Wrocław University of Science and Technology
Faculty of Information and Communication Technology

Field of study: **IST**
Speciality: **CE**

MASTER THESIS

Title

Maciej Sroczek

Supervisor
dr inż. Dariusz Konieczny

Keywords: mobile app performance, native, cross-platform, Kotlin, Swift, Flutter, React Native, Ionic

WROCŁAW 2023

ABSTRACT

There are various aspects affecting the overall perception of quality of a mobile application, with performance being one of the most significant, especially from the perspective of the user. Having that in mind, it is crucial to understand the differences between the available mobile development approaches and in which use cases they are able to provide the highest value.

The purpose of this master's thesis was to perform a comparative analysis of the performance of mobile applications built using both native and cross-platform solutions. Exemplary applications were implemented with Kotlin, Swift, Flutter, React Native, and Ionic to be used as the environment for the experiments. The experiments provided results considering the selected performance metrics, e.g., CPU, memory, and power usage. The results were interpreted in order to find benefits and/or weaknesses for each studied solution, as well as to try to define optimal scenarios for their use.

STRESZCZENIE

Na ogólne postrzeganie jakości aplikacji mobilnej wpływają różne aspekty, przy czym wydajność jest jednym z najistotniejszych, zwłaszcza z perspektywy użytkownika. Mając to na uwadze, kluczowe jest zrozumienie różnic pomiędzy dostępnymi podejściami do wytwarzania aplikacji mobilnych i w jakich przypadkach użycia są one w stanie zapewnić najwyższą wartość.

Celem niniejszej pracy magisterskiej było przeprowadzenie analizy porównawczej wydajności aplikacji mobilnych zbudowanych z wykorzystaniem rozwiązań natywnych oraz cross-platformowych. Przykładowe aplikacje zostały zaimplementowane przy użyciu Kotlin, Swift, Flutter, React Native oraz Ionic, aby posłużyć jako środowisko do przeprowadzenia eksperymentów. Eksperymenty dostarczyły wyniki uwzględniające wybrane metryki wydajności, np. zużycie procesora, pamięci i energii. Wyniki zostały zinterpretowane w celu znalezienia korzyści i/lub słabości dla każdego badanego rozwiązania, a także próby zdefiniowania optymalnych scenariuszy ich wykorzystania.

CONTENTS

1. INTRODUCTION

Over the last few years, mobile devices, such as smartphones, tablets, or even smartwatches, could be seen as a rather essential part of human lives. This is confirmed by the big and still increasing number of over 7 billion mobile users across the world [26]. Because nearly 90 percent of users spend their time using different apps, the number of mobile app downloads is very high, at over 200 billion in 2020, which has a direct impact on the expansion of the mobile app market [27]. The growth of the mentioned market results in the evolution of different implementation methods for mobile development, with native and cross-platform being the most widely used.

Native mobile development implies creating software that can only be run on a specific platform (operating system), such as Android or iOS [9]. In order to do so, platform-specific tools must be utilized. In the case of Android, the programming language Kotlin may be used, and in the case of iOS, Swift. While it can be seen as a limitation, it provides some advantages, such as being able to use different elements of the system directly and, with that, maximize the achievable performance.

Cross-platform mobile development aims to eliminate the need to implement multiple versions of the same mobile app in order to make it available for users of different platforms. This method assumes the use of a single codebase that enables building the app for various operating systems. From the perspective of a user, each of them should perform and look as if they were implemented natively [16]. Such an approach quickly became popular among developers, including successful companies such as Meta and Google [14]. Some examples of cross-platform frameworks are Flutter, React Native, and Ionic.

All of the differences between the above-mentioned implementation approaches can make them more or less applicable in various scenarios. The selection of either native or cross-platform development method as well as the specific technology is really important because it may directly affect aspects such as development time, cost, and overall end-product quality. However, most of the popular solutions are constantly being updated, which leads to the necessity of recurrent comparative analysis in order to obtain the most up-to-date state of the art. Such knowledge will then be helpful to determine in which cases different development approaches and tools should be optimally used.

1.1. THE PURPOSE OF THE THESIS

The purpose of this master's thesis is to carry out research on the performance of selected cross-platform frameworks in comparison to each other and to native development methods. A number of metrics will be selected for analysis based on a literature review and personal experience. Exemplary applications will be prepared as an environment for the experiments. The results will form the basis for defining the advantages and downsides of developing single codebase cross-platform applications. Furthermore, optimal scenarios of use will be proposed for each studied framework and native technology.

