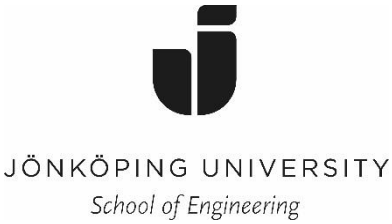

React Native and native application development

A comparative study based on features & functionality when measuring performance in hybrid and native applications

PAPER WITHIN *Computer Science*
AUTHOR: *Oskar Svensson & Marcus Presa Käld*
TUTOR: *Peter Larson-Green*
JÖNKÖPING April 2021

This exam work has been carried out at the School of Engineering in Jönköping in the subject area Computer Science. The work is a part of the three-year university diploma programme, of the Bachelor of Computer Science. The authors take full responsibility for opinions, conclusions and findings presented.

Examiner: Johannes Schmidt

Supervisor: Peter Larson-Green

Scope: 15 credits (first cycle)

Date: 27/4 – 2021

Postadress:	Besöksadress:	Telefon:
Box 1026	Gjuterigatan 5	036-10 10 00 (vx)
551 11 Jönköping		

Summary

This essay has been done at Tekniska Högskolan in Jönköping.

Smartphone apps today have a wide array of different usages & features and several different tools can be used to develop these smartphone apps. These tools can be broken down into three different categories, depending on what type of app they create: Native, hybrid, or web apps. These types of apps come with their advantages and disadvantages when it comes to development, performance, and costs to name a few.

The purpose of this paper seeks to answer performance issues around gradual app development in the native development language Kotlin, in comparison to the hybrid development framework React Native, with a focus on common functionalities. The focus on functionality adds the perspective of not only performance but also how a native and hybrid app may respond to the implementations, to give a wider glance at how native and hybrid compare. This may give a better understanding of how the development will turn out for both hybrid and native, in real-life cases. The chosen components for performance in this study are CPU, RAM, and battery.

The method to carry out this research involves the implementation of two testing apps for smartphones, one for Kotlin and one for React Native who function the same for the corresponding platform. The testing apps are a construct of various functionality that will be gradually measured in experiments. The experiments for the apps have been created to be a mixture of user usage and assurance of representative data from the smartphone's hardware components when the testing app is running.

The experiments conducted in this essay show that React Native has an overall worse performance than Kotlin when it comes to CPU processing and that React Native is more prone to having a negative response in performance when features or functionality are implemented. Memory usage did not show the same clear difference. A functionality that performed somewhat worse than the others involved for React Native compared to Kotlin was GPS, as further investigation of the collected data showed.

Keywords

App development, Functionality, Kotlin, React Native, Performance, Android development, Hybrid, Native.

Abbreviations

JVM – Java Virtual Machine

API – Application Programming Interface

ADB – Android Debug Bridge

GPS – Global Positioning System

App – Application (smartphone)

CPU – Central Processing Unit

RAM – Random Access Memory

GPU – Graphics Processing Unit

UI – User Interface

SDK – Software Development Kit

Contents

1 Introduction

These days, modern smartphone apps can be divided into three major categories: hybrid, native, and web apps, as explained in [1] [2]. When starting the development of an app for a smartphone, one must choose how to implement it and which of these three ways suits best.

Native apps are not web-based like hybrid or web apps are. Native apps are coded and executed in the machine language of the hardware platform they are intended for, which also means that one app can only be developed to one platform at a time. Native apps generally have better performance because of this. Support for device access, such as the smartphone's camera and GPS are also most accessible and extensive in native apps.

Hybrid apps are essentially web apps that have been put in a native shell. These apps behave almost like native apps and have the advantage of developing to multiple platforms such as iOS or Android with one codebase. This is in React Native's case enabled by native code being rendered but implemented with JavaScript. Hybrid apps also have good device access with packages or libraries that enables support for all the smartphone's built-in features such as camera or GPS, along with a UI that is like a native app.

Lastly, web apps are apps that run as a website through a browser with the preference of being viewed on a smartphone. This kind of app traditionally has less device access, has a harder time displaying a native UI appearance, and generally has poorer performance. Web apps are however becoming a more prominent choice as of late for app development, for instance, progressive web apps (PWA) which has more in common with native or hybrid solutions, as written in [3]. The upside to this kind of app is that the app is easier to implement and cheaper to develop and maintain. As seen in figure 1 below where it's illustrated how the three different ways of methods are rendered in the smartphone.

Figure 1. Native, web and hybrid app illustration. [4]

Looking at table 1 shown below which is inspired by [5], a summary is shown of what the pros and cons are with choosing one of the three development ways. Green indicates a pro whilst red indicates a con with orange as a middle ground.

	Native	**Hybrid**	**Web**
Development speed	Slow	Moderate	Fast
Development cost	High	Moderate	Low
Maintenance cost	High	Moderate	Low
Device access	Platform SDK enables access to all device features	Device features can be accessed, but depending on the tool	Most of features are accessible
Code reusability	Code for one platform only works for that platform	Most hybrid tools will enable portability of a single codebase to the major mobile platforms	Browser compatibility are the only concerns
App performance	Great	Good	Moderate
Graphical performance	Great	Good	Moderate

Table 1. Comparison between native, hybrid and web apps.

The process of choosing between one of these three ways can be hard and tiresome since there are so many variables to attune for. However, one should start from the beginning of smartphone app development. The first issue to consider when developing a smartphone app is the need and what kind of problem the app is trying to solve, as explained in [6]. When a need and a problem have been defined the appropriate features should be considered to fill the need and to solve the problem. The features of a smartphone app have a major impact on how the app will perform as it will utilize the smartphone in different ways. Features and functionality are therefore a good standpoint for the choice between native, hybrid, or web. A map app using GPS features behaves much differently from a social media app and conditions and implementations will be different between native, hybrid, and web.

Functionality is an aspect of what software or hardware features can do for the user, such as camera access, Bluetooth communication, or retrieving phonebook contacts from the user's smartphone. This thesis aims to investigate how some of these different features and functionalities affect the performance of an app on Android smartphones when developing for native and hybrid, excluding web. It also aims to investigate app performance that motivates the use of a feature or functionality over the other for native or hybrid, based on the results from experiments. The focus on performance is during development and with the addition of functionalities in a test app, rather than merely the measurements with all functionality combined. This paper goes in-depth on functionality, implementation, and investigation on how these affect an app's performance on smartphones.

The chosen factor to investigate for this thesis is functionalities' impact on performance. Performance refers to the speed and power of the smartphone's hardware components. Hardware components enable the smartphone's capabilities, and its properties decide how efficiently it can execute tasks. The performance transcends all app development and is a major influence on how an app is perceived

by users. The chosen parameters to investigate and monitor for the hardware components in this study are:

- CPU (Central Processing Unit)
- RAM (Random Access Memory)
- Battery

The motivation behind the choice for these three components is that they (except for battery) are central to a device's overall speed. The battery is instead the counterweight and accounts for the speed's impact on the smartphone's battery life. The impact on the smartphone's performance is an important factor when choosing between these different implementations and fits best for the subject of this thesis.

Performances on smartphones can sway between different means of development and it is therefore interesting to look at how much and why different means affect performance. Smartphone apps also have a lot of different usages and purposes which leads to a lot of different implemented functionality. To get more concise insights and the ability to compare more and similar functionality in a smartphone app will only native and hybrid apps be compared in this study, as they have the most in common. This will also limit the scope of the project which enables more focus for the time-constrains of the project.

1.1 Background

The choice between hybrid and native can however become tricky depending on what functionality the app is supposed to have. If roadblocks from the choice of tool for development or the functionalities needed would occur it is a lot of work to either try to work around it or switch tool entirely. The functionalities chosen for further investigation in this study resides in common functionalities that most apps have and that cover different categories i.e., device access or internet communication.

1.1.1 Previous studies & analysis of native and hybrid

In previous studies where React Native and native solutions have been evaluated and compared, the outcome has been that React Native (and similar hybrid frameworks) is somewhat similar both in performance and appearance in these three studies [7] [8] [9].

- [7] Set out to try and replicate an existing native Android app in React Native to measure performances as well as comparing the overall UI differences. The study showed that the UI difference between native and hybrid was not protruding but native performed a bit better when evaluating performance, which included GPU frequency, CPU load, memory usage, and power consumption. The tests were however manual, and it is noted that automatic testing might have led to more accurate results.

- [8] Has a focus on development, system performance, and user experience, with the differences between hybrid and native evaluated through the

development of an iOS, Android, and React Native app. As for system performance did the paper conclude a poorer result for hybrid compared to native. For user experience was it concluded that when compared separately did not React Native show a difference to native but did when compared side by side. The paper also evaluated codebase sizes and time spent on implementations.

- [9] Recreates an existing native app in React Native through a port and has a focus on background processing and Bluetooth but also evaluates functionality such as notifications and graphs. The evaluation is based on performance, functionality, and codebase sizes. The study includes both iOS and Android, where a worse result for hybrid is presented compared to iOS than Android when it comes to CPU utilization including Bluetooth functionality. Memory consumption shows similar results. Native code implementation was however used for React Native when implementing the Bluetooth functionality.

The studies [7] and [8] are from 2016, just a year after React Native was released, which would explain the worse UI elements and system performance presented in the study. Both React Native and Kotlin get updated regularly over the years which could have a big impact on performance and measured parameters, as shown in their release notes [10] [11].

1.1.2 Collaboration

This study is in collaboration with Cybercom Jönköping, who are interested in a study of this kind. Cybercom Jönköping takes part in the development of a wide array of smartphone apps and often gets confronted with what technologies and tools that is best suited for projects. From this study, the students and Cybercom aim to get a good understanding of React Native and natives' performance and the functionalities impact during development and more insight to what way to choose when confronted with the decision to choose hybrid or native development.

1.2 Problem description

As stated previously, the choice between native and hybrid includes many variables. From researching previously published papers [7] [8] [9] has it been noted that there is a gap in the influences of individual increases of implemented functionality, or in other words; only investigation of the finished product of the app has been done.

If performance issues would occur during development, the developers would want to know where and why, and if those issues could lead to bigger problems down the road. If a certain implementation or functionality is causing worse performance one would like to know this from the start and plan accordingly to decide which technology would work best to implement the app. This could be the case for React Native because of its non-native implementations of functionality. This problem also touches upon what features or functionalities have the biggest difference performance-wise between native and hybrid development. Examples of features or functionality are device access such as camera and GPS or internet services such as API requests and database storing.

Drawing from the conclusions of previous works, hybrid development may be the easiest and most comfortable way to develop smartphone apps in some cases. But in a different context would native still be a clear choice, as native has been shown to have a slightly better performance. The overall question has many variables to attune to (as explained in the introduction) but an important one is the app's functionalities impact on the performance of the smartphone. And simply not overall, but during development. If problems with an app would occur during development, it could have big implications both business-wise and technical-wise, as a lot of progress could be lost or be in vain.

1.3 Purpose and research questions

From the problem description, it is stated that the choice of implementation for a smartphone app is not always clear and that this study aims to aid that problem. It is also stated that a gap has been found when it comes to the influence of individual increases in functionality between native and hybrid. This study aims to fill that gap by measuring performance with the increase of functionality in a hybrid and native app, focusing on the functionality's impact. The problem to investigate revolves around initial app implementations that progress to later in the development cycle when the app has more implemented features, and the impact the features have on the smartphone's performance. The purpose of this study is then:

To investigate React Native's contra Kotlin's impact on smartphone performance through multiple app functionalities in a test app, thus exploring functionality suitability for native and hybrid.

The two main research questions of this thesis are:

1. What is the difference between React Native and Kotlin's impact on smartphone performance when implementing functionalities in an app?

2. Is there a breaking point of app implementation where React Native's impact on app performance is considerably worse compared to Kotlin? And if so, with what functionality, and why?

RQ1's purpose is to establish a foundation for further research and study. The question is answered through experiments and observing the results. RQ2's answer extends upon RQ1 and will need additional analysis to how, when and, why the results are happening.

1.4 Scope and delimitations

This thesis will only investigate the performance for Android and not iOS, the reason for this is limited time and the student's assets. The current versions of React Native (0.63) and (1.4.10) for Kotlin are the versions that will be used. The physical Android devices available to us run Android 7.0/API level 24. Our experiments will be conducted on two Android devices: Huawei Honor 8 and Samsung Galaxy S7

both from 2016. Therefore, can this study not answer for how older or newer devices would perform during testing of this paper.

Performance is in this study limited to the percentage load on CPU cores, percentage load on RAM, and monitoring of battery statistics.

The breaking point, which is the main foundation for research question two (RQ2) is defined and limited by the following set of restraints:

- It is the largest difference between React Native and Kotlin between two development stages.
- Several instances of the greatest difference can be found between the same two development stages in different measurements.
- It is in majority.

The breaking point is also explained in the theoretical framework.

1.5 Outline

This section intends to explain how the report was structured and will briefly explain what will be covered in the chapters.

i. Theoretical Background

This chapter covers relevant papers and literature and briefly explains different technical aspects that will be used throughout the paper and their background.

ii. Method and Implementation

Explains how the research questions will be answered and through what tools. This section creates a connection between the research questions and the method and explains in-depth how the method to conduct experiments will be implemented.

iii. Findings and Analysis

When the necessary data has been collected it will be analyzed and presented graphically for the reader to get a better understanding of the finalized results.

iv. Discussion and Conclusion

The author of this paper will discuss the findings and arrive at their conclusions about the study. It will also include the thought about the choice of methods and whether it adequately answered the research questions.

2 Theoretical background

This section will cover the current survey of relevant literature, background research, and explanation of theoretical concepts. Most of the content of the headings will be of different smartphone functionality.

2.1 Previous works & discernment of area

From researching previous papers [7] [8] [9] it is known that when it comes to creating apps with React Native, through porting or replicating there is not a big difference in performance between native and hybrid. However, extending upon the problem description, these papers did not investigate the impact of individual implementations and functionalities in the app and the effects it has on the smartphone's performance, as this thesis aims to do. It arises the question of how the performance can sway during development for the two different ways of creating an app when more and more features are added to the system.

A similar work from 2018 [9] investigates the functionality in React Native and native. However, this paper has a focus on four different functionalities: notifications, graphs, Bluetooth, and background processes. The 'future work' section of this research paper mentions that additional research within this area is interesting, but with different or more functionalities. In our thesis, functionalities have been chosen to support the measurement of several app implementations and to bring something new to the table.

When evaluating the performances from past works the parameters that were looked at were CPU usage, battery drainage, memory usage and, in some cases GPU. Past works have focused mostly on the performance overall together with UI differences, while this study will investigate changes in performance during development as more implementations and functionalities are added to apps. A specific testing app will also be developed instead of trying to replicate or simply port an app from native to hybrid. This study does also not have a focus on graphical performance, as the test app will consist of a minimal number of pages (focusing on functionality) and therefore excluding GPU performance as a parameter to measure.

2.2 React Native & Kotlin

Kotlin [12] and React Native [13] are two tools for smartphone app development when it comes to Android. The difference between these two is that Kotlin is a native programming language for Android whilst React Native is a framework where the implementation code for the app is written in JavaScript and then rendered with native code. React Native was released in 2015 by Facebook and has gained a lot of attention since with their "Learn once, write anywhere" approach and is today a major choice for many to develop apps as mentioned in [14].

2.2.1 React Native

React Native is a framework developed by Facebook which is one of the most popular frameworks together with Flutter as shown in [15], to develop hybrid apps

with. When surveying members of Stack Overflow about which framework was most wanted, React Native ranked highest at 13% and the contender flutter ranked at 6.7% as presented in [5]. Because of the support React Native has over other frameworks, React Native was chosen for the investigations in this paper.

2.2.2 Kotlin

Kotlin is a programming language on the rise to develop native Android apps with. Native Android apps can be developed in two different languages, whereas Kotlin is one and the other is Java. However, Java will not be covered in this thesis. Java has been covered in many previous works while Kotlin has gotten less exposure, which emphasizes the choice of native implementation in this study. Kotlin was first released back in 2011 but had an official release in 2016. Kotlin has begun to become a more common choice to develop Android apps in and is supported on Android developer's official website, but the competitor language Java is still the most common since it has been around for longer (1995).

2.3 Functionalities

A brief background on various app functionalities in a smartphone.

2.3.1 API

API stands for app Programming Interface and enables data exchange between two different software systems, functioning as a bridge for an interaction. These interactions between two software systems can range between login into a certain website, fetching weather from a user's area to uploading files to a database. When databases are referred to in this paper it will mostly refer to an API database

2.3.2 Database

A database is a data structure that stores organized information. The most common database is the relational database which contains tables that often includes multiple fields of information. Databases are used almost anywhere in online systems and apps. In smartphone apps, a database can for example be used to store high scores from a game or contain a person's images and posts from a social media profile.

2.3.3 Bluetooth

Bluetooth is a wireless technology that enables short-range communications between Bluetooth-compatible devices such as smartphones, headsets, or laptops. The technology is based on radio waves with a 2.4 GHz frequency which because of the high frequency limits the range to about 9 meters.

2.3.4 Sound playback

Sound playback refers to utilizing the smartphones speakers in some way to play a sound, in this case from a local mp3-file. This is part of a smartphones background process capabilities since the smartphone can play sounds from an app when it is in the background i.e., not being open on the screen. Audio playback will be utilized as a background process in this paper.

2.3.5 GPS

GPS stands for Global Positioning System and is a worldwide used satellite system used to determine the ground position of an object. GPS receivers are included in a wide array of commercial products such as smartphones and automobiles.

