Kashif Malik

Appsheet vs React Native: Evaluation of performance and development of Android Apps

Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Information Technology

Bachelor's Thesis

26 September 2021

Abstract

Author:	Kashif Malik
Title:	Appsheet vs React Native: Evaluation of performance and development of Android Apps
Number of Pages:	40 pages
Date:	26 September 2021
Degree:	Bachelor of Engineering
Degree Programme:	Degree Programme in Information Technology
Professional Major:	Mobile Solutions
Supervisors:	Ilkka Kylmäniemi, Senior Lecturer

The aim of this thesis was to study the effects on the performance of an Android device while using applications developed with react native and Appsheet, a no code development platform. The thesis also explores the challenges and limitations that exist in the process of app development while using the said platforms.

The study was conducted by developing various applications that can be used to measure the selected performance parameters. This was supplemented with a thorough study of the existing work on the use of react native and Appsheet, along with closely related necessary information. The performance parameters were graphical processing unit usage, central processing unit occupancy and frames per second.

The investigation resulted in a competitive data analysis among the two development platforms. Applications developed with Appsheet exceeded in performance in most of the parameters as compared to the application developed with React Native.

Keywords: React Native, Appsheet, No-Code development, app development, application performance

Contents

List of Abbreviations

App: Application. Application used in mobile devices

CPU: Central processing unit

FPS: Frames per second

GPU: Graphics processing unit

HLL: High level language. A programming paradigm

IDE: Integrated development environment

NCDP: No code development platform. Platform that does not require coding for application development.

SOC2: Framework applicable to all technology service or SaaS companies

Adb Android Debug Bridge – A data gathering tool for Android

JPEG Joint Photographic Experts Group – An Image Format

1 Introduction

The market for mobile devices has expanded rapidly over the past decades. Mobile devices are being used to perform diverse tasks and the usage is growing. Software applications developed for the handheld devices are becoming increasing crucial for the daily life.

Enterprises are using the mobile software applications for a diverse array of tasks. Mobile software applications are being used by enterprises for various tasks associated with employees such as recording of working hours, sick leaves, internal communication etc. Mobile software applications are also being used by enterprises increasingly as a product to sell to customers. As digitalization grows, mobile software applications will be needed in much greater demand in the future. Developing, maintaining, and updating the mobile application software in a cost-effective way whilst maintaining a good quality has been a challenge for the enterprises.

Multiple technologies have been developed to develop the software efficiently. Apart from a few markets, mobile application development is primarily focused on two operating systems i.e., Android and iOS. Android and iOS have become dominant mobile operating systems in the past decades. The providers of these two operating systems (Google for Android and Apple for iOS) also provide the technologies to develop software applications which can be specifically run on their operating systems.

Google has developed the Kotlin programming language and the Android Studio IDE for developing the Android mobile software applications. Similarly, Apple provides the Swift programming language and the Xcode IDE for developing iOS applications. Developing mobile software applications with the technologies provided by the operating system provider is defined as native mobile software development. However, multiple technologies have emerged that allow for the development mobile software applications.

There is a plethora of alternatives to native mobile software development, however, the cross-platform development has become a popular choice for companies who are looking for cost effective solutions for software development. Cross-platform development allows the software developers to develop applications for Android and iOS with a single code. The flagship framework for cross-platform development is React Native.

Recently, the native mobile software development and the cross-platform software development have been challenged by a set of emerging technologies collectively called no-code. No-code frameworks are software design systems that do not require composition of code for developing applications. One of the leading No Code app development platform is Appsheet.

1.1 Objective and boundaries of the thesis

This thesis has been developed to investigate the effects on selected performance parameters of an Andriod device while using applications developed with Appsheet in comparison to applications developed with React Native. Along with that, this thesis also explores the convenience offered by Appsheet for developing a functional application set side by side by the one developed with React Native.

Further, the objective narrated above is bound to the consideration of usage of CPU, usage of GPU and frame per second, as the parameters that define performance.

1.2 Research questions

This thesis is dedicated to exploring the answers to the following research questions. These research questions tend to direct the research work to achieve the objective of this thesis.

– What is the effect on the performance of a device while using an application developed with Appsheet with reference to the performance while using React Native based application?

– Does the No code platform, Appsheet offer an easier development option of an application as compared to React Native?

1.3 Structure of the research

This research work has been completed by following the common research model, which is ubiquitous among the research society. The initiation of the research work was inspired by the online learning about the prevalent No code development platform (NCDP) for creating applications for mobile devices. This led to the curiosity to investigate the importance of NCDP against the most popular mobile applications development platform, React Native. Thus, an objective was formed to be achieved through this research. To attain the objective, research questions were formed.

Appsheet was selected as the NCDP to be investigated against React Native, since it is being maintained and developed by Google. Further, it has been claimed to be the most user friendly and advanced NCDP so far. The reason for choosing React Native as the standard is that it is the most popular development platform that permits cross platform application development simultaneously.

The research work started with the familiarization of the React Native and no code development platforms (NCDPs), by going through the existing literature available on the scientific journal and publication resources. It further propagated with the development of applications with similar features and outlook using Appsheet and React Native.

These applications were installed and run on a smart phone. Samsung Galaxy S10+ was chosen as the test device to investigate the performance of the

applications. The native tool gave the numerical data that led to the conclusion of the research work. Table 1 gives a short description of the structure of the thesis.

Table 1. Structure of thesis

Chapter	Title	Input	Outcome
1.	Introduction	Introduction of the thesis and research work	Familiarization with this thesis and its confines
2.	Background study	Existing literature	Knowledge about applications, programming languages and NCDP
3.	Artifact Evaluation	Information obtained by comparing the performance of apps	Data about selected performance parameters
4.	App Development Comparison	Comparison of app development with React Native and Appsheet	Capabilities of each development platform.
5.	Conclusion	Outcome of results and analysis	Outcome of the thesis

In the Table 1, the five chapters of the thesis are explained. The first two chapters focus on introduction and literature review respectively. The last three chapters explain the experiments, results and conclusion. Table 1 provides an overview of the thesis.

2 Background Study

This section of the thesis is focused on the review of the existing literature to facilitate the familiarization with the code and no code applications. This part further describes the various types of applications, along with different kind of coding languages that are being used. The sources of information were obtained from online resources including sciencedirect and jstor.

