Cross-Platform Empirical Analysis of Mobile Application Development frameworks: Kotlin, React Native and Flutter

Dr Bhawna Suri
Department of Computer Science, Bhagwan Parshuram Institute of Technology, GGSIPU, Delhi, India
bhawnasuri@bpitindia.com

Dr Shweta Taneja
Department of Computer Science, Bhagwan Parshuram Institute of Technology, GGSIPU, Delhi, India
shwetataneja@bpitindia.com

Isha Bhanot
Department of Computer Science, Bhagwan Parshuram Institute of Technology, GGSIPU, Delhi, India
ishabhanot3@gmail.com

Himanshi Sharma
Department of Computer Science, Bhagwan Parshuram Institute of Technology, GGSIPU, Delhi, India
sharma.himanshi2023@gmail.com

Aanchal Raj
Department of Computer Science, Bhagwan Parshuram Institute of Technology, GGSIPU, Delhi, India
aanchalraj13901@gmail.com

ABSTRACT

Mobile application development is currently seen as one of the most indispensable skills due to the recent boom in the population using smartphones and tablets. Due to the abundance and diversity of devices and platforms, developing applications for mobile devices is a complicated task involving multiple alternatives, technologies, and trade-offs. The development of applications in a cross-platform framework has become more prevalent in recent years. However, it is less clear which of the three major frame-works for android development, React Native, Flutter, and Kotlin is the most efficient in general comparisons.

In this work, we have analyzed empirically the performance of three mobile application frameworks: Flutter, React-Native and Kotlin. Three tic-tac-toe games are designed using Flutter, React-Native and Kotlin, and their performance is compared on the basis of various parameters such as CPU usage, Memory usage, packages and their installations to understand the advantages and disadvantages of cross-platform and native mobile development. Our work also aims to assist developers in determining which framework is most suited to their requirements by comparing them in areas such as environment, development languages and ease of use. The results have indicated that Kotlin has a slight advantage over the two cross-platform technologies, Flutter and React Native. Memory usage of React-Native and Flutter were approximately 13.3% and 1.6% more than Kotlin respectively. Similarly, the application sizes of React-Native and Flutter are 98.06MB and 93.32 MB respectively, with Kotlin coming in last at 11.06 MB. In future, further investigation of the topics covered in this work is required to ensure and enhance the findings.

ICIMMI 2022, December 23, 24, 2022, Jaipur, India
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9993-7/22/12…$15.00
https://doi.org/10.1145/3590837.3590897

KEYWORDS

Cross-platform mobile application development, Native mobile application development, Flutter, Kotlin, React Native

ACM Reference Format:
Dr Bhawna Suri, Dr Shweta Taneja, Isha Bhanot, Himanshi Sharma, and Aanchal Raj. 2022. Cross-Platform Empirical Analysis of Mobile Application Development frameworks: Kotlin, React Native and Flutter. In *International Conference on Information Management & Machine Intelligence (ICIMMI 2022), December 23, 24, 2022, Jaipur, India.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3590837.3590897

1 INTRODUCTION

Android is one of the most popular mobile operating systems available in the market. Android users span 2.5 billion people in over 190 countries. Android is one of the most popular mobile operating systems available in the market. Android users span 2.5 billion people in over 190 countries. However, with all its popularity, android does not enjoy the vast market of Japan and the United States. Countries such as Brazil, India, Indonesia, Iran, and Turkey hold 85% of the android market [1].

However, the development of apps for all platforms becomes difficult, as each has its own set of tools, plugins, packages, and programming languages. While this factor won't cause much trouble for experienced developers, it might be a bit distressing for amateur developers. Android and iOS continue to compete for the top spot in mobile operating system popularity, and both ap-pear to be viable and substantial marketplaces for developers. As a result, developers prefer cross-platform programming since it allows for code reusability and a single codebase with potential platform-specific capabilities. Many cross-platform frameworks have gained popularity in recent years, some of which have been adopted by a significant number of developers, including Flutter, React Native, and Xamarin [2]. React Native is extensively used for creating Android and iOS apps, for example, Facebook, Instagram and Discord [3]. Since Flutter is comparatively new, it does not have a lot of popular apps under its belt. However, Google ads and Alibaba are examples of a few popular apps built by Flutter [3].

