Term Paper

Analysis of Cross-Platform Mobile Development Frameworks

October 16, 2021

Team Number: 3

Team Name: Team Helios

Course: Software Metrics and Processes

Course Code: SWE 4701

Submitted To

Md. Jubair Ibna Mostafa

Lecturer, Islamic University of Technology (IUT)

Contents

Analysis of Cross-Platform Mobile Development Frameworks

Sharmin Naj Mou[2,3,4,6] 170042074

Rizwanul Haque Khan[2,3,4,5] 170042078

Zarif Hossain[2,4,5,6] 170042019

Saimul Haque Shanto[1,2,4,5] 170042080

Ahmed Nusayer Ashik[1,2,4,5] 170042086

Fahmida Tasnim Lisa[2,3,4,6] 170042020

1 Abstract

Cross-platform mobile applications are very popular nowadays. It can reduce development time highly due to the fact that cross-platform apps can be made from a singular code base to be used on different platforms. Our study investigates the performance of mobile applications for the major cross-platform frameworks. Based on data collection on the performance metrics, our results indicate that certain frameworks perform equally or better than the other frameworks but there is no best scored framework in case of performance analysis for all the features in this study.

2 Introduction

Mobile applications are becoming more and more popular day by day. As mobile devices have made big advancements in the last couple of years, so did the need for sustainable tools and technology to support this advancement increase. Nowadays, we see that the general population is dependent on mobile applications for their daily lives. Google Play store and the Apple App store are the two leading mobile app markets from where apps can be downloaded. There are about million mobile apps available to download in both of these stores. So it is really competitive out there, to get users' faith and interest. It is stated in the International Data Corporation (IDC) market forecast

Contributions:
[1] Proposal Writing
[2] Literature Review and Research
[3] OSS Research and Selection
[4] Data Collection
[5] Analysis and Observation
[6] Documentation

that Android and iOS covered 92.9 percent of smartphone market shares of the year 2013 alone. (Angulo and Ferre, 2014). That number will be more this year. The app market offers developers and development teams different mobile devices and different platforms for them to create mobile programs and applications. To cover all of them, developers need to create different versions of the app using different frameworks to target each platform.

Cross-platform development frameworks help developing apps for multiple platforms. These frameworks save resources by reusing code that can be used to make the same feature mobile apps for different platforms all from a single code base. These are used because they can help reduce overall development time and cost because developers doesn't need to write codes for every platform, so there is no need to hire separate developer for separate platform. Also reusing code can largely decrease development time. And cross-platform mobile apps have more market reach. Usually, the same app which can be used on multiple platforms can have more user base. And hence more user loyalty. On the other hand, even though developing and deploying native applications is easy and efficient for user experience, it is not possible to reuse code for another platform. Native apps are those apps that were developed for a single operating system or platform or device. So, in case of making the same application, developers need to start from scratch for making the same feature mobile app.

Native app performance is the higher than cross-platform app performance and the eventual goal for cross-platform mobile app development is to get that kind of performance and gain reach on many platforms as possible. In our study, we investigate this issue. For our study, we analyse cross-platform frameworks for single-platform development. Each of the codebases used in our study is built for Android phones. We further expand on this and we propose conducting a similar study for the iOS platform as future work.

Developers often discuss the performance of apps developed using cross-platform frameworks. It has been found that expected performance greatly depends on the choice of suitable technical development framework. (Corbalán et al., 2019). The main motivation of this study is to analyze the efficiency of cross-platform technologies.

We used three technologies Ionic, Flutter and React Native for cross-platform mobile app development. We used 9 codebases to collect data on the metrics we defined. After this, we investigated, analysed and made thorough observations on our findings. And we found that there is no certain best framework. It greatly depends on the features used in this study. Some frameworks perform

equally or even better in some cases of performance analysis which we will further discuss in Section 5.

3 Related Works

Priorly, measuring performance and efficiency of cross-platform apps have been studied and researched by developers and researchers.

Cross-platform development uses a singular code base which is used by a number of platforms. Platforms here are the different operating systems like Android and iOS or Windows. And by cross-platform development frameworks we mean Ionic, React Native, Java, Kotlin, C++, Flutter and also Objective-C and Swift. Previous researches are seen to take both the software and hardware perspectives of evaluation regarding cross-platform applications.

(Xanthopoulos and Xinogalos, 2013) classified mobile app types and trends of these app types in the current market. They showed that generative apps showed more promise. Even though authors in this paper made a simple RSS reader app using JavaScript with no knowledge of target iOS and Android platform using Titanium (an open-source framework for creating native apps), the downside was the complete dependence of the application to the software development environment. So even though native app provide better performance and UI/UX, it is limited in reducing cost and development time and is only limited to the platform which it was created for. That's why we are going to be evaluating cross-platform technologies in our study and not focusing on native application frameworks.

Then we have seen studies that compare major cross-platform mobile development tools like PhoneGap/Cordova, Xamarin, Appcelerator Titanium, and Smartface App Studio in depth (Tunali, V., and S. Zafer, 2018). Because of their "write once, run anywhere" or similar mindset, all of these solutions assist app vendors in developing apps that can execute on several mobile platforms with less effort and cost in less time. It's not easy to select a mobile app development tool because there are so many options with so many varied features and capabilities. As a result, software vendors must be aware of their benefits and drawbacks, evaluate each one in light of their own unique development requirements and limits, and then make informed decisions. This paper helped us to analyse the framework in the best possible way. We have learned how the approach of analysing the frameworks should be from this study.

We have also seen studies made on performance overhead of native and cross-platform mobile development frameworks like in the (Biørn-Hansen et al., 2020) They gathered data manually on a couple of metrics like CPU usage, memory consumption, time to completion etc. And their results showed that native development frameworks have increased performance than compared to the cross-platform approach. The results also indicate that certain cross-platform the same or even better than native on some metrics. So doing a proper analysis of cross-platform frameworks in our study shows promise about the fact that native is not always better in all cases. Cross-platform development frameworks works just as well and if not better in certain cases.