2.1 Types of apps

An application, mostly referred to as an “app” is a software that is set up in a smart electronic device such as a smartphone or a computer to perform a specific purpose. Commonly the apps are developed to serve a specific purpose and meet a focused objective. A single app has different features and components integrated with one another to perform various tasks to achieve the single multi-info output. (Hoehle and Venkatesh, 2015)

Applications are classified into following three basic types. This classification is based on the dependency of the app on the external connectivity to execute a task.

2.1.1 Web-based app

The applications that rely on an internet connection for complete operation are called web-based applications. These apps utilize a notably small space of the memory of the device, since the internet server is used to accumulate the database. The preferred languages that are used to develop web-based apps include HTML5, CSS and JavaScript. Google docs and Netflix are popular web-based applications. (Di Lucca and Fasolino, 2006)

2.1.2 Native app

Native apps are the dedicated software that are created for a specific platform. These apps are developed to furnish the apical functioning by imparting highly compatible functioning components compatible with the particular operating system. As an example, an application developed for iOS, will malfunction if installed into another operating system like Android. To further elaborate it, a calculator app developed for Apple devices is a native app and can give optimum performance only with iOS. (Que, Guo and Zhu, 2016)

2.1.3 Hybrid app

Hybrid apps are an integration of native and web-based apps and are operable with web-based and native platforms. Native app development is an easier and swift process, along with that these apps are diverse and can be blended in different platforms. This quality makes the hybrid apps more agile, yet this compromises the efficiency. Hybrid apps are poorer in performance as compared to web-based or native apps. (Que, Guo and Zhu, 2016)

2.2 What is react native?

React native is a constituent of the JavaScript scheme to create natively renditioning apps for both Android and iOS. It has evolved from React, the JavaScript athenaeum created by Facebook to establish the front end. React native is aimed at the app development of mobile apps. Alternatively, it can be said that react native makes it possible to create mobile applications that are native by using the already popular JavaScript library. Along with that, react native gives the flexibility to dispense the code concurrently between platform thus making it possible to develop applications for both iOS and Android at the same time. (Eisenman, 2015) React Native app development relies on the combination of XML-esque and JavaScript markup, called JSX. Further, in React Native code, Java is employed to natively render the application

programming interface (API) for Android and Objective C is utilized in case of iOS. consequently, the applications depict a genuine mobile user interface with dedicated components. In this way, the application is free from any web views or constituents. Apart from these, React Native provides JavaScript's terminals for connection with the APIs of the platform. This paves the access of the application to the elements device like the camera or global positioning system.

React Native permits the development of applications compatible with Android and iOS operating system, along with that it has the capacity to be adapted by any prospective platforms. React Native is always a cross platform compatible language capable of developing apt software solutions for mobile applications. The success of React Native is getting a recognition and big enterprises like Facebook have started using for their mobile applications development. (Danielsson, 2016)

2.3 Advantages of React Native

The feature of React Native that permits the integration of rendering with the host program, distinguishes it from the other cross platform software creation. Currently WebView's are utilized to render the applications developed from the integration of HTML, CSS, and JavaScript. This procedure serves the purpose, but it comes with a compromise on performance. Along with that, such traditional method does not have the access to the native user interface components of the platform. This results in an extra effort to create the necessities separately for the newly developed app and hence requires separate updates to keep it compatible with the system updates.

Contrary to that, the markup language in React Native is transfigured to genuine and native elements of user interface. These clouts the rendering elements of the platform that is being worked upon. On top of that, the principal user interface string is partitioned from the working of React, thus ensuring a high performance without any compromise on the effectiveness. During the update phase React Native, the views are again rendered. A notable difference between React and React Native is that React Native employs the user

interface repository of the host platform, instead of utilizing the markup from CSS and HTML. (Hansson and Vidhall, 2016) The ability to employ host platforms user interface gives React Native the ability to develop cross-platform applications.

2.4 Application development using React Native

The application development process from React Native, as adapted by fusioninformatics.ae (2021), a prominent software development company based in Dubai starts with the identification of the requirements and expectations. This is succeeded by the creation of user interface and evaluation user experience derived from the interface. This is achieved by presenting a prototype to a group of UX experts. Then the backend if developed and harmonized with the UI. To ensure a smooth and flawless product, rigorous application testing is done to locate any bugs or shortcomings. The final step after approval is the deployment of the application across a global server. This method of application development is traditionally the most popular and is deemed the exceedingly reliable.

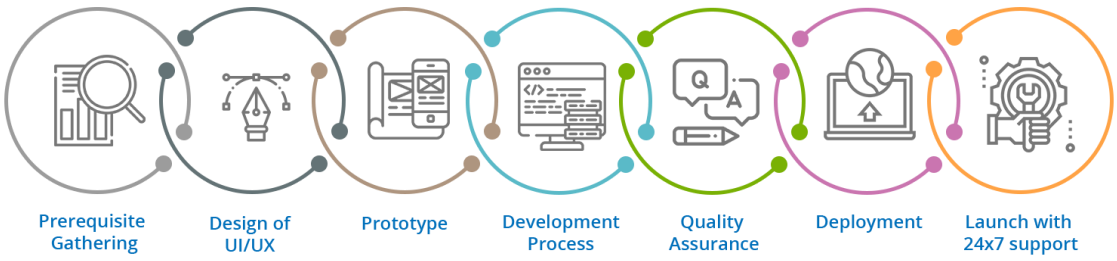

Figure 1. React Native based application development process. (Fusioninformatics.ae. 2021)

The figure above demonstrates the process used to develop a React Native application. The figure highlights the steps involved in making a React Native app that includes designing at the start and launch with technological support at the end. The quality of a React Native app depends on how effectively these steps are executed and how much resources (time etc.) are utilized.

2.5 No Code Application Development

No code development platform referred to as NCDP, provides the possibility to develop software without the need of conventional computer coding, instead the program can be developed by using a pre-developed configuration and a user interface. NCDP significantly reduces the product development cycle time and permits the employment of less skilled developers to equally participate in the development process.

Customarily, NCDP development environment comprises of the following features:

- a user interface that offers various modules or blocks that can be dragged and dropped.
- an ocular tool for modeling the user interface, data configurations and range of capabilities. Further, it also offers the possibility to include the manually developed code as well.
- Junctions to manage the cache and fetch the data.