2.3.6 Camera Integrations

Camera integrations in this paper refer to smartphone apps that utilize the phone's camera capabilities in some way. Which mostly is taking and saving photos and videos. It also involves screening a preview of the camera like a traditional photo app.

2.4 Android Debug Bridge

Android Debug Bridge [16] is a tool for developers to analyse smartphone apps. This tool enables communication with a mobile device from a computer whilst the app is running/debugging, which allows the program to monitor the smartphone's different performance statistics.

Figure 2 below explains how ADB works, by connecting your smartphone to your computer and on the computer, ADB runs through a shell that passes through the commands to the phone.

Figure 2. Illustration of how ADB connects to the phone

2.5 Npm packages

Npm is a package manager for JavaScript and enables programmers to install JavaScript modules/packages for their projects. These packages can have all kinds of different purposes but are mostly used to ease up and enable feature implementation in the apps that is being developed. In the case of this study will npm packages be used in the project for React Native to aid the implementation of the different functionalities.

2.6 Hardware components

The following sub-headers contain an explanation of each relevant component of a smartphone for this study.

2.6.1 Central Processing Unit

A central processing unit (CPU) is the core of the smartphone. It executes calculations and operations which executes the actions that are being made on

smartphones. The CPU operates on different cores with multiple threads which are affected by operations the CPU is conducting, impacting its speed to handle the calculations and operations.

The measurement from the CPU in this study will be from the load on User, System, and app. System refers to the amount of CPU time used by the kernel. The kernel is responsible for low-level tasks, such as interactions with the hardware and memory allocation. User refers to user space processes, which are higher-level processes like an application or a database running on the device. App is the load on, and the time spent by the CPU on the developed app/package.

2.6.2 Random Access Memory

Memory refers to the random-access memory (RAM) in this study. This memory is used to store and access the data that is being used along with machine code. RAM is not used to store long-term data such as images or other files but instead fetches data that allows the smartphone to complete its operations.

2.6.3 Battery

The battery powers the smartphone and is directly affected by the other hardware components. The more power that is required by for example the CPU, the more power will be drained from the battery.

2.7 Breaking point

A breaking point is in this study the result of the greatest increase of a trajectory, relative to another trajectory in a chart. The trajectory is defined between two points. The breaking point must not only be the greatest increase but also have the greatest increase in several occurrences (in the majority) in between the same points in different measurements. An example is shown in figure 3.

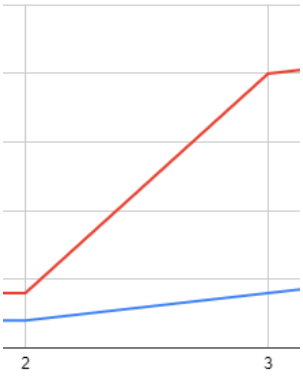

Figure 3. Example of a breaking point.

3 Method and implementation

This section covers the method used for data collection to answer the research questions given in this thesis.

The method for this thesis begins with a literature review and an in-depth look at Kotlin and React Native to pave a path and foundation for experiments, which will be the main subject of the method. The literature study includes which important aspects shall be explored and considered when it comes to creating Android apps for both Kotlin and React Native, and what has previously been explored. The literature study also included research for which functionalities to investigate further, as a pilot study. The experiments, test scenarios, and test cases will be described along with the functionalities that are relevant to each test case.

3.1 Connection between question and method

The first research question; *What is the difference between React Native and Kotlin's impact on smartphone performance when implementing functionalities in an app?* is answered with experiments that progressively include more processes and functionality in the form of a test scenario with several test cases, as it seeks to answer the impact on performance. The goal of this section of the method is to distinguish the differences and gather a possible curve of React Native's and Kotlin's results.

The second research question; *Is there a breaking point of app implementation where React Native's impact on smartphone performance is considerably worse compared to Kotlin? And if so, with what functionality and why?* is directly connected to RQ1, with RQ1 answered can the process of investigating if a curve will appear and where it starts. When adding more functionality and implementations of features in the test apps it will require more RAM, CPU, and battery power. The goal is to investigate if at some point during this process, a conclusion can be made if there is a point where Kotlin is performing notably better than React Native. Depending on the results will the question to 'why?' be answered through a deeper look at the functionality in question, including its implementation, package, and area of use.

3.2 Literature review

The scientific research papers relevant to get this study started were found on Diva and Google Scholar. Diva, which is the database where student's research papers including Jönköping University are uploaded to. Diva was a good place to start since the overall subject of this paper, smartphone development; has previously been explored in different ways.

The focus was finding future work in already published papers as well as the method sections where different experiment concepts were examined. The findings of each paper were also of interest. Literature studies are also important in motivating why our thesis is important by looking at older works so that there is no academic research yet within this field. By researching previous papers, it can be concluded which areas are extra important to conduct research within and how to try and bring something new to the table.

Through literature reviews and comparative, quantitative experiments with a scientific approach as described in [17] [18], will the hybrid framework React Native and the native language Kotlin be evaluated and tested for their impact on performance for Android smartphones. This thesis will evaluate based on the findings how much hybrid apps differ performance-wise with different kinds of functionality implemented to the native way of development.

3.2.1 Choosing functionalities

By looking at Google Play Store's top 20 most downloaded apps of 2019 could it be concluded that several of them are messaging, social media, and entertainment apps as found in [19]. By noticing this a pattern of different functionalities could be extracted, and an early idea of what to look further into was established. The goal with this information was to implement and combine these functionalities in a test app in a somewhat realistic context and in a way that would enable the use of experiments. The functionalities should also be able to work with each other and be initiated with a very small timespan between them. The chosen functionalities also need to vary between what type of hardware or services they are accessing, to get a better perspective on the end results.

For the scope of this project, six functionalities were chosen that could be separated into three different categories: device access, internet communication, and background processing. These three categories cover the major functions that an app usually has. In table 2 below are the chosen functionalities displayed and grouped by their respective category.

Device access	Internet communication	Background processing
Camera	API requests	Audio playback
GPS	Database integration	Bluetooth

Table 2. The chosen functionalities grouped by respective category.

The reasoning behind the choice of these functionalities is that they easily interact with each other (which creates a more realistic scenario for their usage) cover a wide area of smartphone feature access and are commonly featured functionalities in smartphone apps.

3.3 Experiments

To answer the first research question (RQ1) will experiments be conducted by creating one test app in Kotlin and one test app in React Native, with several versions of the same test app containing different amounts of implementations and functionality. The goal of RQ2 is to find out if there is a breaking point performance-wise for React Native. The experiments will be conducted in a way that enables quantitative research as described in [18], which means that the experiments must be able to be conducted multiple times without any unknown variables changing.

This will be done by eliminating all human interaction during the testing phase through automatic scrips that interact with the smartphone. This will in the end lead

to empirical data that can be compared and evaluated for a result that will determine the differences for each added functionality and implementation.
The parameters to be measured during these tests are:

- CPU (Central Processing Unit)
- RAM (Random Access Memory)
- Battery

These parameters are of interest in this study because they play a central role in how fast the smartphone can execute intended actions and how much power these actions draw from the smartphone. The parameters will be measured throughout the whole experiment whereas the CPU's workload is measured in percentage, the memory in megabytes, and battery drainage in voltage. The functionalities that are included in the experiments are also to be executed within milliseconds of each other, to get a big impact as possible on the above parameters. Functionality that is executed separately or far apart from each other will simply not have a meaningful impact on the goal of this study. An exception is a functionality that operates individually in the background, such as audio playback and Bluetooth.

3.3.1 Devices

The experiments were executed on two Android devices and the device model and hardware specifications are presented in table 3 below.

Device	OS	RAM	CPU	Battery
Huawei Honor 8	Android	4 GB	2.3 GHz x 8 cores Cortex-A72 & Cortex A53	3000 mAh
Samsung Galaxy S7	Android	4 GB	2.3 GHz x 8 cores Mongoose & Cortex-A53	3000 mAh

Table 3. Devices used and their hardware specifications.

3.3.2 Test Scenario

By combining several different functionalities seamlessly and in a concrete context can a test scenario be created. A test scenario contains several test cases. An illustration of this test scenario is demonstrated below in figure 4. The numbers between 1 and 6 indicate test cases and the chronological order of implementation and the surrounding boxes represent a contained test case, which is explained under the next header. To further stress test performance, the app will in the later instances include background processes with sound playback and Bluetooth communications (5 and 6) to see if there are any notable changes in the performance. Figure 4 below is a visualization of the increasing functionality and the foundation for the experiments.

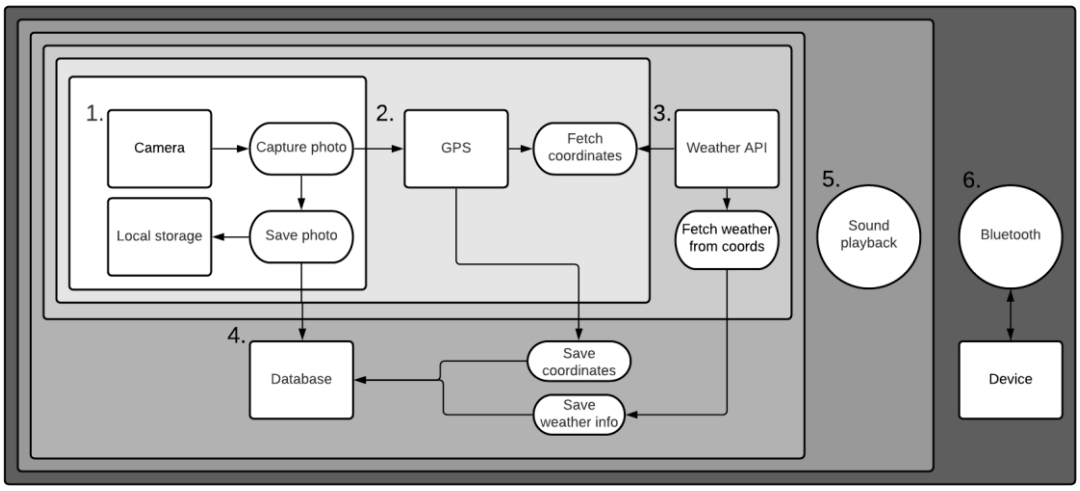

Figure 4. Visualization of the test scenario with several test cases.

3.3.3 Test Cases

A test scenario consists of several test cases in this study, as illustrated 1 to 6 above in Figure 4. A test case represents a certain implementation stage during development. Test cases are tested together in the order that they are logically implemented and together they make up a test scenario. Figure 5 illustrates test cases in experiments and how they are gradually applied. As seen does a newly added test case contain the previous test cases. All test cases will be integrated in a way that allows them to be tested simultaneously under the course of a few milliseconds. Their integration with each other is further explained under "Implementing test cases".

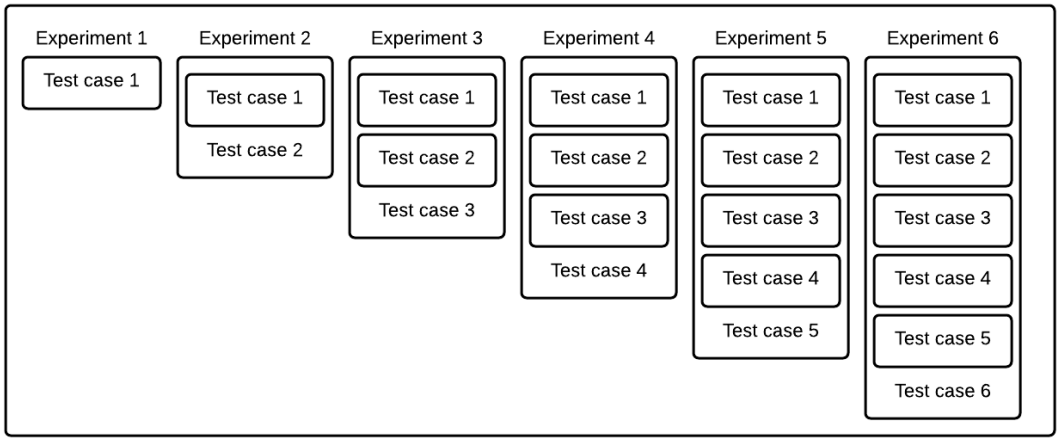

Figure 5. Illustration of how the test cases is gradually added to experiments.

3.3.4 Prerequisites for experiments

Important factors to consider whilst creating the test cases for conducting experiments is that the test cases are equally implemented between hybrid and native so that the results are of value. The experiments shall be conducted on the same smartphone so that the smartphone properties do not have an impact on the final findings for the experiments. The experiments will also be conducted on two different Android smartphones to get a wider angle on the results. The phones that

will be used are Huawei Honor 8 and Samsung Galaxy S7, both from 2016. As mentioned, the reasoning behind utilizing two phones is that it will give a better foundation when analysing the test results.

Test case 2-4 also includes external factors like other platforms or services which has their own impact on the experiments, which must be accounted for. For example, test case 3 includes sending a request to fetch weather information from an API. It is not guaranteed that this API is consistent with its speed of the response during the experiments. To minimize this kind of external impact on the experiments must parameters be set which lessens the impact of such anomalies. The most direct approach is to run several iterations of the tests to gather a larger amount of data which would "flatten out" the anomalies from external sources such as the API from test case 3. This issue will be more touched upon in "Arrangement of experiments".

3.3.5 Coding & structure

The implementation and coding of the apps will be test case-specific, meaning that one test case will be implemented at a time in the corresponding apps. By implementing one test case at a time, it is possible to make the apps as identically as possible, which will create accurate results for the experiments that will be conducted. All code is public and shared on GitHub [20], this creates transparency so that future works may analyze and investigate the code. Utilizing GitHub will help to organize the different test cases. This is done by dividing implementation into different branches, enabling easy access to the various timestamps of the incremental implementation of the test apps. Branching is a feature that GitHub and other remote source control tools have which is used to save implementations in a coding project.

When all test cases have been implemented in each app, will the testing be conducted individually and collectively with the existing previous test cases for any new variables changing in the performance, as described earlier in this section. Each test case will be presented in chronological order to make the reader and other researchers understand more about how and why different functionalities were implemented.

3.3.6 Test apps

The apps developed for native and React Native testing and conduction of experiments consist of a main view which functions as a starting page where nothing from the phone's parameters is measured. The main view also allows the user to navigate to the test scenario where tests and experiments will be conducted. The test scenario view consists of a camera preview with a single button for taking a picture, which also activates the various functionalities that the test scenario inherits. Screenshots are shown below in figure 6 of the main view and the test scenario view from the native app and figure 7 showcases the same of the React Native app. The screenshots are from an emulator for the sake of simplicity but will be tested on real smartphones as previously stated.

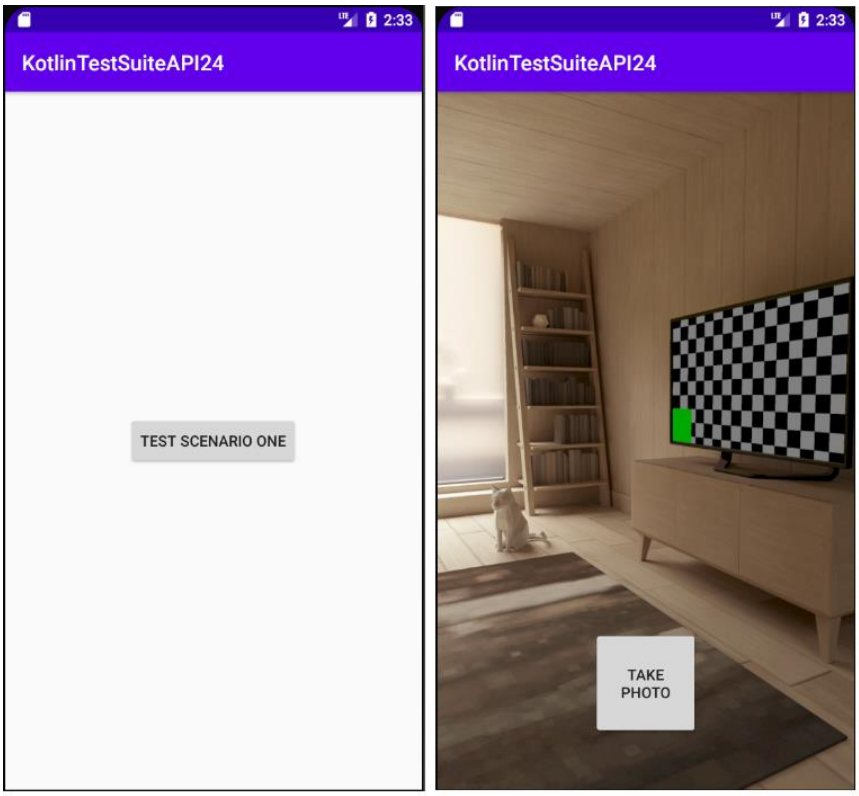

Figure 6. Screenshots of the main-view (left) and the test-scenario (right) from the native test app.

