Evaluating the performance of Android based Cross-Platform App Development Frameworks

Mohammad Mahendra
Faculty of Computer Science
Universitas Indonesia
Depok, Indonesia
mohamad.mahendra@ui.ac.id

Bayu Anggorojati
Faculty of Computer Science
Universitas Indonesia
Depok, Indonesia
bayuanggorojati@cs.ui.ac.id

ABSTRACT

Mobile applications have become a necessity in people's daily lives, and more are being developed at this time. To increase the efficiency in developing mobile apps, cross-platform mobile app development framework is needed. Currently there are various types of mobile app development frameworks with their respective approaches, be it native apps, mobile web apps, widget based, bridging with javascript, and others. However, user adoption and retention to a mobile app is highly influenced by good user experience, which can be quantified by application performance. In this research, a series of experiment are conducted to measure the performance of several Android based mobile app development frameworks based on five parameters, namely cpu usage, memory usage, response time, frame rate and app size. The results of this study show Native Android application that is used as the baseline gives the best performance and then followed by widget based cross-platform development framework, i.e. Flutter.

CCS CONCEPTS

• **Human-centered computing** → *Ubiquitous and mobile computing systems and tools*; **Empirical studies in ubiquitous and mobile computing**; • **Software and its engineering** → *Development frameworks and environments.*

KEYWORDS

Mobile application, cross platform, development framework, performance evaluation

ACM Reference Format:
Mohammad Mahendra and Bayu Anggorojati. 2020. Evaluating the performance of Android based Cross-Platform App Development Frameworks. In *2020 the 6th International Conference on Communication and Information Processing (ICCIP 2020), November 27–29, 2020, Tokyo, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3442555.3442561

1 INTRODUCTION

According to the statistics provided by statista, there are 204 billions mobile app downloads worldwide in 2019[1], including the apps available on iOS App Store, Google Play and third-party Android stores. This indicates that there is a big interest in mobile apps by people around the globe. Consequently, more industries have been focusing their products towards mobile platform, including those that are non-technological companies. Considering big market shared between iOS and Android apps, going for a cross-platform development framework is a rational strategy as it increases the development efficiency in terms of time and cost. However, the efficiency provided by cross-platform app development costs the app's performance and user experience (UX) degradation.

Another statistics provided by statista says that mobile app user retention rate worldwide ranges from 31% to 39% from 2012 to 2019[2]. This shows that nearly one out of three people would discontinue using the mobile app after the first experience. In another words, mobile app performance plays an important role in maintaining high user retention rate. Therefore, it is important to study the performance of mobile apps to get a measurable and quantifiable user experience, especially those of cross-platform apps.

Various research related to mobile app performance, Graphical User Interface (GUI) test as well as cross-platform development tools, have been done in recent years. The study related to mobile app performance have been done in [1] [2]. While Aggarwal et al. in [1] focus on performance evaluation model for mobile app, Chunye et al. in [2] simulate concurrent multiple mobile users in accessing application server. Morgado et al. have a series of research related to an automated mobile app GUI testing tool based on the user interface (UI) patterns [6][5][7]. Studies on cross-platform mobile app development frameworks have been presented in [8][9]. Last but not least, research on evaluating the performance of cross-platform mobile app development approaches have been done in [4][10]. While Jia et al. in [4] could only present up to the experiment design as well as the initial data and observations, Willocx et al. in [10] have performed actual performance evaluation on an mobile app developed by using several different cross-platform frameworks. However, [10] solely focus on measuring the performance and did not develop its own application, thus less flexible. Moreover, more recent cross-platform development frameworks, e.g. Flutter[3] and

[1]https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/
[2]https://www.statista.com/statistics/751532/worldwide-application-user-retention-rate/
[3]https://flutter.dev/

React Native[4], were not tested, and frame rate was not included as one of performance parameter.

In this research, performance of two cross-platform mobile app development framework, i.e. Flutter and React Native, is evaluated, with native Android application as the baseline. A test application that tries to simulate a social media application and two automated UI test scenario are developed in each framework for evaluation purpose. Finally, a series of experiment are conducted to measure the performance of mobile app developed by different frameworks based on the developed test scenario.