1.2. THE SCOPE OF THE THESIS

To begin with, a problem analysis will be performed, which will result in defining the specifications for the experiments to be carried out. Conducted experiments will provide data for further analysis, which will be organized into groups based on the experiment environments, studied platforms, and frameworks. The results will be interpreted in the context of quality and possible optimal use-cases for implementing mobile applications using the selected frameworks and native methods. All of the research must be documented.

1.3. THE STRUCTURE OF THE THESIS

The thesis has been divided into seven chapters. The first chapter aims to provide a brief introduction to the topic. The second chapter contains the literature review, which helps to present the relevancy of the subject matter as well as provide knowledge necessary for the further work. In the third chapter, the research method is mostly defined based on the literature review. The fourth chapter concerns the implementation of testing environments and the realization of prepared experiments. In the fifth chapter, the results from performed experiments are visualized and described. The sixth chapter contains the discussion that emerged from the experiment results and the conclusions drawn. Finally, in the last chapter, the complete work is summarized, and key takeaways are featured. Additionally, limitations are explained, and suggestions for future work are proposed. The dissertation closes with a bibliography as well as lists of figures and tables.

2. LITERATURE REVIEW

2.1. RELATED WORK

2.2. MOBILE DEVELOPMENT RELEVANCY

2.3. DEVELOPMENT APPROACHES

The definition of mobile development can be interpreted in a variety of ways. It can be seen as a broad process of implementing a mobile application, starting with planning and designing and finishing with testing, releasing and maintaining. A more software-oriented definition is that mobile development simply refers to implementing an application for mobile devices by coding it using a selected technology stack [17]. In this thesis, the latter definition is assumed.

Mobile development can become a complex task considering the variety of devices and platforms existing in the market. There are many different approaches available and in order to choose one over another the mobile application requirements should be taken into account as well as target platforms and devices, development and time costs [28].

In this chapter, there are presented selected popular approaches to mobile application development. Each of them is described mainly in the context of architecture, technology stack and tools, platforms supported and possible advantages or disadvantages.

2.3.1. Native mobile development

Native mobile development encompasses building mobile applications that can only be implemented using a platform-specific programming language and deployed to a single operating system [29]. Such an approach brings with it the necessity for creating and maintaining multiple codebases and with that possibly multiple development teams [18]. The number of distinct codebases does not simply equal to the number of target platforms, as different versions of a single platform may require to be implemented independently [8]. Hence, development costs are high from the viewpoint of financing and time.

Native mobile applications are closely integrated with the operating system through using target platform's components [20][22] and most recent features [12]. For that reason, at times they can be referred to as "embedded" [11] and in theory should provide the maximal performance. Furthermore, because native apps are developed according to the operating system's guidelines, such as Material design system for Android and Cupertino for iOS, they are naturally easy to use for users accustomed to that specific platform.

Since almost a decade, the mobile operating system market has been dominated by Android and iOS, reaching 99,3% in March 2023 [23]. For this reason, in the context of this thesis only the above-mentioned operating systems are being taken into consideration.

As can be seen in the Figure 2.1, in case of iOS, almost 90% of devices are running either the most or second-most recent major version of the operating system. Therefore, when targeting the Apple's system, probably a single codebase would be enough.

However, in case of Android, there is a high level of market fragmentation, as nearly 20% of smartphones or tablets are running older versions released as far as in 2015 (Figure 2.2). Because there are limitations such as deprecation of code commands and API (Application Programming Interfaces) behavior changes between distant versions, multiple codebases may be chosen to be maintained separately per a single mobile application. Another issue is the fact, that device manufacturers are able to apply various modifications to the operating system which can lead to errors occurring only on those devices [10], causing difficulties for developers. Because Apple is the exclusive manufacturer of devices running iOS, they do not suffer from such a problem.

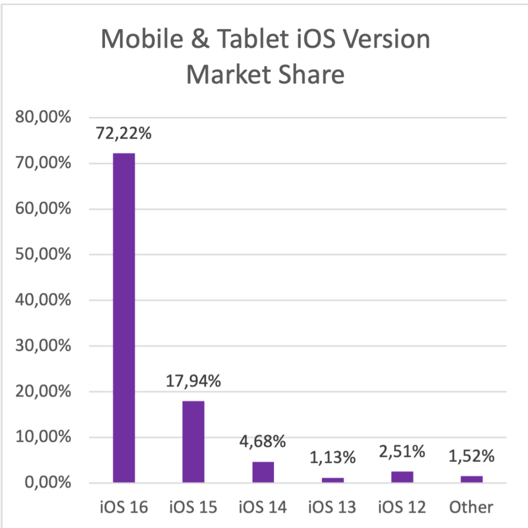

Fig. 2.1. iOS version market share (Source: Own work based on [25])

