A Comparative Study between Applications Developed for Android and iOS

Robert Győrödi
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Vlad Georgian Adrian
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Doina Zmaranda
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Cornelia Győrödi
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

***Abstract*—Now-a-days, mobile applications implement complex functionalities that use device's core features extensively. This paper realizes a performance analysis of the most important core features used frequently in mobile application development: asynchronous multi-threaded code execution, drawing views/elements on the screen and basic network communications. While multiple mobile platforms have emerged in recent years, in this paper two well-established and popular operating systems were considered for comparison and testing: Android and iOS. Thus, two basic applications featuring the same functionality and complexity were developed to run natively on both platforms. Applications were developed by using development languages and tools recommended for each operating system. This paper aims to highlight the differences between the two operating systems by analyzing core feature performance metrics for both functionally identical mobile applications developed for each platform. Results obtained could be further used for guiding the optimization of application's development process for each considered operating system.**

Keywords*—Android; iOS; mobile application development; mobile device core features; common scenario performance comparison; development optimization

I. Introduction

The rapid development of the mobile devices industry has culminated with the rise of modern operating systems, specifically optimized to use the advantages and limits of the hardware environment in order to interface with the user.

While many mobile operating systems have been developed in the recent years, in today's market, the most widely adopted are Android [7], developed by Google and iOS [8] developed by Apple.

Being open-source software, Android has been extended and used by some of the major mobile device manufactures, being advantageous from the development cost perspective and offering a great level of customization.

Apple's approach to a mobile operating system was quite different, as iOS was developed to run on a very specific set of devices, which feature an established list of hardware components. The close relationship between the hardware setup and the operating system development have tied the success of iOS platform to the popularity of its host devices. This approach, however, also represents an advantage, as iOS was optimized to have a responsive and fast interface, designed specifically around its hardware limitations.

The comparative study developed in this paper will concentrate on the analysis of three important core system features that are used extensively in every modern mobile application: asynchronous multi-threaded code execution, drawing views/elements on-screen and basic network communications.

A specific architecture together with several tests was developed to measure the time needed for the operating system to perform tasks that involved each feature. The observed performance differences for individual tasks are expected to be relatively small, with only a few milliseconds separating one device from another. These discrepancies will, however, become noticeable in real-world applications, where core features are combined and used recurrently to introduce new functionalities.

The performance measurements were applied on a basic application developed to run on Android and iOS. During the development phase, the recommended development languages and tools were used: for the Android operating system, the Android Studio [9] environment was used to develop the application and the main programming language chosen was Java [10]; the application authoring tool XCode [11] was used for the iOS implementation alongside the Objective C [12] language.

Finally, an exhaustive analysis of obtained results was made and several guidelines for application development optimization were presented.

II. Related Work

While several comparison studies between the two operating systems exist in the literature, they are merely

focused on comparing existing features and architectures than taking into consideration application development issues.

For example, a comparison related to various factors that influence security on both platforms, such as application provenance, application permissions, application isolation, and encryption mechanisms is presented in [1], [2], [6]; [3], [4] present a comparison of the two operating system architecture together with provided features and frameworks for application development; several tools for cross-mobile application development are proposed in [5]; also, a comparison based on availability and capabilities of different set of UIs is described in [6].

Moreover, several papers in the literature realize comparisons based on detailed analysis of market share of smart phones having different mobile operating systems [3], but also on advertisement and overall impact on the consumers [6].

With the general complexity of both operating systems expanding on each new version iteration, more features become available for application developers. In this context, analyzing from the performance point of view of the most important core features used in mobile application development for both operating systems could be very helpful for further application development processes. Consequently, the paper approaches a very important aspect, by guiding the optimization process to potential slow or inefficient parts of the application specifically on each device.

The paper is structured as follows: next chapter presents the two mobile applications developed for each operating system together with the web platform used by both applications for receiving HTTP requests and sending JSON responses. Chapter IV describes the developed testing architecture and the performance tests carried on. Based on the results of the comparisons, several conclusions regarding optimization issues for application development are drawn and presented in the conclusion chapter.

III. Presentation of the Mobile Applications

A native mobile application was developed for each operating system (Android and iOS) in order to study the performance and development differences. Both mobile applications feature the same functionality and scene structure, with differences only being visible at the user interface level, where some elements diverge in order to respect the design guidelines recommended by each operating system manufacturer.