The no code development platforms are composed to serve particular purpose. In spite of having the above-mentioned features, all the NCDPs are different due to their intended functionality. The difference comes in the modes of operations, assimilation and needs in the market. Applications developed from NCDP might be focused to one segment of business like visualizing the workflow or those can be extensive and might have the enterprise resource planning tool assimilated into it. NCDPs are identical to visual programming languages. (CABALLAR, 2021)

No code platforms are popular among the enterprises that desire a rapid transformation from traditional to digitized system by using the cloud-based solutions of the mobiles apps. No code environment and its accessories are customized for individual users and their needs unlike the customary IT solutions. This facilitates in overcoming the software development curbs like time, finances and expertise. NCDP offer a convenient terminal to connect the application programming interface of the enterprise software. Along with that, it

is quick and easy to incorporate the current software system and all its components of the business with the new NCDP solution. (Harris, 2021)

The use of NCDPs offer the following perks:

- **Access** – The rapidly increasing adaption of codeless mobile application development has resulted in an easy access to the NCDP platforms from the creators. This ease of access has resulted in a paradigm shift in app development industry. Now anyone with basic internet skills and business knowledge can develop apps.
- **Agility** - NCDPs offer a noticeable level of agility in the functionality of the user interface and thus can offer better user experience. Data display and different forms reduce the development time by being supple for different parts of the same app as well as for other apps.
- **Richness** – Unlike the popular view NCDPs are rich in features and functionalities. Apart from that, they offer extensive integration possibilities thus allowing the developers to achieve the desired business requirements. (Woo, 2020)

The no code development platforms are providing an opportunity to reduce the burden on the common worker. By using such platforms less skilled employees can take responsibilities during the app development process and later on the user can enjoy a quick upgrade to the apps according to their changing needs. The rising popularity of NCDPs is contributed by the ease of availability to the end user trying to utilize information technology for business. One dominant reason is that unlike traditional and low code platforms, NCDPs do not require any computer programming skills, thus enabling every member of the business to participate in their app development process actively. On top of that, the NCDP gives the control to the user where benefits from the drag and drop app development process due to the model driven interpretative concept of the platform. Along with this, NCDPs provide a user-friendly interface which supports the app development process for any user by providing a real time preview of the app as it develops. (Harris, 2021)

One of the conspicuous apprehensions about no code development platforms is the security. The rising popularity has sparked a debate on the ability of NCDPs to shield the sensitive data from hackers. This concept about NCDPs originated when the security companies rang the alarm that app developed in NCDPs are created by non-technical personnel. A careful and conscious study showed that apps developed with traditional codes are much more vulnerable in comparison with app developed using NCDPs. A simple explanation to this is that NCDP apps only allow the user to change or manipulate what visible thus restricting the app developer from accessing the actual programming of the platform. This ensures that the functionality of the application remains impervious, and the security persists resolute. Popular and rapidly developing no code development platforms include Microsoft power apps, Airtable, Quickbase and Google Appsheet. (Woo, 2020)

2.6 Appsheet

Appsheet is an NCDP maintained and further developed by Google. It is a SOC2 compliant app development platform which offers natural language programming to facilitate non-technical resources to use simple English to input the commands for app development. It was developed in 2014 and was acquired by Google Inc. in the beginning of 2020. Appsheet uses cloud based platforms like Google drive and office 365 as data provenience, to develop an application. The agility of this platform permits its application in a wide range of business domains including management related apps, technical data apps and reporting apps. (Lystra, 2021)

Appsheet facilitates the app development process by permitting the insinuation of data from the cloud. Along with that, data from spreadsheet can be imported and utilized directly in the app development. This possibility of utilizing the data from various databases or spreadsheets significantly expedites the app development. Appsheet is commercially available for organizations upon obtaining a license and apart form that it can be used as self-service platform by individuals. The licensed version offers better management of the resources, improved features for analysis of data and higher performance. Appsheet

makes it possible for the business components with elementary knowledge of databases and spreadsheets, to develop fully functional applications. (appsheet.com, 2021) The apps developed by using Appsheet perceive data in the form of impressions, illustrations, barcodes and near field communication. The data is harmonized into the online data multitude, thus making it accessible and protected. This synchronization of data is accomplished either manually or mostly automatically. (Nickelsburg, 2021)

The sharing and synchronization of data permits the cooperation among the users over mobile and desktop gadgets. Proclamations and tasks can be made by developing the relevant rules when necessary. Apart from having the possibility of online data access, data is saved on the local memory of the device when it is not possible to upload it to the common cloud. The data is always transmitted to the cloud through an encrypted protocol. Latest versions of Appsheet retrieves the data from the online tools developed by Google including Dropbox and Google drive. (Lystra, 2021)

The data on Appsheet is featured in illustrative and reciprocate formats. Most popular data display formats contain list, chats, calendars, forms, control panel and maps. Depending upon the source of the data, Apps can exhibit various views of the same data distinguished on the basis of source. (appsheet.com, 2021)

According to Nickelsburg (2021) Appsheet follows the declarative language paradigm. Gautier, Le Guernic and Besnard (1987) has described declarative model as computer coding archetype where the desired output is defined rather than the structure on how it should be achieved. Declarative criteria develop a supportive code for parallel processing applications and is in accordance with the explicit rationale systems. coding.

This means that the users of the Appsheet platform can tailor the experience of the user of the application by declaring the rationale of the scheme of the app instead of using the conventional coding. This enables the achievement of adept and secure customized outcome with the minimum utilization of resources as compared to the traditional coding done with the declarative approach. (Nickelsburg, 2021)

The input of data to the apps developed through Appsheet can come from the scrutiny of apparatus, field, safety protocols, reporting of various data inputs or the management of inventory.

With reference to industry 4.0, highlighting the importance of automation in mobile applications, Bauer, Brandl, Lock and Reinhart, (2018) emphasized on the capability of Appsheet to self-manage the mechanisms of a business. As an example, the Appsheet offers the possibility to approve the orders if they meet defined criteria. On top of that, Appsheet is a prominent platform that provides the facility to use the artificial intelligence and host language from Google, to self-create activities and impressions of the user. Further, Appsheet is capable of developing desktop and mobile applications at the same time with minimum need of coding, thus facilitating the non-developers to customize the applications according to their organizational needs.

2.6.1 Features of Appsheet

According to Google (2021) Appsheet offers state of the art and modern application development features. The most distinct and convenient features include the facility to automate the data processing and display, thus facilitating the flow of information. Along with that, Appsheet features the syncing of data offline. The most distinct and lucrative aspect of Appsheet is the availability of Google’s advanced and mature artificial intelligence platform, that could permit machine learning.

