Awel Eshetu Fentaw

Cross platform mobile application development: a comparison study of React Native Vs Flutter

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY
2020

ABSTRACT

Author: Awel Eshetu Fentaw

Contact information: awelk1982@hotmail.com

Supervisors: Dr. Oleksiy Khriyenko

Title: Cross platform mobile application development: a comparison study of React Native Vs Flutter

Project: Master’s thesis

Study line: WISE

Page count: 83 + 3 (83= page count without appendices; 4= page count of appendices)

With a dramatic increase in the usage of handheld devices such as smartphones and tablets, it became a matter of existence for businesses if they do not deliver their services to address mobile users. One critical problem for businesses to address these massively growing users is the diversity of mobile platforms that users prefer to use. Businesses need to find a way for their service to run in different mobile platforms using a single code base or very minimal platform specific tweaks. Hence cross-platform mobile application development comes to the rescue.

Among the widely used cross-platform mobile application development kits are React Native and Flutter. React Native is an open-source mobile application development framework created by Facebook. Developers can develop applications for mobile and web by using React. React is a declarative, component-based JavaScript library for building user interfaces (UI). Another important UI toolkit is Flutter. Flutter is Google’s UI toolkit for developing natively compiled applications for mobile, web and desktop from a single codebase written using Dart.

This thesis presents a comparison study of two widely used cross-platform mobile application development kits. It starts by discussing common application development methodologies. Following that, this thesis details mobile application development approaches with high emphasis on cross-platform mobile application development using React Native and Flutter. There is an implementation of COVID-19 tracking application which consumes REST (Representational State

Transfer) API (Application Programming Interface) from nubentos [3] and coronavirus open API [43]. The application is developed using React Native and Flutter which will be used for performance analysis and comparison between the two applications running on Android and iOS platforms.

Keywords: Application development, Mobile application, React Native, Flutter, Cross-platform application, Performance, Native application

ACKNOWLEDGEMENTS

I would first like to thank my thesis supervisor, Dr. Oleksiy Khriyenko, who provided great suggestions on the flow of the thesis and contents to be added. I would also like to thank all friends who helped me borrow their devices during testing of the application developed for this thesis. Thank you.

Espoo, June 2, 2020

Awel Eshetu

ABBREVIATIONS AND ACRONYMS

UI	User Interface
RAD	Rapid Application Development
CPU	Central Processing Unit
GPU	Graphics Processing Unit
AI	Artificial Intelligence
IoT	Internet of Things
VAR	Virtual and Augmented Reality
REST	Representational State Transfer
COVID-19	Coronavirus disease 2019
API	Application Programming Interface
XP	Extreme programming
GPS	Global Positioning System
IDE	Integrated Development Environment
APK	Android Package
SDK	Software Development Kit
HTML	Hypertext Markup Language
CSS	Cascading Style Sheet
JSX	JavaScript XML
IPA	iOS App Store Package
DOM	Document Object Model
BLoC	Business Logic Component

WHO	World Health Organization
CI	Continuous Integration
CD	Continuous Delivery

FIGUERS

TABLES

TABLE OF CONTENTS

1 INTRODUCTION

Today mobile phones play a key role in peoples’ day to day lives. According to Ericsson mobility report 2019, the total number of global mobile subscriptions was around 8 billion in 2019. Subscriptions associated with smartphones account for more than 70 percent of all mobile phone subscriptions. The number of global smartphone subscriptions is forecasted to reach 7.8 billion in 2025, or 83 percent of all mobile subscriptions [1].

Trends in technology are continuously changing fast bringing new opportunities and capabilities to business and developers. Mobile payment, Internet of Things (IoT), Virtual and Augmented Reality (VAR), Artificial Intelligence (AI), Gaming are among many capabilities. In most cases businesses incorporate these technologies with mobile devices, since mobile devices are increasingly in use and there is a continuous improvement in computing capability, capacious memories, screen sizes and open operating systems that encourage application development. Mobile devices use different platforms. Among the most platforms today are Android, Apple iOS, Windows (Windows phone platform) [2]. This diversity in platforms is a challenge for businesses to develop applications targeting all or most platforms from a single code base. Hence businesses and developers see Cross-platform application development as an alternative to target most platforms rather than following expensive and platform specific application development. Cross-platform application development is mostly done by using Cross-platform application development kit which will compile application code to platform specific native code.

Although there are several Cross-platform application development kits, it is still a challenge for business or developers to choose which development kit will suffice all their application functionality and user experience requirements.

