A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks

Article *in* International Journal of Security and Privacy in Pervasive Computing · January 2021

DOI: 10.4018/IJSPPC.2021010105

3 authors, including:

Spela Pecnik
University of Maribor
11 PUBLICATIONS **32** CITATIONS
SEE PROFILE

Iztok Fister jr.
University of Maribor
234 PUBLICATIONS **4,938** CITATIONS
SEE PROFILE

A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks

Blaž Denko,
University of Maribor, Slovenia
Špela Pečnik,
University of Maribor, Slovenia
Iztok Fister Jr.,
University of Maribor, Slovenia

ABSTRACT

The number of users of smart mobile devices is growing every day. Because of the popularity of using mobile devices, it is important for business stakeholders to develop mobile applications targeting all mobile platforms in order to ensure that the number of users is as large as possible. One possible solution is the creation of hybrid mobile applications. These are applications that combine the properties of web and native mobile applications, and their main advantage is compatibility with multiple mobile operating systems. This paper presents the results of very comprehensive experiments that involved the use of various hybrid mobile development frameworks that were tested under different scenarios. Experiments revealed that the performance of hybrid applications in different scenarios varies considerably, although the results of these applications were comparable to those that were achieved in the experiment with the native application.

Keywords. Android, Flutter, Framework, Hybrid Mobile App, Ionic, NativeScript, React Native

INTRODUCTION

Nowadays, the number of users of smart mobile devices worldwide exceeds 3 billion, and the growth trend of their use will continue in the coming years. This indicates the number of new mobile phone users, which increased from 2018 to 2019 by 300 million. Mobile devices still have great market potential in the markets of the most populated countries, such as China and India (Statista, 2019a). In India, the share of smart mobile device use in 2018 was only 24 % (Statista, 2019b). Mobile operating systems are adapted to the capabilities of the devices, so companies and developers need to choose the right approach for building applications to ensure as many potential users as possible (Fister Jr. et al., 2018; Martinez & Lecomte, 2017). The market is currently dominated by mobile devices with two operating systems - Android and iOS, which, together,

Citation details: Blaž Denko, Špela Pečnik, Iztok Fister Jr. A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks. International Journal of Security and Privacy in Pervasive Computing. 2021, DOI: 10.4018/IJSPPC.2021010105

cover more than 98 % of market share (StatCounter, 2020). The program codes of native mobile applications running on the Android and iOS platforms are incompatible with each other, due to the use of different tools and programming languages when creating an application (Majchrzak et al., 2017). The consequences of this are separate projects and different development environments for developers who create the same applications for both operating systems (Biørn-Hansen et al., 2017). One possible solution is to create hybrid mobile applications. These are applications that combine the features of web and native mobile applications, and their main advantage is compatibility with multiple mobile operating systems. These applications use dedicated frameworks and interfaces that allow the use of native functionalities of mobile devices while simplifying the development of applications using modern web technologies (Vilcek & Jakopec, 2017). Their main drawbacks are lower responsiveness and possible poorer user experience, due to the non-use of native components (Delia et al., 2017). Therefore, it is important for companies and developers to choose the appropriate technology or framework with the least number of shortcomings when developing hybrid mobile applications.

The contribution of this paper is twofold. Firstly, our mission is to present the comparison of the development of hybrid mobile applications, as well as the theoretical foundations of the selected hybrid frameworks for their development. Secondly, the selected hybrid frameworks are used for the development of simple mobile applications and their evaluation under different scenarios.

The structure of this paper is as follows: Section 2 presents a theoretical insight into hybrid mobile applications. This Section is followed by a presentation of selected frameworks for the development of hybrid mobile applications (Sec. 3). In Section 4, we present the selected criteria for performing the experiment, the application development plan, and describe the development of an individual application. In Section 5, we present the results obtained during the experiment, and in the final Section 6, we summarize briefly our findings in the development of applications and the implementation of the experiment, with which we compared their performance.

HYBRID MOBILE APPLICATIONS

Hybrid applications combine the features of native and web technologies, and enable compatibility with multiple mobile operating systems (Vilcek & Jakopec, 2017). The most important features of hybrid applications are the common program code and the development environment, which differ in the case of the development of native applications (Delia et al., 2017). Such an application development concept is called “write once run anywhere”, and refers to the ability of an application to run on different operating systems (Huynh et al., 2017). The development of hybrid mobile applications is carried out using modern technologies that differ depending on the selected framework in which we create the application (Ma et al., 2018). For example, those built in the Ionic framework use the Apache Cordova interface and run within the web view, while applications built in the React Native framework use an interface that allows the application code to be mapped to native application components (Griffith, 2017; Eisenman, 2017).

Most frameworks for developing hybrid mobile applications are based on web technologies such as HTML5, CSS, JavaScript, and web frameworks. Despite the use of web technologies, these applications can also work offline and store data in local databases such as SQLite and RealM, which are installed on mobile devices as part of the application (Huynh et al., 2017; Parihar, 2020). Like native applications, hybrid applications can also access many functionalities (data storage, notifications, contacts, gallery, etc.) and hardware (motion and vibration sensor, camera, navigation, gyroscope, etc.) built into modern mobile devices. Hybrid mobile applications can also

be published in the online stores of individual mobile operating systems, from which users install them on their mobile devices (Griffith, 2017).

FRAMEWORKS FOR MOBILE APPLICATION DEVELOPMENT

In the following subsections, we present the four selected frameworks for mobile application development.

Ionic

Ionic is an open-source framework that enables the development of hybrid mobile applications using web technologies such as HTML, CSS and JavaScript. These are specially adapted and optimized for operation on mobile devices, which allows the development of highly interactive mobile applications. Applications built in the Ionic framework run within a web view and can use the functionalities, camera, and measuring devices built into modern mobile devices (Griffith, 2017) (Ravulavaru, 2017).