This paper is structured as follows. The test app as well as automated UI test scenarios developed in this research are briefly explained in section 2. The performance metrics and measurement tools used in this research are briefly outlined in section 3. The measurement results from the experiments are presented and discussed in section 4. Finally, the conclusion and future outlook are given in section 5.

2 TEST APPLICATION AND SCENARIO

2.1 Test application

According to SensorTower, a company that provides mobile app store intelligence, top mobile apps worldwide by downloads in January 2020 (when this research was started) are dominated by social category[5]. Based on this fact, it can be concluded that social apps are the most used apps by mobile users worldwide. Hence, the social category like app is the most representative app to be evaluated to get the performance comparison with respect to UX.

In this research, a simple social media like application that incorporates main features in social media, e.g. timeline, taking picture and post a status, is developed as test application. However, by considering several factors, i.e. the focus of this research is more into comparative study of cross-platform app frameworks and the uncertainty in network load that may influence the right view of some measurement parameters like response time and frame rate, hence the usage of network is eliminated in the test app. Therefore, features that involve two-way communication among users are not included in the test app, e.g. post status, comment, like, etc. On the other hand, a feature to take picture with camera is included in the test app as using camera involves *native API* and different framework has different implementation of native API. Finally, this test app has two pages, i.e. the main page with timeline that contains statuses from other users and a page to create status. It is then implemented in native Android, Flutter and React Native.

2.2 Automated UI test scenario

Two test scenarios are developed for the performance evaluation purpose, namely *infinite scroll* and taking picture with camera. The infinite scroll scenario is related to timeline feature which is common in social category apps. The infinite scroll is related to social media users behavior who tends to spend time scrolling the timeline indefinitely. This behavior refers to a theory in which a user continues to scroll the app's timeline as long as she is not satisfied since scrolling is an easy thing to do and there is virtually infinite contents on top of that [3]. From the performance perspective, infinite scroll would theoretically consume more memory as more contents are displayed in the app. Furthermore, the an app loads the next content at the bottom of the app has different implementation in different frameworks which influence the app performance.

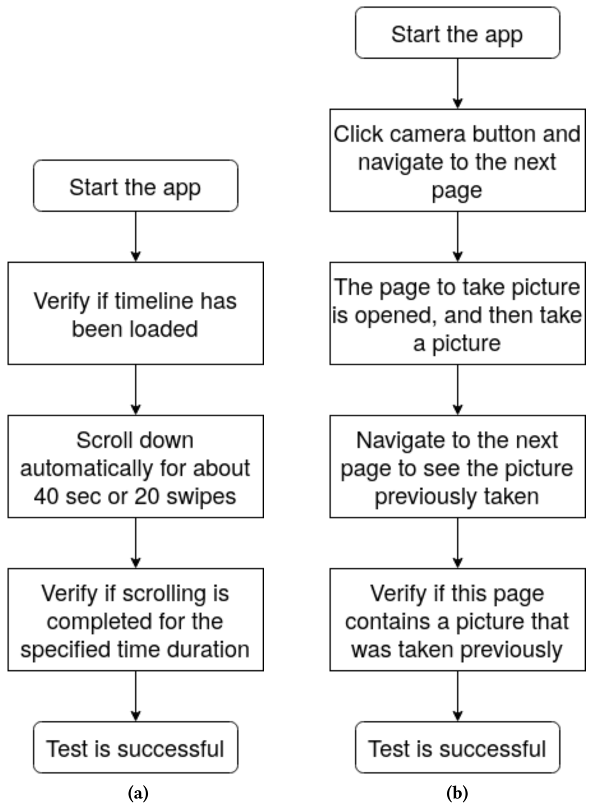

Start the app
Verify if timeline has been loaded
Scroll down automatically for about 40 sec or 20 swipes
Verify if scrolling is completed for the specified time duration
Test is successful
(a)
Start the app
Click camera button and navigate to the next page
The page to take picture is opened, and then take a picture
Navigate to the next page to see the picture previously taken
Verify if this page contains a picture that was taken previously
Test is successful
(b)

Figure 1: The steps performed in infinite scroll test scenario (a) and taking picture by camera test scenario (b)

The second scenario refers to a feature to create status in a social media app but in this case it is limited to status through picture taken from camera. This scenario represents an active social media user behavior who likes to take selfie and updates the status frequently. From the performance perspective, this scenario involves the usage of hardware, i.e. camera, and native API from the device. As earlier mentioned, the way native API works and implemented in different frameworks may be different, thus influence the app performance.