1.1 Background

In recent years it has become more common to see mobile applications which are developed by using cross-platform mobile application development frameworks and tools. As cross-platform mobile application frameworks are becoming more and more versatile, more companies are adopting cross-platform mobile application development to target multiple mobile platforms from a single codebase and reduce development cost and time.

The most natural way of developing an application for a certain platform would be by using platform's native technologies. Besides providing flawless feels and looks, the application runs faster. Moreover, platforms usually come with well-prepared API documentations and there exist solid professional and community support. However, as the number and variety of mobile platforms grows, targeting several platforms using native technologies would need platform specific competence and the development process would be costly.

One of the key advantages cross-platform mobile application development frameworks provide is a means of writing applications once and deploying it on multiple platforms. Now, there are arrays of cross-platform mobile application frameworks in the market. Although comparing each one of them is not the goal of this thesis, Table 1. provides some lists of cross-platform mobile applications frameworks which might indicate why choosing the right cross-platform application framework could be a puzzle for companies and developers.

There exists research on "Comparison and Evaluation of Cross Platform Mobile Application Development Tools" (Dalmasso, I., Datta, S. K., Bonnet, C., & Nikaein, N. ,2013) highlighting different cross-platform application frameworks and making performance comparisons using CPU and Memory usages [79]. However, the work does not include currently leading cross-platform application development frameworks.

Table 1 Cross-platform native application development tools [78]

Framework/Tool	Development Uses
Xamarin	C#
React Native	JavaScript
Flutter	Dart
PhoneGap	HTML5, JavaScript
Ionic	AngularJS, JavaScript
Native Script	AngularJS, Vue.js, TypeScript, JavaScript

Writing code once and deploying it everywhere results in greater development cost and time reduction, since there will be only one code to write and maintain which could target multiple platforms. However, choosing appropriate framework for the cross-platform application development would still cost some time for making research and evaluations of candidate frameworks. Providing a comparison study between the leading cross-platform application development frameworks would benefit companies and developers to make the right choice and reduce their time to research and evaluate the frameworks.

1.2 Objectives and research questions

The main goal of this thesis is to provide a comparison study on two most widely used Cross-platform application development kits, React Native and Flutter. It aims to find out CPU, GPU and memory usage of an application developed by React Native and Flutter which runs on iOS and android platforms.

Although Cross-platform application development kits provide capability to target multiple platforms using a single code base, there are limitations such as UI-construction, hardware and software access, network capabilities, among others. It is clear most Cross-platform application development kits provide a core essence of compiling application code to platform specific native code, but there are still differences among these kits on, how they compile application code to native code, how they map application UI elements to native UI elements, how they access native device capabilities , how much memory they consume during the process of compiling and so on. Hence this thesis aims at answering the following main questions regarding an application developed by using React Native and Flutter.

1. What are the pros and cons of React Native and Flutter mobile application development frameworks?
2. What performance differences will an application developed by the above two frameworks have?

Besides answering the above two mentioned research questions, the thesis aims at helping developers and companies to make right choice of framework to develop a set of functionalities by clustering sample mobile applications based on features they provide. Currently app stores provide theme-based clustering of mobile applications which is mainly end-user oriented. Knowing the set of features that could easily be developed by a framework would include technical requirement to the decision of choosing right framework besides budget and competence. Moreover, this thesis overviews cross-platform application development process. It also presents implementation of Cross-platform mobile application using two most widely used Cross-platform mobile development frameworks such as, React Native and Flutter. After that, the thesis details a range of performance comparisons of the application developed by these two development frameworks.

1.3 Methodology

The selected research questions are related to each other. The second question could be considered as the main objective of the thesis, however it will only make sense to address the second question after providing a fact based comparison study on the pros and cons of the frameworks that will answer the first research question. How the research questions are addressed is described in the following paragraphs of this section.

Introduction and background information on categories of application development and mobile application development approaches is researched and organized through a literature review. Information on the subject is gathered by using widely used online search engines and databases such as Google Scholar, Sci-hub and IEEE Xplore. The keywords used for searching information includes "mobile applications", "application development", "cross-platform mobile application development", "mobile application development" and "cross-platform application frameworks". More information on a specific platform or framework is collected by using the platform's or framework's name as a keyword and from the official website of the platform or framework. Primarily, the literature review includes information from research papers that are available in a digital format. In case of further elaborating a context, information is searched from the web and the quality and relevance of the search result is cross examined from multiple similar sources before use.