React Native

React Native is also an open-source framework for developing hybrid mobile applications based on the React.js framework for developing front-end web applications (Masiello & Friedmann, 2017). It differs from the Ionic framework in the use of native components and that it is not implemented in a web view. The program code of an application, written in JavaScript, is mapped into the native components of the user interface. This improves performance, and adds a sense of authenticity compared to web-based applications (Eisenman, 2017). Like applications built in the Ionic framework, these applications also have access to the functionalities, camera and sensors of the device (Masiello & Friedmann, 2017).

NativeScript

NativeScript is another open-source framework for mobile application development based on the JavaScript programming language, in which the application program logic is constructed. Application view components are defined in XML and can be formatted using CSS (Branstein & Nick, 2017). It allows developers to use JavaScript without other program libraries, or one of the two modern web frameworks Angular and Vue.js (NativeScript, 2020). Like applications built in the Ionic and React Native frameworks, these applications can also access the functionalities, sensors, and other measuring devices built into mobile devices (Branstein & Nick, 2017).

Flutter

Flutter is a framework based on the Dart programming language. Like the rest of the aforementioned frameworks, it is open source, and allows the development of mobile applications by writing only one version of the application, which runs on Android and iOS mobile operating systems (Flutter, 2019). Applications created in this framework are based on visual widgets. These act as individual components that together form the user interface of the application. Like the other frameworks mentioned earlier, this one also has access to the functionalities and measurement

tools built into mobile devices (Windmill, 2019).

COMPARISON OF FRAMEWORKS FOR HYBRID APPLICATION DEVELOPMENT

In order to compare the selected frameworks for hybrid application development, we designed an experiment which consisted of the following stages:

- Definition of the goals,
- Application development planning,
- Development of the application in the selected frameworks,
- Creation of a test environment.

All stages are outlined in detail in the next subsections.

Definition of Goals

The design of the experiment was divided into four phases. Before the implementation of applications, we defined various criteria and parameters, the values of which were obtained during these phases. Some of these parameters are application-specific, while others are specific to the selected criteria. In the first phase, we wanted to capture the basic properties related to the entire application. For this purpose, we measured the following parameters: Time of creating installation packages (APK), size of installation packages and installed applications, application installation time, and the time of the first (cold) and warm launch of applications. In the second phase, we tested the responsiveness of the application in the case of displaying a large amount of data and images. The parameters we measured at this stage were the percentage of CPU usage and the amount of working memory consumption in the case of vertical scrolling around the screen. In the third phase, we tested the performance of applications in the implementation of more complex algorithms, according to the previously selected criteria. We chose three conventional algorithms: The Bubble Sort, the Fibonacci sequence number search algorithm, and the Sieve of Eratosthenes algorithm. The parameters we measured during the execution of the algorithms were the execution time and the percentage of CPU usage. In the last, fourth phase, we included a criterion based on which we measured the performance of applications in the case of longer repetitive operation in real-time. We tested this by performing a classic stopwatch continuously over a period of time. During its operation, we measured the percentage of CPU usage.

Application Development Planning

After determining the goals of the individual phases, we began to plan the structure of the application in order to classify the selected criteria into meaningful layouts within our application. We decided to distribute the criteria on three content-separated screens, which will, together, form the entire structure of the application. To switch between individual screens, we created a fixed bottom bar navigation. By creating a high-fidelity wireframe application (Figure 1), we have unified the look of the user interfaces of applications created in different hybrid frameworks.