4 Methodology

In this section, we present the research methodology of doing a proper case study and analysis of different cross-platform frameworks. We elaborate on the open-source software (OSS) selection, the codebases we used, framework selection, metrics selection and process through which we gathered the data.

4.1 Workflow

The workflow of the study we are doing on analysis of cross-platform development frameworks is given below:

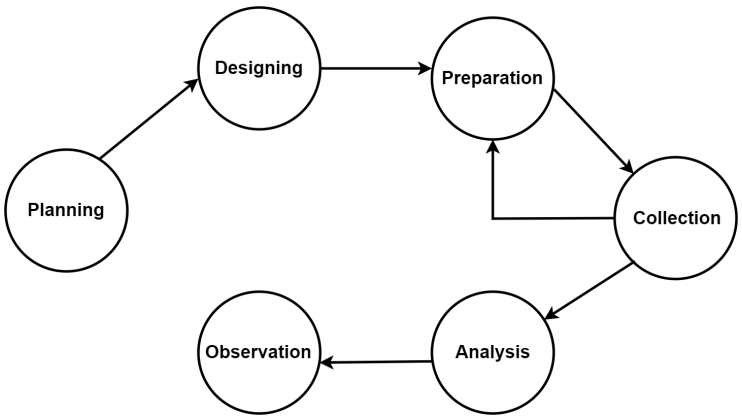

Figure 1: Workflow

- **Planning**

  For doing a study, we defined a problem for our empirical study. Then we need to identified the research questions. We precisely stated the objective of our study which is to evaluate the

performance of cross-platform technologies for determining which technologies are better than the other.

For that, we also studied other case studies and research papers. We also took notes of how the previous studies were done for performing our study. We also identified what the limitations were in other studies and how our study can improve the current cross-platform frameworks market.

- **Designing**

  Here we selected open-source software(OSS) for our study. We researched the internet to find open source code bases and APKs. We wanted to find standard source codes for each of the frameworks Ionic, Flutter and React Native. So we selected 9 repositories from GitHub and we studied them to determine if they are standard and fair for collecting data and choosing metrics. We chose 9, three sets for each framework because we wanted to collect huge amount of data and make thorough analysis of their performance. The OSS details have been discussed in Section 4.3.

- **Preparation**

  In this step, we choose the metrics for doing the study. We also determine if these metrics would be able to help evaluating the performance of cross-platform technologies. The metrics has been discussed in Section 4.4. We also research on software tools/ services for metrics measurement and data collection. And we install and prepare the tools and software on our respective machines. In this study, we used the Android Profiler tool for data gathering.

- **Collection**

  We do data gathering here. We setup the applications and collect readings on the decided metrics. All data extractions using the Android Studio Profiler tool were done manually. And we made documentation of the readings we took. We discussed more on this in Section 4.5.

- **Analysis**

  Based on the data we gathered, we investigated the data readings, and we analysed them. Based on that analysis we checked if we needed to make further calculations on those metrics. Then we analysed the overall results on each of the frameworks.

- **Observation**

  We make the conclusions based on the data analysis we did on the frameworks. Here we also give necessary graphical support like tables, graphs and charts to support those conclusions.

We also assess the limitations of our study and see where we could improve our study. We also give suggestions for future work on our study of performance on cross-platform technologies

4.2 Technical Frameworks

Artifact	Framework	Version	Programming Language	APK size
Ionic app	Ionic	v5.0	TypeScript	10.3 MB
React Native app	React Native	v0.64	JavaScript	9.7 MB
Flutter app	Flutter	v2.5	Dart	32.8 MB
Movie Bank app	Ionic	v5.0	TypeScript	19.2 MB
MobiDemo app	React Native	v0.64	JavaScript	17.5 MB
Movie DB app	Flutter	v2.4	Dart	10.113 MB
KnoWhere E-commerce app	Ionic	v5.0	TypeScript	16.8 MB
React Native E-commerce app	React Native	v0.64	JavaScript	43.675 MB
FLutter E-commerce app	Flutter	v2.5	Dart	11.8 MB

Table 1: List of technologies included in the study based on the OSS used

From previous studies, we can say that evaluating performance and studying in the area of cross-platform framework needs to be in the limelight. We wanted cover the major cross-platform frameworks for our study and we accordingly selected Ionic, Flutter and React Native. Table 4.2 lists three technologies based on the OSS code bases we used. These technologies vary in terms of APK size, programming languages, development approach and other characteristics.

- ***Ionic*** is a complete open-source SDK hybrid approach for mobile app development. Ionic apps are truly cross-platform. The development is easy and it is able to run as Android, iOS, Progressive Web App (PWA) all from a singular code base. For this reason, Ionic is very popular. This project has about 45,400 stars on GitHub

- ***Flutter*** is the framework for mobile-only cross-platform development. It supports Android and iOS platform. Nowadays, it has achieved great attention from developers as seen that it has 1,31,000 stars on GitHub

- ***React Native*** is an open source cross-platform mobile development framework written in JavaScript. It supports Android and iOS platform. And it is very popular (having 98,700 stars on GitHub)

We wanted to include a native approach in our study but we decided against an inclusion of this approach as there has been multiple studies done on the comparative analysis of native and cross-platform mobile development frameworks. And native is better than cross-platform applications in terms of performance. So we decided not to focus on native app. Instead we wanted to evaluate some

of the major frameworks of cross-platform apps. Even though native apps are better in performance, it can be costly to build. With a limited budget, cross-platform apps are the best choice. So we decided to move forward with our study of analysis of cross-platform mobile development frameworks.

The list of technologies referenced in Table 1 provides for descriptive discussion and evaluation of this paper. The APK size mentioned in Table 1 is a very important metric. It helps determine the impact of adopting the app to the market.

4.3 OSS Details

The code bases and artifacts we chose reflect the major frameworks that we are using for cross-platform development analysis. We wanted to have a fair and similar representation across all the frameworks by choosing the open-source softwares (OSSs)

We selected 9 code bases. We selected the OSSs in such a way that there are 3 types for each framework and there are 3 categories of app types which are a movie review app, e-commerce shop app, and benchmark feature app. Benchmark feature app made for Ionic, Flutter and React Native. A Movie App comprised of review and details of films and tv shows. And there are three movie apps for all three frameworks. We also chose three code repositories and APKs of E-commerce mobile applications for three of the frameworks. Table 1 shows the mobile applications we selected for each framework in our study analysis of cross-platform development frameworks

4.3.1 Feature Performance Benchmark Applications

The Benchmark Feature apps that we referenced were collected from (Biørn-Hansen et al., 2020) paper, where we got the APKs of each of the three frameworks Ionic, Flutter and React Native.