The overall steps performed in both test scenarios are visualized in Figure 1. Please note that the time duration or number of swipes limit are used so that we can verify whether the test is successful or not. Furthermore, the 40 seconds duration or 20 swipes are chosen based on the results of our preliminary test, which show that the memory consumption is relatively constant after such period of time.

After defining test scenario, a set of automated UI or instrumented tests is developed. Developing instrumented test in native Android, Flutter and React Native apps require a specific testing

[4]https://reactnative.dev/
[5]https://sensortower.com/blog/top-apps-worldwide-january-2020-by-downloads

library called *espresso*[6], *flutter driver*[7], and *detox*[8] respectively. Finally, the detail implementations as well as UI test are available in a GitHub repository[9] which consists of several branches for each different app development frameworks implementation.

3 PERFORMANCE METRICS AND MEASUREMENT TOOLS

3.1 Performance metrics

In this research, five metrics are used to measure and compare the performance of applications developed by Flutter and React Native as cross-platform app development frameworks and native Android app as the baseline. Those metrics are as follows:

(1) CPU usage
(2) Memory usage
(3) Response time
(4) Frame rate
(5) Application size

In general both CPU and memory usage are basic metrics to measure the performance of any computer system. In the experiment, the average measure of both metrics over the whole duration of test scenario as described in Section 2.2 is considered. Besides, max value of both metrics during the entire test duration are also taken into account to see how much the apps consume both CPU and memory resources at a short moment of time, e.g. when there is an event that consumes the most resources.

The response time and frame rate are two performance metrics that can give more quantitative view on UX level. Frame rate, which is measured in frame per second (fps) unit, tells the user how responsive an app is. A minimum standard of good frame rate is 60 fps because the minimum refresh rate of typical mobile device screen is 60 Hz. If an app experiences less than 60 fps of frame rate, the user starts to see the app appearance is lagging or slowing down. In the experiment, average frame rate over the whole duration of test scenario is considered. Response time can also tell you how responsive an app is. While frame rate only concerns with the appearance, response time measure the responsiveness of an app in executing a specific task. Unlike the other aforementioned metrics, response time is only measured on a specific and interesting event during the entire test scenario.

Lastly, the size of an application gives an idea of how much storage that needs to be allocated by the device to install and run an app. Although it does not directly impact the performance of a running app, it is a necessary consideration for a user, especially those who are faced with limited storage capacity. In this research, the size of app installer and the size of installed app in the device are considered and compared.

3.2 Measurement tool

To measure the CPU and memory usage as well as response time, *android profiler*[10] tool is used, as it is able to provide real-time performance data for any app run on android platform. To be more precise, *cpu profiler* in android profiler is used to record CPU usage and response time, while memory profiler is used to record memory usage throughout the entire test duration. To analyze the recorded measurement data for later use, the data from cpu and memory profilers can be exported into two different files with `.trace` and `.hprof` extensions respectively.

To compute the frame rate in an android mobile app, *choreographer* class which is responsible for coordinating animation, input and rendering, is used. Choreographer receives VSYNC information each second hence it is able to compute the number of frames per second produced by an app. To further measure the frame rate of an app, an external library called Takt[11] is used, since there is no specific tool in android profiler to measure this metric. The results of frame rate measurement can be observed in *logcat* panel of Android Studio and then saved manually into a text editor for later use. Finally, app size can be observed from the device's system preference.