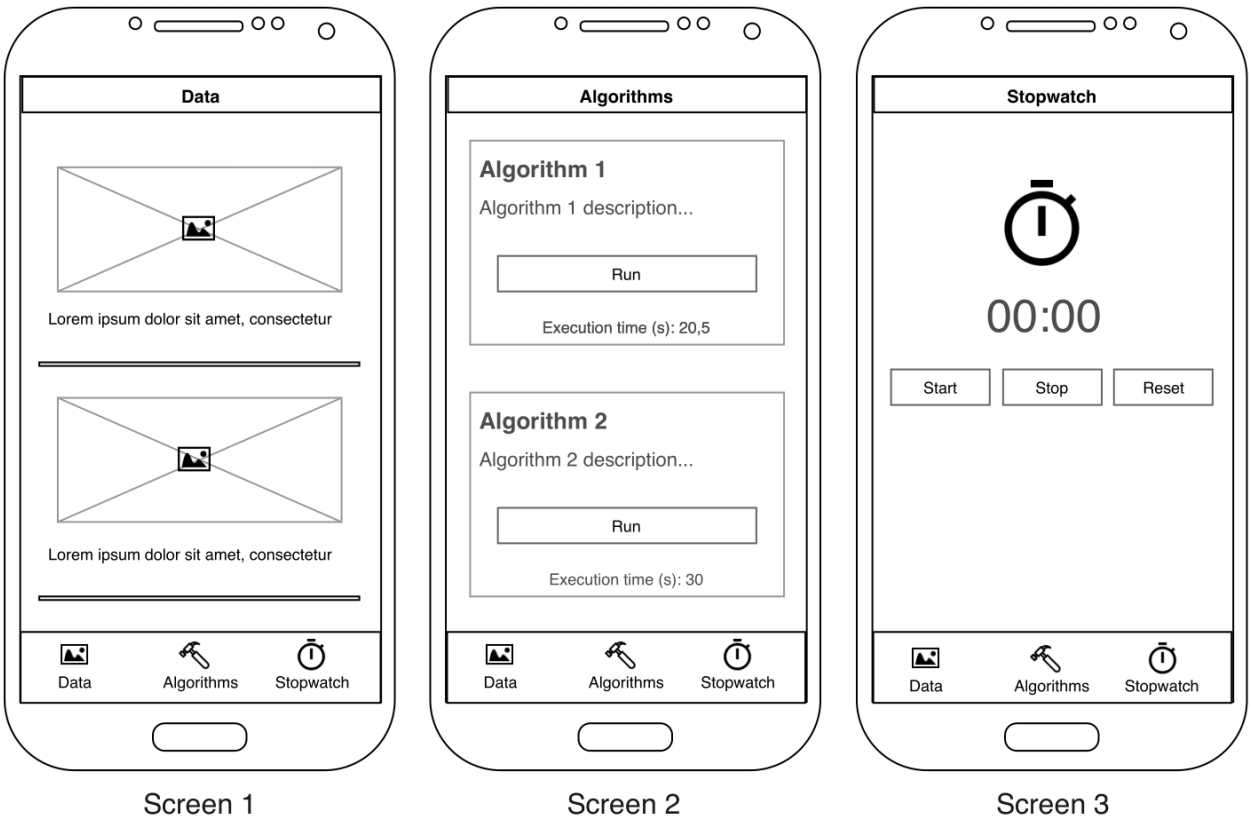


Figure 1: Skeleton of the application user interface

Development of Application in Selected Frameworks

First, we started creating the user interface of the application. For its implementation, we used pre-prepared components offered by individual frameworks. If these were not available, we used additional freely available libraries with preprepared components. The appearance of the individual components composed the user interface and was adapted to the skeleton, which was created in the application development planning phase.
In the next step, we created the program logic of the application. The logic was the same for applications created in hybrid frameworks, based on the use of the JavaScript programming language. The only difference was in the basic structure of the functions, their triggering and data output. The program structure of test algorithms, stopwatch and functions for obtaining data using a web service was the same for all applications. We only had to change and adjust the program logic and code structure slightly for the Flutter framework, which is based on the Dart programming language. When we finished creating the applications, we packed them into APK installation packages. These were then used as part of the experiment.

Creation of the Test Environment

Before starting the experiment, we established all the necessary environments and tools. For the

measurements, creation of APK installation packages and obtaining the results, we used the development environment Android Studio. For conducting the experiment, we used three mobile devices with the Android operating system (Table 1). We reset the mobile devices to the factory settings to ensure consistent performance and, thus, greater comparability of the results in the experiment.

Table 1: Mobile devices used in the experiment.

Mobile device	Operating system	Basic specifications
Samsung Galaxy S10e (2019)	Android 9 One UI 1.1 (PIE)	Chipset: Snapdragon 855, octa-core, RAM: 6 GB, Internal memory: 128 GB
Huawei Mate 10 Pro (2017)	Android 9 EMUI 9.1 (PIE)	Chipset: Kirin 970, octa-core, RAM: 6 GB, Internal memory: 128 GB
Xiaomi REDMI 5A (2017)	Android 7.1 Nougat	Chipset: Snapdragon 425, quad-core, RAM: 2 GB, Internal memory: 16 GB

The values of the parameters in the specific phases were obtained in two ways. The first is that we read the values from the applications by displaying their values on the screen of the device (for example, the execution time of the algorithms). Another way with which we obtained the parameter values is by using a special tool and logs in Android Studio. The tool we used is called Profiler, and displays real-time data on application resource consumption (working memory, CPU, battery). Table 2 defines the ways in which we obtained parameter values in specific phases.

Table 2: Methods of obtaining values for parameters in individual phases

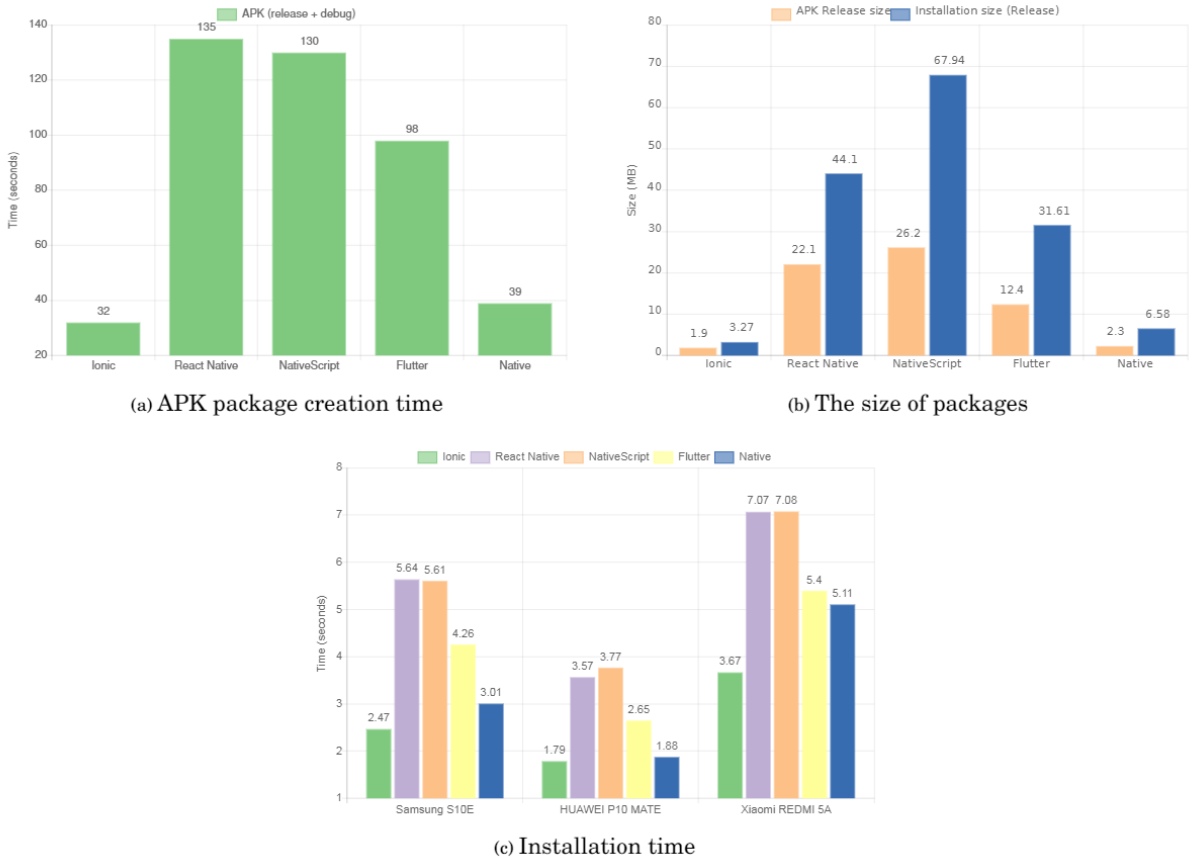
Phase	Parameter name	Method of obtaining value
First phase	Time of creating installation packages (APK)	logcat (after creating was completed)
First phase	Size of installation packages and installed applications	Obtained from the APK file and installed app
First phase	Application installation time	logact (after installation on the device was completed)
First phase	Time of the cold launch and warm launch of applications	logcat: displayed
Second phase	CPU usage in the case of vertical scrolling around the screen	Profiler: CPU
Second phase	RAM consumption in the case of vertical scrolling around the screen	Profiler: Memory
Third phase	Execution time of algorithms	Obtained from the device’s screen
Third phase	CPU usage during the time of algorithms execution	Profiler: Memory