Andreas Biørn-Hansen one of the authors of (Biørn-Hansen et al., 2020) was the main contributor to the Github repository[1] where we collected the source code and the APKs from. These apps were made for the purpose of performance evaluation for the study the authors made in the paper (Biørn-Hansen et al., 2020). There are 3 commits made in this repository. The applications were pretty straightforward. And since the authors used it in their study, the applications were similar in UI and features and we thought it would be perfect for our study.

[1] `https://github.com/mobiletechlab/EMSE-D-19-00180-replication-package`

4.3.2 Movie Applications

We selected three applications each made with three frameworks Ionic, Flutter and React Native. All three of these mobile applications were made using the MovieDB API.

The Ionic Movie App is named Movie Bank, an app that the user can explore 28,000 movies. It has been developed with Ionic 4. The source code is open-source and we collected it from a GitHub. There are about 42 commits in this repository and the app has proper features and it is a standard app for Ionic. Thats why it would be a suitable OSS for our case study.

The Flutter Movie App is also made using the MovieDB API and has been collected from a GitHub. It has about 17 commits and the features are the same as the Ionic app. It also has an APK file which makes our study analysis easier.

The React Native movie application has been collected from GitHub. It has about 62 commits. It also has been made using the MovieDB API and all the features almost match as the Ionic and Flutter movie apps that we chose for our study.

So we think that these three apps have been suitable choices for our study of cross-platform development frameworks for this category.

4.3.3 E-commerce Applications

Similarly, we selected the e-commerce categories of three apps each for the three frameworks Ionic, Flutter and React Native.

The Flutter E-commerce app is a typical mobile e-commerce shop application for products. The source code was taken from GitHub. It has about 43 commits and also provides an APK file. After assessing it, we decided it was a standard e-commerce app with moderate features and that it was suitable for our study in analysing Flutter framework.

The Ionic E-commerce app is named KnoWhere by its contributors. We collected this app from GitHub and it has about 61 commits. And APK has also been provided. It has standard features of an e-commerce application.

We also selected a React Native E-commerce app from GitHub. It has about 10 commits and also a typical e-commerce application.

These three e-commerce apps are almost similar in case of features and are standard applications so it has been suitable for our study on cross-platform frameworks. All the application can be found in this GitHub repository.[2]

4.4 Metrics Selection

Metrics are significant indicators of software process efficiency and effectiveness. The analysis of established metrics aids in the identification of areas for improvement and the formulation of subsequent actions. So we have selected some metrics as per the requirement of our work. The selected metrics are discussed below.

4.4.1 Completion Time(CT)

The time when a job or process is completed. Completion time is used as a metric of the applications that we call Feature Benchmark Applications. We chose this feature because, by comparing CT of each platforms for the same task we can conclude which platform is doing the task faster and efficiently. We chose mostly used features for apps and observed the benchmarks for our conclusion. The features are discussed below.

1. **Accelerometer:** The accelerometer is a sensor that allows users to improve their experience by altering the app screen orientation on their smartphones. The primary goal of the mobile phone accelerometer is to adapt the device's orientation to the device's position, from horizontal to vertical and vice versa. It measures the position and orientation changes of the screens to create a comfortable viewing experience for the users. The accelerometer in a smartphone measures the device's linear acceleration. When the device is in its rest position, regardless of orientation, the figure represents the force of gravity acting on it while also measuring the acceleration on the X and Y axes, which will be zero.

2. **Contacts:** Contacts list is one of the most important things on the device routinely utilized by almost all users of smartphones. It's where contact information is saved including names, phone numbers.

3. **File system:** The ability to read files stored on the file system is tested. Bulk data access requires access to the file system and even a basic app requires access to the file system.

4. **Geolocation:** Geolocation makes usage of a mobile device's built-in GPS to show the exact location of the device and its user. This information is acquired through apps that have permission to access

[2] `https://github.com/lisa1704/smp-empirical-study`

the user's location data.

4.4.2 CPU Usage (CPU)

The CPU's performance capacity affects how fast all programs and operations run. One can check how intensively running programs are being processed by looking at CPU utilization. The percentage of a processor core's total working time that is actually used to process data is indicated by the respective value. CPU usage is used as a metric of these two applications discussed below.

1. **Feature Benchmark Applications:** Three frameworks (Flutter, Ionic, React Native) of Feature Benchmark Application uses these same features for measure.
   (a) Accelerometer
   (b) Contacts
   (c) File System
   (d) Geolocation

   We have already discussed in section 4.4.1 about these features.

2. **Movie Applications:** Three frameworks (Flutter, Ionic, React Native) of Movie Application uses these same features for measure.
   (a) Startup
   (b) Home screen
   (c) Searching

3. **E-Commerce Application:** Three frameworks (Flutter, Ionic, React Native) of E-Commerce Application uses these same features for measure.
   (a) Startup
   (b) Home screen
   (c) Searching

4.4.3 Memory Consumption (RAM)

The amount of memory used by a program during its execution is referred to as memory consumption. The more variables we have, the more RAM we will need. As a result, it's common knowledge that more complex applications require more RAM. To know what amount of memory the applications

actually use, we have run the applications using a profiler tool. The data collection methods will be discussed in 4.5.

1. **Feature Benchmark Applications:** Three frameworks (Flutter, Ionic, React Native) of Feature Benchmark Application uses these same features for measure.

   (a) Accelerometer

   (b) Contacts

   (c) File System

   (d) Geolocation

   We have already discussed in section 4.4.1 about these features.

2. **Movie Applications:** Three frameworks (Flutter, Ionic, React Native) of Movie Application uses these same features for measure.