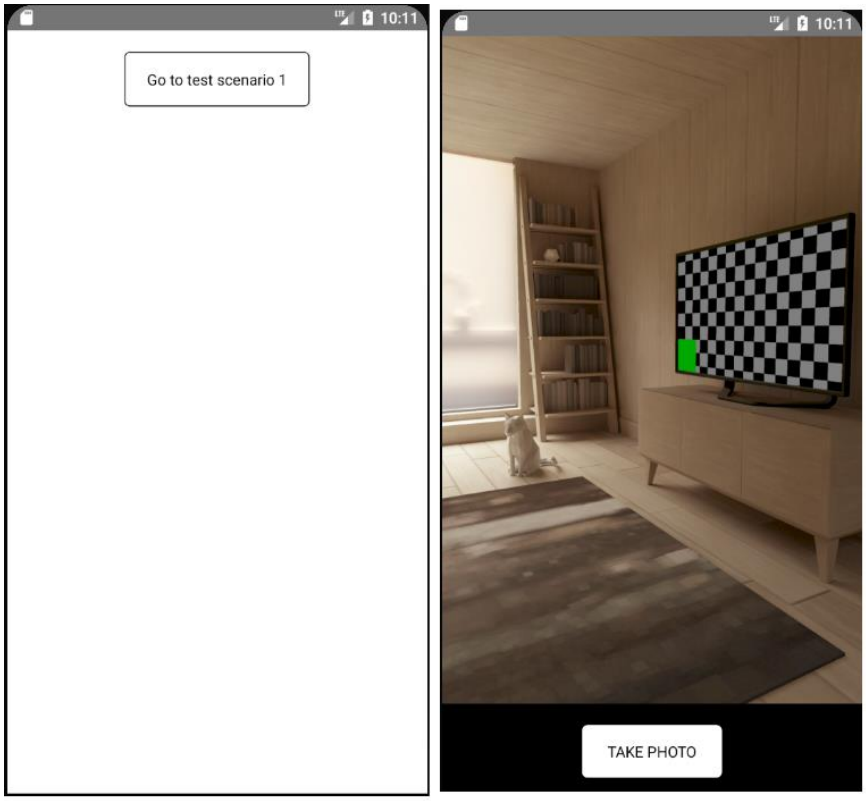

Figure 7. Screenshots of the main-view (left) and the test scenario-view (right) from the React Native test app.

3.4 Test Case Implementation

To present a structured way of reporting the implementation of the test cases, this section will cover each case for each solution for the reader to get a better understanding of the test cases. Each test case is described along with the functionality it contains as well as a summary of how it was implemented. Since all functionality is contained within the same view can no visual representation of each test case be made. Instead, the logical chain of events will be displayed for each test case.

3.4.1 Test case 1: Camera integration

The camera is the foundation of the test scenario. As shown in figure 4 which displays the test scenario, the camera starts all the other processes and test cases when a picture is taken. This test case on its own is simply just a preview of the camera with a button to take a photo and save it to the local storage. This test case utilizes the smartphone's device access as it displays a camera preview and saves the photo to the device. Pressing the button for taking a photo will then initiate the rest of the test cases, like dominoes. The library used for Kotlin is Android's latest camera library called CameraX [21]. For React Native was React Native Camera used [22], or RNCamera for short. RNCamera is a comprehensive camera module for React Native projects and is free to use but has an optional donation system. In figure 8 is the logic of test case 1 showcased.

Figure 8. Test case 1.

3.4.2 Test case 2: GPS

When a picture is taken, the first thing that happens is that location properties are fetched, it is worth adding that GPS is enabled from the beginning. The location is the coordinates presented in the longitude and latitude of the smartphone's current position. This is done by utilizing the smartphone's integrated GPS capabilities. This test case will take a photo and simultaneously get the current location of the smartphone. This is also done by accessing the smartphone's inherited functions as GPS is integrated in the hardware.

In Kotlin was a native class/library used and in React Native was a npm package called "React-Native Get Location" [23] used to ease up development. This package is a small lightweight package and was chosen for its simplicity. The goal of the test case is to fetch coordinates and nothing else. Figure 9 displays the logic of test case 2.

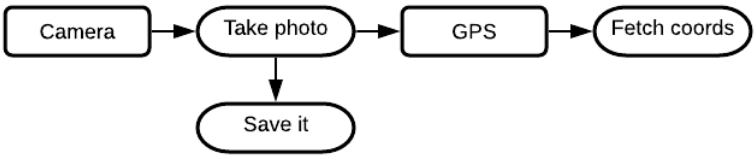

Figure 9. Test case 2.

3.4.3 Test case 3: API

The API integration in this test scenario is a weather API. The API that is used for this test case is called Open weather map (https://openweathermap.org). After fetching the coordinates in the previous test case, this test case utilizes those coordinates to get the current weather data from the weather API. The data, which is sent as a response from the API is of the JSON format, which allows iterations through its data to extract the necessary information. The necessary information is in this test case the main weather status and the city to which the coordinates belong.

This test case takes a photo, fetches the coordinates, and then fetches the current weather from the coordinates. In Kotlin the library Volley [24] was used to send an HTTP GET request to the API. Volley is a common library used to send HTTP requests and is referenced in the documentation on Android developer's official website, which emphasized the choice of this library. React Native did not need a library or a package, to send the API request. Figure 10 shows the logic for test case 3.

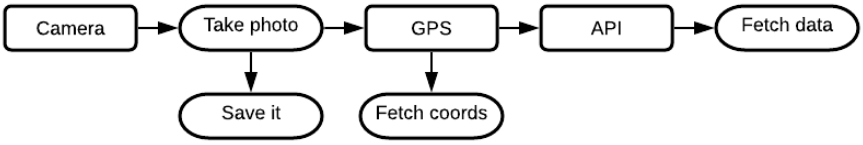

Figure 10. Test case 3.

3.4.4 Test case 4: Database integration

For test case 4 was Firebase [25] used, which is a database developed by Google and suited for integrations with smartphones. The purpose of this test case is to save the photo, the GPS coordinates, and the weather data acquired from the API. The previously gathered data is stored temporarily in variables and is then sent to the database. The weather information is stored as strings in the database whilst the photo is stored as a reference and the GPS coordinates are stored as numbers. In total this test case does all the previous tasks and ultimately stores the data in the database. For Kotlin was Google's default package for firebase connections used and for React Native was the npm package "React Native Firebase" [26] used, which enables similar techniques for storing data to firebase as native. Figure 11 contains the further logic of test case 4.

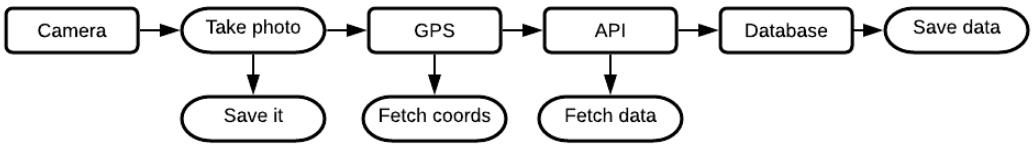

Figure 11. Test case 4.

3.4.5 Test case 5: Audio playback

To further add functionality to the test scenario, background processes will be used. The first background process is audio playback, which means that an audio file will be queued when first starting the test scenario. This audio file will then play in the background during the experiments to add more processing power.

For React Native was a npm package called "React Native Sound" [27] used which enables the developer to queue an mp3 file for playback. React Native Sound is a commonly downloaded npm package and has all the basic functionality such as play/stop, volume, and the ability to play a track from either a file or a network. In this test case will the audio be played from a file. Implementing the audio playback for Kotlin is simple since you only need to utilize the inbuilt class to play any audio file that you include in the project. The logic of test case 5 is presented in figure 12.

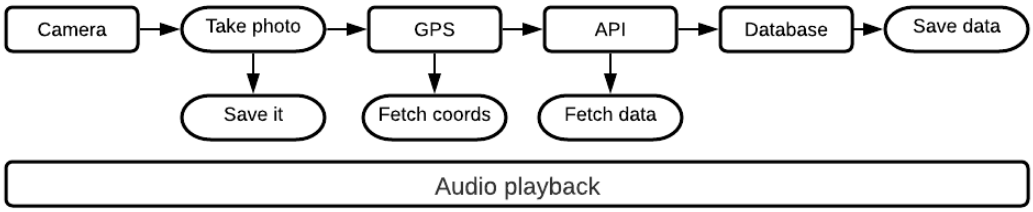

Figure 12. Test case 5.

3.4.6 Test case 6: Bluetooth

The second background process and final test case for the test scenario is Bluetooth. This test case will operate in the background along with the audio playback. This test case will establish a connection with a computer that runs a corresponding Bluetooth protocol to receive data. The data that will be transmitted from the smartphone is dummy data in the form of a string, which continuously is sent to the computer program. This test case will start at the same time as the previous test case and run simultaneously.

In React Native was the npm package "React-native-bluetooth-classic" [28] used which enables serial Bluetooth communication with another device, where the React Native app acts as a client. For the implementation on Kotlin were sockets used. Where the app acts as a client and connects to the server with the help of the MAC address and UIID is used when sending data.

Figure 13 shows the logic of the last test case.

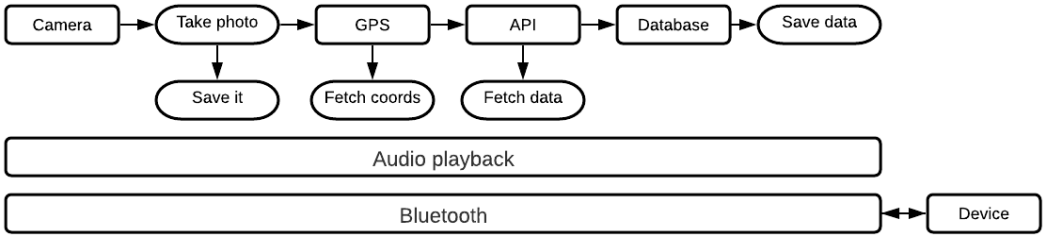

Figure 13. Test case 6.

3.4.6.1 SPP Bluetooth

To test the Bluetooth functionality and its impact on performance was the decision made to implement the SPP (Serial Port Profile) [29] which requires more energy than its counterpart BLE (Bluetooth Low Energy). The reason behind SPP is that it will send more data streamlined than BLE and it will have a greater impact on the smartphone's hardware performance.

The approach for the Bluetooth part was to implement a server-side on the computer that was created with the help of python. A client-side was implemented on the smartphone test app to create a scenario where the client continuously sends data to the server to simulate a likely scenario that could happen with the use of Bluetooth.

3.5 Arrangement of Experiments

Test case 1-4 is initiated by simply taking a photo during the experiments. The test cases will then be executed within milliseconds of each other and use each other's information in different ways. Test cases 5-6 will be started automatically when the test scenario is initiated and then run in the background as the other test cases are executed through automatic input. Table 4 below shows the structure of the experiments. The execution is illustrated from left to right in chronological order. From the table, it is also shown that in total will six experiments be conducted for each smartphone.

Experiment	**Test case 1**	Take photo					
Experiment	**Test case 2**	Take photo	Fetch GPS coordinates				
Experiment	**Test case 3**	Take photo	Fetch GPS coordinates	API request			
Experiment	**Test case 4**	Take photo	Fetch GPS coordinates	API request	Send to database		
Experiment	**Test case 5**	Start audio playback	Take photo	Fetch GPS coordinates	API request	Send to database	
Experiment	**Test case 6**	Start Bluetooth connection	Start audio playback	Take photo	Fetch GPS coordinates	API request	Send to database

Table 4. Structure of experiments with test cases.

The choice of this approach resides in comparing as many functionalities as possible for the scope of this project and the ability to connect them in a context with each other to simulate gradual app implementations. Other comparative cases could include functionalities that are compared for a longer amount of time, for example navigating through an app that contains multiple views while parameters of the performance are monitored. This kind of test is not suitable for this study as increasing functionality implementation would be more complicated to measure.

3.5.1 Test case 1-4 experiments

Each experiment and the corresponding test case will be initiated with a script through a console with ADB commands to eliminate any human interaction. These scripts can be found in appendix 1. Experiments for test cases 1-4 will execute the test cases 12 times in a row with an intermediate pause of 5 seconds, a duration of one minute. The reasoning behind this cycle is to imitate a real-life scenario where a user is taking multiple pictures through an app during a period.

Multiple cycles of the test cases will also give more accurate results and eliminate some of the anomalies that may occur in the parameters during testing. The smartphones used for testing will be restarted for the first experiment to get more accurate data. The smartphone will then be restarted between every test case. The reasoning behind this is to give each test case the same conditions. A diagram of the structure for test cases 1-4 is presented in figure 14 below.

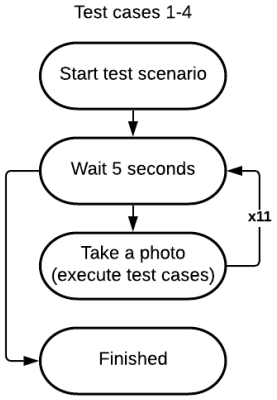

Figure 14. Experiment cycle for test case 1-4.

3.5.2 Test case 5-6 experiments

Test case five and six contains background processes and are therefore treated a bit differently than the previous test cases. The arrangements for these test cases will carry on for a longer amount of time and execute the previous test cases less frequently. The reasoning for this is to get a better understanding from the data of the background processes, which would yield more accurate results over a longer amount of time. The experiments for these test cases will run for 10 minutes and execute the previous test cases each 15 seconds, a total execution of 40 times. A diagram of the structure of test case 5-6 is presented in the diagram below in figure 15.

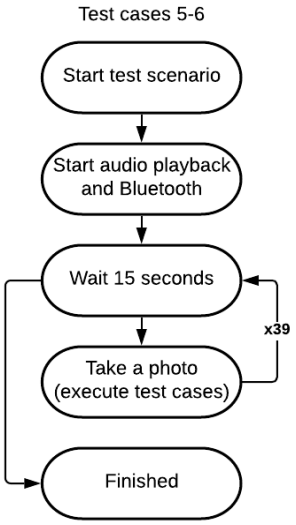

Figure 15. Experiment cycle for test case 5-6.

3.5.3 Measurement cycle for experiments

The measurements that are done to extract data from the smartphone's hardware components during experiments are the same for all test cases and all the components, CPU, RAM, and battery. To get the most possible accurate sample from the implemented functionality will all the components' data be extracted directly when they are executed (i.e., when the photo is taken). This way of extracting data will give as many samples as the experiment is looped in the script, which yields a definitive amount of data to expect when starting the tests. A diagram of the measurements during experiments can be seen in figure 16 below.

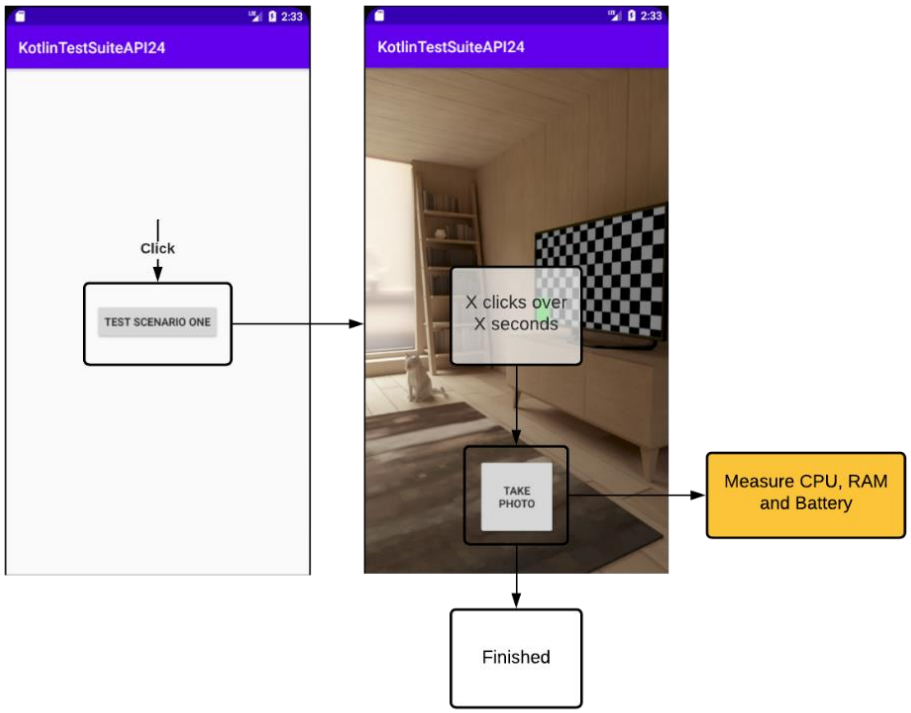

Figure 16. Logic of how measurements are done during experiments.

3.5.4 Handling Data from Experiments

When defined data has been extracted from the experiments must the results be compiled. This will be done by recording each result in a document so that data from each implementation can be evaluated. The gathered data will be of a reasonable size which empathizes a calculation of an average for each test case. The first research question (RQ1) will be directly answered through this comparison as its answer simply relies on gathering hard data. Secondly, a curvature will be created of the performance for multiple and different test cases. The curvature will contain all gathered data from the experiments that are meant to answer research question two. With a curvature of the results can the second research question (RQ2) be investigated. Which is finding a breaking point where the performance gap is the biggest. If a breaking point (or several) were to be found further conclusions can be made and to which some may be subjective and must be discussed. This can differ for each test case or type of functionality.

4 Findings and analysis

In this chapter will the gathered data from the experiments be presented and examined. The data presented is firstly ordered in headings by research question where the first research question is secondly divided by test case and thirdly divided by which devices the data has been collected from.

The data presented for research question two is simply divided by the measured component and secondly by smartphone. The data presented in this chapter is an average of the total amount of data gathered. The whole amount of data contains some anomalies but has been accounted for in the method. All data collected can be found in appendix 4 to 25. Some measurements for CPU for Samsung Galaxy S7 overextend 100% because the measurements made are across all cores in the CPU.

4.1 Research question 1

The data necessary to answer the first research question; *What is the difference between React Native and Kotlin's impact on smartphone performance when implementing functionalities in an app?* Is presented under this header where each test case is accounted for and grouped by the smartphone from which the data has been collected from.

4.1.1 Test Case 1

The data found for test case 1 is presented in table 5 and 6 below.