Fourth phase	CPU usage during continuously performing a stopwatch over a period of time.	Profiler: CPU

THE RESULTS OF THE EXPERIMENT

Phase 1

As mentioned in previous chapters, one of the parameters measured in the first phase was the APK package build speed. Graph 2a shows that the time was the shortest for the application made in the Ionic framework (32 s). The second fastest time was achieved by the native application, followed by applications created in the Flutter (91 s) and NativeScript (130 s) frameworks. It took the longest to create a package for application built in the ReactNative framework (135 s).
Depending on the size of the built packages (Graph 2b), the applications follow each other in the same order, with one difference, namely, that here, the size of the NativeScript package is the largest (67,94 MB), and the second largest is from React Native (44,1 MB). The installation time was also the fastest for the application in the Ionic framework on the device Huawei P10 Mate. The worst installation times were achieved again by applications with React Native and NativeScript frameworks. In general, however, the longest installation times were on the device Xiaomi REDMI 5A (Graph 2c).
Graph 3a shows the time of the first (cold) launch of applications on devices. Among the hybrid applications, the fastest running application was the one created in the React Native framework on the Samsung S10E (0,17 s). On all devices, the application built in the NativeScript framework was the slowest to run. It achieved the worst time on a Xiaomi REDMI device (6,52 s).
On the warm launch, the fastest (0,14 seconds, on the Samsung S10E) among the applications created in hybrid frameworks was the application created with the React Native framework. As with the first application launch criterion, the slowest application on all devices was built in the NativeScript framework, with the longest time of 6,46 seconds on a Xiaomi REDMI device (Graph 3b).

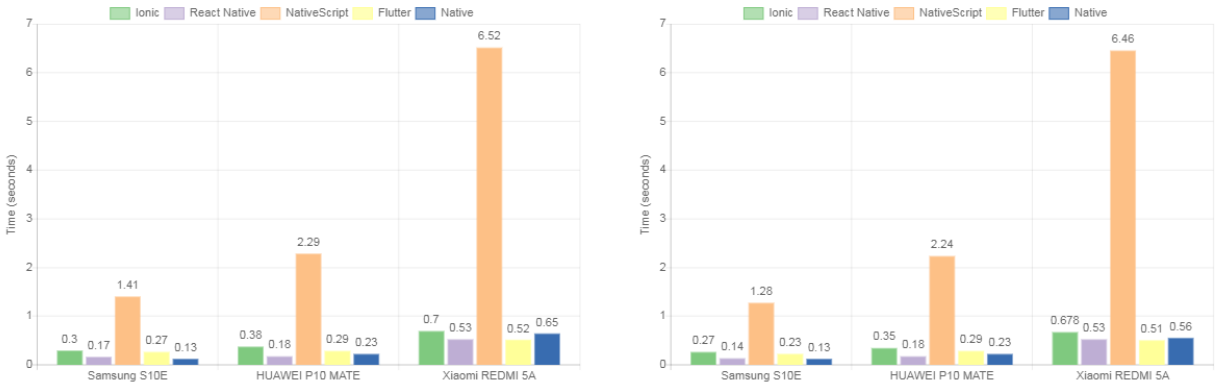


(a) APK package creation time

(b) The size of packages

(c) Installation time

Figure 2: Results of measurements of package creation and application installation in phase 1



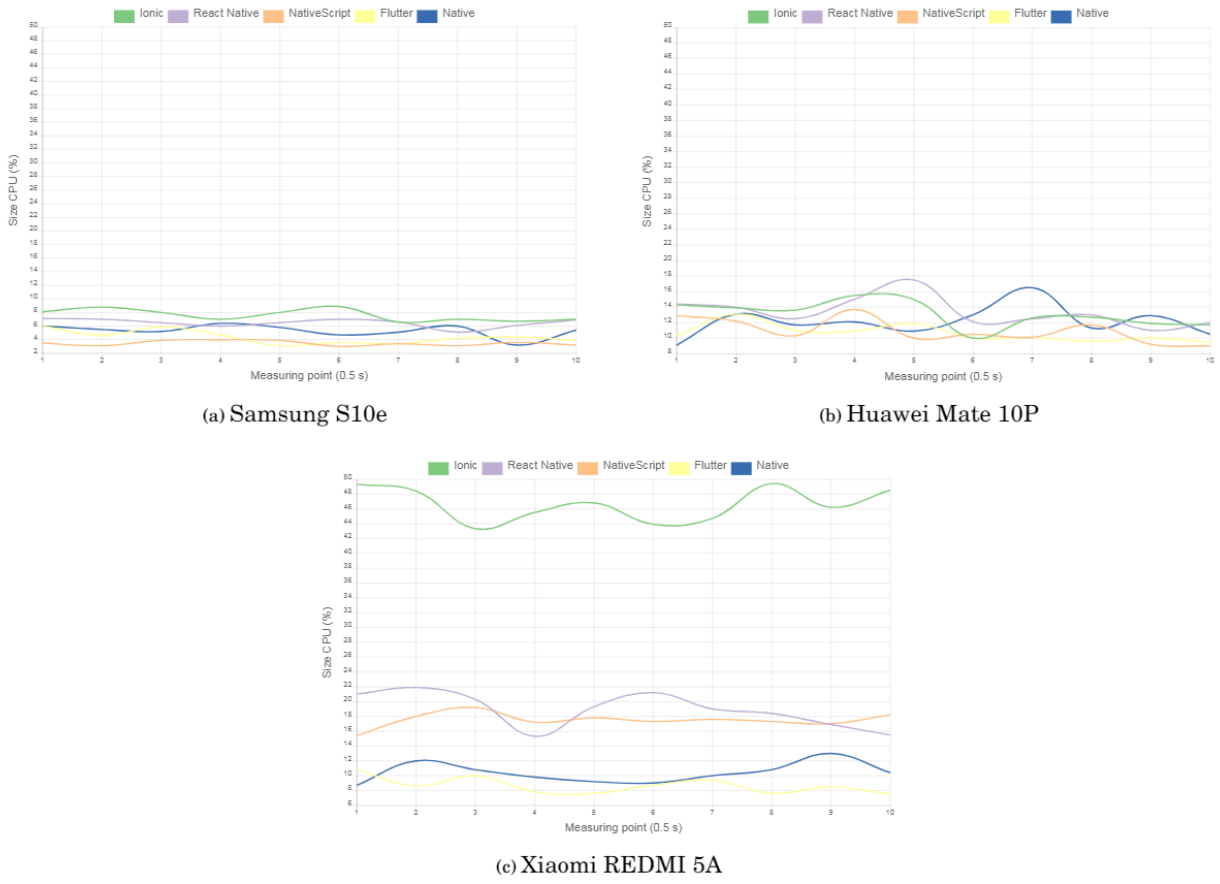
(a) Time of the first launch of applications on devices