1.1 Native vs Cross-platform frameworks

When an application is created with the intent of working on only one platform, it is called a native app. Such apps are created using platform-specific programming languages and tools. For example, one can use Kotlin to create a native Android app and Swift to create an iOS app.

Native applications have certain advantages, some of which are discussed below:

- A native application tends to rank higher on the respective platform's app store due to its performance and execution time.
- We can have access to every tool of the platform on which we are working. It is comparatively easier for developers to debug and familiarize themselves with the built-in tools, as opposed to looking for external tools or software.
- Significantly higher level of user satisfaction, as the application adheres to the look and feel of the operating system.
- The enhanced compatibility between the code and device resources ensures the high performance of native apps.
- A native app is comparatively easier to publish than a cross-platform app.

However, there are certain disadvantages of native applications:

- With increasing diversity in the number of platforms available, there is limited use of an application that works on a sin-gle platform.
- Native applications are costlier and more time-consuming than cross-platform applications, as there are different sets of code for different platforms [4].

An application that works on several platforms rather than a single platform is called a cross-platform application. For example, we can use frameworks such as React Native or Flutter to create applications for Android as well as iOS.

The advantages of cross-platform applications are:

- Since a cross-platform application requires a single code base, it gains the upper hand over native applications in terms of cost. It is easier to organize and maintain the code in cross-platform apps.
- Development time is significantly reduced as we do not have to make multiple applications for multiple platforms.
- Cross-platform apps can be accessed by a larger audience, as they are available on multiple platforms [5].

While these factors have been successful in propelling cross-platform apps into the limelight, there are certain disadvantages of cross-platform applications:

- Native UX tools are not available to cross-platform apps. Therefore, they are unable to give the same UX experience as their native counterparts.
- The performance of cross-platform applications is comparatively slower than native applications.

This paper aims to compare cross-platform frameworks with native frameworks on various parameters, which in turn can help to decide which framework is to be used per user requirements.

2 RELATED WORKS

Simon Stendor et al. in their thesis compared android flutter and react-native by creating two similar stop-watch apps using both technologies [4]. In this case, the Flutter app performed somewhat better than the React Native app. However, the difference is negligible. The performance tests for both applications were done using profiler functionality in Android Studio (Tests were done only on Android). Matilda Olsson compared android as well as ios flutter and react-native by creating 2 similar push notifications apps using both technologies [5].

Jakub Jagiełło measured the performance in terms of the number of dropped frames under a certain time for two different applications [6]. Tests were done only on Android. Kamil Wasilewski et al. built three applications using Java, Kotlin and Flutter. Comparisons were done using parameters like Build Time and Application File Size, Start-Up of the Application and Idle Usage of RAM, Collection Operations, REST—GET and POST, Serialization and Deserialization and File Operations [7]. Their results stated that despite achieving the best time results in most tests, the Flutter application was outclassed in the database operations testing, where rival applications were up to ten times faster. In terms of RAM use, the Flutter programme performed the poorest, while the Java application performed the best. In terms of CPU consumption, the opposite was true.

Yoonsik Cheon and Carlos Chavez explored the challenges of rewriting an android native application into flutter [8]. They came to the conclusion that Flutter lacked a built-in, formal way of doing things. Applications often incorporate guidelines as well as available libraries and application programming interfaces. Another source of concern could be the lack of a large number of third-party libraries, or packages, that allow platform-independent access to the underlying platform's capabilities or services. Mika Kuitunen discusses the different options available for cross-platform development and how they attempt to duplicate the same high performance and positive user experience as seen in native applications [9].

Mehmet Isitan and Murat Koklu compared various cross-hybrid frameworks based on their application size, CPU usage, network usage and energy consumption [10]. They observed that Unity3D is one of the most popular frameworks. Unlike other popular development tools, Unity3D is appropriate for creating applications with more games or visuals. According to data from SO, GitHub, and GT, React Native and Flutter have gained the most popularity and developer support in recent years. According to this criterion, Flutter has recently surpassed React Native. In terms of performance, Flutter clearly outperforms React Native. Re-act Native is a step ahead because it can be built with JavaScript. Dart, the development language used by Flutter, must be mas-tered for third-party software. Hina Hussain et al. compare the app performance in terms of reaction time, memory consumption, processing power, and app size after providing an overview of all key technologies used [11].