The use of Appsheet not only presents the convenience of developing applications without a code but it integrates the application with the machine learning algorithms developed by Google, thus making it one of the most advanced yet easy to use platforms form application development. (Petrović, Roblek, Radenković, and Nejković, n.d.) This enhanced offering not only save time and resources, but it also makes it possible to give the application autonomy in running some aspects of the software. (Haan, 2021)

From the industrial point of view, the software developed with appsheet can locally store the data and synchronize it once the internet is accessible. What makes the data more attractive is the complementary information that come due to the artificial intelligence of the Appsheet. The app stores the location through the global positioning system of the device and can link the pictures and scanned info with the data. Along with that, all of this can be shared quickly as a link through email, thus making the process of data collection, collaboration and exhibition convenient and easy. (Petrović, Radenković, and Nejković, 2020)

Due to the possibility of inhouse customization, Appsheet offers a better control and deliverable to the organizations. It makes the possibility to rapidly change the application with the changing ecosystem. Along with that, the policies, features and information can be updated without the needs of expert IT personnel. (Bucchiarone et al., 2021)

2.6.2 App development using Appsheet:

According to Harraz (2018) application development using Appsheet follows a simple yet effective approach. The usual development process includes the planning of the application, management, and security of the app as shown in Figure 2.

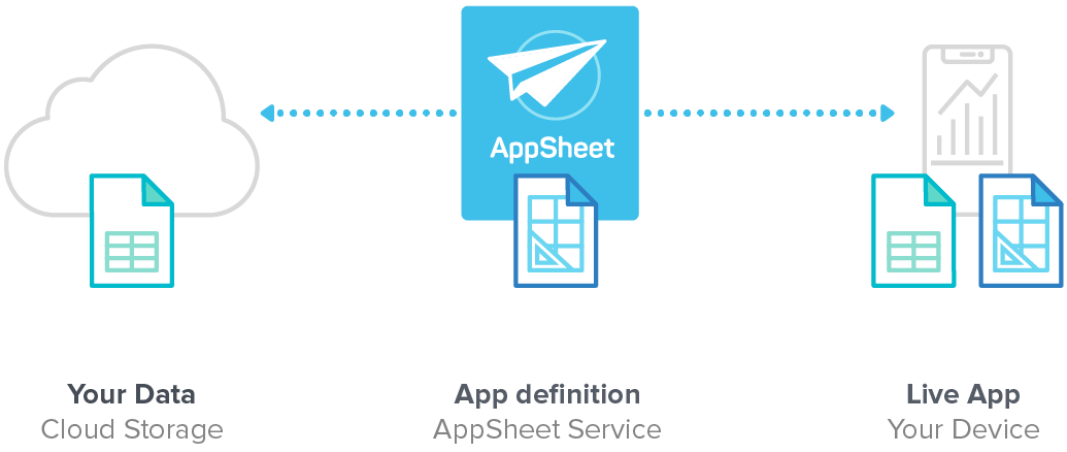

Figure 2. Simple steps in application development using Appsheet (Uribe, 2021)

The founding component of the Appsheet derived software is the creation of spreadsheets to act as source of data, this is followed by the creation of application by dragging and dropping Google add-ons. Basically, at this point the application is ready but it can still be customized, and interface can be updated. The application is used for monitoring on shelf availability, it is developed using the mobile app builder Appsheet. Four development requirements were planned followed by the management mode of the application and at the end the security mechanism of the application was set. (Harraz, 2018). Figure 2 shows the simple process of application development using Appsheet.

3 React Native & Appsheet: An Evaluation of tools, metrics, and performance

To investigate the answers to the research questions of this study, a comprehensive comparison of React Native and Appsheet has been conducted. A total of 4 artifacts have been developed to assess the performance of the React Native and Appsheet technologies. Different artifacts highlight different aspects of app development technologies. Of the four artifacts developed, two were developed with React Native and two were developed with Appsheet.

The frameworks used to develop the artifacts are also associated with a development approach as explained in the earlier chapters. The React Native framework represents the cross-platform application development while the Appsheet represents the No Code application development.

The artifacts developed with the frameworks were android apps. Both the frameworks offer the possibility to develop apps which are identical to a native android app developed using the Android Studio IDE and the Kotlin programming language.

It is important to note that no optimization has been done to any of the artifacts. There are multiple technologies that offer further optimization of the applications; however, such technologies are beyond the scope of this thesis. This study focused on what the software developers can expect from the React Native and Appsheet frameworks.

3.1 Experimental Setup

The tests were conducted using the mobile device Samsung Galaxy S10+ which at the time of tests ran the latest operating system. S10+ is popular Android device. During the tests, the Bluetooth and GPS were turned off. The

tests were also conducted in a way that only the performance of the artifact in focus was measured. The apps and services running in the background were ignored.

3.2 Metrics and Data Gathering Tools

The metrics measured to compare the performance of the no-code and cross platform frameworks are as following.

3.2.1 Metrics

- **Frames Per Second (FPS):** It is considered an important metric to measure the fluidity and responsiveness of the screen. If the screen displays 60 FPS at a constant rate, the fluidity of the screen is perfect. Transitions and animations that are not displayed at 60 FPS are termed as “Janky”. This study measures the FPS of the artifacts and detects the Janky frames. The experiments on the artifacts were conducted twice and the data from the second readings was collected. During the first time, the applications displayed some lag which could skew the data.

- **Central Processing Unit (CPU):** The CPU usage of the apps generated with Cross-Platform and No Code platforms offers important insights regarding the performance of the apps. CPU plays an important role in the user experience of the apps and is considered an important metric for performance. This thesis measures the CPU usage of the artifacts in the context of transitions. The applications differed in their CPU usage while performing the similar transitions.

- **Graphics Processing Unit (GPU):** GPU memory usage in Android operating system provides important insights in the performance of the app. The GPU usage of the artifacts was measured for the different

transitions that involved different type of views. GPU performance was compared using the ADB Dumpsys tool which provides the total GPU memory usage for the action performed by the app.

3.2.2 Tools Overview

Many tools were used to measure the performance metrics explained previously. The performance of the Android device was measured using both the CLI (command line interface) and graphic profiling tools. The CLI tool was adb (Android Debug Bridge) that was used for both the FPS and the GPU metrics measurement. The graphics profiling tool was “GPU Watch” which is native to the Samsung Galaxy S10+ device used for the experiments.