(b) Time of warm launch of applications on devices

Figure 3: Results of time measurements of the first (cold) launch and warm launch of the application in phase 1

Phase 2

In phase 2, we obtained results on device processor consumption. Graph 4a shows the obtained results for the Samsung S10E. The share of processor usage is between 4 % and 10 % of total processor usage (100 %). The highest use (8,9 %, point 6) was for the application created in the Ionic framework, and the lowest (3 %) for the application created in the NativeScript framework.
The next graph (Graph 4b) shows the data for the device Huawei P10 MATE. CPU usage has increased slightly compared to the previous device, and ranges between 9 and 18 % of total usage. The highest usage (17,5 %) was measured for the application (point 5) created in the React Native framework, while the lowest usage (3 %) was found for the application created in the NativeScript framework (point 1).
The highest CPU consumption was detected for the Xiaomi REDMI (Graph 4c) device, where the maximum usage was 49,4 %, for the application created in the Ionic framework (point 8). The minimum consumption on this device was 7,5 % for the application made in the Flutter framework (point 10).

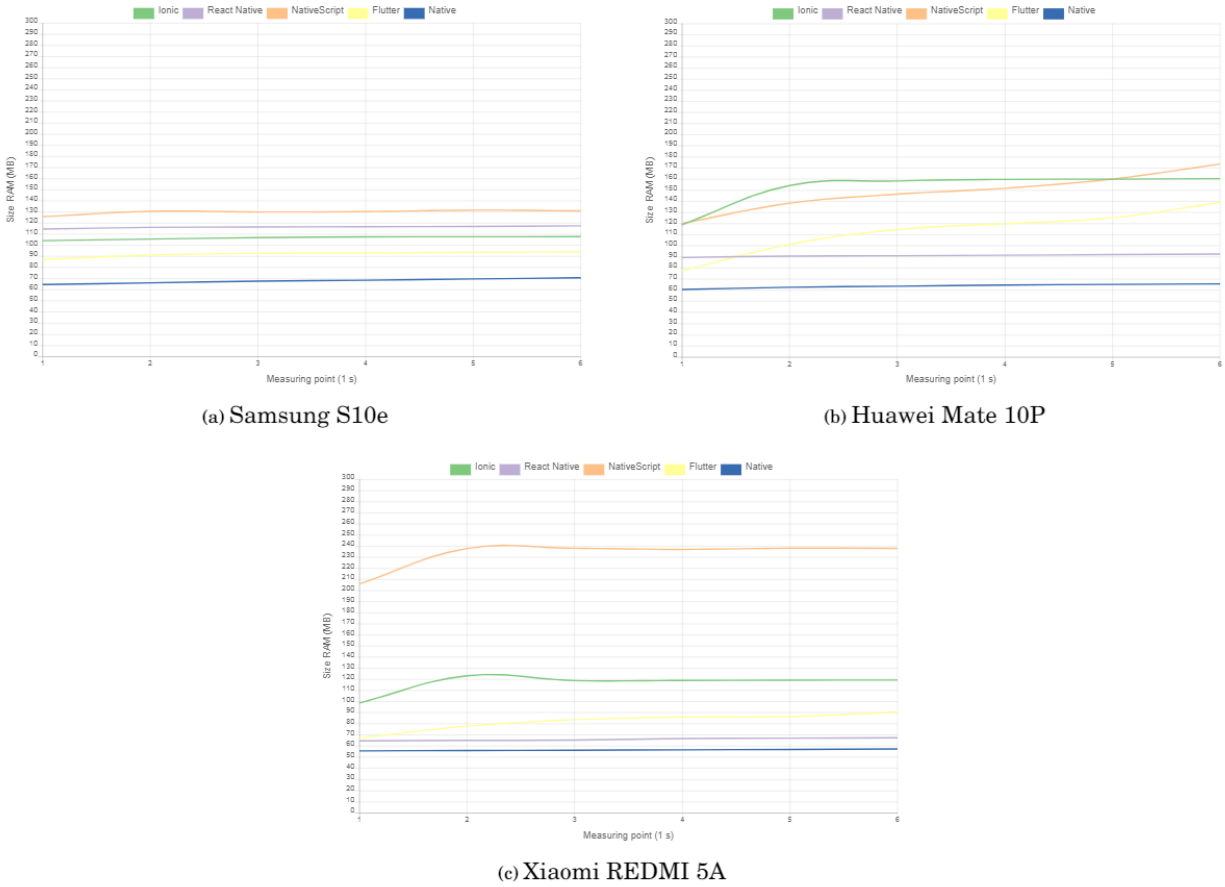


(a) Samsung S10e

(b) Huawei Mate 10P

(c) Xiaomi REDMI 5A

Figure 4: The share of applications' CPU usage when scrolling vertically on the device screen in phase 2