The mobile applications are complemented by a web platform built on top of the Laravel [13] framework. The platform receives signed data requests through the HTTP protocol and it then sends back responses containing JSON [14]-encoded structured data that is extracted and compiled from a MySQL [15] database.

From a functionality standpoint, each application allows the user to view promoted commercial locations and related events or picture galleries for a specific geographical area. The web platform provides the data, which is displayed within the mobile applications, allowing registered users to perform CRUD operation over the datasets representing the locations, events and galleries.

The mobile applications were designed to use a hierarchical navigation system that guides the user to the desired content. Using this approach, different category and entry lists were created for each data type alongside shortcut paths that allow the user to reach the content in an efficient manner. The general structure of the scenes is described in Fig. 1.

In recent years, several frameworks such as Xamarin [16], Cordova [17] or React Native [18] were created, allowing the development of mobile applications that run on multiple operating systems using a single codebase solution. Using the hybrid application development approach, while it does have its advantages, was not preferred in this case because the purpose-built frameworks introduce another layer over the native code, making testing much more difficult and the results inaccurate.

Therefore, a native approach was chosen for the application development process on each platform, using the tools recommended by each operating system manufacturer. This allowed each codebase to exploit the advantages of its operating system separately, emphasizing the major differences in implementation and optimization between the platforms.

Both applications followed similar MVC (Model-View-Controller) architectural pattern [19], having clear delimitations between classes and code sections that handle the application behavior, the user inputs and the information representation form. Model classes were created to describe and handle the structured data displayed using the user interface.

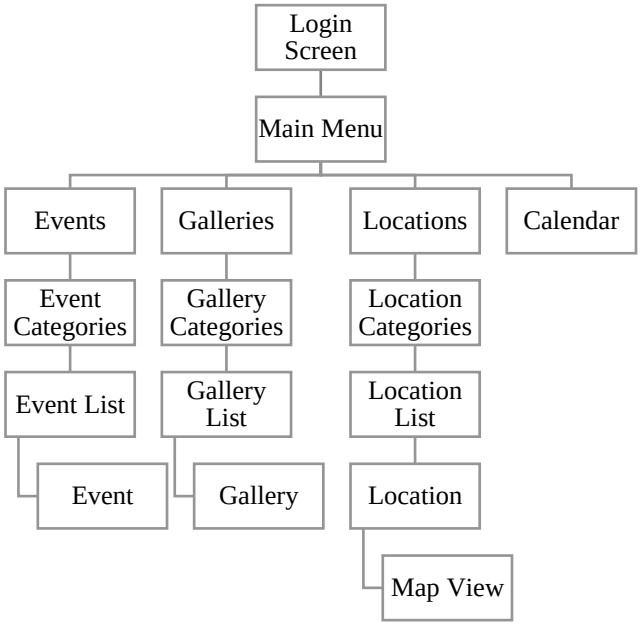

Fig. 1. The application scene structure 1.

A. Android Application Development

For the Android operating system, the choice for the development language was straightforward as only the Java language is supported natively. The visual structure for each scene was built using the default method of declaring UI elements in separate XML [20] files.

Since the Android application development process lacks a tool for scene navigation management, the rules that define the order of scenes were described within the *Activity* and *Fragment* derived classes [7].

The responsiveness of the user interface was facilitated by isolating all long-lasting or complex operations in secondary threads. This approach reduced the amount of workload on the main thread, which was then able to handle user interface updates and input detection without further delays.

Slow operations, such as establishing a network connection or data decoding, were implemented by deriving the *AsyncTask* class form Java [10]. Once a data set is prepared, the main thread is notified of this change using the observer design pattern optimized for multiple listeners.

Activities were created for each section context, leaving all subsequent scenes to be handled by using Fragments. For scenes that involved grids and lists, the application took advantage of the reusable item view approach, minimizing the amount of memory used to store complex arrays of data.

B. iOS Application Development

The iOS application authoring tool XCode offers two native options regarding the main development language: Objective C and Swift. Currently, Swift is being promoted for the development of new application that run in the Apple ecosystem. But, for this implementation process, Objective C was chosen as it is much more mature language with clearly outlined best practices, coding styles and an existing suite of well tested and stable third-party libraries.