3.3 Experiment scenario

Before experiment is conducted, there are things that need to be prepared, especially choosing the device for experiment. Without loosing the generality, the comparison of cross-platform frameworks performance have similar patterns regardless whether emulator or real device is used. Besides, several interruptions that may impact performance measurement, such as notifications, incoming phone overheat, etc, may occur if real device is used. Hence, Google Pixel 2 emulator in Android Studio with Android API level 28 (Pie) is used as the device for the experiment.

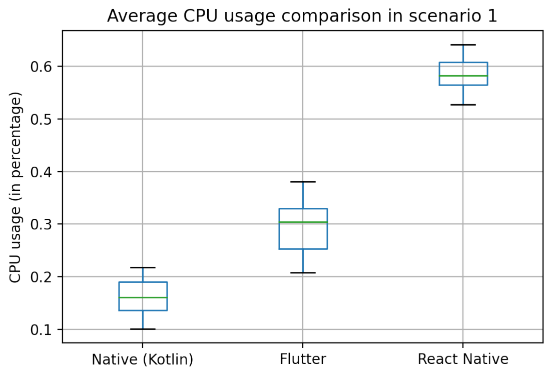
The experiment itself follows five general steps as follow: 1) run test scenario, 2) start android profiler, 3) start cpu activity recording, 4) observe memory usage of the app and then perform *memory dump*, 5) obtain frame rate data from logcat. Although, the automated UI test is performed, the overall experiment scenario involves human intervention, hence it is a semi-automatic one. Furthermore, each of this experiment will be repeated for thirty times for each app development framework for each test scenario to get enough samples for statistical analysis.

4 EXPERIMENT RESULTS AND DISCUSSION

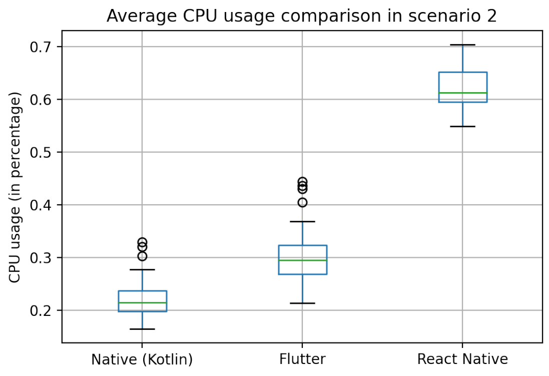
4.1 CPU usage

The boxplot as well as the overall mean of average CPU usage measurements comparison in the first scenario, i.e. infinite scroll, are depicted in Figure 2a and Table 1 respectively. The results show that native Android application being the baseline gives the best performance, and then followed by Flutter and React Native. It can also be observed that the CPU usage percentage increases almost double, i.e. it tends to exponentially increase from native Android to Flutter and to React Native. This means that native Android application is the most optimized in utilizing computation resources provided by CPU while React Native application is the worst among the three. However, the maximum CPU usage in the whole experiment shows a bit different pattern, in which native application still superior to the other two and then followed by React Native and Flutter applications, as shown in Table 1.

[6]https://developer.android.com/training/testing/espresso
[7]https://api.flutter.dev/flutter/flutter_driver/flutter_driver-library.html
[8]https://github.com/wix/Detox
[9]https://github.com/mohmahendra/AndroidComparison
[10]https://developer.android.com/studio/profile/android-profiler
[11]https://github.com/wasabeef/Takt



(a)



(b)

Figure 2: Average CPU usage comparison in infinite scroll scenario (a) and taking picture by camera scenario (b)

Table 1: Overall mean and max comparison of CPU usage in both scenario

	Scenario	*Native*	*Flutter*	*React Native*
Mean	1	15.96%	29.34%	58.64%
	2	22.57%	30.70%	62.01%
Max	1	48.90%	85.40%	79.80%
	2	96.30%	97.70%	93.20%

Table 2: Overall mean and max comparison of memory usage (in MByte) in both scenario

	Scenario	*Native*	*Flutter*	*React Native*
Mean	1	67.863	102.41	121.808
	2	51.433	104.533	125.570
Max	1	188.3	213	271.7
	2	166.4	210.4	220.4

In the second scenario, i.e. taking picture with camera, similar performance pattern is still observed as depicted in Figure 2b and Table 1. However, while the native application experiences higher CPU usage of about 5 to 10%, both Flutter and React native applications only experience 1 to 5% higher CPU usage compared to the results obtained in the first scenario. On the other hand, the maximum CPU usage in the whole experiment for all three frameworks reach more than 90% in a certain situation that is when the camera is accessed by the application for the first time. Based on this finding, accessing camera costs a significant computational resource no matter which development framework is used.