ADB Systrace is a tool which allows to capture a trace of the system. The data recorded during the trace can be visualized in an HTML web page using the Perfetto UI. Similarly, ADB Dumpsys is a tool that captures the data for the entire lifecycle of the process under observation. ADB Dumpsys is managed from the command line. It can be reset and will provide data from the reset onwards.

3.3 Artifact Design & Development

The artifacts developed for this thesis study were made using the React Native and Appsheet framework. The artifacts are identified as A1, A2, R1 and R2. A1 and A2 were developed using Appsheet while R1 and R2 were developed using React native.

3.3.1 Navigation Transition

Navigating between pages in an app involves a transition animation that indicates a change in the context or the page. Navigating from one page to another was tested in artifacts R1 and A1. A1 and R1 had a similar design

consisting of two pages. The first page had a list item. Clicking on the list item leads to a second page which displays a picture of the bird. The navigation between pages differs in various platforms. React Native does not have an in-built navigation available to use for software developers. React Native requires a third-party navigation library and multiple options are available to choose from. React Navigation is a library that is presented on the official documentation of the React Native framework. React Native Navigation is an important alternative to the official React Navigation and offers more native experience. However, the focus of the study is to evaluate the standard technologies of React Native and Appsheet.

Options to optimize the performance are beyond the scope of this study. Hence, React Navigation library was chosen for this thesis study.

3.3.2 Bottom Tab Navigation

Bottom tab navigation is commonly used in mobile applications that require multiple page usage. Different platforms offer bottom tab navigation in a variety of ways. In Appsheet, the bottom tab navigation is in-built and can be applied while designing the app without the need for any additional software libraries. For React Native, the bottom tab navigation can be obtained the React Navigation library that was installed for navigation in the app. For bottom tab

navigation, an additional installation was executed using the following command:

```
npm install @react-navigation/bottom-tabs
```

This installation provided efficient APIs that allowed the formation of bottom tab navigation in the app.

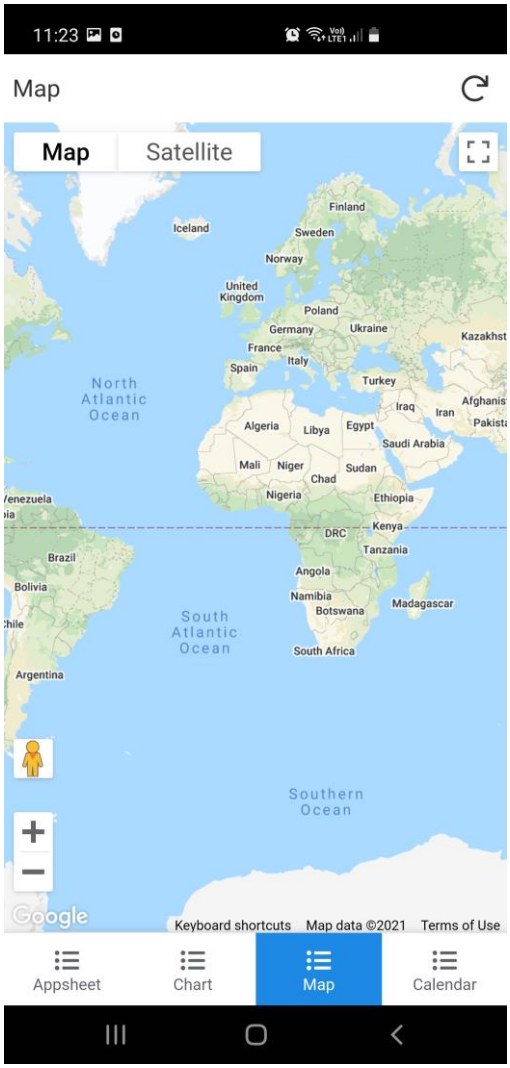

Figure 3. Screenshot of the Map screen in Appsheet A2 Artifact

The bottom tab navigation was applied to A2 and R2 artifacts as shown in Figure 3. The two artifacts were identical in design. A2 was developed using

Appsheet while R2 was developed using the React Native framework. Both the apps had a bottom tab navigator which allowed the user to navigate between four screens. The four screens are as follows:

1. Home: The home screen contained a list of text items.

2. Map: This screen displayed a map. The Appsheet has an in-built option to apply the map in the app. For React Native, a third-party library is needed to deploy the calendar. “react-native-maps” was chosen as the preferred library for maps in React Native app. It is the most widely used library for maps in React Native framework and updated regularly by its software developers.

3. Chart: The chart screen displayed a pie chart two quantities i.e., sunny, and rainy days. Appsheet provides the option of using all kinds of charts without any external library. React Native requires the usage of the external libraries for charts. “Victory Native” is the library used for pie chart in the React Native app. Victory Native is a popular library for data visualization tools in React Native. It was chosen based on its popularity and up-to-date maintenance.

4. Calendar: Calendar screen displayed a calendar view. For Appsheet, the calendar was available as an in-built option. For React Native, an external library “react native calendars” was used. “react native calendars” is among the most widely used libraries for React Native.

3.4 Results

The results from the experiments provided important insights into the performance of the frameworks. The experiment involving the navigation from one page to another was conducted on A1 and R1. The experiment was

conducted twice on each app and the data was gathered from the second round. For the data on frames per second (FPS), the experiment was conducted using the ADB Dumpsys and ADB Systrace tools. ADB Dumpsys measures the data of the whole process of the app i.e., since the app was started. The data was obtained using the following command:

```
adb shell dumpsys gfxinfo package-name
```

To measure the data while going from one screen to another or for measuring the data of a certain action on the user interface, the ADB Dumpsys was reset before the action was performed.

The values measured during the experiments are as follows:

a) FPS: "$Count^{DUR.}$ (jank)" provides the count of frames as reported by the ADB Systrace and ADB Dumpsys tools. The "count" is the total number of frames rendered, duration is the time it took to render those frames and "(jank)" is the number of Janky frames rendered during that period. As an example, $22^{186\ ms}$ (1) would mean 22 frames rendered in 186 ms and 1 frame was janky. In case of ADB Systrace, there is an additional parameter of color added to the number of janky frames. The color is attached to a jank frame depending on the how much additional time it took for

rendering in addition to the 16-millisecond time needed for a smooth 60 FPS rendering.

b) CPU: The CPU has been measured in percentage. The CPU peak shows the highest point of CPU usage when the action was started and ended.

c) GPU: The total GPU memory usage was observed using the ADB Dumpsys tool of the action observed.

