e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:04/Issue:06/June-2022 Impact Factor- 6.752 www.irjmets.com

DISTINCTION OF MOBILE FRAMEWORKS: FLUTTER VS NATIVE APPS

Dr. P. Srinivasa Rao*1, Baradi Pavan*2, Atharva Srivastava*3,
K. Venkata Amani*4, Aakriti Sharma*5

*1Professor, HOD, Dept. Of Computer Science And Engineering, JB Institute Of Engineering & Technology, Hyderabad, Telangana, India.

*2,3,4,5Students, Dept. Of Computer Science And Engineering, JB Institute Of Engineering & Technology, Hyderabad, Telangana, India.

ABSTRACT

To keep up with the new apps that appear every day, mobile apps need to withstand high demands. Substantial performance and creative aesthetics are preconditions for designing mobile applications. There are various tool options at the time of creation, one of which is a native application, which is more powerful and suitable for mobile environments. Another option is a tool that requires only one codebase on many platforms, making it easier to maintain. Flutter is Google`s open-source user interface (UI) toolkit that you can use to create cross-platform apps with a single code base that looks native. In this paper, we will discuss Flutter and how native applications are said to be greater when it comes to performance and behavior. From a computational speed perspective, experiments were conducted to evaluate Flutter as a cross-compiler for two native programs consisting of Kotlin and Android Studio, and Swift and XCode. A survey was conducted to investigate whether there are differences in user perceptions of appearance and animation. A literature review was conducted to complement the results of the experiments and surveys and provide a background to the problem. Flutter is a brand-new tool that is rapidly gaining attraction. The conclusion is that a Flutter application can compete with a native program in terms of CPU performance, but it is less developed in terms of animation. Flutter does not require sophisticated code to create a basic application and utilizes many fewer lines of code in development than native. The conclusion is that Flutter is best suited for developing small to medium-sized apps, but it can evolve to overcome its existing shortcomings in the animation field.

Keywords: Performance, Flutter, Native-Applications, User-Interface, Development.

I. INTRODUCTION

The purpose of this paper is to gauge Flutter as a UI framework for developing mobile applications. A method that is used for producing mobile applications is by developing them from a specific platform. Another method is to write a code base which can be compiled into different platforms. It is a cross-platform procedure and is popular as it is flexible and fast. Choosing between native and cross-platform is often a question of cost and the proper way of developing. Flutter is an open-source framework and works with the programming language Dart that can create and develop mobile applications with a single code base and compiles the code into both Android & iOS. It was developed by Google in 2018 to create applications that inherit the same type of feel, visuals, and performance like would have been emerged as a native mobile application. This paper deals with Flutter's comparison to native applications in the performance of the CPU, UI, space & time complexity, and code needed to perform its designated task. Many of the terms that are mentioned in this section are explained in the paper.

II. LITERATURE SURVEY

Amatya (2013) investigates the platform, development environment, and code base to determine the differences between cross-platforms and native. Amatya concluded that while native is a good match for larger applications, it is not the best match for every application because it is more costly and tedious than cross-compilers.

Axelsson and Carlstr om's report is also a performance comparison of native versus cross platform. It distinguishes between React native and Swift and a Java application. The study's findings indicated that the performance of the application builds was similar with slight changes. The animations were the area where the discrepancies were most visible.

Reading similar work led to the conclusion that while there are numerous articles exploring cross-platform and native techniques, there is a lack of performance testing comparing native and cross-platform approaches because most simply compare code and flexibility. However, Dalmasso et al. (2013) published a report that investigated how cross-platforms fared against each other. The authors investigated CPU and memory utilization, as well as cross-platform compatibility. The changes in these metrics were highlighted based on the dependency the program was using.

III. PROPOSED METHODOLOGIES

This section gives a deep understanding about Flutter, Swift and Kotlin, and the development, building environments that were used in the experiment. The aim is to help the reader and author understand how each language and environment works and what differentiate them in terms of functioning, management of looks and development by the means of this methodologies.