Kewal Shah et al. discuss the advantages and disadvantages of cross hybrid development with respect to native development [12]. The four types of cross-platform mobile applications which can be classified based on their implementation include interpreted, web, hybrid, and widget-based apps. As a result, there is a thorough examination of the flutter framework and its bene-fits. Aleksander

Kaczmarczyk has determined that, despite their numerous benefits, cross-platform frameworks are still nascent in the market [13]. Although it has just been a decade since the introduction of interpreted apps, newer inventive ways like Flutter continue to be created, propelling the future of cross-platform mobile app development. There have been many attempts to throw some light on the methods to use while creating such apps. Further research can be conducted by testing our results with quantitative numbers obtained.

Erik Blokland tracks the energy consumption of React Native, Flutter, and the Android API while executing UI activities [14]. These measurements were carried out by building applications in each of the selected frameworks that administer a variety of User Interface elements and executing a test script in MonkeyRunner, an experimental tool that allows programmatic testing of app UIs via an API that supports all forms of movements. The energy consumption was calculated by utilizing the built-in Android power monitoring utilities.

3 METHODOLOGY

For comparing the features of various frameworks, a basic tic tac toe game has been implemented using flutter, react native and kotlin. The features compared include code reusability, environment in which the application is made, code complexity in terms of number of moves, packages and its installations, application sizes, CPU usage, memory usage etc.

Code reusability is the practice of using an existing piece of code for a new function. Code reusability increases productivity, re-duces cost and improves the quality of the app. Its advantages include saving time as there is no need to write the specific piece of code again, saving cost, reduced development risk. A code written is tried and tested. By reusing the cost, there is no need to test it again so the code is highly reliable and free from defects.

Code complexity is a function represented in the terms of number of moves taken by the users while playing the game. It is represented in the form of a linear time complexity function O(n) which grows proportionally as per the input. It describes the up-per bound of growth rate of a function and the worst case. There are two types of code complexity namely space complexity and time complexity.

Application size is the amount of space taken by the application in the system in which it is running. The average size varies as the system in which the app is run. For example, for IOS it is about 35 MB and for android the number is around 12MB. It also varies as per the category the apps are assigned to. CPU usage is the computer's usage of processes resources or the work handled by the CPU. Actual CPU usage depends on the type of work handled by the CPU. Memory usage is determined by the amount of memory application uses while running. The program instruction takes up memory and more memory are reserved for data.

3.1 Implementation of Apps

A tic tac toe game is implemented using basic logic in three frameworks namely flutter, react native and kotlin. Flutter app is implemented on Android Studio version Arctic Fox/2020.3.1. The React Native app is made on Visual Studio Code Version 1.73. Kotlin app is