The table 2 displays the data obtained from apps A1 and R1. This data was collected by navigating from the first page of the apps to the second page.

Table 2. Data on FPS of R1 and A1 artifacts

Technology	$Count^{DUR.}$ (jank) adb systrace	$Count^{DUR.}$ (jank) adb dumpsys
R1 (React Native)	$236^{4066\ ms}$ (2 yellow)	$22^{186\ ms}$ (1)
A1 (Appsheet)	$40^{652.5\ ms}$ (0)	$27^{186\ ms}$ (21)

The ADB Systrace tool data showed that the React Native app was rendering the frames for a longer period. The React Native app rendered frames for almost all the time the trace was recorded. The Appsheet app rendered fewer frames in shorter period. The adb Dumpsys data was recorded for the same amount of time in milliseconds because it was recorded with CLI command. React Native app rendered 22 frames as compared to 27 in 186 milliseconds. The ADB Systrace showed that the React Native app had more Janky frames, however the ADB Dumpsys showed that the Appsheet had almost 70% of the frames as Janky compared to only 1 Janky frame in 27 by React Native.

Table 3 and Table 4 show data of A2 and R2 apps. The left column, in both tables, shows the screen that was transitioned to from home i.e., from Home to Chart, Home to Map, Home to Calendar. The FPS data is displayed in similar manner to the Table 1.

Table 3. Data on FPS of A2 artifact

A2 (Appsheet) Transition	$Count^{DUR.}$ (jank) adb systrace	$Count^{DUR.}$ (jank) adb dumpsys
Chart	$124^{2075\ ms}$ (0)	$33^{240\ ms}$(24)
Map	$59^{1304\ ms}$ (0)	$56^{240\ ms}$ (2)
Calendar	$39^{636\ ms}$ (0)	$36^{244\ ms}$ (2)

In Table 3, the data from the two tools (systrace and dumpsys) was considerably different for similar actions. The adb dumpsys recorded that more frames were rendered in shorter period than systrace. The Table 4 also showcased different values by the two tools for similar actions in the React Native app. The adb dumpsys recorded more frames being rendered in a shorter amount of time than adb systrace data. However, a pattern was visible from the data from both tools about the two frameworks under consideration which is discussed in the analysis and discussion session.

Table 4. Data on FPS of R2 artifact

R2 (React Native)	$Count^{DUR.}$ (jank) adb systrace	$Count^{DUR.}$ (jank) adb dumpsys
Chart	$240^{4017\ ms}$ (1 red)	$2^{258\ ms}$ (2)
Map	$217^{3869\ ms}$ (1 red)	$40^{258\ ms}$ (2)
Calendar	$254^{4302\ ms}$ (2 yellow)	$21^{258\ ms}$ (3)

Apart from the FPS, the other two metric under study, the CPU and GPU, were measured with a single tool. The GPU memory was observed from adb dumpsys while CPU was observed from the “GPU watch” tool.

Table 5. Data on GPU and CPU of R2 artifact

R2 (React Native)	GPU Memory	CPU Peak
Chart	19.85 MB	22%
Map	4.11 MB	37%
Calendar	4.15 MB	39%

In the table 5, the GPU memory usage of the React Native (R2) app was significantly higher when navigating from home to chart screen compared to the Map and Calendar screens. However, the CPU usage was the highest when navigating to the Calendar screen.

A2 (Appsheet)	GPU Memory	CPU Peak
Chart	4.07 MB	33%
Map	4.07 MB	37%
Calendar	4.07 MB	28%

Table 6. Data on GPU and CPU of A2 artifact

On the other hand, table 6 shows that the GPU usage of the Appsheet was similar when navigating to different screens. The CPU usage was highest for the Map screen.

3.5 Discussion and Analysis

Large number of janky frames were observed in the case of Appsheet while navigating to a page with a JPEG picture in A1 and a page with a pie chart (picture) in A2 by adb dumpsys. In both these cases, the adb systrace recorded an almost normal rendering of 60 FPS. For the React Native apps, similar pattern was observed. The data of R1(navigating to picture) and pie chart (picture) of R2 showed that the adb systrace observed rendering of around 60 FPS in a similar fashion to the Appsheet. The ADB Systrace and ADB Dumpsys had different data for the two frameworks which is beyond the scope of this study. The pattern obtained from the data from both tools shows that Appsheet and React Native exhibit similar behaviour when it comes to FPS rendering and Janky frames for pictures in the apps.

In the case of navigating to Map and Calendar, the React Native rendered less frames than Appsheet in a period and had more Janky frames. Appsheet had a more fluid and smooth navigation transition to Map and Calendar screens compared to React Native.

The GPU usage of React Native and Appsheet was measured in artifacts R2 and A2. React Native had a higher GPU usage than Appsheet in all cases. A significant spike of GPU usage was observed while navigating to pie chart in the R2 React Native app. Appsheet used a fixed amount of 4.07 MB GPU Memory in all cases. Appsheet had a smoother performance compared to React Native in GPU usage. It indicated that Appsheet had loaded the items after the app was opened while React Native fetched data from the APIs only after navigation transition.

CPU usage of React Native and Appsheet was observed in R2 and A2 as well. Both the apps had a CPU peak of 37% for navigating to Maps screen. The Maps screen of both the apps had the same API of Google Maps. In case of chart as shown in Figure 4 below, React Native was more efficient while in case of Calendar, Appsheet was more efficient.