4.2 Memory usage

The boxplot as well as the overall mean of average Memory usage measurement comparison in the infinite scroll scenario are presented in Figure 3a and Table 2 respectively. The results illustrates almost similar trends as compared to the CPU usage measurement results, where the native Android application being the most superior and then followed by Flutter and finally React Native. However, unlike the exponential pattern that is observed in CPU usage results, the increase of memory usage from native application to Flutter and to React Native tends to follow logarithmic pattern, i.e. there is a bigger gap between memory usage of native and Flutter applications than the memory usage between Flutter and React Native applications. Furthermore, the maximum memory usage in the entire experiment also shows similar pattern as the overall mean.

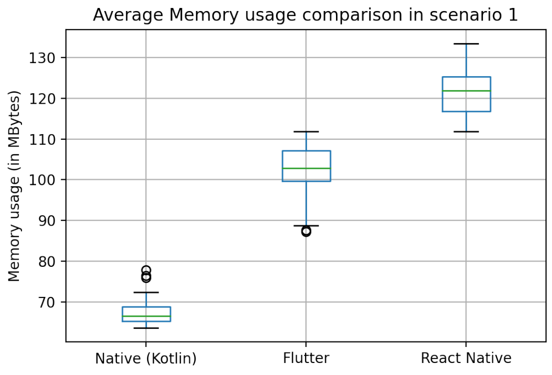
Similar performance comparison pattern on memory usage is also observed in the second scenario. Interestingly, native Android application consumes lower memory resource in this scenario than the first one, while the memory usage of both Flutter and React Native show the opposite. This shows that performing infinite scroll naturally consumes more memory than taking picture by camera in native Android application. However, both Flutter and React Native do not have direct access to camera, unlike the native application, thus needing more memory resource to process and translate more data to access the camera through native API. Based on this finding it can also be observed that in general the ratio of memory usage between Flutter and native apps is higher than the ratio of CPU usage between Flutter and native apps. Lastly, the maximum memory usage in the entire experiment of the second scenario shows similar pattern as that of the first scenario. In another words, Flutter is better at managing the application's CPU usage than the memory consumption.

4.3 Response time

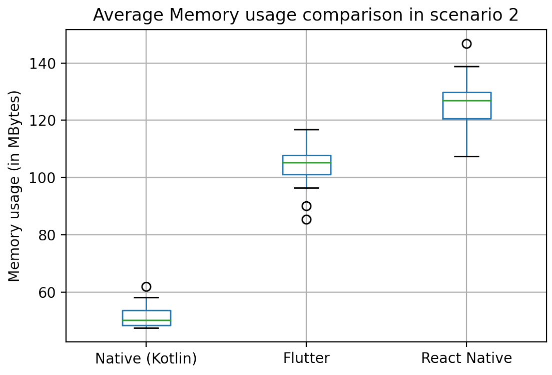
Measuring the response time in infinite scroll scenario is challenging because the way limiting scroll is handled differently in each framework, either by time limit or number of scroll. For this reason, the response time results in the first scenario are omitted.