(a) Samsung S10e

(b) Huawei Mate 10P

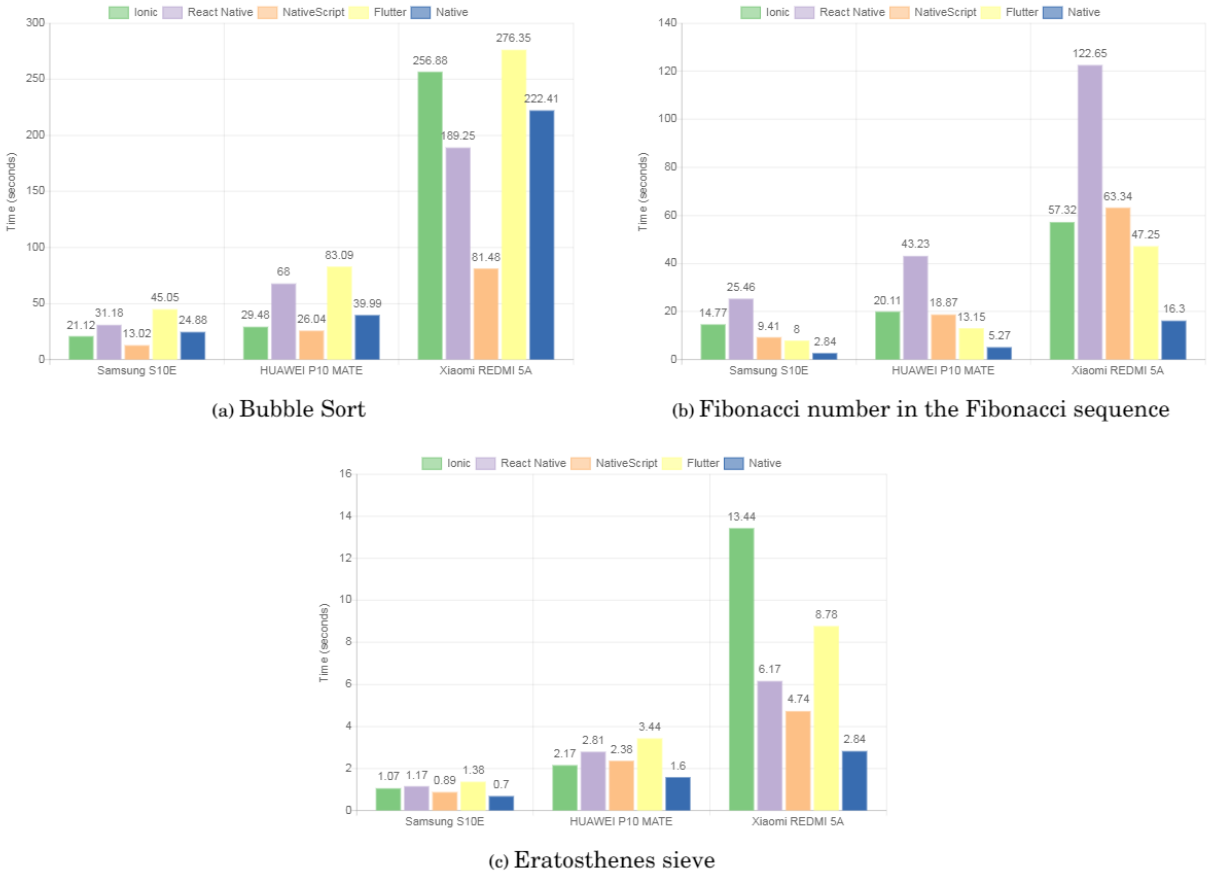
(c) Xiaomi REDMI 5A

Figure 5: The amount of working memory consumption in the case of scrolling vertically on the device screen in phase 2

Regarding the amount of working memory consumption for the Samsung S10E device, we can see from Graph 5a that the biggest difference between the initial and final value of consumption was 6,8 MB for the application we created in the NativeScript framework. The smallest difference (2,9 MB) was in the case of the application created in the React Native framework.

The final value of working memory consumption of the Huawei P10 differs significantly from the initial value compared to the Samsung S10E in some applications. The largest difference between initial and final consumption (61,5 MB) was found for the application created in the Flutter framework, and the smallest (3,1 MB) for the application created in the React Native framework (Graph 5b).

Graph 5c shows the working memory consumption measured on the Xiaomi REDMI device. The difference between the initial and final consumption was larger compared to the first device, and slightly smaller compared to the second. The largest difference between the initial and final consumption (31,9 MB) was measured in the application created in the NativeScript framework, and the smallest (2,7 MB) for the React Native framework.