1. Native mobile applications:

Native applications in the field of mobile applications refer to the applications that are built to run on a specific platform or OS. Plenty of languages can be used for building native mobile applications like - Kotlin, Java and Swift.

Mainly, developers focus on building native applications because of the customization that is enabled on native devices such as camera access.

1.1 Native implementation of UI:

Native mobile applications are more fluid looking, with better animations and easier integration with the device. Native applications inherit the target platform looks and apply it to their appearance, which is known as Native UI, allowing the native applications to behave and look and feel the same as” native” system giving the user an experience same as the mobile platform OS itself.

2. Cross-platform mobile application development:

Cross-platform applications are software which can be used on another platform than what it was developed for. The principle of cross-platforming is to build and maintain only one code base, saving time in development.

3. Flutter

Flutter is a mobile SDK and UI tool developed by Google, based on Dart programming language, officially released in 2018. The developers’ goal is to deliver applications that provide performance, look, and feel of native applications.

4. Dart

Dart is a client-optimized language developed by Google, that can be used on multiple platforms. It provides a C-style syntax and is object oriented with compilation into both JavaScript and native based code.

4.1 Flutter/Dart UI management

Flutter uses widgets as the main concept for every component part that is built in Flutter. The widgets are created to look native, and developers can fully customize these to their liking. Flutter uses a material components library by default letting the developer use components that are ready from the beginning. Flutter has a functionality called hot reloading i.e., when changes are made, they are injected into the dart Virtual Machine and Flutter rebuilds the structure as soon as the changes were made.

4.2 Flutter compiling

Flutter works on Android by compiling the Dart code to native code with help of AOT compiling. Combining this with x86 libraries, created when compiling the dart code, are put into a runner, and built as an APK. Flutter runs similarly on an iOS system, where Dart code compiles into an ARM library and put as a runner, built as an. iPad. Flutter suggests the process is like how game engines work.

5. Kotlin

Kotlin is an open source, statically typed programming language developed by JetBrains. Kotlin is completely compatible with Java and supported on the Android platform.

5.1 Kotlin compiling

Kotlin compiles into compatible bytecode for Java when targeted on the JVM. Kotlin is a versatile programming language that functions on multiple platforms. When targeted to native applications, Kotlin will go through the LLVM to produce specific code for the platform.

5.2 Kotlin/Android UI management

The layout and UI elements can either be declared in XML or in code at run time. Drag and drop is available for the non-programmable layout creation. Both options need to have the UI parts connect to the code and will need to be initiated upon creation in the on Create method for the class layout. The object in the layout is referred to as view Groups and views. The views are called widgets which can be a text object or an image.

6. Swift

Swift is a programming language by Apple, made to be able to work on multiple of platforms and aims to be a replacement to the C-languages. Swift is managed as a group of projects where it is split into swift compiler, standard library, core libraries, LLDB library, swift package manager and Xcode playground support.

6.1 Swift compiling

Swift code is compiled down to machine code with seven level steps in the compiler. The parser is run at the beginning to check if there are any grammatical errors and show warnings. After the parser, a semantic analysis is carried out to see if it's safe to compile the code without errors. Clang importer then imports clang modules that can be referred from the analyzer. This is followed by something called a SIL generation and SIL guaranteed transformations. The first SIL runs another analysis on the code to improve optimization. The second SIL do data flow diagnostics to check so there are no uninitialized variables and results in a canonical SIL, meaning that it is a SIL that exists after the optimization and analyses has been done. There is an additional SIL step that is called the SIL Optimizations where extra high-level swift optimizations in additions to the ones before, are carried out. At the end there is an LLVM IR Generation which means that SIL is transformed into machine level code with the help of LLVM, which are compiler tools.

6.2 Xcode UI management

When developing a UI in Xcode, the Interface Builder is used to create storyboards. Scenes are used in storyboard to represent what is seen and what is happening on the device screen. Segues connect the scenes and holds uphold their relation. Scene objects can be dragged and dropped to create a new item to view. These items and controllers can be connected to code manually or with the help of Xcode assistant that can create or generate code automatically.