Huawei Honor 8

Right from the start in test case 1, there is a major difference between native and hybrid when it comes to the load on the CPU for the processes in user and the app, as seen in table 5. React Native has a higher load on user processes, which amongst other top-level tasks, is time spent on the app. This also correlates for the time spent on the test app which also showcases a higher percentage for React Native. System processes are lower and similar between React Native and Kotlin. The memory shows the same low difference between the two.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	11.83	12.5	2.25	2129.33	56.33	4.30
React Native	18.33	12.08	10.08	2025.75	53.59	4.23

Table 5. Data extracted from test case 1 with Honor 8.

Samsung Galaxy S7

The difference here from the Honor 8 smartphone is that the CPU load is higher for Kotlin on all measured processes compared to React Native, as seen in table 6. However, it is noted that React Native has a higher RAM usage than Kotlin where the difference is not that distinguishable.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	90.9	94	70	1399	38.6	4.22
React Native	60	61.75	47.45	1615	44	4.29

Table 6. Data extracted from test case 1 with Galaxy S7.

4.1.2 Test Case 2

The data found for test case 2, which is presented in table 7 and 8 below.

Huawei Honor 8

In test case 2 with GPS functionality added are the parameters about the same and even lower in some cases. Both native and hybrid however shows a small increase in system which handles the hardware. As previously seen, hybrid has worse performance for CPU but this time only a small amount. Hybrid also continues to have a slightly better performance for the memory.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	13.91	12.91	1.91	2129.83	56.34	4.22
React Native	14	12.33	12.16	1971.16	52.14	4.20

Table 7. Data extracted from test case 2 with Honor 8.

Samsung Galaxy S7

For test case 2 there is a shift in the CPU user and system usage however this time CPU app has a higher usage in React Native compared to Kotlin. There is still a pattern here with a bigger RAM usage for React Native than Kotlin.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	108.667	98	59.66	1416	39	4.28
React Native	130	118.83	62.925	1728.75	47.7	4.26

Table 8. Data extracted from test case 2 with Galaxy S7.

4.1.3 Test Case 3

The data found for test case 3, which is presented in table 9 and 10 below.

Huawei Honor 8

In the third test case seems hybrid to be affected the most, displaying worse performance for CPU but about the same for the memory. Native however shows a minor increase in system time and a decrease for both the app and user processes, remaining mainly unaffected by the added functionality.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	11.5	13.66	1.83	2128.91	56.32	4.24
React Native	19	15	12.08	2066	54.65	4.19

Table 9. Data extracted from test case 3 with Honor 8.

Samsung Galaxy S7:

In the third test case it is noted that performance for CPU user and system has increased for both Kotlin and React Native, whilst app usage stays the same for Kotlin but decreases for React Native. The RAM usage stays around the same with just a slight increase for React Native.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	124.4	104	59.7	1424	39.34	4.2765
React Native	169.25	145.48	50.525	1822.75	50	4.26

Table 10. Data extracted from test case 3 with Galaxy S7.

4.1.4 Test Case 4

The data found for test case 4, which is presented in table 11 and 12 below.

Huawei Honor 8

With the addition of database integration, the results are like previous test cases. Native remains mostly unaffected with a minor decrease in some instances such as user and system measurements but also suffers a minor increase in time spent for the CPU in the app processes. Hybrid showcases as previous test cases (except test case 2) an overall increase in almost all parameters.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	9.58	10.5	2.41	1906.66	50.44	4.28
React Native	21.33	14.58	14.91	2065.66	54.64	4.19

Table 11. Data extracted from test case 4 with Honor 8.

Samsung Galaxy S7

In the fourth test case there is still a higher usage for CPU user and system on React Native whereas Kotlin has a decrease in said processes. CPU app stays around the same as test case three for Kotlin but React Native sees a decrease in usage. RAM usage is increased by about 1 percent point for both Kotlin and React Native.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	98.25	87.66	58	1484	40	4.27
React Native	192.83	158.6	37.875	1859.25	51	4.25

Table 12. Data extracted from test case 4 with Galaxy S7.

4.1.5 Test Case 5

The data found for test case 5, which is presented in table 13 and 14 below.

Huawei Honor 8

The first addition of a background process shows again an increase in most parameters for React Native (especially CPU) while Kotlin only suffers a minor increase. The data shows a small decrease load on the app which may explain the increase on both system and user, as the audio playback is operating between kernel level with the hardware and the top level of the app.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	10.15	11.12	1.92	1957	51.1	4.20
React Native	25.42	18.07	13.5	2015	53.33	4.09

Table 13. Data extracted from test case 5 with Honor 8.

Samsung Galaxy S7

Looking at table 14 it is easy to note that the parameters that has changed noticeably is the CPU performance for React Native, where the parameters has decreased by quite a bit. The table shows that Kotlin has decreased for CPU system and app but not user, and that RAM has increased very small.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	99.25	83.775	51.425	1508.3	41.6	4.33
React Native	156.725	133.9	46.312	1794.95	49.6	4.15

Table 14. Data extracted from test case 5 with Galaxy S7.

4.1.6 Test Case 6

The data found for test case 6, which is presented in table 15 and 16 below.

Huawei Honor 8

With the second addition of a background process, Bluetooth; can the biggest difference in performance be seen for Kotlin, at least for all CPU processes as memory remains largely unaffected. The user CPU process has an increase of roughly 18 percent points from the last test case while the processes in app sees an increase of roughly 10 percent points and system a smaller increase of about 8 percent points. Hybrid also suffers a major increase in CPU processing power in user and app measurements. User processes has an increase of roughly 10 percent points while app increases about the same with roughly 9 percent points. Both process powers are also the highest recorded of all test cases for both Kotlin and React Native. System processes remain largely the same however for React Native and the same goes for the memory.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	28.65	18.77	12.15	1941	51.13	4.18
React Native	34.72	17.9	21.4	2051	54	4.03

Table 15. Data extracted from test case 6 with Honor 8.

Samsung Galaxy S7

The biggest change in all the CPU performance parameter can been seen here, with a huge increase for React Native CPU user with a 116 percent points increase. With the Bluetooth implemented we can see that the CPU app is bigger now for React Native rather than Kotlin, also the CPU system sees a notable spike in the percentage of usage for both Kotlin and React Native. RAM however stays around the same as previous test cases but with more usage for React Native.

	CPU User (%)	CPU System (%)	CPU App (%)	RAM (KB)	RAM (%)	Battery (V)
Kotlin	170.4	120.1	68.8275	1553.15	42.8	4.26
React Native	272.775	173.526	93.325	1873.125	51.7	4.23

Table 16. Data extracted from test case 6 with Galaxy S7.

4.2 Research question 2

In this chapter is the same data used but with all test cases and displayed in graphs to easier display the points of test case implementation and their differences, both relatively and objectively. This showcase of recorded data also means to give a basis for research question two; *Is there a breaking point of app implementation where React Native's impact on app performance is considerably worse compared to Kotlin? And if so, with what functionality and why?* Each graph is grouped by their measured hardware component.

4.2.1 CPU

The processing power from the CPU has the most diverse results between the test cases and is the component to which most conclusions can be made.

Huawei Honor 8; CPU User

In chart 1 can the results from the Honor 8's CPU user processes be seen, which is top-level tasks for the processor. React Native has a deviation decrease in processing power for test case 2 (where it is about the same as Kotlin) but then steadily increases for the rest of the cases. React Native also has a higher processing power than Kotlin for all measurements except test case 2, where test case 5 is the biggest difference. Kotlin is however catching up to React Native in terms of processing power in the last case, test case 6 where the difference is the smallest since test case 2. Kotlin instead has a decrease in processing power after test case 2 and until test case 4, which might suggest that native is unaffected by the implemented functionality.

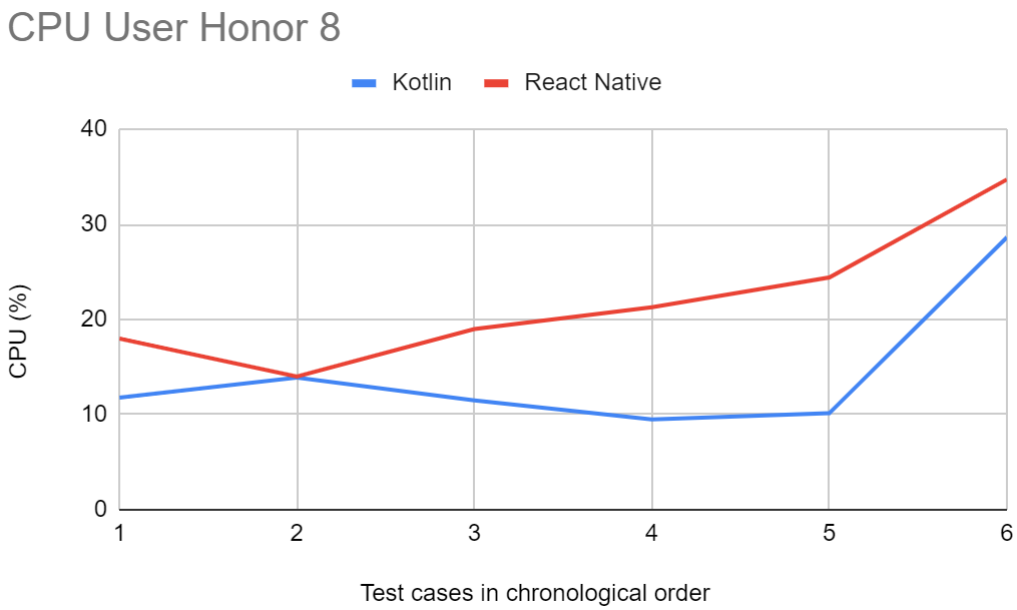

Chart 1. Data from user CPU processes with Huawei Honor 8.

Samsung Galaxy S7; CPU User

The Galaxy S7's results for CPU user processes (chart 2) shows that Kotlin has an initial higher processing power than React Native, which then React Native is catching up to and over exceeding in test case 2. React Native then increases in processing power for the next test case to then flatten out and have a small decrease for test case 4 and 5. As for the Honor 8, the Galaxy S7 is also experiencing a big increase in power for the last test case. Kotlin have about the same curvature as React Native, but overall a smaller percentage in processing power for all the test cases after test case 1.

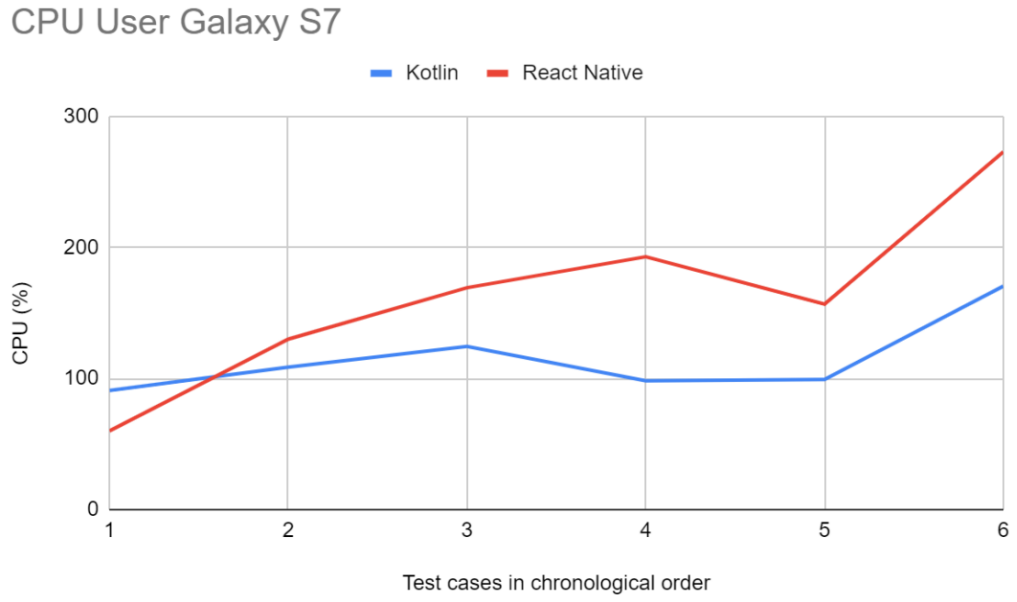

Chart 2. Data from user CPU processes with Samsung Galaxy S7.

Huawei Honor 8; CPU System

For Honor 8's CPU system processes (chart 3), which are the kernel level processing tasks that communicates with the hardware has notable increases in processing power in every other test case for React Native. The biggest increase for React Native can be seen between test case 2 & 3 and 4 & 5. Test case 3 adds the weather API functionality and therefore seems to have an increase in the smartphone's internet communication components. In test case 4 is the processing power about the same as for test case 3 which may be explained by that test case 4 also utilizes the smartphones internet communication components like test case 3, and therefore remaining on the same level. React Native also sees an increase with the first background process being added, audio playback and like between test case 3 and 4 remains about the same with the second background process being added; Bluetooth.

Kotlin has a more strange and unpredicted result between the test cases. At first decreasing a bit to then fall in processing power drastically and to then see an increase again. It can either be a deviation for test case 4 or that internet communication does not have a big impact on the performance, resulting in non-linear measurements. However, after test case 4 there is again an increase in power where test case 6 sees a huge amount of increase for Kotlin, which cannot be seen for React Native.

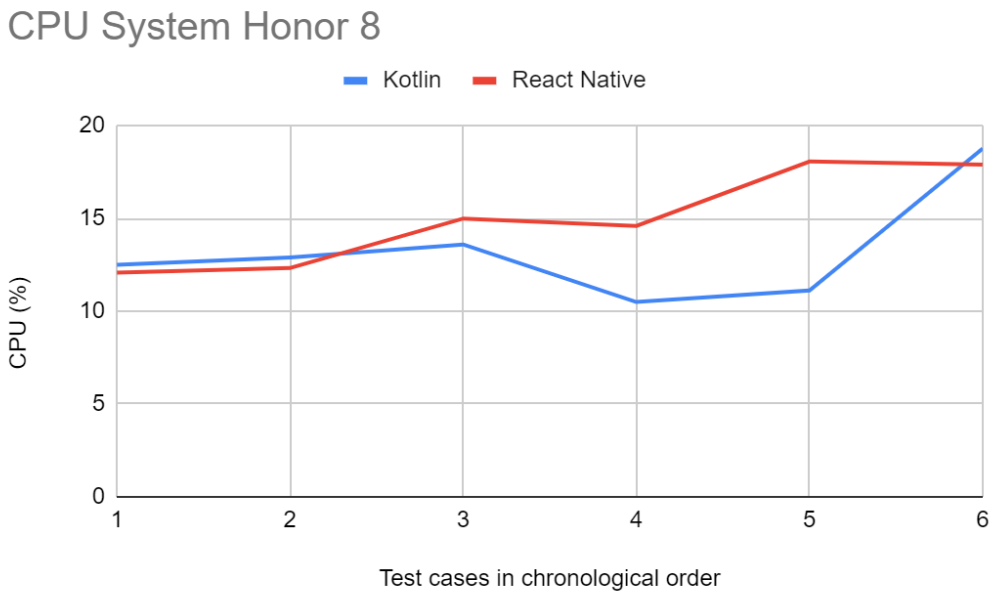

Chart 3. Data from system CPU processes with Huawei Honor 8.

Samsung Galaxy S7; CPU System

The Galaxy S7's kernel-level tasks (chart 4) show about the same scenario as for CPU user, whereas Kotlin is starting off higher to then be overtaken by React Native for the rest of the test cases. React Native has a steady climb in processing power until test case 5 where a dip in percentage is seen. The longer test case 5 with audio playback seems to not be affecting the processing tasks as much as previous test cases. However, Bluetooth holds up the trend of taking the percentage of

processing to the highest of the measured test cases. Kotlin starts off very steady with only minor increases in power, but then unlike React Native has a decline for test case 4, the addition of database integration. Kotlin also suffers a big increase for Bluetooth on a kernel-level.

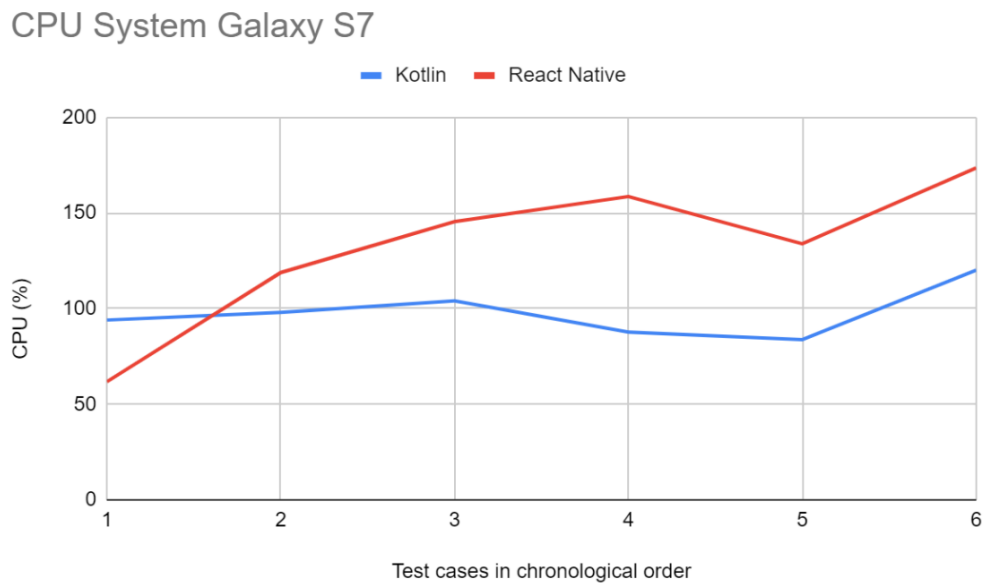

Chart 4. Data from system CPU processes for Samsung Galaxy S7.