Unlike the approach used by Android Studio, the visual structure of the whole application can be managed in a single file using the Storyboard [21] environment. Each individual scene was constructed using static View Controllers for standalone pages and Collection View Controllers to list structured data [12].

The navigation paths between the main scenes (*segues*) were described using the graphical user interface and references were created inside the header files for each view controller, allowing for scene transitions to be performed automatically for events triggered when a background task is complete or for user inputs.

Asynchronous tasks were handled using NSOperation [12] instances that notified the main application thread once all the processing stages were completed. The network connections were managed using the AFNetworking [22] library that extends and simplifies the networking abstractions already available in Cocoa [23], the application development environment for iOS and OS X.

IV. Performance Tests

After developing the mobile applications, the differences between iOS and Android were highlighted by analyzing the specific performance metrics and signature. Since both applications were created using the native tools and development languages, they take advantage of optimizations offered by each operating system.

A. Testing Architecture

From a development standpoint, each application uses the advantages of multi-threading, an approach which improves the responsiveness of the user interface. Standalone long-lasting tasks such as network downloads, data decoding and image conversions were executed in separate threads, leaving the operating system to decide which hardware cores to use in order to perform each operation.

Within the applications, the most computational intensive section was used to highlight the differences between the operating systems. As such, a predefined *location* scene was loaded on each tested device. The data received from the server represents a JSON-encoded string containing the location information; the full data size is 1 MB total, including HTTP headers. The amount of time required to complete each test is expected to be directly proportional to the size of the source data.

The user interface for the *location* scene was created using the following native graphical elements available on both operating systems: adjustable text labels, an image view and structural layout groups. The components of the user interface were displayed prior to running each test in order to maintain the computational cost low for each draw cycle.

At the time of writing this paper, there are no official devices that offer support for both operating systems, meaning that the hardware components must also be taken into consideration while interpreting the results. The discrepancies at a hardware level were minimized by also emulating real devices in a shared environment.

For each device, a total of three tests were performed, measuring the time needed to complete each specific task. The tests were performed on the following physical devices running the latest versions of their respective operating system: Samsung Galaxy S8+ (using Android 7.0) and Apple iPhone 7 Plus (using iOS 10.0). Well-established and leading benchmarking tools, such as GeekBench [24], position both devices very close to each other from a performance perspective.

The iOS device is roughly 72.5% faster in single-core operations however it loses its edge in multi-core tasks where it is 8.6% slower than the Android counterpart. Table 1 presents the most relevant hardware differences between the two devices.

TABLE I. THE PHYSICAL DEVICES USED IN THE TESTING PHASE

Device	**Samsung Galaxy S8+**	**Apple iPhone 7 Plus**
CPU (cores)	8	4
CPU (clock)	4 x 2.35 GHz 4 x 1.9 GHz	4 x 2.34 GHz
RAM Memory	3GB	4 GB
GPU	Mali – G71	PowerVR Series7XT Plus

In an effort to reduce the hardware differences to a minimum, the performance tests were also executed on emulated devices. For the emulation process, the devices with the most advanced specifications were chosen from the available options, ensuring several criteria such as memory size or display resolution remained consistent.

For the Android platform, a virtual device that used the Google Pixel definition file was created by using the native tools embedded in Android Studio. For iOS, a Simulator [25] instance was launched form the XCode environment for the iPhone 7 Plus device.

The emulated devices and the web-based platform that supplies data for the mobile applications used a host computer with the hardware/software setup presented in Table 2.

TABLE II. HARDWARE ARCHITECTURE OF THE SERVER

CPU	Intel 3570K
CPU (cores)	4
CPU (clock)	4x4.4 GHz
RAM Memory	16GB
GPU	AMD Radeon 280x
Network Link State	1000 Mbps, Full Duplex
Storage Type	SSD

Network related delays and issues were minimized by constructing a local network where only the server and the tested device were able to interact. The mobile devices were connected to the local network using 802.11n standard over the 5GHZ band (Wi-Fi). For the emulated devices, a bridged connection over the host computer adapter was used in order to connect to the local network. The testing architecture components are described in Fig. 2.