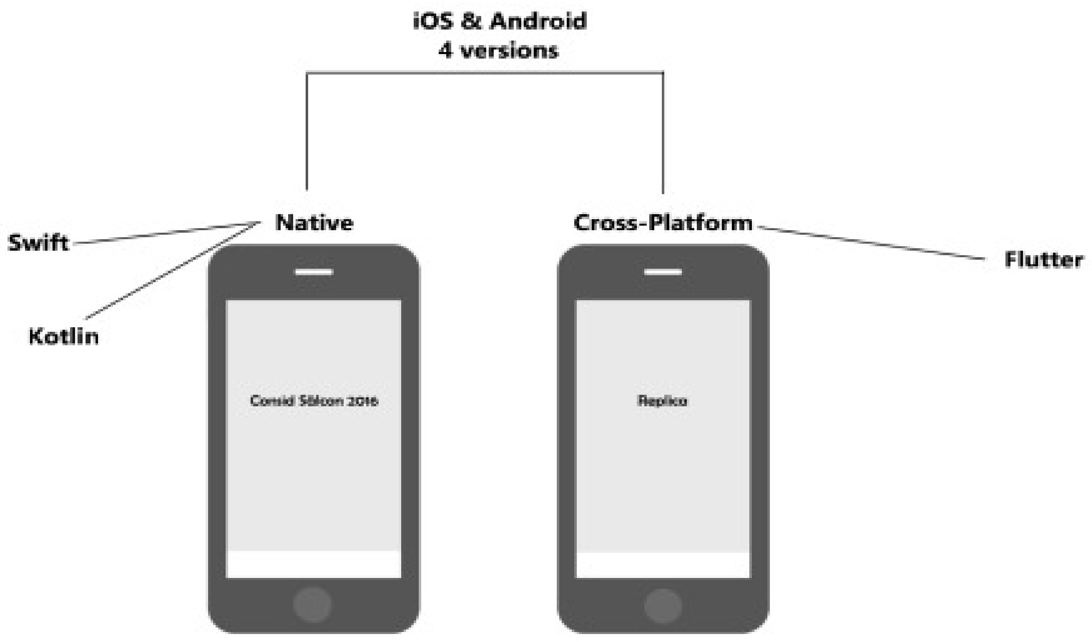


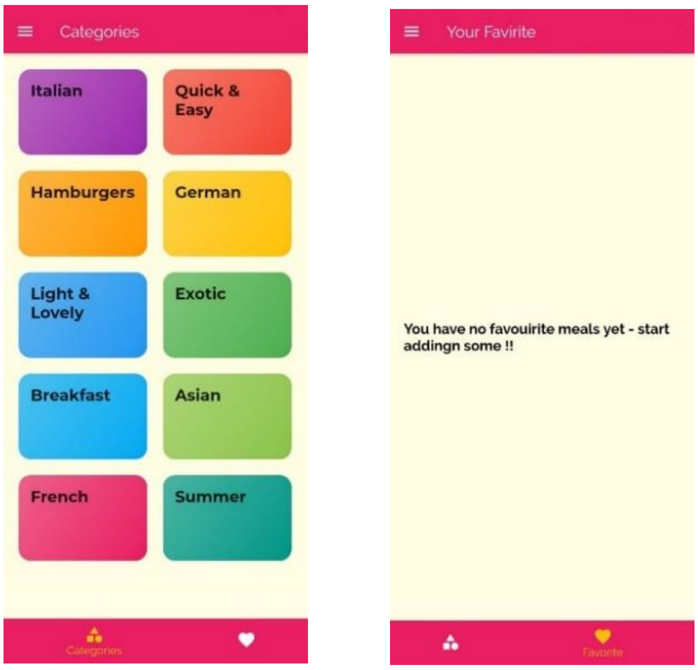
Figure 1: Graphic representation of the applications and their technology

IV. EXPERIMENT

This section describes the experiment and how it was planned, as well as its objectives and the procedure itself. It is done to help the reader understand how the experiment was carried out and planned. We created four applications with Flutter (Android and iOS), Kotlin and Swift. Applications are made to look alike and have code included with each document. Because of the project's time constraints and the need to handle complicated code, a very simplistic layout was chosen. The program interface had a navigation bar as well as two page views: "Categories" and "Favourites." The dashboard featured a category of different types of foods. At the top of each view was an app bar displaying the view name.