   (a) Startup

   (b) Home screen

   (c) Searching

3. **E-Commerce Application:** Three frameworks (Flutter, Ionic, React Native) of E-Commerce Application uses these same features for measure.

   (a) Startup

   (b) Home screen

   (c) Searching

4.5 Data Gathering

For each data points we manually gathered the data.We used android devices to complete this tasks. The list of those devices are given in Table 2.

Model	CPU	Memory	OS
Pixel 3a XL	quad core	4Gib	Android 8.1
Pixel 5	octa core	8Gib	Android 11
Pixel 5a	octa core	6Gib	Android 11

Table 2: List of devices used for measuring performance

For this study, we gathered data on CPU usage, Completion-Time(CT) for precise tasks and memory consumption (RAM). For CPU usage and memory consumption we recorded using the Android Studio Profiler tool using the default sampling method for data collection and for completion time of the task we used precise event handler of the task. When testing a specific app, the Android Studio Profiler provides the performance values and we are then able to report on the maximum value i.e the peak value by making observations for that specific benchmark feature and the minimum value of and then average the readings of the process for mean value. In most studies for performance evaluation of any framework, the CPU usage and memory usage(RAM) is always used. Here the Completion-Time is measured in milliseconds(ms) and reports on the time between starting a benchmark task and getting the results.

Figure 2 is a pictorial representation of our data gathering process. For example, a user requesting accelerometer data and getting it, the time in between is CT. RAM consumption is measured in peak of memory consumption and also the minimum memory consumption during the whole process and then average the values for mean memory consumption. Its recorded in megabytes(MB) when executing a benchmark task. The percentage of available processing power consumed at peak when the feature is tested is the measure of CPU usage.

For more generalized and precise benchmarks, we ran each app 10 times and recorded benchmarks in Android-Studio Profiler and then took the average of those results.

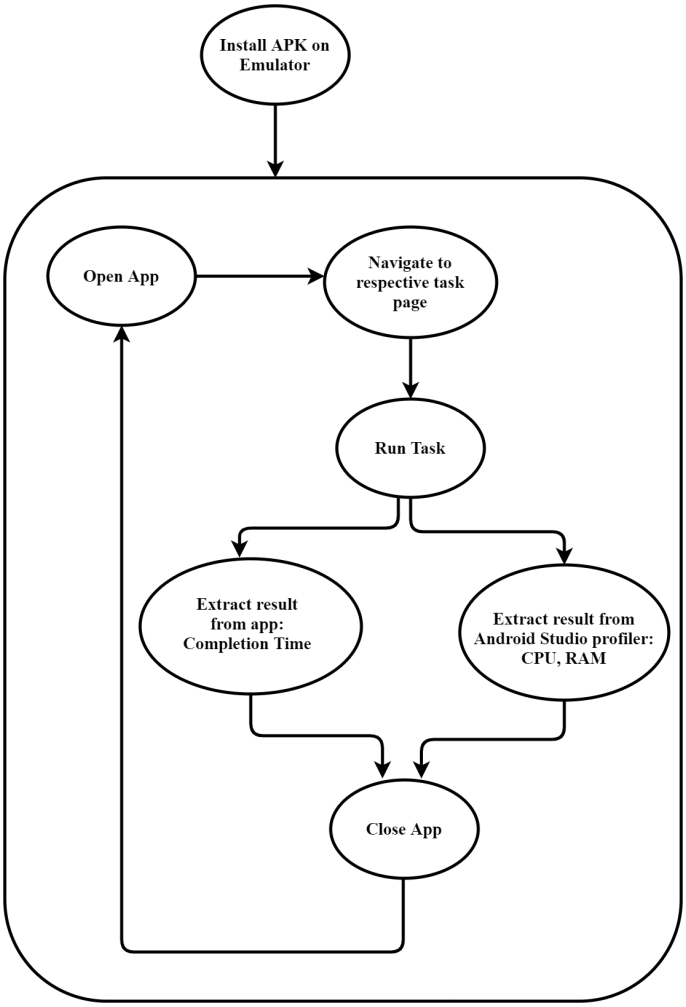

Figure 2: Data Gathering Process

In Android Platform, it is not possible to query the current value of the accelerometer through a platform-provided **API** call. Instead, when changes are observed, the sensor sends system events, which can subsequently be handled by appropriate event listeners. We experimented with the update Interval option supplied by the observable-based accelerometer plugin for the React Native implementation. During development, we discovered that the accelerometer benchmark reported results both above and below the set interval at a 50 ms interval, making it more comparable to the other implementations.

In terms of the contacts' information, we provide each contact object with a name and a mobile phone number and saved it. There is little reason to believe that storing more data(e.g. email addresses) will have a substantial influence on performance. calculated completion time for each cases and computed CPU and RAM usage.

In terms of File System Access, we use a benchmark PNG picture with a resolution of 528 by 528 pixels and a file size of 613 KB. We measure the time until the base64-encoded string is decoded in memory and ready for assignment to a view element (but not the actual rendering) in order to distinguish the device access from the UI representation.

For geolocation we measured the CT and CPU and RAM usage. This particular benchmark retrieves the device's longitude and latitude values.current location, as determined by the location retrieval mechanism specified by the precise platforms.

5 Result

In this section, we evaluate the data we collected on the metrics and decide the overall performance of each of the frameworks based on the metrics and features.

5.1 Feature Benchmark Application Data Tables

5.1.1 Completion-Time for Accelerometer

Inspecting the results in Table 3, we find that React Native and Ionic are statistically significantly different from the Flutter implementation in terms of completion time results for the accelerometer task. Based on the result, we find that mean completion time for Flutter was much higher than the rest and even maximum and minimum value for Flutter was much higher than the rest. The least time was taken by React Native app.

Framework	Mean	Max	Min
React Native	62.36	65.2	54.12
Ionic	70	76	90
Flutter	133.88	136.86	132.76

Table 3: Results of accelerometer performance per framework, Metric: Completion Time (ms)

5.1.2 Completion-Time for Contacts

Inspecting the results in Table 4, we find that in saving contacts Flutter takes the least amount of time. on the other hand React Native takes greater completion time than all the other platforms.