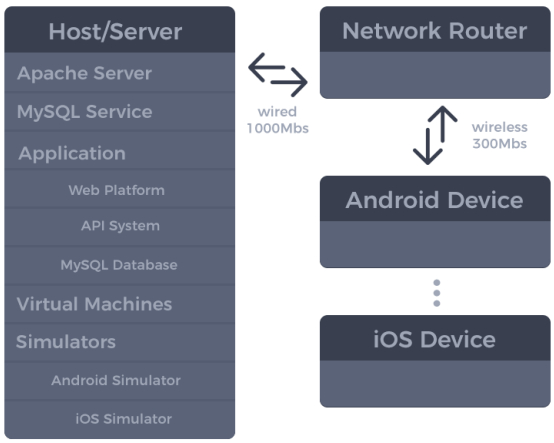

Fig. 2. The testing architecture.

B. *Testing Results*

Before running each test, the mobile devices were restarted and all non-essential background processes and applications were closed.

The time needed to perform an operation was determined by analyzing the timestamp values echoed in the development platform console. This approach allows accurate measurements down to 1ms as it relies on the mechanisms used by the operating systems. Each test was performed several times *(p=10*) in order to obtain the average values.

The first test measures the time needed to establish a connection with the server and to retrieve the location information. The data payload is small in order to prevent any network related delays.

The values obtained by running the first performance test can be visualized in Table 3 and Fig. 3.

TABLE III. NETWORK PERFOMANCE RESULTS

Device	**Transfer Time (ms)**		
	MIN	**MAX**	**AVG**
Physical devices			
Samsung Galaxy S8+	165	183	171.5
Apple iPhone 7 Plus	187	382	272.5
Emulated Devices			
Google Pixel XL	173	252	206.5
Apple iPhone 7 Plus	347	517	401.5

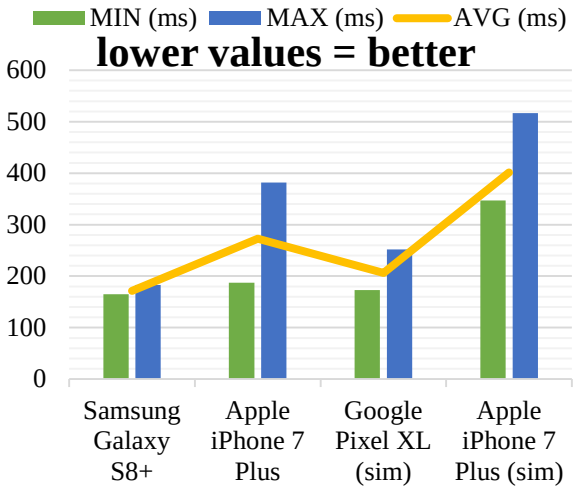

Fig. 3. Network performance results chart.

By analyzing the average values for each device, it can be observed that the Android platform is faster by a small margin in terms of data download times. One aspect that must be emphasized is that on iOS, establishing the initial connection to the server took longer than expected on each device, only with subsequent network calls being more consistent. The implications of using emulated devices become clear as substantial performance differences (slower by more than 30ms) are measured, even in such cases, where the host computer has more computational power than the original device.

Once the data is downloaded and available, a second test is executed, measuring the time needed to transform the raw JSON data into string values that are processed afterwards into model instances. The test results are highlighted in Table 4 and Fig. 4.

The JSON parsing task is launched in a new thread in order to minimize any interference with the main thread that controls the user interface.

Several conclusions can be drawn from the second test results. Since both operating systems allow for tasks of other applications to persist in the background, the performance of the current task is directly controlled by the available core count and the efficiency of the operating system's task scheduler. The multi-threaded approach taken during the development phase has improved the performance on devices that are advantaged by a high number of physical cores.

For both real and emulated devices, the iOS platform had faster average execution times and lower limits. In a simulated environment, Android needed twice the amount of time to process the same amount of data.

The developed application processes structured data in small bursts meaning that higher core clocks do not necessarily improve the overall performance.