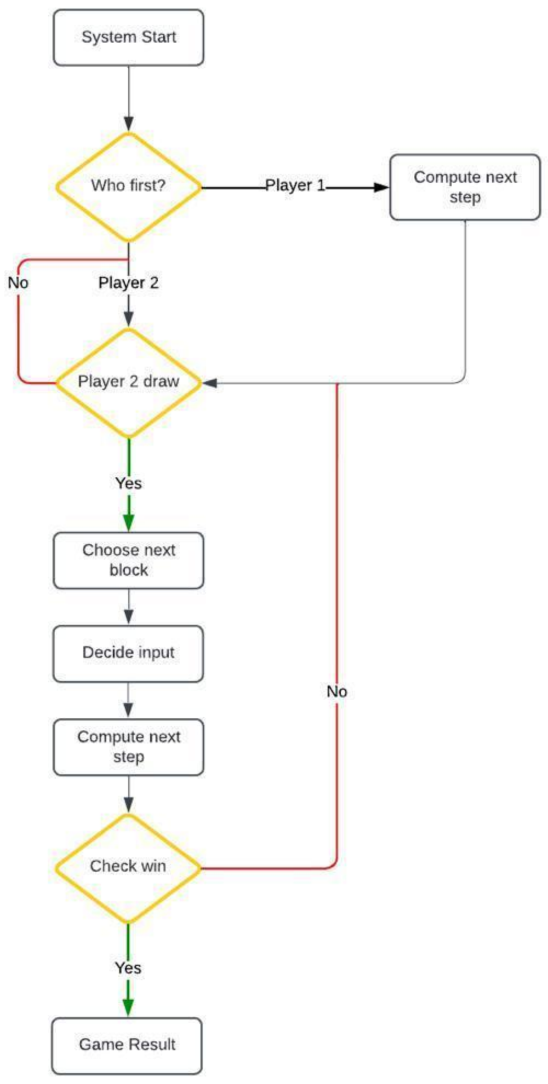

Figure 1: Workflow of the tic-tac-toe game

made on Android Studio version Arctic Fox/2020.3.1. The UI is just like a basic tic tac toe game. The apps are tested on an android vivo u10 smartphone. The application size was seen from the system on which the app is run, CPU usage and memory usage determined by the use of profilers present on the platforms on which the apps are made. Code complexity is measured using simple O(n) Function as mentioned above. Code Reusability is found out by taking into consideration the same piece of code being repeated in the app's code. Figure 1 shows the workflow of the tic-tac-toe game.

3.1.1 Flutter App UI. Figure 2 and figure 3 show the tic tac toe game in the flutter framework.

3.1.2 React Native App UI. Figure 4 shows Player X's turn in the tic tac toe game created using React Native Framework, while figure 5 shows Player O's turn.

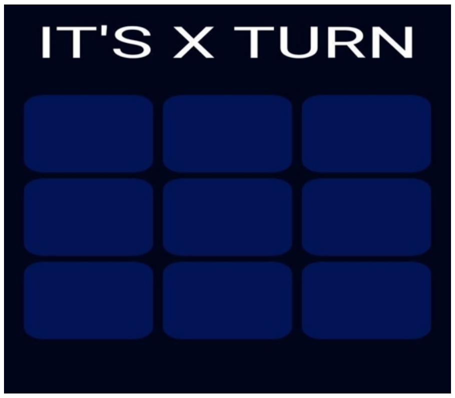

Figure 2: Tic tac toe game in flutter framework

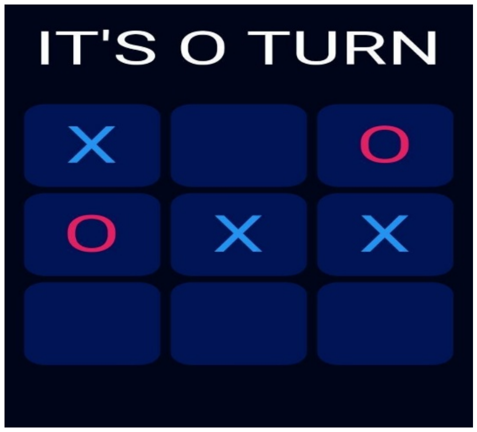

Figure 3: Tic tac toe game in flutter framework

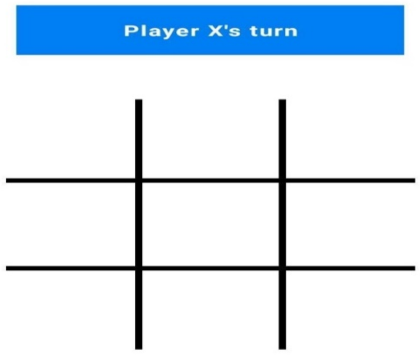

Figure 4: Tic tac toe game in native framework

3.1.3 *Kotlin App UI.* Figure 6 shows Player X’s turn in the Tic tac toe game created using Kotlin. Figure 7 is showing a progressed game.

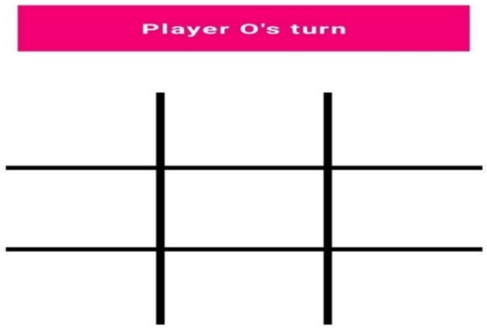

Figure 5: Tic tac toe game in native framework

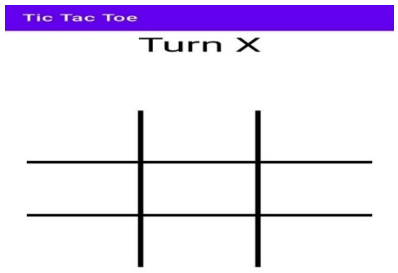

Figure 6: Tic tac toe game in kotlin framework

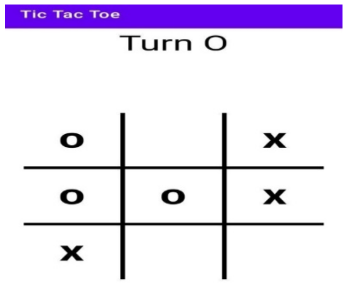

Figure 7: Tic tac toe game in kotlin framework

4 RESULTS

The apps made using the various frameworks are compared on various parameters and the results are as follows.