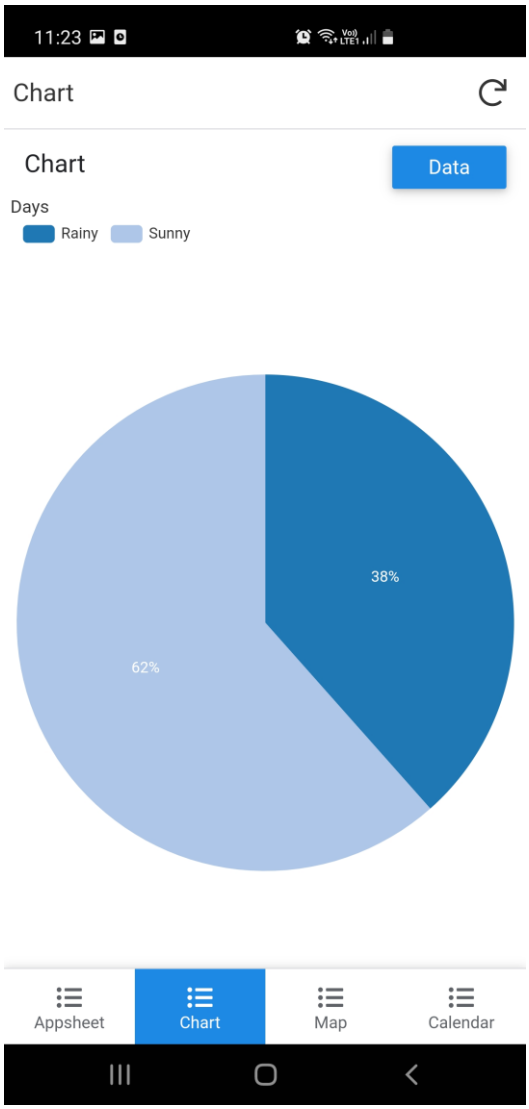

Figure 4. Chart Screen in A2 Appsheet artifact

The experiments indicated that Appsheet performs better than React Native in all metrics observed in this thesis study. An important limitation of the results was that React Native required use of external libraries while Appsheet had all the features available built in e.g., Figure 4 shows the in-built pie chart by Appsheet. This precedent could contribute to the React Native's perceived performance lag in the experiments. However, based on what the two frameworks have to offer a software developer for building applications, the performance of Appsheet is superior to React Native. In the case of React Native, different external libraries may produce different results in performance metrics but this study chose the most used libraries for the respective purposes i.e., how the software development industry is using the React Native framework.

4 React Native & Appsheet: A Development Perspective

In this chapter, the development process of the frameworks under consideration is analysed. React Native and Appsheet differ in how their applications are made. React Native is widely used in the industry and has established practices for meeting the needs of the industry. Appsheet is in early stage of adoption by the industry. Appsheet is currently far less flexible and configurable as compared to React Native. However, Appsheet's ability to deliver service to a wide variety of audience is increasing. Appsheet's ability to be build apps without code gives it a significant advantage over React Native in terms of cost and time consumed in app development. If Appsheet is able to offer a service similar in quality and flexibility as React Native, Appsheet will become the platform of choice for app development. This chapter explores how Appsheet compares to React Native when it comes to delivering the services that industry needs.

4.1 Artifacts

For a comparison of app development, two more artifacts were developed. The artifact developed with React Native is R3 and with Appsheet A3. These artifacts are restaurant search apps which use the Yelp API. Yelp is a popular service for rating the businesses and provides API's for obtaining data on the services of businesses.

To fetch the data from the Yelp Fusion API, an account was created on the Yelp website. An app with the name "Yelp Test App" was created on the Yelp Fusion API service. The app generated the API key which is needed to fetch the data from the API to the client app.

4.2 Appsheet and Apigee

To fetch the data from an external API and display it in the Appsheet app, another service from Google called Apigee is needed. Apigee's latest version is called Apigee X which requires a registered business to sign up to avail its services. Apigee's account allows the user to create an Apigee app which can fetch data from an external API. To fetch data from the Yelp Fusion API, the Apigee app is requires the API key generated on the Yelp Fusion website. In the Apigee account, an Apigee environment can be created which can store the Yelp Fusion API specifications in the API proxies. The Apigee account allows the creation of an API product which is a data object that the client applications will be able to avail and display as shown in Figure 5.

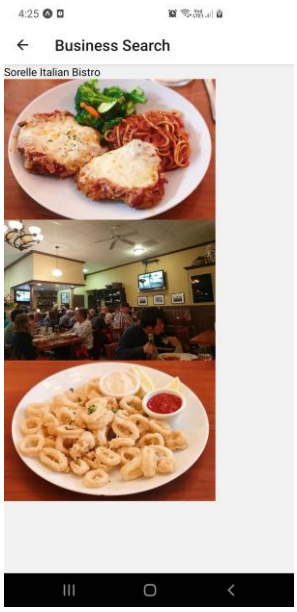

Figure 5. A Screenshot of the R3 Restaurant Search App

The Apigee account also allows the user to register a client Appsheet application which is authorized to use the Apigee API. Once the Apigee API development is complete, the user can head over to Appsheet account and choose Apigee as the data source of the app. The API key and relevant credentials will be needed to get access to the Apigee API data. The Appsheet will then create an app, as shown in Figure 5, with the data obtained from the Apigee API product. Once the Appsheet has generated an app based on Apigee API data, the user can modify the user interface and user experience in the Appsheet account.

4.3 React Native – Restaurant search app

For React Native app, a comprehensive app development plan was needed. Once the Yelp Fusion API key has was obtained, a React Native app was initiated using an external client (e.g., Expo) or the React Native CLI which provides the command line tools to interact with React Native project. For this app (R3), Expo was chosen for its convenience. After the React Native app was initiated, the navigation of the app was configured. After the stack navigation was installed and configured, the API was managed through Axios which is a promise-based HTTP client. Once the data was obtained with from the Yelp Fusion API, the react hooks, Flatlist and navigation with parameters was used to complete the app. The React Native App required significant amount of time, skill and coding effort compared to the Appsheet app.

4.4 Comparative Analysis

Appsheet offers fast and cost-effective solutions compared to React Native, however, the versatility of the services available for React Native still makes it the choice of preference. The software development industry is more focused on the app development with traditional coding methods (React Native, Android, iOS etc.). The payment service providers and banks often provide documentation of their services for programming frameworks like React Native. No documentation is available for Appsheet and Apigee from payment service providers.

The gaming industry also does not currently support any documentation for no code platforms like Appsheet. Leading gaming providers like Evolution only provide documentation focused on the frameworks like React Native.

Appsheet provides apps for Android and iOS but not for web sites. In this regard, the Appsheet is in similar position to the Android native programming and iOS native programming which also do not support website. React Native is a cross-platform framework and it produces applications for Android, iOS, Web, Windows etc.