TABLE IV. JSON ASYNCHRONOUS PARSING PERFORMANCE RESULTS

Device	Processing Time (ms)		
	MIN	MAX	AVG
Physical devices			
Samsung Galaxy S8+	16	29	24.2
Apple iPhone 7 Plus	10	21	14.25
Emulated Devices			
Google Pixel XL	10	14	12.75
Apple iPhone 7 Plus	5	7	6

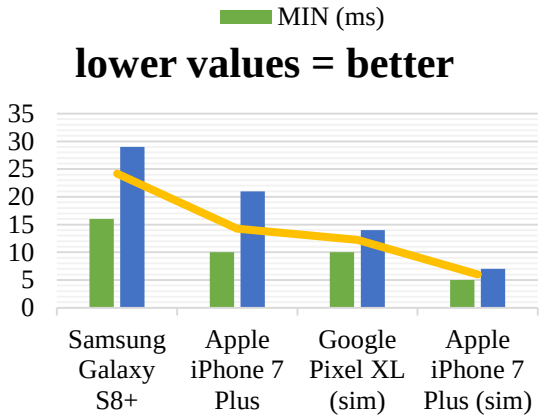

Fig. 4. JSON asynchronous parsing performance results chart.

With the location data downloaded and processed, the third test measured the time needed to update the UI elements on the screen. This actually measured the time needed to display only static data, ignoring elements, which still have to be handled asynchronously, such as image downloads.

From a structural perspective, the location scene has a container element, a *RelativeLayout* on Android and a ViewController on iOS. Inside the container, there is a scroll view that enables all the child elements to be visible on the screen. The relevant data is displayed using a set of labels and a single *ImageView* [7]. For the purpose of this test, we did not take into account the time needed for the image to be displayed, since this would require additional network transfers.

TABLE V. DRAW PERFORMANCE RESULTS

Device	Draw Time (ms)		
	MIN	MAX	AVG
Physical devices			
Samsung Galaxy S8+	19	25	22.5
Apple iPhone 7 Plus	13	20	16
Emulated Devices			
Google Pixel XL	13	19	17
Apple iPhone 7 Plus	3	5	4

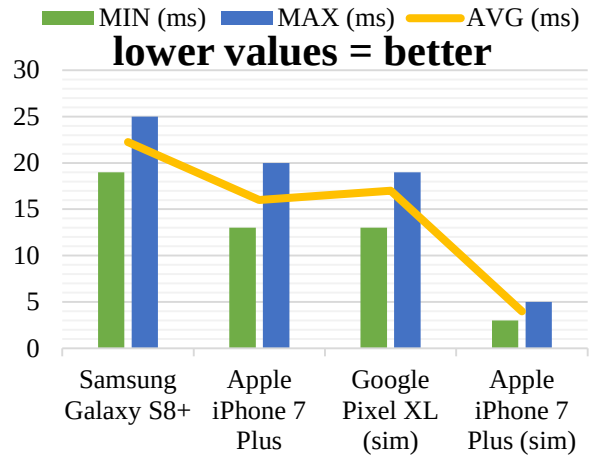

Fig. 5. Draw performance results chart.

The results of the third test, displayed in Table 5 and Fig. 5, show a distinct advantage of the iOS platform over Android for the time needed to draw a scene. The performance difference can mainly be attributed to optimizations at the operating system level for GPU-accelerated UI elements draws.

On physical devices, the time needed to update the UI was relatively close to one display frame, with Android being slower. Tasks which require more than 16ms (60 frames/second = 16.67ms) will affect the fluidity of the user interface. When tested in a simulated environment, the iOS device was significantly faster, taking advantage of the host hardware.

V. Conclusions

A set of two basic applications featuring the same functionality and complexity was developed to run natively on Android and iOS platform. The tests that were performed and presented in this paper analyze several important core features by creating, for each feature, a particular scenario in the implemented applications and architecture. Tests have not outlined any operating system to be more efficient than the other, at least not from an overall application developer perspective, each platform being more efficient and excelling for different tasks.

For network related tasks, Android had a clear edge over iOS, however the time difference was spent mostly on establishing the connection to the server, while the relevant data was retrieved in a similar time frame on both operating systems.

The JSON parsing and decoding test was intended to display the efficiency of the task scheduling part of the operating system and the processing speed of big strings. The results of this test also reflected the hardware differences between real devices, however, in the end, both platforms performed similarly, with iOS being ahead by only a few milliseconds.

On the draw performance test, iOS was clearly faster than Android with views and UI elements being drawn on the screen within the time frame limit to not cause user interface fluidity issues. The emulation process also proved to be much more efficient with iOS devices.

The design constraints of each application might create a situation that would benefit more from the device hardware and the software advantages or limits of one platform over the other. For example, according to the performed tests, an application that relies heavily on views being drawn on the screen as soon as possible will perform better on iOS, while other application that use a lot of network communications will behave better on Android.