The purpose of this experiment was to determine the difference in run-time CPU utilization between Flutter and native programs. Another aspect that was investigated was the amount of code required by each development environment to achieve the desired aesthetics and functionality of the apps. This was a side-by-side comparison that showed how the native code bases compared to the Flutter code base.

The CPU utilization was measured manually three times each program build, and the maximum, lowest, and mean figures were recorded. Values greater than 0 were chosen as the lowest, as the CPU displayed 0 while the programs were inactive. Android studio profiler (Android) and XCode instruments were used to collect data (iOS). Because the output of the measuring tools was solely in the form of graphs, sampling was employed to calculate the mean value and facilitate measurement. After that, the standard deviation was determined for each set of numbers.



You have no favouirite meals yet - start addingn some !!

Figure 2: Images of final Android Flutter application

When testing CPU utilization, the travel path shown below was manually followed. It was done to collect a large amount of data on the application's functionality and to guarantee that all test measures adhered to the same rules.

1. Start the application
2. Wait for Dashboard view to load
3. Navigate to Favorite view
4. Navigate back to categories view
5. Navigate to Favorite view
6. Scroll up and down in list 4 times
7. Return to the Dashboard view

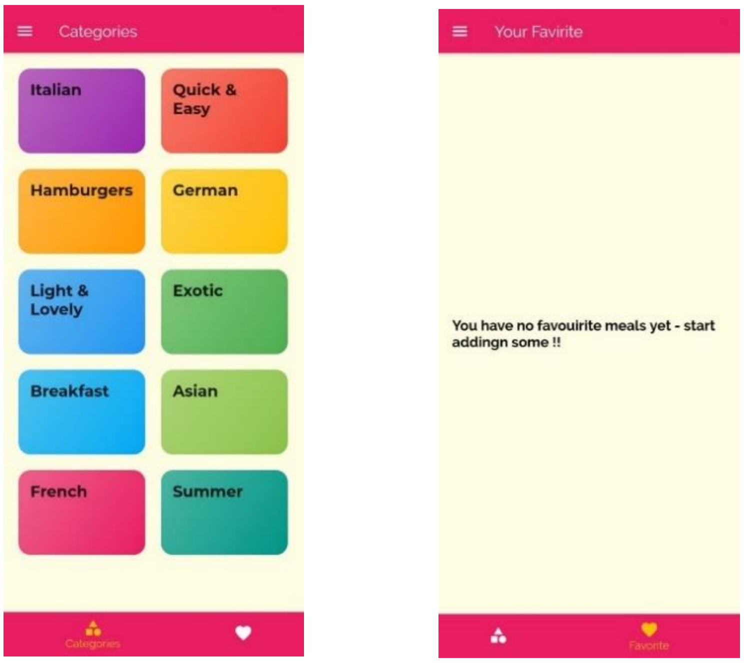


Figure 3: Images of final Android Native application

Although the Android Flutter program had a lower maximum CPU consumption, the native Android application had a lower mean value, and both had the same lowest value. Overall, the CPU performance of the Android applications was high at first, although not as high as that of the iOS applications.

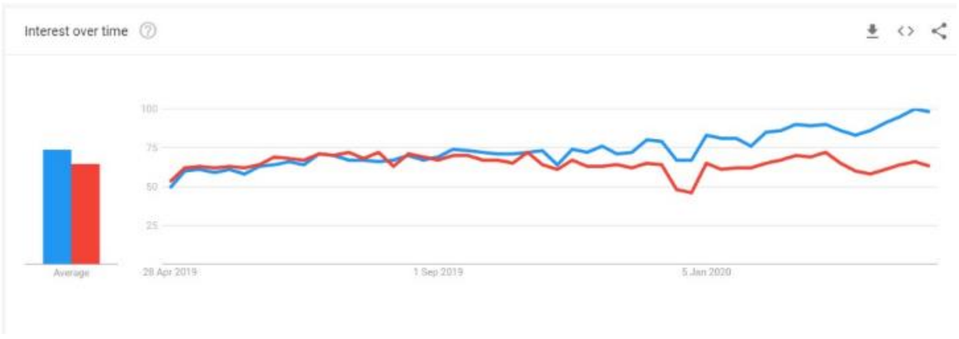


Figure 4: Graph of React Native and Flutter interest of search from Google trends