Table 3: Response time overall mean comparison in accessing camera

Scenario	*Native*	*Flutter*	*React Native*
2	1532.667 ms	1353.233 ms	1725.480 ms



(a)



(b)

Figure 3: Average memory usage comparison in infinite scroll scenario (a) and taking picture by camera scenario (b)

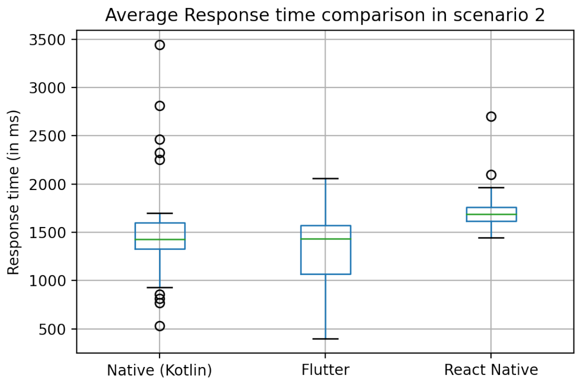


Figure 4: Overall mean comparison of response time in accessing camera

In the second scenario, the response time is only measured specifically when the application is accessing the camera because this part consumes the most CPU usage. The results of response time performance measure are shown in Figure 4 and Table 3. Surprisingly, Flutter experiences the fastest average response time upon accessing the camera and then followed by native Android and React Native applications. As we can see, the average response time is between 1 to 2 seconds in all frameworks, but in some cases it can reach more than 2 seconds (see Figure 4) which is quite long. Furthermore, the overall mean difference between each framework is about 150 to 200 ms which is not so significant, but when it is added with the other processes in the scenario the difference would be more significant or less, depending on the response time performance in the other specific process.

4.4 Frame rate

The boxplot as well as the overall mean of average of the frame rate measurement comparison in infinite scroll scenario are displayed in Figure 5a and Table 4 respectively. The results show that native Android and Flutter have comparable performance in terms of frame rate, although native Android is slightly better than Flutter. Based on this result, the frame rate performance in all framework is below 60Hz which is the typical mobile device refresh rate. However, if we look closer into the entire infinite scroll scenario, it is started when the application is launched and then the timeline is loaded into the application. These two events in the scenario would explain why the overall average frame rate is below 60 frames per second in all frameworks.

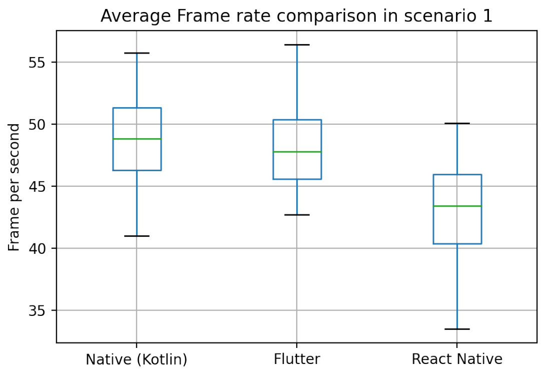
Table 4: Overall mean comparison of frame rate in both scenario

Scenario	*Native*	*Flutter*	*React Native*
1	48.626 fps	48.613 fps	43.90 fps
2	27.704 fps	25.05 fps	22.21 fps

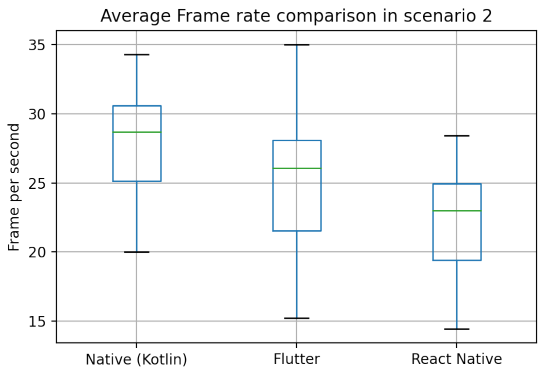
Similar trend in response time comparison can also be observed in the second scenario as shown in Figure 5b and Table 4. However, the difference between native Android and Flutter's frame rate in this scenario is more noticeable than that of the infinite scroll scenario. Moreover, overall mean of average frame rate in this scenario is way lower than 60 frames per second. The reason for such low frame rate in this scenario would be because it involves several activities that are more static in comparison to the first scenario, including access to camera and storing the taken picture into a temporary storage.