Framework	Mean	Max	Min
React Native	60.78	66.6	54.12
Ionic	43	56	32
Flutter	21.05	25.86	15.78

Table 4: Results of contacts performance per framework, Metric: Completion Time (ms)

5.1.3 Completion-time for File System Access

From the descriptive statistics in Table 5, we find that React native and Ionic showing almost similar results and lower mean and lower maximum completion time compared to Flutter. Results indicate that React native has the overall lowest mean value in completion time between all the platforms.

Framework	Mean	Max	Min
React Native	42.49	46.86	37.82
Ionic	46	58	29
Flutter	65.744	66.7	64.9

Table 5: Results of Files performance per framework, Metric: Completion Time (ms)

5.1.4 Completion-Time for Geolocation

From the statistics in table 6, we find that there's a statistically significant difference in mean completion time among Ionic and other platforms. We observed that the app in Ionic Platform required the most mean completion time among other platforms and Flutter takes the least time to complete the task.

Framework	Mean	Max	Min
React Native	57.44	60.84	53.76
Ionic	350.35	644.33	129.54
Flutter	17.76	19.34	17.05

Table 6: Results of geolocation per framework, Metric: Completion Time (ms)

5.1.5 CPU Usage for Accelerometer

Inspecting the results in Table 7, we find that benchmark results from Flutter is lower than the rest of the framework. Mean CPU time usage for react native is higher whereas for Flutter is lower.

During the whole process Flutter substantially consumed lower CPU. Even CPU consumption for flutter dropped down to as low as 0.1%.

Framework	Mean	Max	Min
React Native	23.6	68.2	2.1
Ionic	19	50	3
Flutter	18.5	32.3	0.9

Table 7: Results of Accelerometer per framework, Metric: CPU Usage (%)

5.1.6 CPU Usage for Contacts

Inspecting the results in Table 8, we find that benchmark results from Ionic is lower than the rest of the framework. Mean CPU time usage for react native is again higher than the rest whereas for Flutter is lower. During the whole process Flutter substantially consumed lower CPU.

Framework	Mean	Max	Min
React Native	23.5	64.3	5.5
Ionic	22.5	60.2	3.4
Flutter	19.5	57.3	0.1

Table 8: Results of Contacts per framework, Metric: CPU Usage (%)

5.1.7 CPU Usage for File System Access

From the statistics in table 9, we find that benchmark results from Ionic beats all the other frameworks in benchmark results. Mean CPU time usage for react native is again higher than the rest whereas for Flutter is lower.Here we can see that mean CPU usage of flutter and ionic both are almost similar but there's a statistical difference among Flutter and the rest of frameworks.

Framework	Mean	Max	Min
React Native	28.5	66.5	2
Ionic	27.6	55.2	1.3
Flutter	19.2	37	0.2

Table 9: Results per framework on file system performance, Metric: CPU Usage (%)

5.1.8 CPU Usage for Geolocation

From the statistics in table 10, we find that mean CPU consumption for Ionic is lower than the rest but not significantlty. All the readings from different platforms were almost same but for Flutter the minimum CPU consumption was 0%.

Framework	Mean	Max	Min
React Native	20.5	42.5	2
Ionic	18.54	46.7	8.9
Flutter	22.2	42.9	0

Table 10: Results of Geolocation per framework, Metric: CPU Usage (%)

5.1.9 Memory Consumption for Accelerometer

Inspecting the results in Table 11, we find that React Natve consumes least memory whereas Flutter consumes much higher memory than the rest. Mean memory consumption for for Ionic and Flutter are not significantly far apart from each other.

Framework	Mean	Max	Min
React Native	59.2	62.4	36.5
Ionic	95.5	100.4	60
Flutter	101.01	132.5	63

Table 11: Results of accelerometer per framework, Metric : Memory Consumption(MB)

5.1.10 Memory Consumption for Contacts

Inspecting the results in Table 12, we find that React Native consumes least memory whereas Flutter consumes much higher memory than the rest. Mean memory consumption for for Ionic and Flutter are not significantly far apart from each other. But for the Flutter app, the memory consumption sometime dropped down significantly.

Framework	Mean	Max	Min
React Native	59	68.54	31.5
Ionic	94.5	101.32	63.2
Flutter	102.6	110.59	28.7

Table 12: Results of contacts per framework, Metric: Memory Consumption(MB)

5.1.11 Memory Consumption for File System Access

Inspecting the results in Table 13, we find that again React Native consumes least memory whereas Flutter consumes much higher memory than the rest. Mean memory consumption for for Ionic and Flutter are not significantly far apart from each other.

Framework	Mean	Max	Min
React Native	58.3	65.87	35.44
Ionic	94.6	105.58	36.5
Flutter	100.5	110.23	55.5

Table 13: Results per framework on file system performance, Metric: Memory Consumption(MB)

5.1.12 Memory Consumption for Geolocation

From the descriptive results in Table 14, We discovered that React Native uses the least amount of memory, while Flutter uses a lot more. The average memory consumption of Ionic and Flutter are not much different.

Framework	Mean	Max	Min
React Native	55.4	7.42	23.5
Ionic	91.5	110.32	19.5
Flutter	103.4	112.56	36.5

Table 14: Results of Geolocation per framework, Metric: Memory Consumption(MB)

5.2 Movie Application Data Table

5.2.1 CPU Usage for Startup

From table 15 we can see that Flutter apps mean and max CPU usage is highest among other 2 frameworks. On the other hand, React Native has the least mean and max CPU usage but it holds highest min CPU usage. Here Ionic lies in between React Native and Flutter.

Framework	Mean	Max	Min
React Native	78	86	20
Ionic	67.2	72.3	9.5
Flutter	69.1	81.5	10.9

Table 15: Results of Movie App's Startup functionality per framework, Metric: CPU Usage (%)

5.2.2 Memory Consumption for Startup

From table 16 we can see that Flutter apps mean and max RAM performance is highest among other 2 frameworks even though it has the least min average. On the other hand React Native has the least mean and max RAM performance. Here Ionic lies in between React Native and Flutter.