Huawei Honor 8; CPU app

CPU app (chart 5) is the time spent on the testing app by the processes. These processes have the most consistent difference between them and do not have any crossings or meeting points in performance between React Native and Kotlin. Like previous processes does React Native have an overall progressively increasing curvature, whilst Kotlin remains largely unaffected by the implementations until the last test case comes into play, Bluetooth. React Native has its biggest increases in processing power between the first test case & the second, the third & the fourth and then the biggest in the last test case with Bluetooth communication. Kotlin does not have any notably big changes in performance until test case 6, for the context of the charts.

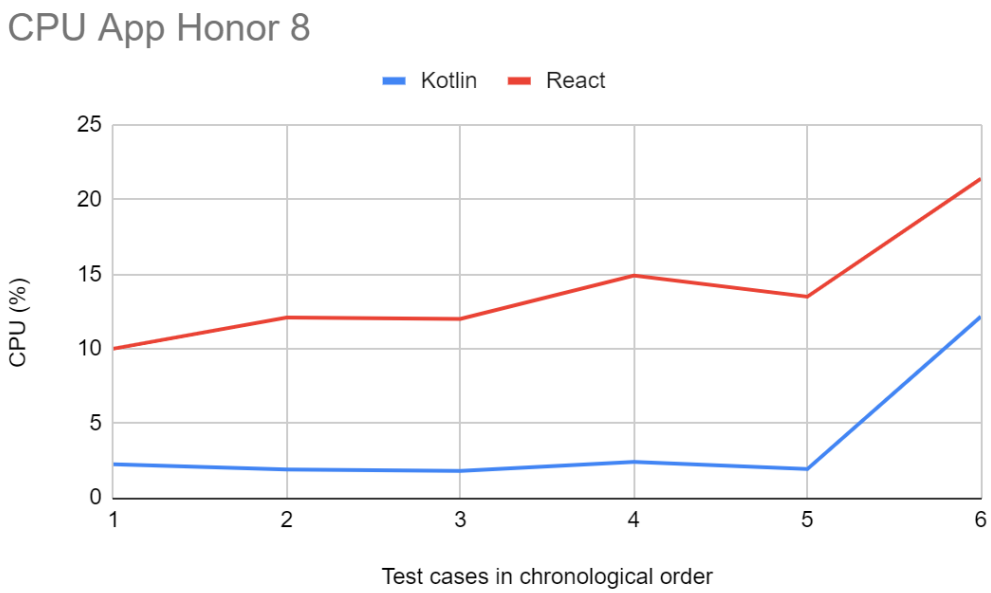

Chart 5. Data from app CPU processes for Huawei Honor 8.

Samsung Galaxy S7; CPU app

The measurements for CPU app for Samsung Galaxy S7 (chart 6) is all over the place with lots of different increases and decreases, especially for React Native. This is also the only instance where React Native most consistently have a better performance, percentagewise to Kotlin. An anomaly is that both React Native and Kotlin are experiencing decreases in processing power after test 2 whereas React Native is seeing an increase after two test cases, one test case earlier than Kotlin, which is only seeing an increase for the last test case.

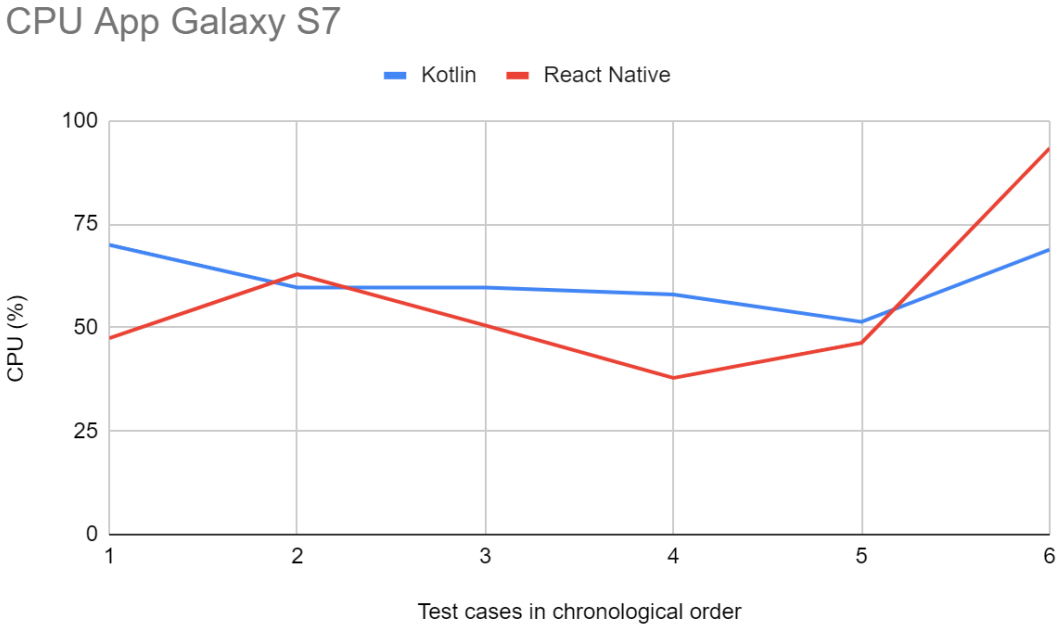

Chart 6. Data from app CPU processes for Samsung Galaxy S7.

4.2.2 RAM

The measurements from the RAM show different results between the smartphones.

Huawei Honor 8; RAM

The memory that handles working data and machine code (chart 7 and 8), the Random-Access Memory shares about the same results for both React Native and Kotlin with no protruding differences between all test cases. React Native has a more consistent results with an arguably steady line for all test cases. Kotlin does however see a decrease in usage after test case 3 and stays on about that level for the remaining test cases.

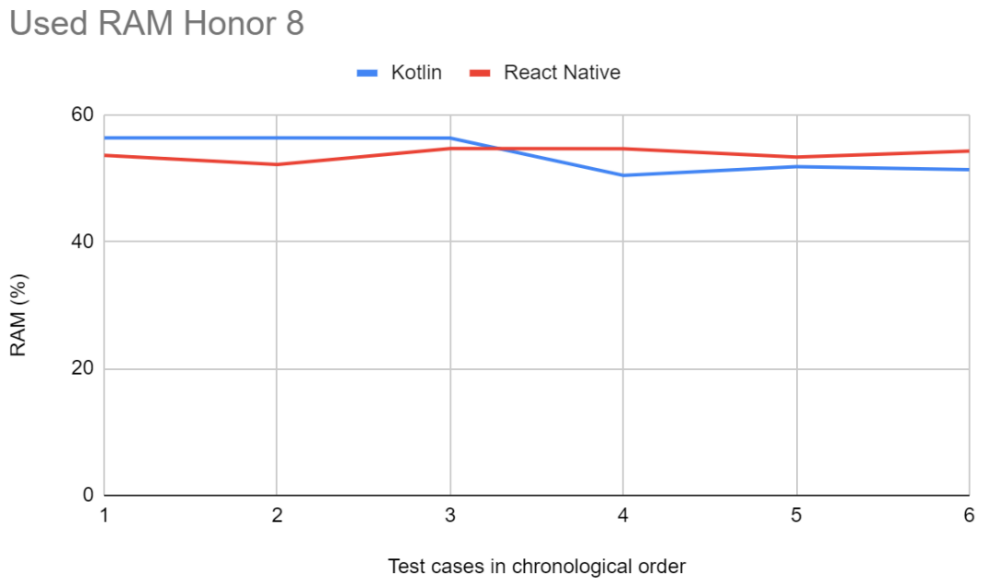

Chart 7. Data from RAM for Huawei Honor 8.

Samsung Galaxy S7; RAM

There is a clear difference between Kotlin and React Native for RAM usage compared to the previous smartphone. Both have a steady growth with a slight decrease for React Native at test case 4 to 5 and observing the graph we can see that React Native utilizes more RAM than Kotlin.

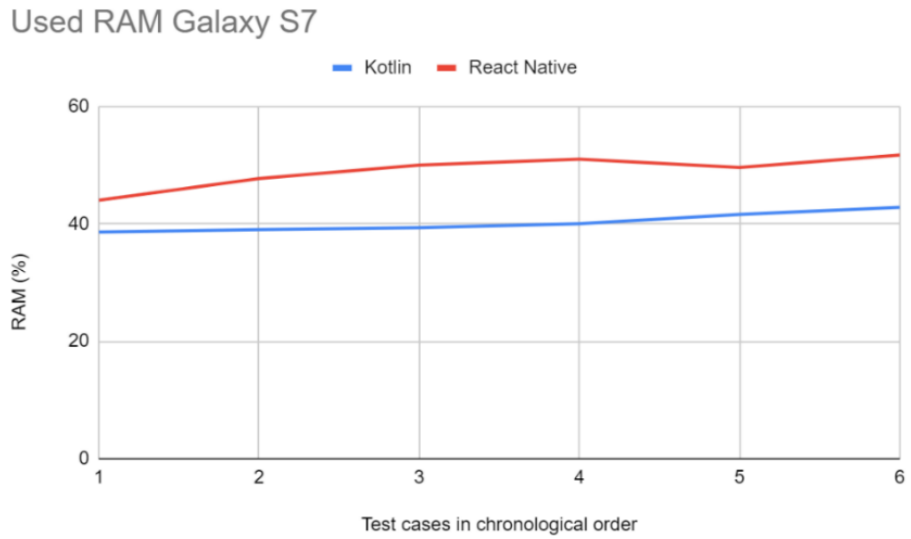

Chart 8. Data from RAM for Samsung Galaxy S7.

4.2.3 Battery

The battery shows a slightly higher than expected voltage for both.

Huawei Honor 8; Battery voltage

Most smartphone batteries work at the same voltage, which is 3.8 volts. From the experiments can a bit larger voltage be seen (around 4.2 V) for both React Native and Kotlin (chart 9 & 10). The measurements between React Native and Kotlin are also about the same for all test cases. After test case 4 can a decline in voltage be seen for both React Native and Kotlin, nearing the expected value of 3.8 Volts.

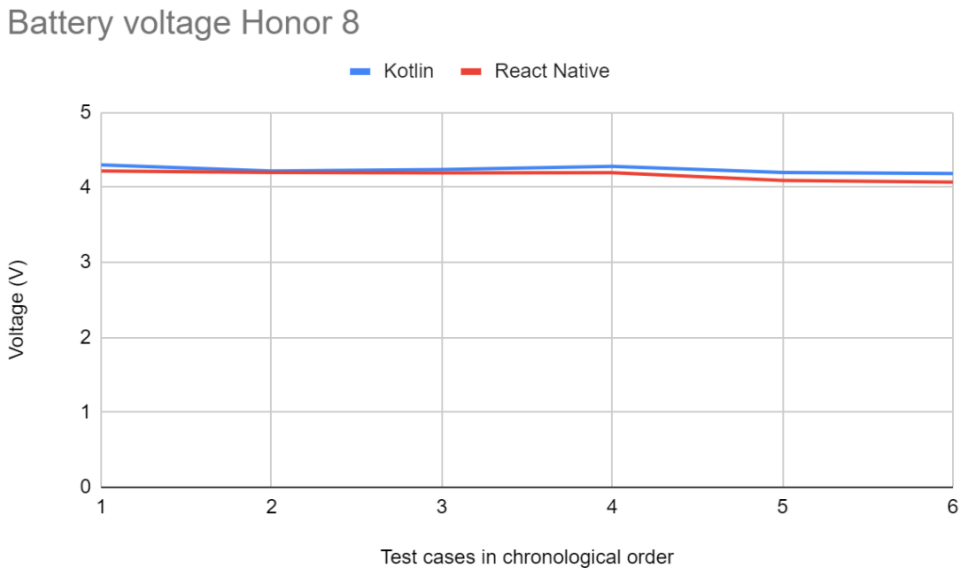

Chart 9. Data from the battery for Huawei Honor 8.

Samsung Galaxy S7; Battery voltage

Both Kotlin and React Native has about the same voltage when being tested however we see a slight increase in Kotlin between test case 4 and 5 but afterwards it goes back to the same level.

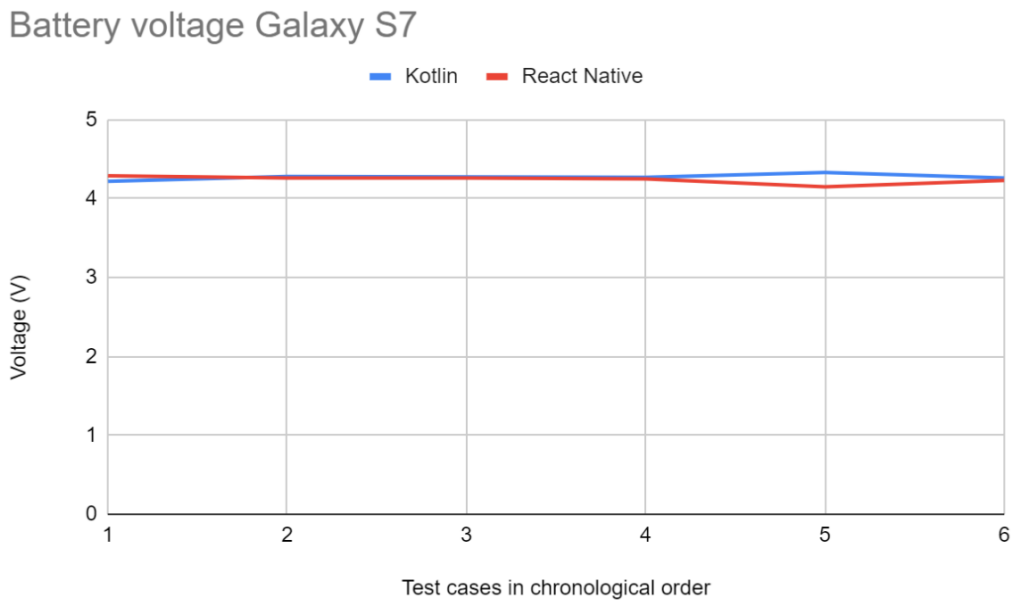

Chart 10. Data from the battery for Huawei Honor 8.

4.3 Further analysis

To have more material for the discussion and findings section of this paper and to find a breaking point, a further analysis was made for the CPU measurements. CPU was chosen due to having the most relevant data. The first calculations and analysis that was made was to seek out where the biggest difference was for all test cases. This was made for each CPU process that was recorded. Table 17 below shows the biggest difference between Kotlin and React Native for each CPU process as well as their induvial performance for corresponding test case. Positive numbers in the column for differences shows how much React Native is performing worse compared to Kotlin in percent points (pp). From this could it also be calculated in how many cases React Native was performing worse than Kotlin, which was in 27 out of 36 cases. The whole table is available in appendix 26.

Honor 8	**Test Case**	**Kotlin CPU APP (%)**	**React Native CPU APP (%)**	**Difference (pp)**
	4	2,4	14,9	12,5
Honor 8	**Test Case**	**Kotlin CPU USER (%)**	**React Native CPU USER (%)**	**Difference (pp)**
	5	10,15	24,425	14,275
Honor 8	**Test Case**	**Kotlin CPU SYSTEM (%)**	**React Native CPU SYSTEM (%)**	**Difference (pp)**
	5	11,125	18,075	6,95
Galaxy S7	**Test Case**	**Kotlin CPU APP (%)**	**React Native CPU APP (%)**	**Difference (pp)**
	6	68,8275	93,325	24,4975
Galaxy S7	**Test Case**	**Kotlin CPU USER (%)**	**React Native CPU USER (%)**	**Difference (pp)**
	6	170,4	272,775	102,375
Galaxy S7	**Test Case**	**Kotlin CPU SYSTEM (%)**	**React Native CPU SYSTEM (%)**	**Difference (pp)**
	4	87,66	158,6	70,94

Table 17. Table of biggest difference in CPU processing per test case.

From the differences found could then the differences in processing power between test cases be calculated, which is the foundation for finding a breaking point. The breaking point indicates where and with what addition of functionality that increases the processing power the most, and most often.

Table 18 below shows the biggest difference between React Native and Kotlin for each CPU process as well as their individual differences between test cases. Positive numbers in the column for Kotlin/React Native difference again means a poorer performance for React Native. From these calculations could it be shown that React Native had worse development between test cases in 19 out of 30 cases. From this table can also the breaking point be found which is located between test case one and two for Samsung Galaxy S7 in the system process. This is supported by the second biggest difference which also is located between test case one and two for Samsung Galaxy S7, but in the user process. Huawei Honor 8 does however not have a breaking point since at least two of the biggest difference are not between the same test case. The whole table is available in appendix 27.

Honor 8	**Test Cases**	**Kotlin CPU APP (pp)**	**React Native CPU APP (pp)**	**Kotlin/React Native Difference (pp)**
	Difference 1–2	-0,35	2,1	2,45
Honor 8	**Test Cases**	**Kotlin CPU USER**	**React Native CPU USER (pp)**	**Kotlin/React Native Difference (pp)**
	Difference 2–3	-2,4	5	7,4
Honor 8	**Test Cases**	**Kotlin CPU SYSTEM**	**React Native CPU SYSTEM (pp)**	**Kotlin/React Native Difference (pp)**
	Difference 4–5	0,625	3,475	2,85
Galaxy S7	**Test Cases**	**Kotlin CPU APP**	**React Native CPU APP (pp)**	**Kotlin/React Native Difference (pp)**
	Difference 5–6	17,4025	47,013	29,6105
Galaxy S7	**Test Cases**	**Kotlin CPU USER**	**React Native CPU USER (pp)**	**Kotlin/React Native Difference (pp)**
	Difference 1–2	17,767	70	52,233
Galaxy S7	Test Cases	**Kotlin CPU SYSTEM**	**React Native CPU SYSTEM (pp)**	**Kotlin/React Native Difference (pp)**
	Difference 1–2	4	57,08	53,08

Table 18. Table of biggest difference between test cases in CPU processing.