Consequently, the developed performance tests and their results can be used to anticipate where the slow or inefficient parts of the application will be on each device. Developers can then author applications that will behave and perform similarly on both operating systems, either by optimizing their source code or by designing functionality around these limits.

References

[1] K. Jamdaade1, A.Khairmode, and S. Kamble, “A Comparative study between Android & iOS” in International Journal of Current Trends in Engineering & Research (IJCTER) e-ISSN 2455–1392 vol 2, no. 6, pp. 495 – 501, June 2016

[2] I.Mohamed and D. Patel, “Android vs iOS Security: A [2] Comparative Study”, *in IEEE 12th International Conference on Information Technology - New Generations* (ITNG), INSPEC Accession Number: 15180414, DOI: 10.1109/ITNG.2015.12, 2015

[3] S. Jaiswal and A. Kumar, “Research on Android app Vs Apple app Market: Who is Leading?” ISSN: 2319-7242, vol. 3, no. 4, pp. 5553-5556, April 2014

[4] D. Singla and L. Mendiratta, “ANDROID VS IOS”, *IJIRT*, vol. 1, no. 5, ISSN: 2349-6002, 2014

[5] N. M. Hui, L. B. Chieng, W. Y. Ting, H. H. Mohamed and M. R. H. M. Arshad, “Cross-Platform Mobile Applications for Android and iOS”, *IFIP WMNC*, 2013

[6] S. Annapurna, K.V.S. Pavan Teja, Y. S. Murty, “A Comparative Study on Mobile Platforms (Android vs. IOS)”, *International Journal of Advanced Research in Computer Engineering & Technology* (IJARCET), vol. 5, no. 3, March 2016

[7] B. Phillips, C. Stewart, K. Marsicano, Android Programming: The Big Nerd Ranch Guide, Big Nerd Ranch Guides, 3rd Edition, 2017

[8] C. Keur, A. Hillegass, iOS Programming: The Big Nerd Ranch Guide, Big Nerd Ranch Guides, 6th Edition, 2017

[9] Neil Smyth, Android Studio 2.3 Development Essentials - Android 7, CreateSpace Independent Publishing Platform; 1st edition, 2017

[10] B. Abazi, Android Development with Java: Step by step guide to build applications, CreateSpace Independent Publishing Platform, 2017

[11] Apple xCode Introduction – Accessed September 2017 – https://developer.apple.com/xcode/

[12] S. G. Kochan, Programming in Objective-C (Developer's Library), Addison-Wesley Professional, 6th edition, 2013

[13] A. Nutile, Laravel 5.x Cookbook , Packt Publishing, 2016

[14] JSON language– Accessed September 2017 – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

[15] MySQL Documentation – Accessed September 2017 – https://dev.mysql.com/doc/

[16] Xamarin developer introduction – Accessed September 2017 - https://developer.xamarin.com/

[17] Cordova developer introduction – Accessed September 2017 - https://cordova.apache.org/docs/en/latest/

[18] React Native developer introduction – Accessed September 2017 - https://facebook.github.io/react-native/docs/getting-started.html

[19] E. Buck, D. Yacktman, Cocoa Design Patterns, 1st Edition, Addison-Wesley Professional, 2009

[20] XML language – Accessed September 2017 - https://developer.mozilla.org/en-US/docs/XML_Introduction

[21] Storyboard developer introduction – Accessed September 2017 - https://developer.apple.com/library/content/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

[22] AFNetworking Framework introduction – Accessed September 2017 - https://github.com/AFNetworking/AFNetworking

[23] M. Weiher, iOS and macOS Performance Tuning: Cocoa, Cocoa Touch, Objective-C, and Swift (Developer's Library), Addison-Wesley Professional, 1st Edition, 2017

[24] GeekBench iOS and Android benchmark results – Accessed September 2017 – https://browser.geekbench.com/android-benchmarks , https://browser.geekbench.com/ios-benchmarks

[25] iOS Simulator – Getting Started – Accessed September 2017 - https://developer.apple.com/library/content/documentation/IDEs/Conceptual/iOS_Simulator_Guide/GettingStartedwithiOSSimulator/GettingStartedwithiOSSimulator.html