Framework	Mean	Max	Min
React Native	172.47	187.27	137.4
Ionic	181.03	197.31	113.66
Flutter	187.46	216.37	92.18

Table 16: Results of Movie App's Startup functionality per framework, Metric: Memory Consumption (MB)

5.2.3 CPU Usage for Home Screen

From table 17 we can see that Flutter apps mean and max CPU performance is highest among other 2 frameworks even though it has the least min average. On the other hand React Native has the least mean and max CPU performance.

Framework	Mean	Max	Min
React Native	81.3	89	32
Ionic	77.6	83.7	6.5
Flutter	71	76.3	9.1

Table 17: Results of Movie App's home screen functionality per framework, Metric: CPU Usage (%)

5.2.4 Memory Consumption for Home Screen

From table 18 we can see that Flutter apps mean, max and min RAM performance is highest among other 2 frameworks. On the other hand Ionic has the least mean and max RAM performance.

Framework	Mean	Max	Min
React Native	204.7	216	199.75
Ionic	149.1	160.1	125.03
Flutter	379.4	393.63	212.99

Table 18: Results of Movie App's home screen functionality per framework, Metric: Memory Consumption (MB)

5.2.5 CPU Usage for Search Functionality

In table 19, the Flutter application's min CPU usage is the lowest but it mean and max value is the highest. React Natives min CPU performance is the highest here though it's mean value is a bit lower than the Ionic application.

Framework	Mean	Max	Min
React Native	73	82	36
Ionic	62.7	59.6	9.8
Flutter	67.4	78.5	12.9

Table 19: Results of Movie App's search functionality per framework, Metric: CPU usage(%)

5.2.6 Memory Consumption for Search Functionality

Here in 20 the mobile apps RAM performance result indicates that Flutter app is the most resource hungry followed by the React Native application. The Ionic application is consumes the least amount of resource while searching.

Framework	Mean	Max	Min
React Native	204.1	211.5	198.55
Ionic	130.8	132.95	123.01
Flutter	387	419.55	307.12

Table 20: Results of Movie App's search functionality per framework, Metric: Memory Consumption (MB)

5.3 E-Commerce Application Data Tables

5.3.1 CPU Usage for Startup

From table 21 we can see that Flutter apps mean, max and min CPU usage is highest among 3 frameworks. On the other hand, Ionic has the least mean, max and min CPU usage in the case of our selected E-commerce app. Here React Native lies in between Flutter and Ionic.

Framework	Mean	Max	Min
React Native	57.4	70.6	23.8
Ionic	51.8	67.9	20
Flutter	63.1	74.2	43.5

Table 21: Results of E-Commerce App's Startup functionality per framework, Metric: CPU usage (%)

5.3.2 Memory Consumption for Startup

From table 22 we can see that Flutter apps score lowest mean, max and min memory space among 3 frameworks. On the contrary, in our E-commerce application React Native takes the maximum memory space.

Framework	Mean	Max	Min
React Native	130.7	145.7	122.3
Ionic	160.2	183.1	80.89
Flutter	157.1	164.39	113.3

Table 22: Results of E-Commerce App's Startup functionality per framework, Metric: Memory Consumption (MB)

5.3.3 CPU usage for Home screen

Here in table 23, React Native has the highest mean and max value average while Ionic has the lowest mean and max value.

Framework	Mean	Max	Min
React Native	63.4	74.2	32.7
Ionic	39.7	50.6	18.5
Flutter	47.4	50.8	42.57

Table 23: Results of E-Commerce App's home screen functionality per framework, Metric: CPU usage (%)

5.3.4 Memory Consumption for Home screen

Here in table 24, React Native has the highest mean and max value average while Ionic has the lowest mean and max value.

Framework	Mean	Max	Min
React Native	132.3	145.6	125.8
Ionic	143.5	151.6	90.04
Flutter	151.8	160.54	148.76

Table 24: Results of E-Commerce App's home screen functionality per framework, Metric: Memory Consumption (MB)

5.3.5 CPU usage for Search

In table 25 , for CPU performance in search functionality, React native takes the highest amount of CPU considering it's mean and max CPU performance. After that Flutter comes. Here the Flutter application has the highest min CPU performance. Ionic falls in between React Native and Flutter.

Framework	Mean	Max	Min
React Native	59.9	76.4	23.2
Ionic	42.5	50.8	19.1
Flutter	47.5	52.56	32.9

Table 25: Results of E-Commerce App's search functionality per framework, Metric: CPU usage (%)

5.3.6 Memory Consumption for Search

In table 26, considering mean, max and min RAM performance result in search functionality, React Native uses the most amount of RAM than the other 2 frameworks. Flutter comes at next and Ionic uses the least amount of RAM in this experiment.

Framework	Mean	Max	Min
React Native	136.9	143.75	132.7
Ionic	149.7	155.43	96.83
Flutter	158.4	168.04	151.84

Table 26: Results of E-Commerce App's search functionality per framework, Metric: Memory Consumption (MB)

6 Discussion

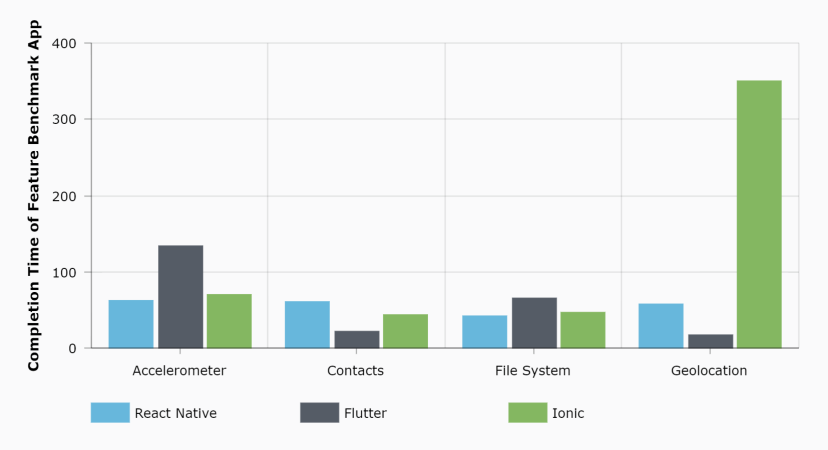

Figure 3: Comparison of Completion Time Among 3 Frameworks in Feature Benchmark App