5 Discussion and conclusions

This chapter will summarize and discuss the results based on the authors own conclusions on the outcome of the study.

5.1 Discussion of method and implementation

The methods used to answer the research questions in this thesis was quite unique in the sense that it had not been utilized in related works before. The result satisfied the research questions which proved that there was a good bridge between the method and the research questions. The method did however prove to give a slightly less accurate answer for parts of our second research question, due to the data that was extracted. There were some studies that were related to the thesis however the studies only briefly covered some parts of the performance and were mostly focused on other aspects. With help of the literature review, the research questions managed to be formed into more unique and specific questions that had not been researched before.

The use of test cases had several benefits, for one it was easy to simply step back and see how the functionality affected the performance for both Kotlin and React Native. Test cases also make it more clear and easier for the reader and other interested parties to see how and what has been implemented if further studies were to be done. Also, each test case provides clear data that shows exactly how both Kotlin and React Native affect the performance of the smartphones.

One of the downsides with ADB when investigating the performance parameters on the phones was that ADB responded differently when utilized on different smartphones. ADB is a great tool for reading the different hardware performance whilst connected to the phone but when it responded differently depending on which smartphone that was used, some compromises had to be made. For example, with the experiment collection on the Huawei smartphone only one script needed to be utilized to collect the necessary data. With the Samsung smartphone, the scripts needed to be broken into two different scripts and then used separately, this meant more time for the collection of data. Overall ADB is a great tool for diving into the smartphone and accessing information that was useful for this study. The values received for battery consumption were not a good way to discuss and conclude how much battery the apps consumed.

To summarise this chapter, we believe that our choice of methods has thoroughly answered the research questions, for the most part. The methods providing empirical data have been presented in a reliable way but that also leaves room for future research. However, it is worth noting that Huawei and Samsung responded differently to the performance tests, this leaves a question should the study have included more smartphones, and would this have a significant impact on the result? Also, would a rearrangement of the different test cases leave different results? These are good questions to ask and would need a change in the method to be able to be answered.

5.2 Discussion of findings

The purpose of this essay was *to investigate React Native's contra Kotlin's impact on smartphone performance through multiple app functionalities in a test app, thus exploring functionality suitability for native and hybrid* with the research questions:

1. What is the difference between React Native and Kotlin's impact on smartphone performance when implementing functionalities in an app?

   And

2. Is there a breaking point of app implementation where React Native's impact on app performance is considerably worse compared to Kotlin? And if so, with what functionality and why?

Each research question will be now be accounted for under the next-coming headlines.

5.2.1 Research question 1

The difference between React Native and Kotlin's impact on performance has been like previous investigations on React Native and native. That React Native has an overall worse performance than native, as for CPU power. To add to that finding does React Native has worse performance than Kotlin in 27 out of 36 cases, 75% in total of varying degrees in this study. This is across all CPU measurements and both smartphones. The worst difference for both React Native and Kotlin was also recorded and can be seen in Table 19 below. It can be observed that React Native has much larger differences in percentage points compared to Kotlin. React Native compared the worst in the top-level user processes and Kotlin compared the worst in the kernel-level system processes, which also indicate on which level the tools mostly operate.

App	Smartphone	Worst difference	Test case	CPU process
React Native	Honor 8	14,27 percent points	5	User
React Native	Galaxy S7	102,37 percent points	6	User
Kotlin	Honor 8	0,87 percent points	6	System
Kotlin	Galaxy S7	32,25 percent points	1	System

Table 19. Showcase of the worst recorded difference for both React Native and Kotlin.

This essay can also add that React Native shows an overall worse performance that correlates with the implementation of functionality. This means that we have found that the more functionality that is added to the app; the worse will React Native's impact be on the smartphone's CPU compared to Kotlin. Even though some small deviations have occurred in this statement. This can also be seen in table 19 as React Native's worst difference is in test cases 5 and 6. Kotlin does not show the same behaviour for increased implemented functionality and has much more anomalies such as decreases in CPU power in latter test cases, which we have interpreted as Kotlin being in some cases unaffected by the increased number of functionalities.
For memory usage, a certain statement cannot be made in terms of a difference in performance. For Huawei Honor 8 was there not a big difference between React

Native and Kotlin in memory usage, they were about the same. However, on Samsung Galaxy S7 React Native utilized more memory than Kotlin on all test cases. Unlike the CPU measurements where both smartphone's data were mostly similar, does the RAM data contradict each other entirely. This results in an unsureness for memory usage between React Native and Kotlin. More tests on smartphones could result in more confident answers in memory usage.

The original idea behind measuring battery was to record milliampere-hours to calculate battery drainage. This did however prove to be difficult since the smartphone was charging when debugging and ADB did not account for this. Instead, a voltage was recorded which is not a measurement of battery drainage and instead acts as a battery health indicator.

The summarized conclusion for the first research question is that React Native has worse performance in terms of CPU than Kotlin, both overall and in response to implementation of functionality. Memory usage is unclear and further testing on more smartphones would be needed.

5.2.2 Research question 2, part one

The second research question investigates where React Native has a worse development in performance compared to Kotlin between test cases, to find the functionality that has the worst development, or in other words; a breaking point. React Native had worse development on CPU load in 19 out of 30 cases, as described in 4.2.4 Further analysis. This means that across both smartphones and all CPU processes, React Native handles individual implemented functionality worse than Kotlin ~63% of the time in our measurements. The biggest difference out of these 19 cases is considered a breaking point if it occurs between the same test cases in at least two of the three CPU process measurements: app, user, and system. Since it then also will be in majority. With this definition could one definitive breaking point be found for Samsung Galaxy S7 where the difference between test case 1 and 2 was 53.03 percent points for React Native compared to Kotlin. For Huawei Honor 8 could not a definitive breaking point be found.

From this can it be speculated that the npm packages that are used for implementing functionalities in React Native, are in majority across the measurements showing a worse performance than the built-in libraries of Kotlin and native development. Since a breaking point could also only be bound for one of the smartphones, the smartphones also play a key part. Part two of this heading will cover it further.

5.2.3 Research question 2, part two

Through calculating the difference between each test case for Kotlin and React Native and finding where the biggest difference is most often, can it be concluded that this occurs most often in test case 2, which involves the GPS functionality. As stated in the method, will the further investigation of this test case be of its implementation, package, and area of use.

Accessing GPS functionality in a smartphone means accessing specific hardware, which hybrid apps will have a harder time doing since they are to a degree limited

in their native shell. The specific hardware is in this case the receiver in the smartphone that communicates with satellites through radio waves to triangulate the smartphones position. Native apps will have direct access to all the smartphone's hardware since they are written in the language specific to the smartphone's platform.

The GPS functionality was implemented with a simple npm package in React Native, which would then fetch the current coordinates of the smartphone. The implementation for Kotlin was also simple but instead with native's built-in functionality for GPS. The reason behind why this change has taken place can be broken down into two things; React Native and the npm package used. The functionality can be implemented with a different npm package and from that change have a different result, for better or worse. This is something that this study cannot make any further investigation into since it would require an entirely new method for the new additions to the experiments. The implementation for Kotlin would be the same to a varying degree depending on the quantity of functionality since native has a class for accessing GPS functionality. The conclusion to React Native having the most increases in CPU load in test case 2 compared to Kotlin resides in that React Native still have more of a hard time accessing specific hardware (than any other functionality) in a smartphone, such as GPS.

5.3 Conclusions

While the overall results indicate that React Native performs worse than Kotlin it is worth noting that these differences are not big other than for the CPU performance. The tests also indicate that differences are depending on which smartphone you decide to run it on, which is worth considering. Though React Native may have a worse overall CPU performance, which was 75% of the cases, individual implementations of functionality are not as high at ~63%.

This leads to the conclusion that React Native will have worse overall performance than native and that individual functionality implementations do not differ that much from native. However, it is worth remarking that React Native is more prone to a negative response in performance when implementing functionality compared to Kotlin (in our measurements), nonetheless. This would emphasize the choice of React Native when first starting development, but as the app that is being developed grows one may consider switching to native. Because at some point due to React Native's worse exponential growth in CPU load relative to native (as shown in this study), it may lead to larger implications.

It was also shown that test case 2 involving the GPS functionality was where React Native experienced the most increases in CPU load, suggesting that it is at least one of the areas where hybrid struggles the most. If this is sufficient information to conclude how to choose between one over the other is entirely subjective and must be taken into consultation with over development issues.

5.3.1 Future research

A future endeavor to take on is newer, different, and more smartphones to take into the equation when performing tests. Different smartphones could shed some light on the memory usage between React Native and Kotlin as the smartphones used in this essay did not give enough data to give an accurate conclusion on that issue. Different or more functionalities are also something that might be interesting to investigate further. The same goes for different usages of functionality that could include more in-depth integration with the testing app.

6 References

[1] C. Griffith, ”Ionic Framework,” 2020. [Online]. Available: https://ionicframework.com/resources/articles/what-is-hybrid-app-development. [Använd Oktober 2020].

[2] R. Budiu, ”Mobile: Native Apps, Web Apps, and Hybrid Apps,” *Nielsen Norman Group,* September 2013.

[3] S. Richard och P. LePage, ”web.dev,” [Online]. Available: https://web.dev/what-are-pwas/.

[4] Kernel Stable, ”Native, Web & Hybrid: Let's Talk About Mobile Apps,” [Online]. Available: https://stablekernel.com/article/native-web-hybrid-lets-talk-about-mobile-apps/. [Använd 02 02 2021].

[5] J. Stangarone, ”Mrc Productivity,” [Online]. Available: https://www.mrc-productivity.com/blog/2019/10/the-mobile-app-comparison-chart-hybrid-vs-native-vs-mobile-web/.

[6] R. Varshneya, ”Entrepreneur,” [Online]. Available: https://www.entrepreneur.com/article/231145.

[7] W. Danielsson, ”React Native application development: A comparison between native Android and React Native,” Linköpings Universitet, Linköping, 2016.

[8] O. Axelsson och F. Carlström, ”Evaluation Targeting React Native in Comparison to Native Mobile Development,” Lund University, 2016.

[9] O. Lifh och P. Lidholm, ”Recreating a Native Application in React Native: Feasibility of Using React Native With Bluetooth & Background Processing,” Blekinge Tekniska Högskola, Blekinge, 2018.

[10] ”React Native versions,” [Online]. Available: https://reactnative.dev/versions. [Använd September 2020].

[11] ”Kotlin Realeses,” [Online]. Available: https://kotlinlang.org/releases.html. [Använd September 2020].

[12] ”Kotlin,” [Online]. Available: https://kotlinlang.org/. [Använd October 2020].

[13] ”React Native,” [Online]. Available: https://reactnative.dev/. [Använd October 2020].

[14] ”Octoverse,” September 2020. [Online]. Available: https://octoverse.github.com/#top-languages.

[15] A. Ravichandran, ”React Native or Flutter - What Should I Pick To Build My Mobile App?”.

[16] ”Android Debug Bridge,” Android, [Online]. Available: https://developer.android.com/studio/command-line/adb. [Använd October 2020].

[17] B. Davidsson och R. Patel, i *Forskningsmetodikens grunder : att planera, genomföra och rapportera en undersökning*, Studentlitteratur, 2011, pp. 49-52.

[18] S. Sukamolson, ”Fundamentals of quantitative research.,” Language Institute Chulalongkorn University, 2007.

[19] K. Jones, ”Visual Capitalist,” [Online]. Available: https://www.visualcapitalist.com/ranked-most-downloaded-apps/. [Använd 30 September 2020].

[20] M. Presa Käld och O. Svensson, ”GitHub,” [Online]. Available: https://github.com/TETP16ReactKotlinOSMPK.

[21] ”Developer Android, CameraX,” Google, [Online]. Available: https://developer.android.com/training/camerax.

[22] ”React Native Camera,” Github, [Online]. Available: https://github.com/react-native-camera/react-native-camera. [Använd October 2020].

[23] ”React-Native Get Location,” NPM, [Online]. Available: https://www.npmjs.com/package/react-native-get-location. [Använd October 2020].

[24] ”Volley,” Android, [Online]. Available: https://developer.android.com/training/volley. [Använd October 2020].

[25] ”Firebase,” Google. [Online]. [Använd October 2020].

[26] ”React Native Firebase,” [Online]. Available: https://rnfirebase.io/. [Använd November 2020].

[27] ”react native sound,” [Online]. Available: https://www.npmjs.com/package/react-native-sound. [Använd December 2020].

[28] ”bluetooth classic,” [Online]. Available: https://www.npmjs.com/package/react-native-bluetooth-classic. [Använd December 2020].

[29] ”Serialio,” [Online]. Available: https://www.serialio.com/faqs/whats-difference-between-bluetooth-le-and-bluetooth-spp-ble-vs-spp. [Använd 21 January 2021].

7 Appendices

Appendix 1 Text defining the appendix
Appendix 2 etc.

Appendix 1 Script used for test case 1-4 in the experiments.

```
adb shell input tap 550 1000;

for /l %%x in (1, 1, 12) do (
   echo %%x
   TIMEOUT /T 5
   adb shell input tap 550 1400;
   adb shell dumpsys meminfo --oom;
   adb shell top -m 6 -n 1;
   adb shell dumpsys battery;
)

echo "Done"
exit
```

Appendix 2 Script used for test case 5-6 in the experiments.

```
adb shell input tap 550 1000;

for /l %%x in (1, 1, 40) do (
   echo %%x
   TIMEOUT /T 15
   adb shell input tap 550 1400;
   adb shell dumpsys meminfo --oom;
   adb shell top -m 6 -n 1;
   adb shell dumpsys battery;
)

echo "Done"
exit
```

Appendix 3 All measurements for Huawei Honor 8 from test case 1 for Kotlin.

Test case 1						
Honor 8	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	23	13	2	2015	0,5330687831	4,381
2	6	9	3	2057	0,5441798942	4,381
3	6	12	3	2060	0,544973545	4,381
4	2	17	2	2120	0,5608465608	4,29
5	28	6	3	2121	0,5611111111	4,284
6	12	9	1	2147	0,567989418	4,284
7	5	16	5	2160	0,5714285714	4,284
8	11	17	2	2188	0,5788359788	4,284
9	8	16	2	2185	0,578042328	4,266
10	25	8	2	2175	0,5753968254	4,266
11	10	18	1	2151	0,569047619	4,266
12	6	9	1	2173	0,5748677249	4,266
	Average	Average	Average	Average	Average	Average
	11,83333333	12,5	2,25	2129,333333	0,5633156966	4,30275

Appendix 4 All measurements for Huawei Honor 8 from test case 2 for Kotlin.

Test case 2						
Honor 8	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	8	14	1	1997	0,5283068783	4,229
2	24	24	3	2023	0,5351851852	4,293
3	11	8	2	2045	0,541005291	4,293
4	14	8	1	2078	0,5497354497	4,293
5	12	12	2	2107	0,5574074074	4,293
6	13	8	2	2125	0,5621693122	4,166
7	13	10	3	2176	0,5756613757	4,166
8	29	24	2	2180	0,5767195767	4,166
9	9	12	1	2184	0,5777777778	4,166
10	18	12	1	2210	0,5846560847	4,166
11	11	11	2	2232	0,5904761905	4,158
12	5	12	3	2201	0,5822751323	4,254
	Average	Average	Average	Average	Average	Average
	13,91666667	12,91666667	1,916666667	2129,833333	0,5634479718	4,22025

Appendix 5 All measurements for Huawei Honor 8 from test case 3 for Kotlin.

Test case 3						
Honor 8	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	15	10	1	2046	0,5412698413	4,371
2	14	17	2	2041	0,5399470899	4,371
3	9	15	2	2055	0,5436507937	4,371
4	10	7	3	2105	0,5568783069	4,152
5	19	19	1	2110	0,5582010582	4,265
6	6	18	2	2128	0,562962963	4,265
7	8	15	2	2161	0,5716931217	4,265
8	5	8	2	2178	0,5761904762	4,265
9	12	14	1	2180	0,5767195767	4,166
10	12	18	3	2175	0,5753968254	4,166
11	11	12	1	2183	0,5775132275	4,166
12	17	11	2	2185	0,578042328	4,166

	Average	Average	Average	Average	Average	Average
	11.5	13,66666667	1,833333333	2128,916667	0,5632054674	4,249083333

Appendix 6 All measurements for Huawei Honor 8 from test 4 for Kotlin.

Test case 4						
Honor 8	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	8	8	2	1874	0,4957671958	4,376
2	9	6	3	1898	0,5021164021	4,376
3	6	9	3	1895	0,5013227513	4,376
4	12	12	1	1900	0,5026455026	4,376
5	17	8	1	1894	0,5010582011	4,207
6	18	9	1	1897	0,5018518519	4,27
7	10	13	2	1951	0,5161375661	4,27
8	9	6	3	1914	0,5063492063	4,27
9	6	10	6	1909	0,505026455	4,27
10	5	19	1	1934	0,5116402116	4,204
11	3	17	3	1910	0,5052910053	4,204
12	12	9	3	1904	0,5037037037	4,204
	Average	Average	Average	Average	Average	Average
	9,583333333	10,5	2,416666667	1906,666667	0,5044091711	4,283583333

Appendix 7 All measurements for Huawei Honor 8 from test case 5 for Kotlin.