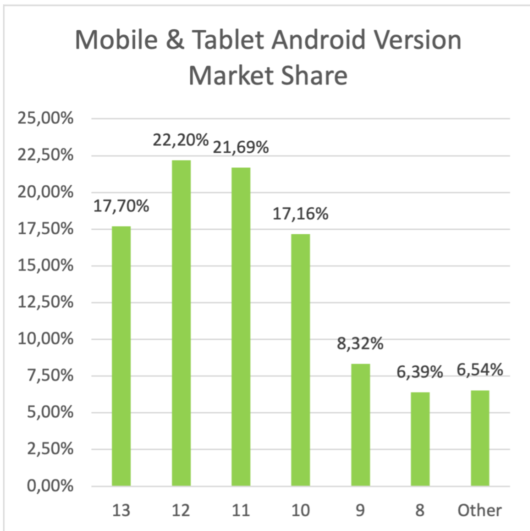

Fig. 2.2. Android version market share (Source: Own work based on [24])

2.3.1.1. Android

Android is an open-source operating system developed by the Open Handset Alliance and Google to run mainly on mobile devices such as smartphones and tablets but also TVs and cars [1] [10]. It is based on the Linux kernel and has a multiple-component structure [21], as can be seen in the Figure 2.3. Each part takes responsibility for different tasks, e.g., Android Runtime provides optimized garbage collection and View System (included in

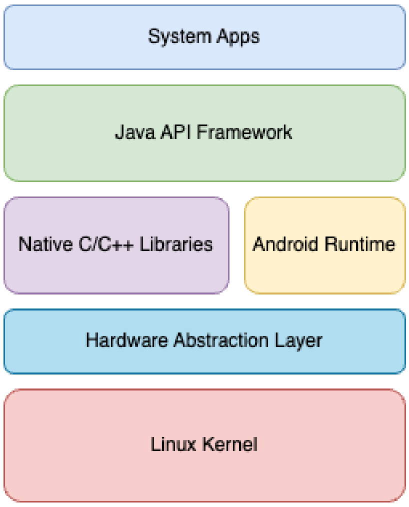

Fig. 2.3. Android architecture (Source: Own work based on [2])

Java API Framework) enables developers to implement the user interface layouts using various elements such as lists and buttons [2].

It is not necessary to use an IDEs (Integrated Development Environment) to build most software. Despite that, most programmers tend to reach out to them because of the guaranteed development comfort and productivity increase. Android Studio is the primary IDE for native mobile development of Android applications. It is created on top of IntelliJ's IDEA and provides numerous features such as Gradle, advanced debugging tools and profilers, etc [3].

For many years, Java has been the official language for Android development. However, since Google established Kotlin as the default choice in 2019 [5], over 60% of programmers have switched to it [4]. Furthermore, data shows that almost 90% of the Google Play Store Top 500 USA mobile apps have been developed with it [15]. Kotlin's popularity is certainly going to grow in the upcoming years, considering the undeniable benefits it brings. Still, there are some scenarios in which Java could remain the first choice.

Table 2.1 shows that all the features necessary for Android application development, such as e.g., Android SDK or AndroidX support, are fully provided by both Java and Kotlin. Furthermore, the latter introduces additional advantages in the form of enabling the usage of Jetpack Compose toolkit and even the ability to create multi-platform projects (Kotlin Multiplatform is currently only accessible in Beta version [13]).

Java is a high-level object-oriented programming language introduced as far back as 1995. It is one of the most popular languages in the world which is regularly updated, reaching version 20 this year. However, in the context of Android development the supported versions are 8 and 11, with the latter requiring high Android API version in order to use all the offered elements (although upgraded API desugaring announced in February this year broadens the range of libraries available without increasing the app's minimum supported API level [6]). The advantages of Java mainly result from the fact it is present for a very long time. During the last 30 years it gathered a big community of developers with high-level experience. Therefore, it may be easier to form a competent team for the project. Moreover, there are numerous applications that had been created with Java which owners might not seek for a migration, which as a result maintains Java's importance in the market [15].