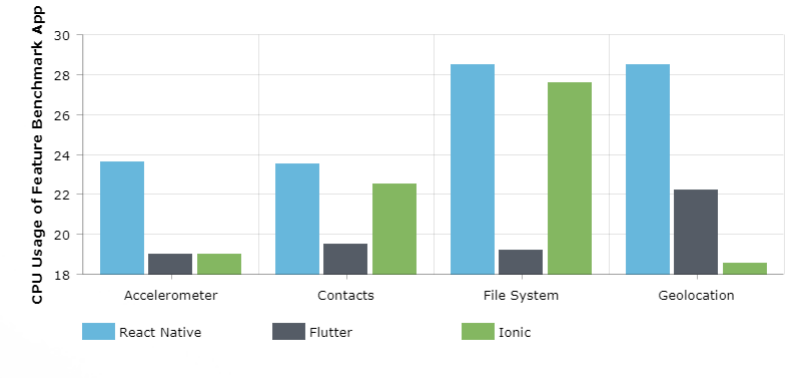

Figure 4: Comparison of CPU Usage among 3 framework in Feature Benchmark app

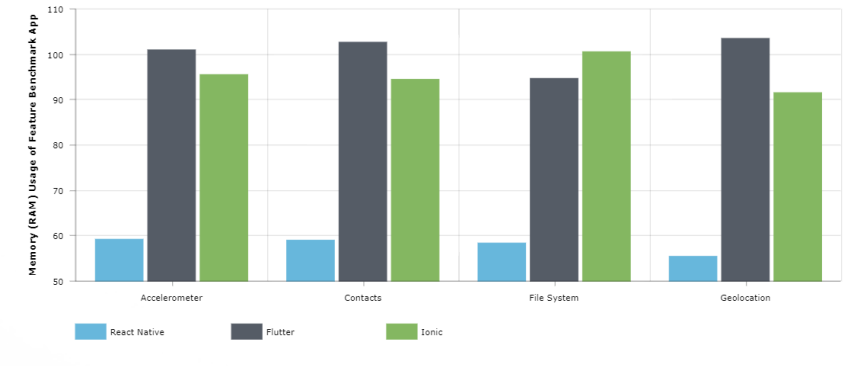

Figure 5: Comparison of Memory (RAM) Usage Among 3 Frameworks in Feature Benchmark App