Test case 5						
Honor 8	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	3	12	3	1942	0,5137566138	4,291
2	15	9	1	1965	0,5198412698	4,219
3	3	13	1	1965	0,5198412698	4,219
4	13	8	2	1919	0,5076719577	4,196
5	11	8	1	1966	0,5201058201	4,196
6	9	7	2	1922	0,5084656085	4,2
7	6	15	1	1969	0,5208994709	4,2
8	10	10	2	1943	0,514021164	4,195
9	13	11	2	1941	0,5134920635	4,191
10	13	9	1	1934	0,5116402116	4,191
11	1	12	2	1934	0,5116402116	4,186
12	10	16	2	1942	0,5137566138	4,186
13	20	6	1	1924	0,508994709	4,274
14	9	11	1	1925	0,5092592593	4,274
15	9	11	4	1927	0,5097883598	4,204
16	16	10	2	1929	0,5103174603	4,125
17	14	12	2	1940	0,5132275132	4,125
18	9	7	2	1965	0,5198412698	4,176
19	11	13	2	1935	0,5119047619	4,2
20	11	8	2	1934	0,5116402116	4,185
21	6	13	2	1935	0,5119047619	4,185
22	12	12	3	1957	0,5177248677	4,219
23	9	15	3	1966	0,5201058201	4,213
24	22	5	1	1967	0,5203703704	4,213
25	6	15	2	1957	0,5177248677	4,211
26	8	8	2	1959	0,5182539683	4,211
27	9	9	2	1960	0,5185185185	4,204
28	11	15	2	1988	0,5259259259	4,19
29	8	15	1	2017	0,5335978836	4,19
30	8	16	2	1990	0,5264550265	4,21

31	8	13	1	1971	0,5214285714	4,201
32	6	13	2	1962	0,519047619	4,198
33	12	10	2	1963	0,5193121693	4,198
34	17	4	2	1963	0,5193121693	4,193
35	8	20	2	2016	0,5333333333	4,169
36	10	10	2	1972	0,5216931217	4,169
37	13	8	2	1993	0,5272486772	4,215
38	11	7	2	1979	0,5235449735	4,215
39	8	15	2	1965	0,5198412698	4,223
40	8	14	4	2010	0,5317460317	4,223
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	10,15	11,125	1,925	1957,775	0,5179298942	4,202075

Appendix 8 All measurements for Huawei Honor 8 from test case 6 for Kotlin.

Test case 6						
Honor 8	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	21	18	10	1924	0,508994709	4,162
2	25	21	11	1931	0,5108465608	4,162
3	17	17	8	1925	0,5092592593	4,162
4	39	17	12	1923	0,5087301587	4,162
5	16	19	16	1922	0,5084656085	4.17
6	28	15	21	1923	0,5087301587	4.17
7	33	20	10	1924	0,508994709	4,122
8	23	13	10	1937	0,5124338624	4,122
9	38	22	8	1933	0,5113756614	4,168
10	21	25	9	1934	0,5116402116	4,168
11	32	18	10	1936	0,5121693122	4,169
12	25	11	9	1935	0,5119047619	4,173
13	34	16	10	1933	0,5113756614	4,173
14	29	14	9	1936	0,5121693122	4,146
15	21	19	10	1935	0,5119047619	4,146
16	31	23	13	1938	0,5126984127	4,158
17	28	15	15	1938	0,5126984127	4,158
18	27	24	11	1939	0,512962963	4,171
19	31	20	10	1941	0,5134920635	4,22
20	36	18	17	1943	0,514021164	4,22
21	27	24	12	1945	0,5145502646	4,216
22	34	20	14	1950	0,5158730159	4,216
23	36	29	10	1947	0,5150793651	4,216
24	28	20	12	1958	0,517989418	4,207
25	40	17	11	1951	0,5161375661	4,189
26	23	21	12	1950	0,5158730159	4,189
27	29	15	11	1952	0,5164021164	4,195
28	33	13	10	1961	0,5187830688	4,195
29	32	18	12	1944	0,5142857143	4,201
30	26	22	19	1944	0,5142857143	4,158
31	40	16	12	1948	0,5153439153	4,158
32	29	19	18	1948	0,5153439153	4,188
33	21	21	12	1948	0,5153439153	4,188
34	25	23	14	1944	0,5142857143	4,218
35	31	22	19	1944	0,5142857143	4,227
36	23	21	11	1947	0,5150793651	4,227
37	32	11	10	1948	0,5153439153	4,234
38	29	19	14	1953	0,5166666667	4,229
39	28	18	11	1953	0,5166666667	4,229
40	25	17	13	1955	0,5171957672	4,228
	Average	**Average**	**Average**	**Average**	**Average**	**Average**

	28,65	18,775	12,15	1941	0,5134920635	4,18525

Appendix 9 All measurements for Huawei Honor 8 from test case 1 for React Native.

Test case 1						
Honor 8	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	24	7	7	2027	0,5362433862	4,36
2	22	8	14	2027	0,5362433862	4,214
3	15	6	12	2026	0,535978836	4,214
4	6	18	12	2022	0,5349206349	4,214
5	17	11	14	2029	0,5367724868	4,214
6	16	11	11	2026	0,535978836	4,214
7	22	14	5	2020	0,5343915344	4,233
8	16	8	16	2025	0,5357142857	4,233
9	25	11	1	2027	0,5362433862	4,233
10	17	20	8	2029	0,5367724868	4,233
11	26	6	10	2024	0,5354497354	4,233
12	14	25	11	2027	0,5362433862	4,211
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	18,33333333	12,08333333	10,08333333	2025,75	0,5359126984	4,233833333

Appendix 10 All measurements for Huawei Honor 8 from test case 2 for React Native.

Test case 2						
Honor 8	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	9	6	1	1964	0,5195767196	4,199
2	3	10	3	1958	0,517989418	4,221
3	8	11	20	1963	0,5193121693	4,221
4	20	15	15	1960	0,5185185185	4,221
5	11	13	11	2020	0,5343915344	4,221
6	18	16	12	1968	0,5206349206	4,221
7	22	18	10	1973	0,521957672	4,195
8	6	11	20	1968	0,5206349206	4,195
9	22	11	13	1965	0,5198412698	4,195
10	6	13	13	1967	0,5203703704	4,195
11	20	11	15	1974	0,5222222222	4,195
12	23	13	13	1974	0,5222222222	4,178
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	14	12,33333333	12,16666667	1971,166667	0,5214726631	4,20475

Appendix 11 All measurements for Huawei Honor 8 from test case 3 for React Native.

Test case 3						
Honor 8	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	21	16	10	2056	0,5439153439	4,211
2	19	21	9	2072	0,5481481481	4,211
3	18	16	14	2065	0,5462962963	4,21
4	28	13	13	2057	0,5441798942	4,193
5	12	16	16	2059	0,5447089947	4,193

6	16	16	12	2070	0,5476190476	4,193
7	18	18	12	2063	0,5457671958	4,193
8	27	8	6	2072	0,5481481481	4,189
9	18	11	11	2068	0,5470899471	4,189
10	15	15	15	2073	0,5484126984	4,189
11	14	14	12	2066	0,5465608466	4,189
12	22	16	15	2071	0,5478835979	4,189
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	19	15	12,08333333	2066	0,5465608466	4,19575

Appendix 12 All measurements for Huawei Honor 8 from test case 4 for React Native.

Test case 4						
Honor 8	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	20	20	16	2162	0,571957672	4,273
2	21	15	10	2161	0,5716931217	4,273
3	19	10	13	2126	0,5624338624	4,273
4	13	11	13	2014	0,5328042328	4,273
5	11	14	14	2014	0,5328042328	4,067
6	19	17	20	2049	0,5420634921	4,178
7	24	13	15	2038	0,5391534392	4,178
8	19	17	14	2032	0,5375661376	4,179
9	30	17	12	2048	0,5417989418	4,166
10	28	15	16	2040	0,5396825397	4,166
11	30	12	15	2055	0,5436507937	4,166
12	22	14	21	2049	0,5420634921	4,166
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	21,33333333	14,58333333	14,91666667	2065,666667	0,5464726631	4,1965

Appendix 13 All measurements for Huawei Honor 8 from test case 5 for React Native.

Test case 5						
Honor 8	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	25	16	9	1985	0,5251322751	4,116
2	21	11	11	1987	0,5256613757	4,116
3	20	13	13	1989	0,5261904762	4,116
4	22	20	10	1983	0,5246031746	4,057
5	18	16	12	2012	0,5322751323	4,098
6	20	22	4	2007	0,530952381	4,094
7	26	19	14	2006	0,5306878307	4,094
8	23	17	14	2014	0,5328042328	4,108
9	23	21	13	2011	0,532010582	4,109
10	28	15	15	2020	0,5343915344	4,105
11	28	15	13	2011	0,532010582	4,068
12	29	12	14	2022	0,5349206349	4,068
13	29	19	13	2015	0,5330687831	4
14	32	15	11	2023	0,5351851852	4,122
15	33	11	17	2005	0,5304232804	4,107
16	26	16	13	2021	0,5346560847	4,096
17	34	17	17	2027	0,5362433862	4,096
18	24	22	17	2024	0,5354497354	4,114
19	25	22	20	2015	0,5330687831	4,114
20	25	16	22	2019	0,5341269841	4,1
21	25	19	8	2017	0,5335978836	4,077
22	27	17	12	2029	0,5367724868	4,077

23	26	20	14	2031	0,5373015873	4,106
24	28	20	13	2035	0,5383597884	4,089
25	23	22	15	2029	0,5367724868	4,102
26	22	21	12	2016	0,5333333333	4,102
27	17	25	10	2016	0,5333333333	4,102
28	26	18	19	2013	0,5325396825	4,078
29	31	16	14	2021	0,5346560847	4,085
30	23	27	10	2026	0,535978836	4,085
31	31	16	15	2024	0,5354497354	4,072
32	25	20	15	2019	0,5341269841	4,072
33	28	18	10	2021	0,5346560847	4,039
34	30	18	13	2023	0,5351851852	4,093
35	19	19	19	2026	0,535978836	4,093
36	25	16	13	2020	0,5343915344	4,109
37	22	20	16	2013	0,5325396825	4,109
38	27	15	11	2016	0,5333333333	4,017
39	26	20	15	2012	0,5322751323	4,1
40	25	21	14	2023	0,5351851852	4,1
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	25,425	18,075	13,5	2015,65	0,5332407407	4,093175

Appendix 14 All measurements for Huawei Honor 8 from test case 6 for React Native.

Test case 6						
Honor 8	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	32	23	20	2009	0,5314814815	4,136
2	32	16	19	2038	0,5391534392	4,136
3	42	12	18	2036	0,5386243386	4,094
4	25	20	23	2024	0,5354497354	4,094
5	28	20	14	2038	0,5391534392	4,081
6	28	18	18	2039	0,5394179894	4,081
7	31	14	22	2033	0,5378306878	4,073
8	32	11	14	2034	0,5380952381	4,073
9	27	15	18	2035	0,5383597884	4,034
10	16	23	25	2032	0,5375661376	4,034
11	39	13	23	2035	0,5383597884	4,025
12	36	19	13	2033	0,5378306878	4,039
13	42	14	21	2051	0,5425925926	4,058
14	31	15	25	2065	0,5462962963	4,053
15	36	17	19	2054	0,5433862434	4,053
16	41	20	18	2054	0,5433862434	4,035
17	39	19	19	2050	0,5423280423	4,04
18	27	25	27	2059	0,5447089947	4,088
19	35	18	25	2065	0,5462962963	4,1
20	39	17	25	2064	0,546031746	4,1
21	35	17	23	2042	0,5402116402	4,082
22	46	14	19	2082	0,5507936508	4,082
23	31	19	21	2055	0,5436507937	4,062
24	37	20	22	2050	0,5423280423	4,079
25	37	16	21	2066	0,5465608466	4,079
26	42	19	23	2066	0,5465608466	4,09
27	32	18	25	2061	0,5452380952	4,084
28	25	22	24	2065	0,5462962963	4,084
29	34	23	20	2050	0,5423280423	4,066
30	36	20	24	2057	0,5441798942	4,062
31	34	12	27	2059	0,5447089947	4,062
32	33	22	24	2050	0,5423280423	4,045
33	42	22	23	2061	0,5452380952	4,045
34	38	20	20	2065	0,5462962963	4,074
35	42	11	21	2070	0,5476190476	4,086
36	40	17	22	2057	0,5441798942	4,086

37	32	22	22	2059	0,5447089947	4,094
38	36	21	23	2066	0,5465608466	4,094
39	37	19	25	2050	0,5423280423	4,09
40	42	13	21	2062	0,5455026455	4,082
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	34,725	17,9	21,4	2051,025	0,5425992063	4,073875

Appendix 15 All measurements for Samsung Galaxy S7 from test case 1 for Kotlin.

Test case 1						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	76	124	48.4	1386	0,382661513	4.26
2	91	91	62.5	1399	0,3862506902	4.192
3	132	29	74.1	1396	0,3854224186	4.229
4	77	84	74.1	1397	0,3856985091	4.229
5	113	110	70.9	1398	0,3859745997	4.225
6	97	100	77.4	1399	0.3862506902	4.228
7	63	94	59.3	1400	0,3865267808	4.226
8	87	100	70.9	1401	0.3868028713	4.226
9	113	110	87	1402	0.3870789619	4.227
10	97	106	74.1	1402	0.3870789619	4.223
11	61	106	74.1	1404	0.387631143	4.223
12	84	74	70.9	1405	0.3879072336	4.193
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	90.91666667	94	70.30833333	1399.083333	0.3862736978	4.223416667

Appendix 16 All measurements for Samsung Galaxy S7 from test case 2 for Kotlin.

Test case 2						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	180	167	40	1428	0.3942573164	4.293
2	94	81	65.6	1423	0.3928768636	4.293
3	100	103	59.3	1411	0.3895637769	4.293
4	97	103	59.3	1414	0.3903920486	4.293
5	97	117	63.3	1415	0.3906681391	4.293
6	97	65	63.3	1415	0.3906681391	4.283
7	97	65	61.2	1417	0.3912203203	4.283
8	116	77	70.9	1426	0.3937051353	4.283
9	116	119	61.2	1427	0.3939812258	4.283
10	94	70	54.5	1410	0.3892876864	4.283
11	103	106	56.2	1404	0.387631143	4.248
12	113	103	61.2	1404	0.387631143	4.288
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	108.6666667	98	59.66666667	1416.166667	0.3909902448	4.284666667

Appendix 17 All measurements for Samsung Galaxy S7 from test case 3 for Kotlin

Test case 3						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	206	131	94.4	1415	0.3906681391	4.281
2	202	89	84	1429	0.394533407	4.281
3	138	84	37.5	1426	0.3937051353	4.281
4	124	129	65.7	1428	0.3942573164	4.281

5	164	145	39.3	1430	0.3948094975	4.279
6	127	112	39.3	1431	0.3950855881	4.279
7	97	94	54.5	1433	0.3956377692	4.279
8	91	94	62.5	1434	0.3959138597	4.279
9	109	94	62.5	1415	0.3906681391	4.279
10	68	100	58	1417	0.3912203203	4.247
11	61	74	61.2	1418	0.3914964108	4.276
12	106	106	58	1419	0.3917725014	4.276
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	124.4166667	104.3333333	59.74166667	1424.583333	0.393314007	4.2765

Appendix 18 All measurements for Samsung Galaxy S7 from test case 4 for Kotlin.

Test case 4						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	100	97	96.6	1565	0.4320817228	4.3
2	81	106	56.2	1462	0.4036443954	4.297
3	65	68	35.4	1478	0.4080618443	4.297
4	81	119	58	1465	0.404472667	4.231
5	89	75	78.5	1480	0.4086140254	4.286
6	90	74	45.1	1468	0.4053009387	4.286
7	191	125	62.5	1484	0.4097183876	4.25
8	81	53	43.7	1472	0.4064053009	4.279
9	81	97	43.7	1489	0.4110988404	4.277
10	100	90	61.2	1477	0.4077857537	4.277
11	81	71	51.6	1492	0.4119271121	4.27
12	139	77	64.5	1479	0.4083379348	4.27
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	98.25	87.66666667	58.08333333	1484.25	0.4097874103	4.276666667

Appendix 19 All measurements for Samsung Galaxy S7 from test case 5 for Kotlin.