On the other hand, Kotlin offers many assets because of which it has replaced Java as an official first-choice language for Android development, as mentioned before. Kotlin is a comparatively recent programming language introduced in 2016 by JetBrains. First and foremost, it is fully interoperable with Java and therefore, it is possible to call Kotlin code inside Java code and vice versa. Secondly, Kotlin's syntax is very concise and null-safe, thus increasing the speed of development and reducing the project's code lines and lowering the possibility of mistakes. For those reasons, implementation of new apps using Kotlin is fairly straight-forward and comfortable from the point of view of developers. Moreover, the migration of existing projects from Java to Kotlin is uncomplicated and can lead to size decrease and simplification of the codebase with much smaller Null Pointer Exception occurrence in runtime [5].

Table 2.1. Java and Kotlin comparison (Source: Own work based on [5])

Feature	Java	Kotlin
Platform SDK support	Yes	Yes
Android Studio support	Yes	Yes
Lint	Yes	Yes
Guided docs support	Yes	Yes
API docs support	Yes	Yes
AndroidX support	Yes	Yes
AndroidX Kotlin-specific APIs (KTX, croutines, ...)	N/A	Yes
Online training	Best effort	Yes
Samples	Best effort	Yes
Multi-platform projects	No	Yes
Jetpack Compose	No	Yes
Compiler plugin support	No	Yes (Kotlin Symbol Processing API)

Since Kotlin has already been established as the preferred programming language for Android development, all considerations in the scope of this thesis will be limited to it rather than Java.

One of the possibilities acquired when selecting Kotlin for Android development is the ability to use Jetpack Compose. It is a powerful toolkit used for building User Interface layouts introduced in 2021 with a goal improve that process compared to the previous XML approach [7]. The main difference between the above-mentioned methods is the fact that they represent declarative and imperative approach, respectively. The former greatly reduces the amount of boilerplate code improving readibility as well as build time and therefore, increases development efficiency. Additionally, just as Kotlin offers interoperability with Java, Jetpack Compose provides the same in regards to XML. In short, XML approach implies the creation of layouts in XML markup files and later referencing them in the code while implementing the behavior. On the other hand, Jetpack Compose makes use of prebuilt components and intuitive state management [19].

2.3.1.2. iOS

Include Cupertino!!!

2.3.2. Web development

2.3.3. Cross-platform mobile development

Most cross-platform frameworks require a middle layer that connects the app with the system and translates the commands to be natively called. This is considered to be a possible root of performance decrease [8].

2.3.3.1. Flutter

2.3.3.2. React Native

2.3.3.3. Ionic

2.3.3.4. Comparison

tutaj tabelka z frameworkami I w kolumnach rozne elementy, np., supported platforms, itd.?

Table 2.2. Cross-platform frameworks comparison (Source: Own work based on ...)

Framework / Element	Flutter	React Native	Ionic
Initial release	2017		
Current stable version			
Implemented with	C, C++, Dart		
Supported platforms	Android, iOS, Web, Windows, macOS, Linux		
Supported IDEs??			
Programming language	Dart		
Rendering	Canvas drawing	Native platform components	Native platform components

2.4. EVALUATION OF CROSS-PLATFORM FRAMEWORKS

2.5. PERFORMANCE MEASUREMENT

2.5.1. Mobile development

2.5.2. Web development

3. RESEARCH METHOD

3.1. PERFORMANCE METRICS

3.1.1. Mobile environment

3.1.2. Web environment

3.2. RESEARCH SCENARIOS

3.3. TESTING TOOLS

3.3.1. Mobile environment

3.3.2. Web environment

3.4. TESTING DEVICES

4. IMPLEMENTATION OF SAMPLE APPLICATIONS

4.1. APP 1???

5. RESEARCH RESULTS

5.1. MOBILE ENVIRONMENT

5.1.1. App 1???

5.2. WEB ENVIRONMENT

5.2.1. App 1???

6. DISCUSSION

6.1. MOBILE ENVIRONMENT

6.2. WEB ENVIRONMENT

7. SUMMARY

7.1. CONTRIBUTION

7.2. LIMITATIONS

7.3. SUGGESTIONS FOR FUTURE WORK

Flutter Impeller

BIBLIOGRAPHY

[1] Android, `https://www.android.com/what-is-android/`. Accessed 24 Apr. 2023.

[2] Android Developers, `https://developer.android.com/guide/platform`. Accessed 24 Apr. 2023.

[3] Android Developers, `https://developer.android.com/studio/intro`. Accessed 25 Apr. 2023.

[4] Android Developers, `https://developer.android.com/kotlin`. Accessed 25 Apr. 2023.