(a) Bubble Sort

(b) Fibonacci number in the Fibonacci sequence

(c) Eratosthenes sieve

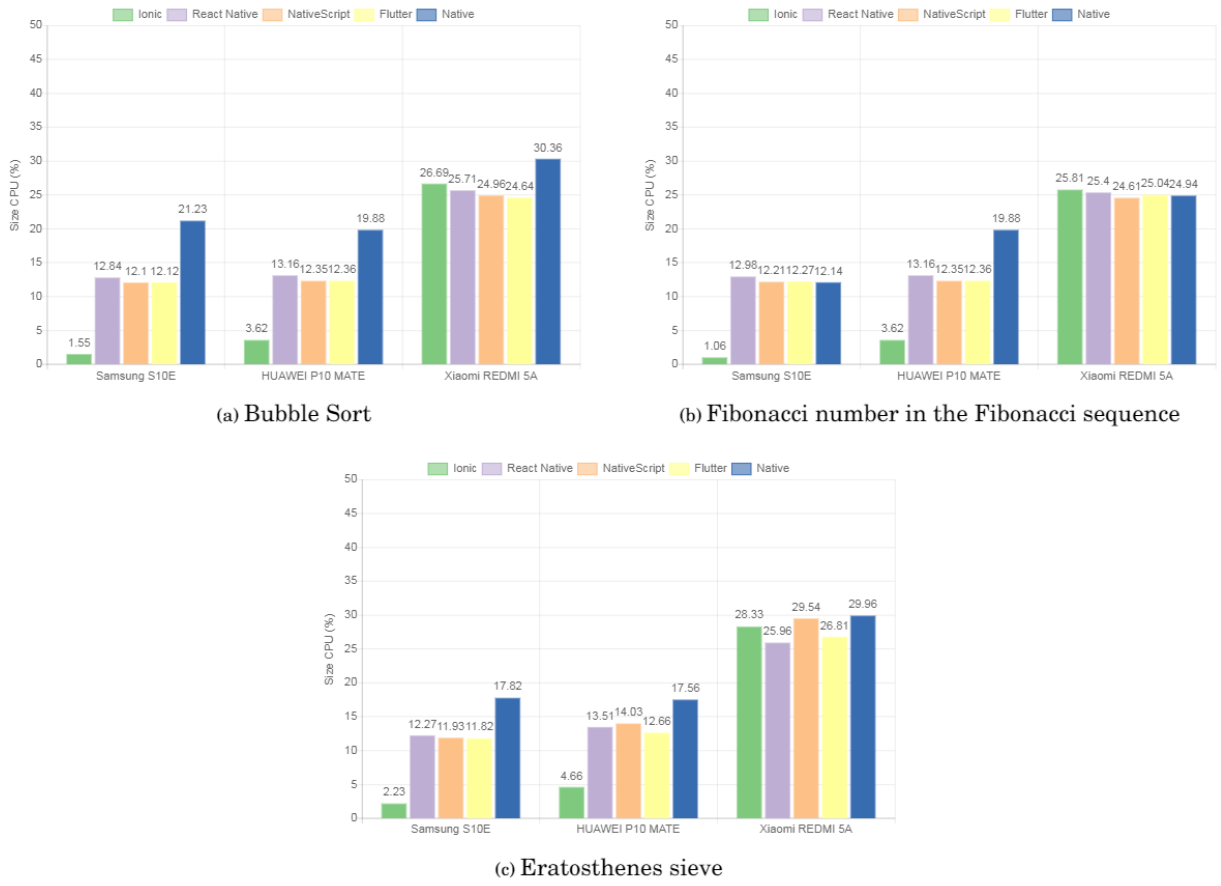
Figure 6: Algorithm execution time in phase 3

Phase 3

In the third phase, we tested the execution time of individual algorithms. The fastest, the Bubble Sort algorithm, was implemented in the application, which we created in the NativeScript framework, with a time of 13,02 seconds (Graph 6a), on the Samsung S10E device. The slowest, Bubble Sort algorithm, was implemented in an application we built in the Flutter framework. The maximum time, 276,35 seconds, was achieved on the Xiaomi REDMI device.

Graph 6b indicates the execution time of the algorithm for finding the number in the Fibonacci sequence. Among the applications we built in hybrid frameworks, the fastest running algorithm was the one we built in Flutter, with a time of 8 seconds on the Samsung S10E. On all devices, the algorithm ran the slowest in the application we created in the React Native framework, with the slowest time of 122,65 seconds on the Xiaomi REDMI device.

Among the hybrid applications, the fastest in the implementation of the Eratosthenes sieve algorithm was the one made in the React Native framework, with a time of 0,89 seconds on the Samsung S10E. The slowest application for execution of this algorithm, was created in the Ionic framework, and installed on a Xiaomi REDMI device (13,44 s) (Graph 6c).



(a) Bubble Sort

(b) Fibonacci number in the Fibonacci sequence

(c) Eratosthenes sieve

Figure 7: Share of CPU usage during the algorithm execution in phase 3

The share of CPU usage in the implementation of the Bubble Sort algorithm was distinctly the highest on the Xiaomi REDMI device. It is also interesting that the native application achieved a much higher share of CPU usage than the hybrid application. Among hybrid applications, the lowest share (1,5 %) was achieved in the application made in the Ionic framework, namely on the Samsung S10E device (Graph 7a).

In the execution of the Fibonacci sequence number search algorithm, the CPU consumption shares were very similar to the consumption of the Bubble Sort algorithm. However, the results of the native application are, here, more comparable to the results of the the hybrid applications. Interestingly, we measured the smallest and largest share of usage here in an application built in the Ionic framework. The largest was on the Xiaomi REDMI device (25,81 %) and the smallest on the Samsung S10E device (1,06 %) (Graph 7b).

Even when implementing the third algorithm (Eratosthenes sieve) on devices, the measured results were very similar to the previous two. The graph 7c shows that the lowest usage in the execution of the algorithm (2,24 %) was again found in the application developed in the Ionic framework (on the Samsung S10E device). Comparing hybrid applications, the highest usage (29,54 %) was for applications built in the NativeScript framework on the Xiaomi REDMI device.