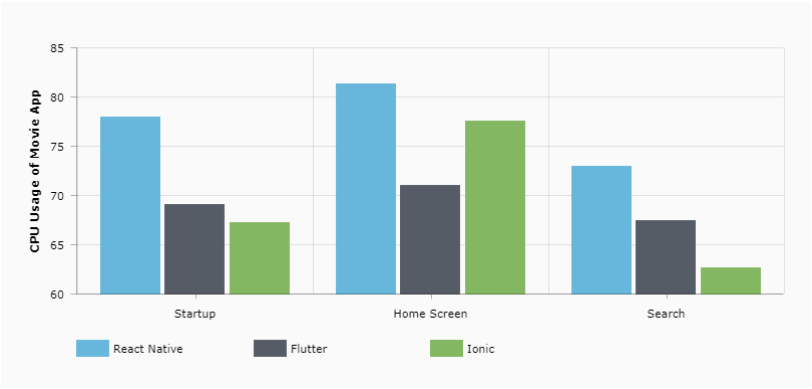

Figure 6: Comparison of CPU Usage Among 3 Frameworks in Movie App

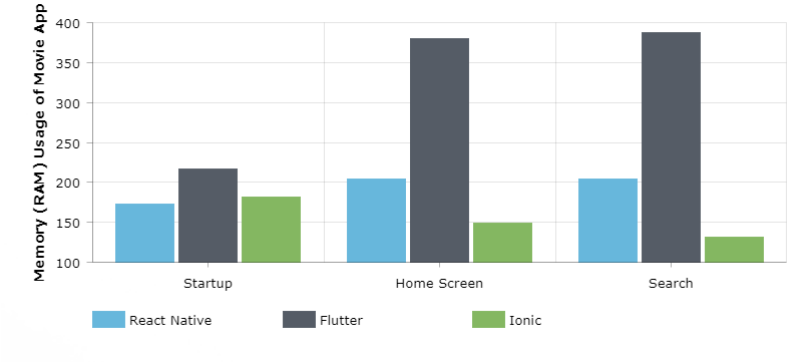

Figure 7: Comparison of Memory consumption among 3 framework in Movie app

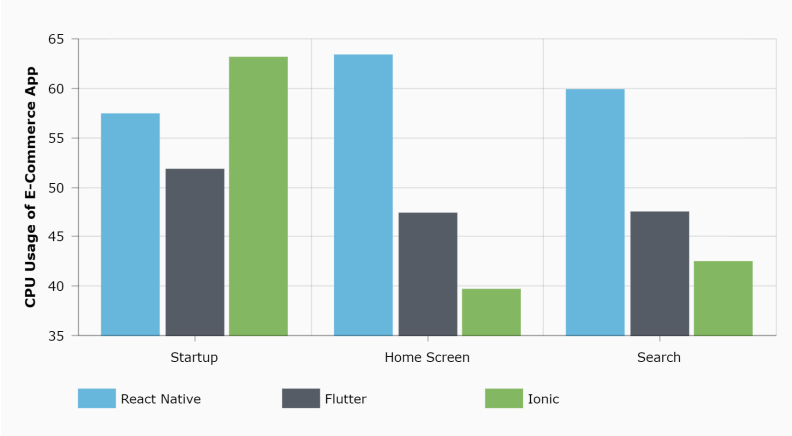

Figure 8: Comparison of CPU Usage Among 3 Framework in E-Commerce App

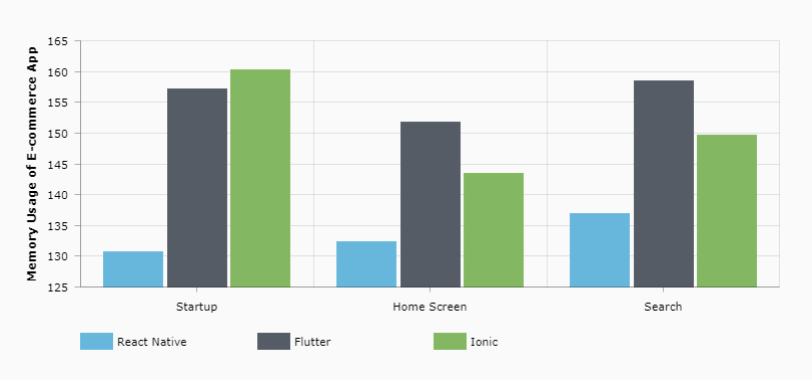

Figure 9: Comparison of Memory (RAM) Usage Among 3 Framework in E-Commerce App

As we can see from the bar charts of feature benchmarks apps, for the Accelerometer feature React Native app gives the lowest completion time(CT) but CPU usage is much more higher than all other platforms. For saving Contacts feature, React Native gives higher Completion Time than all other platforms and Flutter gives the lowest CT. Across all the features, React Native consumes most CPU but consumes substantially less memory than all other platforms. When it comes to File System Access Feature, we see that Ionic and React Native gives almost same CT and Flutter gives the highest CT among the other two. Although Flutter gives slightly higher CT among the other but consumes much less CPU and memory among React Native and Ionic. When it comes to Geolocation feature, we see that Ionic struggles to manage each location and gives abnormally higher CT than React Native and Flutter. We can say that when it comes to a Geolocation for an app, Ionic would not be the best choice to build. Although during the testing of this feature Ionic consumed less CPU and memory but gave abnormally higher CT. Flutter did the best among the others in Geolocation features and consumed less CPU than React Native. So when it comes to CPU consumption we observed that in these features CPU consumption of React Native was highest, and memory usage was lowest. Whereas, memory consumption of Ionic was higher than all the other platforms.

For the movie application, considering CPU consumption we can see that React Native consumes the most amount of CPU than the other two frameworks in all the functionalities. Flutter comes in next in terms of CPU consumption in startup and search functionality but this behaviour differs in homes screen functionality. Finally the Ionic application is the least CPU consumer in startup and search functionality but it's value is higher in home screen. Is we look at memory consumption, Flutter consumes way more memory than the other two frameworks. Even though React Native consumes less memory than Ionic in startup, it consumes more in the other two functionalities. The Ionic application has the lowest memory usage here in home screen and search functionality.

For the e-commerce application, React Native is consuming the highest amount of CPU than the other 2 frameworks though it consumes the lowest amount of memory than the others. Ionic consumes the highest amount of CPU during startup, but it consumes the lowest amount of CPU in home screen and during searching e-commerce items. In case of memory consumption, Ionic lies in between React Native and Flutter. While analyzing features of Flutter e-commerce app, we saw that it uses the highest memory resource in home screen and during searching.

From the above discussion, we can say that when it comes to CPU consumption, in most of the cases React native consumes CPU significantly higher than all other platforms, on the other hand when it comes to memory consumption React Native consumes significantly less memory than all the other platforms. For memory consumption we observed that feature wise Flutter consumes ram significantly higher than all other platforms.

7 Limitations

We discuss the threats to validity (Fenton and Bieman, 2019) as follows:

Internal Validity:

Since our study explores which cross-platform frameworks have the best performance, there isn't any cause-effect relationship. Our study purposes are the performance evaluation of Ionic, Flutter and React Native and which framework performs best based on the metrics. So, there is low internal validity in our study as we don't explore any causal link in our study.

Construct Validity:

We found meaningful measures for our study of measuring performance of the cross-platform frameworks. Measuring CPU Usage (CPU), Memory Consumption(RAM) and Completion Time(TC) gave us enough insight into the performance of each framework. During OSS selection, we struggled to find code bases that we could consider as standard and fair open source repositories for our study. And eventually, we found 9 such OSSs for our study having similar features. So, our study has high construct validity.

Conclusion and External Validity:

We analysed 9 code repositories (OSS). Of them, some of the repositories doesn't have the common developer. So due to this, some apps are a bit different from the other. This can be a threat of validity.

8 Conclusion

For this study, we did performance analysis of cross-platform mobile development frameworks in Android operating system. We gathered data on three metrics: CPU usage, memory consumption and completion time(CT). These metrics were collected manually through a thorough and time consuming data gathering process. When it comes to performance, it greatly depends which feature is the key part of the application. And it varies for each of these frameworks. Our results indicate that certain cross-platform framework performs equally or even better than the other frameworks but no framework performs the best in all features in the study. Our findings are significant to the studies doing on cross-platform technologies. And since cross-platform apps are very popular our study can contribute to the improvement of these technologies.

For future work, we will analyse more applications with more frameworks. Since, we did our cross-platform technology study on Android operating system, in the future we will also include iOS, Windows and other operating systems. For this study, we completed the research from the users' point of view. We plan to explore the insights from developers' perspective on cross-platform frameworks in the future.

References

Angulo, Esteban and Xavier Ferre. 2014. A case study on cross-platform development frameworks for mobile applications and UX. In *Proceedings of the XV International Conference on Human Computer Interaction*. pp. 1–8.

Biørn-Hansen, Andreas, Christoph Rieger, Tor-Morten Grønli, Tim A Majchrzak and Gheorghita Ghinea. 2020. "An empirical investigation of performance overhead in cross-platform mobile development frameworks." *Empirical Software Engineering* 25:2997–3040.

Corbalán, Leonardo, Pablo Thomas, Lisandro Delía, Germán Cáseres, Juan Fernández Sosa, Fernando Tesone and Patricia Pesado. 2019. A study of non-functional requirements in apps for mobile devices. In *Conference on Cloud Computing and Big Data*. Springer pp. 125–136.

Fenton, Norman and James Bieman. 2019. *Software metrics: a rigorous and practical approach*. CRC press.

Xanthopoulos, Spyros and Stelios Xinogalos. 2013. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*. pp. 213–220.