[5] Android Developers, *Android's Kotlin-first approach*, `https://developer.android.com/kotlin/first`. Accessed 25 Apr. 2023.

[6] Android Developers, *Api desugaring supporting Android 13 and java.nio*, `https://android-developers.googleblog.com/2023/02/api-desugaring-supporting-android-13-and-java-nio.html`. Accessed 26 Apr. 2023.

[7] Android Developers Blog, *Jetpack compose is now 1.0: announcing android's modern toolkit for building native ui*, `https://android-developers.googleblog.com/2021/07/jetpack-compose-announcement.html`. Accessed 27 Apr. 2023.

[8] AppDynamics, *Mobile app performance explained*, `https://www.appdynamics.com/media/uploaded-files/mobileapp.pdf`. Accessed 23 Apr. 2023.

[9] Competition Markets Authority, *Mobile ecosystems. Market study final report.* Updated 4 Aug. 2022.

[10] Crha, J., *Comparison of Technologies for Multiplatform Mobile Applications Development*, Master's thesis, Masaryk University. 2021.

[11] Fentaw, A.E., *Cross platform mobile application development: a comparison study of React Native Vs Flutter*, Master's thesis, University of Jyväskylä. 2020.

[12] Hjort, E., *Evaluation of React Native and Flutter for cross-platform mobile application development*, Master's thesis, Åbo Akademi University. 2020.

[13] Kotlin, `https://kotlinlang.org/docs/multiplatform.html`. Accessed 26 Apr. 2023.

[14] Kotlin, *The six most popular cross-platform app development frameworks*, `https://kotlinlang.org/docs/cross-platform-frameworks.html`. Accessed 18 Apr. 2023.

[15] Krusche Company, *Kotlin vs Java: strengths, weaknesses and when to use which*, `https://kruschecompany.com/kotlin-vs-java/`. Accessed 26 Apr. 2023.

[16] Lachgar, M., Hanine, M., Benouda, H., Ommane, Y., *Decision framework for cross-platform mobile development frameworks using an integrated multi-criteria decision-making methodology*, International Journal of Mobile Computing and Multimedia Communications. 2022, Volume 13, 1.

[17] Microsoft, *What is mobile application development?*, `https://azure.`

microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-mobile-app-development/#definition. Accessed 22 Apr. 2023.

[18] Mota, D., Martinho, R., *An approach to assess the performance of mobile applications: A case study of multiplatform development frameworks*, Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE,. 2021.

[19] Mustafa Yiğit, *Say hello to Jetpack Compose and compare with XML*, https://blog.kotlin-academy.com/say-hello-to-jetpack-compose-and-compare-with-xml-6bc6053aec13. Accessed 27 Apr. 2023.

[20] Olsson, M., *A Comparison of Performance and Looks Between Flutter and Native Applications*, Bachelor's thesis, Blekinge Institute of Technology. 2020.

[21] Shaheen, J.A., Asghar, M.A., Hussain, A., *Android os with its architecture and android application with dalvik virtual machine review*, International Journal of Multimedia and Ubiquitous Engineering. 2017, Volume 12, 7.

[22] Singh, M., Shobha, G., *Comparative analysis of hybrid mobile app development frameworks*, International Journal of Soft Computing and Engineering (IJSCE). 2021, Volume 10, 6.

[23] Statcounter, *Mobile operating system market share worldwide*, https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202303-202303-bar. Accessed 23 Apr. 2023.

[24] Statcounter, *Mobile tablet android version market share worldwide*, https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide/#monthly-202303-202303-bar. Accessed 23 Apr. 2023.

[25] Statcounter, *Mobile tablet ios version market share worldwide*, https://gs.statcounter.com/ios-version-market-share/mobile-tablet/worldwide/#monthly-202303-202303-bar. Accessed 23 Apr. 2023.

[26] Statista, *Forecast number of mobile users worldwide 2020-2025*, https://www.statista.com/statistics/218984/number-of-global-mobile-users-since-2010/. Accessed 18 Apr. 2023.

[27] TechJury, *55+ jaw dropping app usage statistics in 2023*, https://techjury.net/blog/app-usage-statistics/#gref. Accessed 18 Apr. 2023.

[28] Velvetech, *5 key mobile development approaches*, https://www.velvetech.com/blog/5-key-mobile-development-approaches/. Accessed 22 Apr. 2023.

[29] Vilček, T., Jakopec, T., *Comparative analysis of tools for development of native and hybrid mobile applications*, 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). 2017.

LIST OF FIGURES

LIST OF TABLES