4.1 Application Size

The applications run on the same system and their sizes were measured when they were loaded into the computer. Kotlin app used the least amount of space in the system, while flutter and react native apps used more space with react native app using the most space, as shown in figure 8.

Table 1: Energy Usage Comparison

Application	Energy Usage
React-Native	Very High
Flutter	Very High
Kotlin	Medium

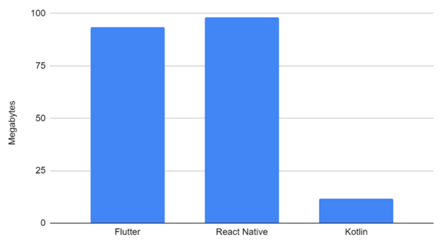

Figure 8: Application size comparison

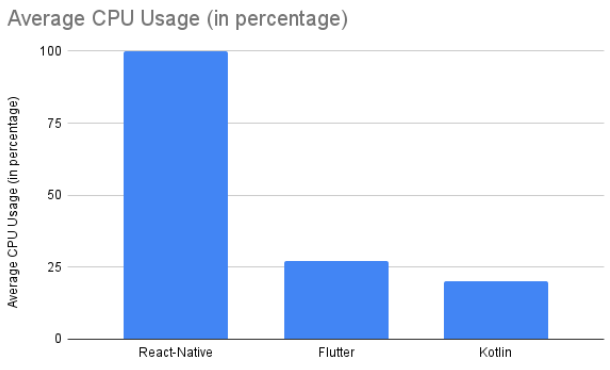

Figure 9: CPU usage comparison

4.2 Performance Comparison

The performance of the app is measured as per various parameters like CPU Usage, Memory Usage, Network Usage and Energy Usage. Graphs are attached below indicating the same.

4.2.1 CPU Usage. The CPU usage is found out using the profiler of each platform. React Native has the highest CPU usage, as shown in figure 9.

4.2.2 Memory Usage. The memory usage is found out using the profiler of the platform only. Figure 10 shows that react native has the highest memory usage.

4.2.3 Energy Usage. Table 1 shows the energy usage comparison. It is found using profiler tool only. Both react native and flutter have very high energy usage.

4.3 Code Reusability

Code Reusability is measured in the code by the number of times the same code is repeated throughout the whole code. In this case, as shown in Table 2 react native is the best suited code as about 95 percent code is repeated, kotlin and flutter have 85 and 90 percent reusability respectively.

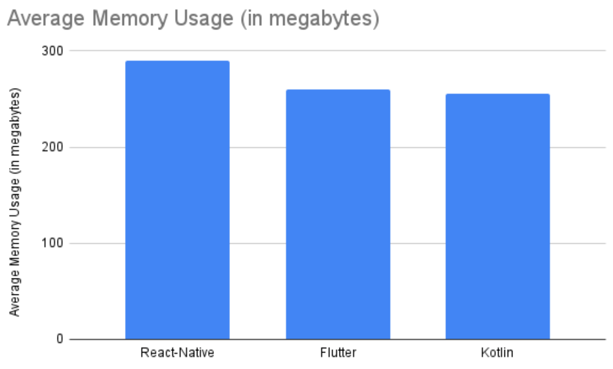

Figure 10: Memory usage comparison

4.4 Code Complexity

It is determined using the function O(n) and due to the same logic implemented in all three codes, the complexity remains the same, that is O(n) and space complexity is n, where n is the number of moves taken. This is shown in Table 2.

5 DISCUSSION

This section discusses the test results that were acquired after implementing the applications. We observed that Kotlin has the smallest application size, while React Native has the largest application size. Comparing the code complexity of the development code of each of the applications, we found it to be the same. The highest code reusability percentage is in React Native, followed by Flutter and Kotlin.

In addition, Kotlin was simple, while other apps had some animation, so our results may have been impacted by that as well. The applications could be compared based on other parameters, such as application security, development cost, and license publishing to markets.