https://trends.google.com/trends/explore?q=Flutter,React%20Native

(React Native is red and Flutter is blue)

V. CONCLUSION

Flutter hence proves to be an important kit for the creation of new applications. It is gaining huge popularity and is seen as a replacement for React Native and the way is distinctive from native applications. The result of this experiment revealed that the CPU performance of Flutter iOS and native Swift iOS respective Flutter android and flutter native had a very small difference. But when compared to the performance of Flutter on iOS and Android, it was seen that Flutter performs better than native applications. There are multiple metrics that could still be used to consider the distinction between Flutter and Native Applications in performance. The code size of the Flutter application has around 125 lines which were comparatively lower than the lines of code that were written separately for iOS (363 lines) and android (217 lines). The best part was that Flutter did not require to write the code for different platforms separately instead the code was incorporated for both. When it comes to appearance, the differences that were drawn were very few. The major difference spotted was when it came to animations and certain items like menu, list, font and spacing. But animations would differ according to the requirements of the user. To summarize Flutter proves as a promising tool with great features for a single application base and to build cross-platform applications. The future of Flutter has high potential in various fields and metrics.

VI. FUTURE WORK

The focus of this paper has been to represent how Flutter can be distinguished from native applications and how a huge part of Flutter is continuously evolving in terms of features like adding widgets et cetera. As future work we would like to focus on how Flutter can be developed and used, also test its code.

Flutter supports materials that are implemented such that it could add to the look and feel of the Native applications. Another feature of Flutter to be investigated is the completely customizable widgets that can hugely impact the development positively. The Future Scope also includes if Flutter can work in a complete system that is a full system comprising back end and web page front end.

VII. REFERENCES

[1] S.Amatya, “Cross-platform mobile development: An alternative to native mobile development,” Linn´ euniversitetet, Fakulteten for teknik (FTK), Institutionen for datavetenskap (DV), diva2:664680: Diva, 2013, p. 67.

[2] O. Axelsson and F. Carlstr¨om, “Evaluation targeting react native in comparison to native mobile development,” Ergonomics and Aerosol Technology, LUP student papers, 2016, p. 105. [Online]. Available: https ://lup.lub.lu.se/student-papers/search/publication/8886469

[3] G. Flutter. (2020). Flutter setup, [Online]. Available: https://flutter. dev/docs/get-started/install

[4] Android. (2020). Android studio setup, [Online]. Available: https : / / developer.android.com/studio

[5] A. Inc. (2020). Xcode setup, [Online]. Available: https://developer. apple.com/xcode/

[6] Google. (2019). Google flutter sdk releases, [Online]. Available:
https: //flutter.dev/docs/development/tools/sdk/releases

[7] A. Inc. (2020). Swift compiling, [Online]. Available:
https://swift.org/ swift-compiler/#compiler-architecture

[8] G. Flutter. (2020). Google flutter animations, Online].Available:
https://flutter.dev/docs/development/ui/animations

[9] M. Bhatnagar. (2018). Statically typed languages vs dynamically typed languages, [Online]. Available:
https://android.jlelse.eu/magic-lies-here-statically-typed-vs-dynamically-typed-languagesd151c7f95e2b

[10] Jetbrains. (2020). Kotlin faq: What is kotlin? [Online]. Available:
https: //kotlinlang.org/docs/reference/faq.html

[11] Google. (2020). Material io components, [Online]. Available: https : // material.io/components

[12] O. Lifh and P. Lindholm, Recreating a native application in react native, 2018. doi: diva2:1213248.