mention also is fact, that interface is tested heavily where the most of available applications execute less operations. In that cases, differences should be imperceptible.

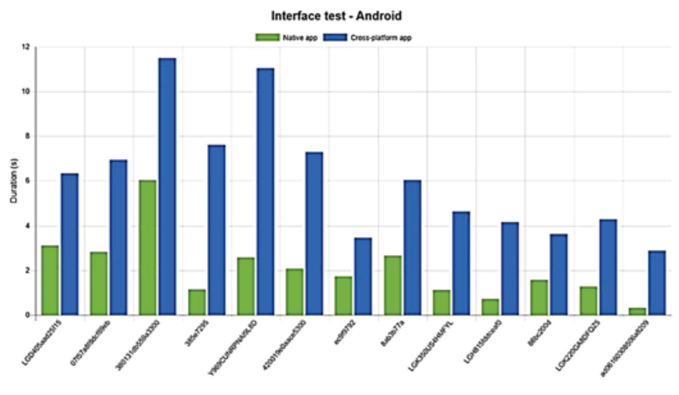


Figure 13. Interface test - Android.

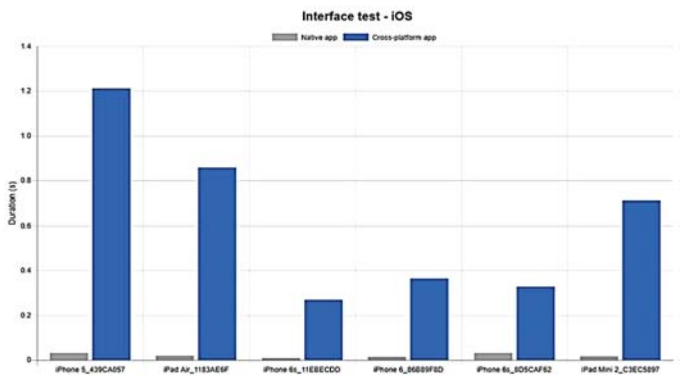


Figure 14. Interface test - iOS.

VI. Summary

The project purpose was to compare performance between native applications and multiplatform one. Tests and their results do not clarify problem – if multiplatform application can be as efficient or better than native technologies. It is important that Xamarin.Forms allows to make application writing process easier but it does not replace knowledge about native technologies. Taking into consideration executed tests, it is seen that Xamarin is not much slower. Moreover it is developing dynamically. Xamarin.Forms is worth considering platform while choosing appropriate technology for multiplatform application creating. However, all depends on functionality of creating application.

References

[1] Roman Wantoch-Rekowski, „Android w praktyce. Projektowanie aplikacji", Wydawnictwo Naukowe PWN SA, 2014.

[2] Andrzej Stasiewicz, „Android Studio. Podstawy tworzenia aplikacji", Helion, 2015.

[3] Carmen Delessio, Lauren Darcey, Shane Conder, „Android Application Development in 24 hours", Pearson 2014.

[4] Mattew Mathias, John Gallagher, „Swift Programming: The Big Nerd Ranch Guide", Pearson Education 2017.

[5] Charles Petzold, "Cross-platform C# programming for iOS, Android and Windows", Microsoft Press, 2016.

Dawid Dobrzański was born in Dąbrowa Górnicza, Poland, in 1993. He lives in Pabianice. He received the Master's degree in information technology from Lodz University of Technology (LUT), Poland in 2017. He is focused on web technologies and he is enthusiast of mobile technologies.

Wojciech Zabierowski (Assistant Professor at Department of Microelectronic and Computer Science Technical University of Lodz) was born in Lodz, Poland, on April 9, 1975. He received the MSc and PhD degrees from the Technical University of Lodz in 1999 and 2008, respectively. He is an author or co-author of more than 100 publications: journals and most of them - papers in international conference proceedings. He has been reviewer in dozens international conferences. He supervised more than 200 Eng. and Msc theses. He is focused on internet technologies, MEMS technologies and automatic generation of music. He is working in linguistic analysis of musical structure.