Phase 4

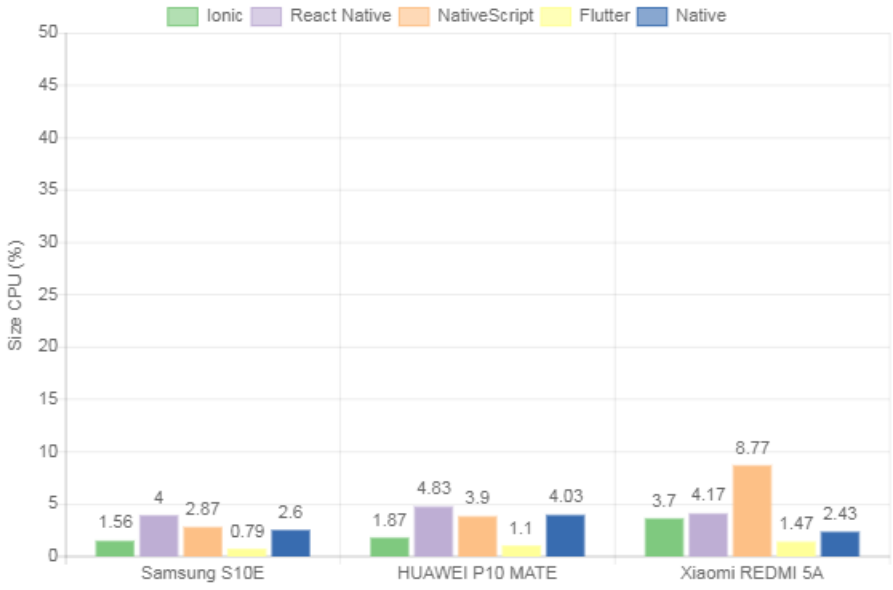


Figure 8: Share of CPU usage during stopwatch execution in phase 4

From Graph 8 we can read the average share of CPU usage when performing a stopwatch on individual devices. We see that the values on all devices were low. The lowest values were achieved by an application created in the Flutter framework. Its lowest value (0,79 %) was achieved on the Samsung S10E. The highest value (8,77 %) was achieved in an application built in the NativeScript framework on a Xiaomi REDMI device.

CONCLUSION

There is no doubt that hybrid mobile application development frameworks change the way we develop modern mobile applications. Despite the big number of different frameworks for hybrid mobile application development, there is no universal framework that would be appropriate for all development tasks. According to our study, there were big differences when using different frameworks in different test scenarios. In fact, some frameworks ensured high execution speeds of applications, while some ensured lower memory consumption. On the other hand, some frameworks offered fast prototyping, as well as better documentation and, therefore, faster development of applications than the others. In the future, we can expect that the way of developing mobile applications will continue to change, as companies will have to take into account the latest trends in mobile application development, such as the use of Artificial Intelligence and Machine Learning, the integration of Virtual Reality in personalization, user experience and use of smart chat bots. In order for hybrid mobile applications to remain competitive with native applications, developers will need to adapt the design and utilization of such applications to current trends. Furthermore, developers will need to continue to ensure that hybrid mobile applications remain as accessible as they are now.

REFERENCES

Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T.-M. (2017). Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. In *Proceedings of the 13th International Conference on Web Information Systems and Technologies* (pp. 344–351). Porto, Portugal.

Branstein, M., & Nick, B. (2017). *The nativescript book*. Louisville, Kentucky: Brosteins.

Delia, L., Galdamez, N., Corbalan, L., Pesado, P., & Thomas, P. (2017, July). Approaches to mobile application development: Comparative performance analysis. In *2017 Computing Conference* (pp. 652–659). London.

Eisenman, B. (2017). *Learning React Native*. Sebastopol, California: O'Reilly Media.

Fister, I., Jr., Deb, S., & Fister, I. (2018, November). Near real-time performance of population-based nature-inspired algorithms on cheaper and older smartphones. In *2018 5th International Conference on Soft Computing & Machine Intelligence (ISCMI)* (p. 12-16). Nairobi, Kenya.

Flutter. (2019). *Flutter architectural overview*. Retrieved 2019-12-21, from https://flutter.dev/docs/resources/technical-overview

Griffith, C. (2017). *Mobile App Development with Ionic, Revised Edition*. Sebastopol, California: O'Reilly Media.

Huynh, M., Ghimire, P., & Truong, D. (2017). Hybrid App Approach: Could It Mark the End of Native App Domination? *Issues in Informing Science and Information Technology*, *14*, 49–65.

Ma, Y., Liu, X., Liu, Y., Liu, Y., & Huang, G. (2018, May). A Tale of Two Fashions: An Empirical Study on the Performance of Native Apps and Web. *IEEE Transactions on Mobile Computing* ,*17*(5), 990–1003.

Majchrzak, T. A., Biørn-Hansen, A., & Grønli, T.-M. (2017). Comprehensive Analysis of Innovative Cross-Platform App Development Frameworks. In *Proceedings of the 50th Hawaii International Conference on System Sciences* (p. 6162-6171). Hawaii.

Martinez, M., & Lecomte, S. (2017, May). Towards the Quality Improvement of Cross-Platform Mobile Applications. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems* (p. 184- 188). Buenos Aires, Argentina.

Masiello, E., & Friedmann, J. (2017). *Mastering React Native leverage frontend development skills to build impressive iOS and Android applications with Native React*. Birmingham, UK: Packt Publishing.

NativeScript. (2020). *Native mobile apps with Angular, Vue.js, TypeScript, JavaScript - NativeScript.* Retrieved 2019-12-17, from https://www.nativescript.org/

Parihar, A. (2020, January). *Five of the Most Popular Databases for Mobile Apps.* Retrieved 2020-01-07, from https://blog.trigent.com/five-of-the-most-popular-databases-for-mobile-apps

Ravulavaru, A. (2017). *Learning Ionic 2, Second Edition*. Birmingham, UK: Packt Publishing.

StatCounter. (2020, January). *Mobile Operating System Market Share Worldwide*. Retrieved 2020-01-16, from https://gs.statcounter.com/os-market-share/mobile/worldwide

Statista. (2019a). *Number of smartphone users worldwide 2014-2020*. Retrieved 2019-12-29, from https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

Statista. (2019b). *Smartphone penetration rate by country 2018*. Retrieved 2019-12-29, from https://www.statista.com/statistics/539395/smartphone-penetration-worldwide-by-country/

Vilcek, T., & Jakopec, T. (2017, May). Comparative analysis of tools for development of native and hybrid mobile applications. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1516–1521).

Opatija, Croatia.
Windmill, E. (2019). *Flutter in Action*. Shelter Island, New York: Manning Publications.