4.5 Application size

Table 5 shows the size of application installer with .apk extension and the application size after it is installed in the device. Based on this finding, it can be concluded that in general native Android application has the lowest application size and then followed by React Native and then Flutter applications. While the difference of .apk size among all frameworks is quite comparable, the size of application after installation differs significantly between the



(a)



(b)

Figure 5: Frame rate comparison in infinite scroll scenario (a) and taking picture by camera scenario (b)

native Android application and both cross-platform development frameworks. With similar feature and nearly identical appearance, React Native application size is more than 7 times larger than the native Android one while Flutter is almost 10 times larger. This is because both cross-platform frameworks require several files to allow the execution in multiple platforms, including iOS and Android. Besides, unlike native Android app that is directly compiled to its operating system's API, both React Native and Flutter require some additional components to run the application, i.e. *bridge* in React Native, and *flutter engine* as well as *icu data* in Flutter.

Table 5: APK size and application size comparison

	Native	***Flutter***	***React Native***
APK size	2.9 MByte	5.4 MByte	4.1 MByte
Application size	10.43 MByte	102 MByte	76.08 MByte

5 CONCLUSION

In this research, a comparative analysis of cross-platform frameworks apps, i.e. Flutter and React Native, with native Android app has been conducted, along with the development of a social media like test app and automated UI test scenario. The performance measurement results show that native Android app as the baseline has the best performance, while Flutter has better performance than React Native in terms of CPU usage, memory usage, response time and frame rate.

This research may be continued into several directions. First, native iOS app as well as other cross-platform app development frameworks may be considered, such as Xamarin and Ionic. Second, real devices may be considered for testing instead of device emulator to get more realistic performance results. Third, other native APIs, such as location, orientation, etc, may also be considered in the testing scenario to get more insight on the cross-platform frameworks performance with respect to those native APIs. Lastly, networking aspect may be considered as well to have more realistic app usage, especially in the context of social media app.

ACKNOWLEDGMENTS

This research is supported by Universitas Indonesia under "Publikasi Terindeks Internasional (PUTI) Prosiding" research grant NKB-884/UN2.RST/HKP.05.00/2020. The authors wish to thank the Faculty of Computer Science, Universitas Indonesia, for the support of this research.

REFERENCES

[1] P. K. Aggarwal, P. S. Grover, and L. Ahuja. 2019. A Performance Evaluation Model for Mobile Applications. In *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)* (Ghaziabad, India). 1–3.
[2] D. Chunye, S. Wei, and W. Jianhua. 2017. Based on the analysis of mobile terminal application software performance test. In *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (Kanazawa, Japan). 391–395.
[3] M. Gilroy-Ware. 2017. *Filling the Void: Social Media and The Continuation of Capitalism.* Watkins Media.
[4] Xiaoping Jia, Aline Ebone, and Yongshan Tan. 2018. A Performance Evaluation of Cross-Platform Mobile Application Development Approaches. In *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)* (Gothenburg, Sweden). 92–93.
[5] Inês Coimbra Morgado and Ana C. Paiva. 2018. Mobile GUI Testing. *Software Quality Journal* 26, 4 (Dec. 2018), 1553–1570.
[6] I. C. Morgado and A. C. R. Paiva. 2015. Testing Approach for Mobile Applications through Reverse Engineering of UI Patterns. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)* (Lincoln, NE, USA). 42–49.
[7] Inês Coimbra Morgado and Ana C. R. Paiva. 2019. The IMPAcT Tool for Android Testing. *Proc. ACM Hum.-Comput. Interact.* 3, EICS, Article 4 (June 2019), 23 pages.
[8] C. M. Pinto and C. Coutinho. 2018. From Native to Cross-platform Hybrid Development. In *2018 International Conference on Intelligent Systems (IS)* (Funchal - Madeira, Portugal). 669–676.
[9] K. Shah, H. Sinha, and P. Mishra. 2019. Analysis of Cross-Platform Mobile App Development Tools. In *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)* (Bombay, India). 1–7.
[10] M. Willocx, J. Vossaert, and V. Naessens. 2016. Comparing Performance Parameters of Mobile App Development Strategies. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)* (Austin, TX, USA). 38–47.

UX	user experience
UI	user interface
GUI	Graphical User Interface
OEM	Original Equippment Manufactur
VM	Virtual Machine
fps	frame per second