Test case 5						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	103	71	141	1513	0.4177250138	4.333
2	91	91	46.8	1515	0.4182771949	4.332
3	81	97	48.3	1514	0.4180011044	4.332
4	110	84	51.6	1512	0.4174489232	4.335
5	103	84	48.3	1500	0.4141358366	4.336
6	103	81	45.1	1501	0.4144119271	4.332
7	103	81	48.3	1497	0.4133075649	4.331
8	106	77	45.1	1500	0.4141358366	4.333
9	94	74	45.1	1502	0.4146880177	4.329
10	129	48	48.3	1504	0.4152401988	4.329
11	88	97	46.8	1504	0.4152401988	4.331
12	123	71	48.3	1507	0.4160684705	4.332
13	113	100	51.6	1512	0.4174489232	4.335
14	97	68	48.3	1515	0.4182771949	4.333
15	106	65	45.1	1496	0.4130314743	4.333
16	110	74	58	1499	0.413859746	4.333
17	88	84	65.5	1501	0.4144119271	4.333
18	90	103	45.1	1501	0.4144119271	4.324
19	110	77	48.3	1513	0.4177250138	4.331
20	100	84	48.3	1517	0.418829376	4.335

21	84	74	38.7	1500	0.4141358366	4.331
22	110	97	51.6	1498	0.4135836554	4.333
23	87	84	48.3	1498	0.4135836554	4.333
24	106	71	48.3	1503	0.4149641082	4.329
25	88	97	46.8	1508	0.416344561	4.331
26	120	87	46.6	1669	0.4607951408	4.332
27	100	77	45.1	1515	0.4182771949	4.333
28	52	100	35.4	1508	0.416344561	4.33
29	94	68	41.9	1511	0.4171728327	4.334
30	93	83	46.6	1500	0.4141358366	4.335
31	89	89	35.7	1499	0.413859746	4.315
32	113	100	45.1	1501	0.4144119271	4.334
33	94	87	45.1	1506	0.4157923799	4.333
34	125	82	53.5	1511	0.4171728327	4.332
35	77	52	74.1	1491	0.4116510215	4.333
36	100	84	45.1	1492	0.4119271121	4.338
37	100	125	85.7	1493	0.4122032027	4.341
38	97	106	45.1	1497	0.4133075649	4.341
39	100	77	51.6	1502	0.4146880177	4.342
40	93	100	53.5	1507	0.4160684705	4.342
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	99.25	83.775	51.425	1508.3	0.4164273882	4.33285

Appendix 20 All measurements for Samsung Galaxy S7 for test case 6 for Kotlin

Test case 6						
Gaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	153	138	150	1505	0.4155162893	4.29
2	176	136	66.6	1523	0.4204859194	4.301
3	169	125	65	1544	0.4262838211	4.291
4	186	125	65.6	1543	0.4260077305	4.286
5	186	125	65.6	1573	0.4342904473	4.288
6	186	121	93.1	1533	0.423246825	4.287
7	141	117	72.4	1530	0.4224185533	4.282
8	171	119	65.5	1534	0.4235229155	4.28
9	161	119	61.2	1539	0.4249033683	4.278
10	177	106	74.1	1542	0.42573164	4.271
11	169	116	59.3	1546	0.4268360022	4.276
12	172	134	62.5	1526	0.4213141911	4.276
13	168	126	61.2	1530	0.4224185533	4.273
14	194	129	61.2	1535	0.4237990061	4.271
15	181	106	64.5	1550	0.4279403644	4.27
16	178	128	75	1563	0.4315295417	4.269
17	156	116	62.5	1565	0.4320817228	4.264
18	159	128	71.8	1546	0.4268360022	4.266
19	193	143	78.5	1569	0.433186085	4.264
20	188	97	65.6	1565	0.4320817228	4.263
21	168	116	64.5	1562	0.4312534511	4.259
22	159	119	71.8	1566	0.4323578134	4.256
23	177	106	65.5	1569	0.433186085	4.261
24	163	91	68.7	1550	0.4279403644	4.259
25	139	129	74.1	1552	0.4284925456	4.262
26	148	126	61.2	1555	0.4293208172	4.261
27	124	112	69.6	1558	0.4301490889	4.261
28	181	106	61.2	1562	0.4312534511	4.256
29	188	116	65.6	1569	0.433186085	4.261
30	200	116	65.6	1551	0.428216455	4.261
31	142	103	61.2	1552	0.4284925456	4.256
32	158	126	61.2	1557	0.4298729983	4.249
33	184	129	61.2	1562	0.4312534511	4.251

34	161	90	61.2	1565	0.4320817228	4.246
35	172	128	65.6	1568	0.4329099945	4.25
36	229	118	71.4	1622	0.4478188846	4.244
37	190	126	58	1555	0.4293208172	4.245
38	168	116	61.2	1560	0.43070127	4.246
39	148	158	80.6	1571	0.4337382662	4.246
40	153	119	62.5	1559	0.4304251795	4.246
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	170.4	120.1	68.8275	1553.15	0.4288100497	4.26555

Appendix 21 All measurements for Samsung Galaxy S7 for test case 1 for React Native.

Test case 1						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	58	45	67.7	1600	0.4417448923	4.344
2	42	48	41.9	1609	0.4442297073	4.306
3	88	94	37.5	1600	0.4417448923	4.305
4	70	43	50	1621	0.447542794	4.295
5	62	86	48.2	1617	0.4464384318	4.295
6	76	59	58.6	1618	0.4467145224	4.257
7	62	59	51.7	1621	0.447542794	4.287
8	69	62	44.8	1615	0.4458862507	4.288
9	34	66	31	1624	0.4483710657	4.297
10	41	59	48.2	1618	0.4467145224	4.297
11	57	63	46.6	1619	0.4469906129	4.27
12	63	57	43.3	1618	0.4467145224	4.291
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	60.16666667	61.75	47.45833333	1615	0.4458862507	4.294333333

Appendix 22 All measurements for Samsung Galaxy S7 for test case 2 for React Native.

Test case 2						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	115	145	75.7	1724	0.4759801215	4.316
2	128	131	56.2	1732	0.4781888459	4.28
3	210	65	338	1726	0.4765323026	4.278
4	81	74	29	1728	0.4770844837	4.251
5	210	139	29	1729	0.4773605743	4.251
6	80	160	33.3	1721	0.4751518498	4.243
7	244	159	28.1	1725	0.476256212	4.269
8	155	145	38.7	1720	0.4748757592	4.269
9	57	127	40	1734	0.4787410271	4.269
10	142	84	29	1738	0.4798453893	4.236
11	78	84	28.1	1736	0.4792932082	4.246
12	70	113	30	1732	0.4781888459	4.265
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	130.8333333	118.8333333	62.925	1728.75	0.4772915516	4.264416667

Appendix 23 All measurements for Samsung Galaxy S7 for test case 3 for React Native.

Test case 3						
Galaxy S7	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	206	182	45.4	1833	0.5060739923	4.303
2	179	144	41.1	1815	0.5011043622	4.278
3	285	179	39.3	1807	0.4988956378	4.267
4	291	206	128	1807	0.4988956378	4.267
5	161	188	45.4	1826	0.5041413584	4.251
6	234	153	78.1	1828	0.5046935395	4.272
7	84	87	25.8	1814	0.5008282717	4.276
8	70	67	30	1818	0.5019326339	4.26
9	90	143	43.3	1829	0.50496963	4.26
10	194	172	50	1828	0.5046935395	4.247
11	57	73	26.6	1821	0.5027609056	4.265
12	180	153	53.3	1847	0.5099392601	4.265
	Average	Average	Average	Average	Average	Average
	169.25	145.5833333	50.525	1822.75	0.5032440641	4.267583333

Appendix 24 All measurements for Samsung Galaxy S7 for test case 4 for React Native.

Test case 4						
Galaxy S7	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	132	229	38.7	1861	0.5138045279	4.26
2	262	103	54.7	1855	0.5121479845	4.262
3	172	97	33	1857	0.5127001657	4.262
4	203	177	48.3	1856	0.5124240751	4.226
5	245	203	35	1856	0.5124240751	4.257
6	210	187	39	1867	0.5154610712	4.259
7	134	138	21.8	1857	0.5127001657	4.253
8	194	135	48.3	1857	0.5127001657	4.253
9	184	172	34.3	1849	0.5104914412	4.235
10	185	118	35	1864	0.5146327996	4.256
11	190	213	25.8	1866	0.5151849807	4.257
12	203	131	40.6	1866	0.5151849807	4.242
	Average	Average	Average	Average	Average	Average
	192.8333333	158.5833333	37.875	1859.25	0.5133213694	4.251833333

Appendix 24 All measurements for Samsung Galaxy S7 for test case 5 for React Native.

Test case 5						
Galaxy S7	CPU User(%)	CPU System (%)	CPU App (%)	Used RAM (MB)	Used RAM (%)	Battery voltage (V)
1	84	61	38.7	1775	0.4900607399	4.203
2	103	93	33.3	1773	0.4895085588	4.213
3	242	93	33.3	1781	0.4917172833	4.15
4	200	109	348	1785	0.4928216455	4.212
5	113	103	33.3	1779	0.4911651022	4.202
6	97	81	25.8	1775	0.4900607399	4.205
7	207	177	43.3	1773	0.4895085588	2.14
8	177	163	43.3	1780	0.4914411927	4.216
9	103	116	22.5	1787	0.4933738266	4.213
10	170	150	30	1792	0.4947542794	4.202

11	150	110	26.6	1794	0.4953064605	4.203
12	167	153	30	1780	0.4914411927	4.215
13	107	120	26.6	1800	0.4969630039	4
14	97	103	26.6	1799	0.4966869133	4.214
15	147	178	50	1785	0.4928216455	4.211
16	147	193	36.6	1792	0.4947542794	4.207
17	178	97	50	1795	0.4955825511	4.189
18	87	119	16.1	1792	0.4947542794	4.206
19	157	157	43.3	1795	0.4955825511	4.212
20	107	97	30	1792	0.4947542794	4.208
21	133	143	26.6	1806	0.4986195472	4.196
22	113	130	30	1792	0.4947542794	4.208
23	187	129	48.3	1806	0.4986195472	4.196
24	134	159	53.1	1816	0.5013804528	4.207
25	120	87	26.6	1780	0.4914411927	4.202
26	123	133	30	1796	0.4958586416	4.209
27	119	156	25	1800	0.4969630039	4.215
28	180	120	53.3	1799	0.4966869133	4.214
29	210	127	50	1813	0.5005521811	4.207
30	103	94	28.1	1810	0.4997239094	4.209
31	178	172	34.3	1804	0.4980673661	4.187
32	153	153	53.1	1814	0.5008282717	4.2
33	193	153	50	1801	0.4972390944	4.195
34	328	131	50	1819	0.5022087245	4.197
35	172	103	43.7	1806	0.4986195472	4.195
36	233	213	56.6	1805	0.4983434567	4.197
37	237	150	43.3	1802	0.497515185	4.207
38	177	190	53.3	1813	0.5005521811	4.195
39	153	147	56.6	1805	0.4983434567	4.203
40	183	193	53.3	1787	0.4933738266	4.2
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	156.725	133.9	46.3125	1794.95	0.4955687465	4.151875

Appendix 25 All measurements for Samsung Galaxy S7 for test case 5 for React Native.

Test case 6						
Galaxy S7	**CPU User(%)**	**CPU System (%)**	**CPU App (%)**	**Used RAM (MB)**	**Used RAM (%)**	**Battery voltage (V)**
1	219	116	100	1833	0.5060739923	4.284
2	347	240	106	1834	0.5063500828	4.28
3	438	175	106	1840	0.5080066262	4.273
4	277	119	400	1850	0.5107675318	4.264
5	309	182	78.7	1836	0.5069022639	4.259
6	242	145	87	1855	0.5121479845	4.25
7	209	156	78.1	1841	0.5082827167	4.251
8	242	165	74.1	1836	0.5069022639	4.232
9	252	181	83.8	1852	0.5113197129	4.252
10	243	153	86.6	1841	0.5082827167	4.23
11	270	160	86.6	1842	0.5085588073	4.249
12	340	257	106	1837	0.5071783545	4.251
13	253	207	90	1850	0.5107675318	4.241
14	250	183	93.3	1845	0.509387079	4.241
15	369	159	78.1	1840	0.5080066262	4.241
16	273	183	83.3	1850	0.5107675318	4.239
17	294	178	78.1	1843	0.5088348978	4.237
18	297	290	80	1853	0.5115958034	4.234

19	243	203	80	1856	0.5124240751	4.239
20	247	167	83.3	1847	0.5099392601	4.239
21	243	163	83.3	1853	0.5115958034	4.232
22	255	161	80.6	1899	0.5242959691	4.226
23	297	129	83.3	1879	0.5187741579	4.236
24	261	133	75.7	1880	0.5190502485	4.23
25	269	191	78.1	1885	0.5204307013	4.233
26	240	157	90	1903	0.5254003313	4.211
27	300	177	96.6	1903	0.5254003313	4.224
28	275	134	78.1	1913	0.5281612369	4.221
29	257	183	90	1915	0.528713418	4.227
30	247	193	83.3	1884	0.5201546107	4.231
31	263	147	83.3	1914	0.5284373274	4.227
32	248	181	80.6	1916	0.5289895086	4.232
33	267	207	90	1919	0.5298177802	4.227
34	291	138	84.5	1915	0.528713418	4.223
35	220	197	80	1899	0.5242959691	4.225
36	277	167	83.3	1920	0.5300938708	4.225
37	260	177	83.3	1904	0.5256764219	4.231
38	277	180	80	1915	0.528713418	4.23
39	300	130	90	1905	0.5259525124	4.228
40	250	177	80	1923	0.5309221425	4.23
	Average	**Average**	**Average**	**Average**	**Average**	**Average**
	272.775	173.525	93.325	1873.125	0.5171521259	4.238375

Appendix 26 All calculations for performance difference for each test case.

Honor 8	**Test Case**	**Avg. Kotlin CPU APP (%)**	**Avg. React Native CPU APP (%)**	**Difference (pp)**
	1	2.25	10	7.75
	2	1.9	12.1	10.2
	3	1.8	12	10.2
	4	2.4	14.9	12.5
	5	1,925	13.5	11,575
	6	12.15	21.4	9.25
Honor 8	**Test Case**	**Avg. Kotlin CPU USER (%)**	**Avg. React Native CPU USER (%)**	**Difference (pp)**
	1	11.8	18	6.2
	2	13.9	14	0.1
	3	11.5	19	7.5
	4	9.5	21.3	11.8
	5	10.15	24.425	14.275
	6	28.65	34.725	6.075
Honor 8	**Test Case**	**Avg. Kotlin CPU SYSTEM (%)**	**Avg. React Native CPU SYSTEM (%)**	**Difference (pp)**
	1	12.5	12.08	-0.42
	2	12.9	12.33	-0.57
	3	13.6	15	1.4
	4	10.5	14.6	4.1
	5	11.125	18.075	6.95
	6	18.775	17.9	-0.875
Galaxy S7	**Test Case**	**Avg. Kotlin CPU APP (%)**	**Avg. React Native CPU APP (%)**	**Difference (pp)**
	1	70	47.45	-22.55
	2	59.66	62.925	3.265
	3	59.7	50.525	-9.175
	4	58	37.875	-20.125
	5	51.425	46.312	-5.113
	6	68.8275	93.325	24.4975
Galaxy S7	**Test Case**	**Avg. Kotlin CPU USER (%)**	**Avg. React Native CPU USER (%)**	**Difference (pp)**
	1	90.9	60	-30.9
	2	108.667	130	21.333
	3	124.4	169.25	44.85
	4	98.25	192.83	94.58
	5	99.25	156.725	57.475
	6	170.4	272.775	102.375
Galaxy S7	**Test Case**	**Avg. Kotlin CPU SYSTEM (%)**	**Avg. React Native CPU SYSTEM (%)**	**Difference (pp)**
	1	94	61.75	-32.25
	2	98	118.83	20.83
	3	104	145.48	41.48
	4	87.66	158.6	70.94
	5	83.775	133.9	50.125
	6	120.1	173.526	53.426

Appendix 27 All calculations for performance difference between test cases.

Honor 8	**Test cases**	**Kotlin CPU APP (pp)**	**React Native CPU APP (pp)**	**Kotlin/React Native difference (pp)**
	diff 1–2	-0.35	2.1	2.45
	diff 2–3	-0.1	-0.1	0
	diff 3–4	0.6	2.9	2.3
	dif 4–5	-0.475	-1.4	-0.925
	diff 5–6	10.225	7.9	-2.325
Honor 8	**Test cases**	**Kotlin CPU USER (pp)**	**React Native CPU USER (pp)**	**Kotlin/React Native difference (pp)**
	diff 1–2	2.1	-4	-6.1
	diff 2–3	-2.4	5	7.4
	diff 3–4	-2	2.3	4.3
	dif 4–5	0.65	3.125	2.475
	diff 5–6	18.5	10.3	-8.2
Honor 8	**Test cases**	**Kotlin CPU SYSTEM (pp)**	**React Native CPU SYSTEM (pp)**	**Kotlin/React Native difference (pp)**
	diff 1–2	0.4	0.25	-0.15
	diff 2–3	0.7	2.67	1.97
	diff 3–4	-3.1	-0.4	2.7
	dif 4–5	0.625	3.475	2.85
	diff 5–6	7.65	-0.175	-7.825
Galaxy S7	**Test cases**	**Kotlin CPU APP (pp)**	**React Native CPU APP (pp)**	**Kotlin/React Native difference (pp)**
	diff 1–2	-10.34	15.475	25.815
	diff 2–3	0.04	-12.4	-12.44
	diff 3–4	-1.7	-12.65	-10.95
	dif 4–5	-6.575	8.437	15.012
	diff 5–6	17.4025	47.013	29.6105
Galaxy S7	**Test cases**	**Kotlin CPU USER (pp)**	**React Native CPU USER (pp)**	**Kotlin/React Native difference (pp)**
	diff 1–2	17.767	70	52.233
	diff 2–3	15.733	39.25	23.517
	diff 3–4	-26.15	23.58	49.73
	dif 4–5	1	-36.105	-37.105
	diff 5–6	71.15	116.05	44.9
Galaxy S7	**Test cases**	**Kotlin CPU SYSTEM (pp)**	**React Native CPU SYSTEM (pp)**	**Kotlin/React Native difference (pp)**
	diff 1–2	4	57.08	53.08
	diff 2–3	6	26.65	20.65
	diff 3–4	-16.34	13.12	29.46
	dif 4–5	-3.885	-24.7	-20.815
	diff 5–6	36.325	39.626	3.301