Regarding Android app development, the Appsheet framework has so far provided every service that React Native has to offer. However, an important area that Appsheet lags React Native is the user interface. React Native allows the software developers to choose from external libraries and the components can also be built the software developers. React Native allows a high degree of customization when it comes to building user interfaces. Appsheet has a very limited customization available for user interface.

The two apps, A3 and R3, developed with React Native and Appsheet showcase important development differences in the two frameworks. The restaurant search apps developed with the two frameworks offer similar user experience. The React Native app development involved more skill and time as compared to Appsheet. Appsheet offers fast and efficient way to develop the app which does not involve coding. However, the level of customization possible with React Native allows far more possibilities as compared to Appsheet.

5 Conclusion

To conclude, it can be stated that the thesis was able to evaluate the performance of the React Native and Appsheet in Android app development. The data of the performance metrics indicated that Appsheet has better overall statistics compared to React Native. Appsheet has most services built-in while React Native relies on external libraries. Appsheet can offer a more fluid and smoother user interface as compared to React Native. The built-in components give Appsheet an edge over the React Native apps. The built-in components in the Appsheet apps work cohesively as compared to React Native.

Appsheet is advancing with time and offers more and more options with the passage of time. The gap between the versatility of React Native and the limited services of Appsheet is reducing. As shown in Chapter 4, Appsheet can fetch and send data to an external API via the Apigee service. Apigee is an effective tool that allows the formation of an API for the client Appsheet app. Despite the apparent advantages in performance and development, the Appsheet is not suitable for wider adaption the industry. Most of the leading payment providers (e.g., Revolut) and gaming providers such as Evolution only provide their API documentation for frameworks like React Native. Although it is possible to integrate the financial and gaming services in the Appsheet app but the customization and debugging options offered by React Native are unmatched. React Native also offers high flexibility and customization for the user interface while Appsheet’s scope in this regard is limited.

References

appsheet.com, 2021. How to create a no-code app. [online] Solutions.appsheet.com. Available at: <https://solutions.appsheet.com/how-to-create-an-app> [Accessed 11 August 2021].

Bauer, H., Brandl, F., Lock, C. and Reinhart, G., 2018. Integration of Industrie 4.0 in Lean Manufacturing Learning Factories. *Procedia Manufacturing*, 23, pp.147-152.

Bucchiarone, A., Ciccozzi, F., Lambers, L., Pierantonio, A., Tichy, M., Tisi, M., Wortmann, A. and Zaytsev, V., 2021. What Is the Future of Modeling?. *IEEE Software*, 38(2), pp.119-127.

CABALLAR, R., 2021. Programming Without Code: The Rise of No-Code Software Development. [online] IEEE Spectrum. Available at: <https://spectrum.ieee.org/programming-without-code-no-code-software-development#toggle-gdpr> [Accessed 13 August 2021].

Danielsson, W., 2016. React Native application development: A comparison between native Android and React Native.

Di Lucca, G. and Fasolino, A., 2006. Testing Web-based applications: The state of the art and future trends. Information and Software Technology, 48(12), pp.1172-1186.

Eisenman, B., 2015. *Learning React Native*. 1st ed. CA: O'Reilly Media Inc., pp.1-14.

fusioninformatics.ae, 2021. *Top React Native App Development Company in Dubai UAE*. [online] Fusioninformatics.ae. Available at: <https://www.fusioninformatics.ae/top-react-native-app-development-company-in-dubai.html> [Accessed 1 July 2021].

Gautier, T., Le Guernic, P. and Besnard, L., 1987, September. Signal: A declarative language for synchronous programming of real-time systems. In Conference on Functional Programming Languages and Computer Architecture (pp. 257-277). Springer, Berlin, Heidelberg.

Google, 2021. *AppSheet: No-code App Development | Google Cloud*. [online] Google Cloud. Available at: <https://cloud.google.com/appsheet#section-1> [Accessed 10 September 2021].

Haan, J., 2021. *Introducing AI-Assisted Development to Elevate Low-Code Platforms to the Next Level - Mendix*. [online] Mendix. Available at: <https://www.mendix.com/blog/introducing-ai-assisted-development-to-elevate-low-code-platforms-to-the-next-level/> [Accessed 14 September 2021].

Hansson, N. and Vidhall, T., 2016. Effects on performance and usability for cross-platform application development using React Native.

Harraz, N., 2018. Online Monitoring of On-Shelf Availability. In *International Conference on Industrial Engineering and Operations Management* (pp. 450-461).

Harris, R., 2021. Low code and no code app development benefits| App Developer Magazine. [online] App Developer Magazine. Available at: <https://appdevelopermagazine.com/low-code-and-no-code-app-development-benefits/> [Accessed 18 July 2021].

Hoehle, H. and Venkatesh, V., 2015. Mobile Application Usability: Conceptualization and Instrument Development. MIS Quarterly, 39(2), pp.435-472.

Lystra, T., 2021. AppSheet's new 'Spec' feature aims to make apps easier to build using natural language, without code. [online] GeekWire. Available at: <https://www.geekwire.com/2018/appsheets-new-spec-feature-aims-make-apps-easier-build-using-natural-language-without-code/> [Accessed 24 August 2021].

Nickelsburg, M., 2021. *Startup Spotlight: AppSheet lets non-developers build custom mobile apps*. [online] GeekWire. Available at: <https://www.geekwire.com/2015/appsheet/> [Accessed 15 August 2021].

Petrović, N., Radenković, M. and Nejković, V., 2020. Data-Driven Mobile Applications Based on AppSheet as Support in COVID-19 Crisis. In *IcETRAN 2020* (pp. 1-6).

Petrović, N., Roblek, V., Radenković, M. and Nejković, V., n.d. Approach to Rapid Development of Data-Driven Applications for Smart Cities using AppSheet and Apps Script. In *Proceedings of the 10th International Conference on Applied Internet and Information Technologies (AIIT). University of Novi Sad, Technical faculty Mihajlo Pupin, Zrenjanin*.

Que, P., Guo, X. and Zhu, M., 2016, December. A comprehensive comparison between hybrid and native app paradigms. In 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN) (pp. 611-614). IEEE.

Uribe, S., 2021. *Discover AppSheet Resources*. [online] Blog.appsheet.com. Available at: <https://blog.appsheet.com/resources-to-learn-more-about-appsheet> [Accessed 12 September 2021].

Woo, M., 2020. The Rise of No/Low Code Software Development—No Experience Needed?. Engineering, 6(9), pp.960-961.