6 CONCLUSION

In order to assist users in choosing the most appropriate framework among the major ones, such as Kotlin, React Native and Flutter for Android development, we designed three tic-tac-toe games in Flutter, React Native and Kotlin for a general comparison of cross-platform and native frameworks, and the results obtained will be useful for users to find the framework that best meets their needs.

Table 2: Various Parameter Comparison

Technology Used	App. size	Environment	Code Complexity	Package and it's installations	Code Reusability
Flutter	Storage used is 93.32 MB.	Android Studio Version Arctic Fox/2020.3.1	Time complexity - O(n). Space complexity - n	Flutter and dart plugins.	90 percent of code can be reused.
React-Native	Storage used is 98.06 MB.	Visual Studio Code Version 1.73	Time complexity - O(n). Space complexity - n	React native tools plugin.	95 percent of code can be reused.

We developed relatively simple applications using Flutter, Kotlin, and React Native. Their performance compared based on various parameters, including CPU usage, memory usage, packages, and installations, Kotlin was found to have a slight advantage in terms of memory consumption and application size over Flutter and React-Native. The results would have varied accordingly if the testing apps had been more dynamic by accessing GPS, camera, and other smartphone features.

Despite being a cross-platform solution, Flutter performs well on a single application base in comparison to native applications. However, the results indicate significant potential for the future, although further research is needed for a more definitive answer.

REFERENCES

[1] https://www.businessofapps.com/data/android-statistics/ (last accessed on 30th November 2022)

[2] Singh M. and Shobha G. 2021. Comparative Analysis of Hybrid Mobile App Development Frameworks. International Journal of Soft Computing and Engineering (IJSCE). 10, 6 (July 2021), 1. DOI = https://doi.org/10.35940/ijsce.F3518.0710621

[3] https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose (last accessed on 30th November 2022)

[4] Stender S. and Åkesson H. 2020. Cross-platform Framework Comparison Flutter and React Native. Bachelor Thesis. Blekinge Institute of Technology.

[5] Olsson M. 2020. A Comparison of Performance and Looks Between Flutter and Native Applications: When to prefer Flutter over native in mobile application development. Bachelors Thesis. Blekinge Institute of Technology.

[6] Jagiełło J. 2019. Performance comparison between React Native and Flutter. Bachelors Thesis. UMEÅUniversity.

[7] Wasilewski K. and Zabierowski W. 2021. A Comparison of Java, Flutter and Kotlin/Native Technologies for Sensor Data-Driven Applications. Technical Report. Lodz University of Technology.

[8] Cheon Y. and Chavez C. 2021. Creating Flutter Apps from Native Android Apps. 2021. Technical Report. University of Texas at El Paso.

[9] Kuitunen M. 2019. Cross-Platform Mobile Application Development with React Native. Bachelor Thesis. Tampere University.

[10] Isitan M. and Koklu M. 2020. Comparison and Evaluation of Cross Platform Mobile Application Development Tools. International Journal of Applied Mathematics, Electronics and Computers. 8, 4 (Dec. 2020), 273-281. DOI= https://doi.org/10.18100/ijamec.832673

[11] Hussain H., Khan K., Farooqui F., Dr. Arain Q. A. and Dr. Farah I. 2021. Comparative Study of Android Native and Flutter App Development. KSII The 13th International Conference on Internet (ICONI) (Jeju Island, Korea December 12-14, 2021). ICONI'21. KSII, Korea 2-3.

[12] Shah K., Sinha H. and Mishra P. 2019. Analysis of Cross-Platform Mobile App Development Tools. 5th International Conference for Convergence in Technology (12CT) (Pune, India March 29-31, 2019). IEEE'19. IEEE, New Jersey, 1-2. DOI=https://doi.org/10.1109/I2CT45611.2019.9033872

[13] Kaczmarczyk A., Zając P. and Zabierowski W. 2022. Performance Comparison of Native and Hybrid Android Mobile Applications Based on Sensor Data-Driven Applications Based on Bluetooth Low Energy (BLE) and Wi-Fi Communication Architecture. Technical Report. Lodz University of Technology.

[14] Blokland E. 2019. An Empirical Evaluation of the User Interface Energy Consumption of React Native and Flutter. Bachelor Thesis. Delft University of Technology.