Clustering of mobile applications is performed by using an open source machine learning library called Scikit-learn. Data for the clustering analysis is collected by extracting features of mobile applications that are presented in Table 8. In addition to those applications, three email communications mobile applications and three instant messaging applications are included in the clustering analysis. The features that are extracted from the app store description of each app is presented in Appendix B. The analysis is done by converting the extracted features of each app to a vector relative to the corpus prepared from the set of all unique features in all apps. Later the vector is transformed to a cosine distance to signify the similarity between each app and to perform clustering analysis. The vector which contains the cosine distance is presented in Appendix C. The result of clustering and related plots is presented in Figure 9.

Comparison of the frameworks is presented by mentioning facts about the frameworks from relevant sources such as frameworks official website, professional blogs, and developer community websites. Moreover, performance analysis is done by comparing the implemented application which is discussed in chapter 4. The screenshot containing functionalities of the implemented testbed application is presented in Appendix A. The most common mobile application performance parameters such as CPU, GPU and Memory usage are used in chapter 5. The performance analysis is conducted by running five tests for each performance parameter, on each platform and for each framework. The arithmetic mean of the measurements is taken as a single representative measurement. The results of the performance analysis are presented using bar charts and screenshot in Figures 24 to 32. Furthermore, a summary of all performance analysis charts is presented in Figure 33. platform specific performance analysis is summarized in Figure 34.

Furthermore, aggregated percentage suggestion of a framework is presented in section 6.2 as a guideline for choosing a framework to develop cluster of features for the umbrella class of companies. The umbrella class of companies and relevant framework decision making variables are presented in Figure 35. In addition, Figure 36 presented summary of clusters and included features per cluster. Weighting of decision variables for choosing a framework for sample large company class is presented in Figure 37 and similar data for other classes of companies is presented in Appendix D. The resulted aggregated percentage suggestion for choosing a framework is presented in Figure 38.

1.4 Scope and limitation

This thesis focuses on comparison study of React Native and Flutter cross-platform mobile application development frameworks. It presents fact-based comparisons on the pros and cons of the frameworks, followed by performance analysis comparison using a testbed application which is implemented for this purpose. Due to time constraints, the testbed application contains features such as navigation to multiple screens, list views, image loading, loading external WebView, search and filtering list contents. These included features are only a subset of the wide range of features that most industry scale applications might contain. Hence the performance analysis should not be considered as a generalized result and it should only be viewed under the scope of the implemented features.

What the thesis is not trying to provide is a generalized performance analysis of React Native and Flutter cross-platform mobile application development frameworks for industry scale applications. However, the conducted performance analysis on the testbed application can be used as an overview and guideline for companies or other interested groups to further conduct in depth comparison and performance analysis between the frameworks.

1.5 Structure of the Thesis

After the introduction, the rest of the thesis is organized as follows. Chapter 2 presents a general overview of different categories of application development. In Chapter 3 different mobile application development approaches are discussed in detail. In Addition to that it presents clustering of mobile applications based on features extracted from their app store descriptions. Chapter 4 provides a detailed explanation of the implementation process of an application developed by two most widely used Cross-platform application development frameworks such as React Native and Flutter. Chapter 5 describes the performance comparison. It discusses performance comparison processes and presents bar charts of results obtained during the performance comparison. Chapter 6 presents the summary of the thesis. It contains the analysis of performance comparisons that are presented in chapter 5 followed by framework choice guidelines for clusters of mobile applications for different classes of companies. The final chapter provides conclusions to the thesis

2 CATEGORIES OF APPLICATION DEVELOPMENT

An application can be defined as a collection of programs that satisfy certain specific requirements towards resolving a problem. Hence an application can be considered as a solution or collection of solutions to related problems. This solution or solutions could target and reside on specific platform or on different platforms, from a hardware or operating system perspective [4]. Before an application is ready for use, it passes through various steps and processes which can collectively be called application development lifecycles.

Application development lifecycle include the following steps:

- **Planning and preparing requirements**: include identifying the need for the application, considering existing applications as a solution, specifying user target group, specifying target platform, considering features of existing applications.
- **Analysis**: include defining and documenting functional requirements for the application and predicting potential problems that may be faced during the lifecycle of the application.
- **Design**: include defining and documenting application features, components, and parts. It also defines the integration of different parts of the application, their use, and the use of the application in general.
- **Construction (Coding)**: programming the application as per the guidelines of the verified requirements and design.
- **Testing**: trying out the application primarily for finding pitfalls and errors. Testing should also confirm if specified requirements and design are met.
- **Implementation (production)**: making the application ready for use after all functionalities and requirements have been met. It includes preparation of documentation for application usage and operational procedures.
- **Support (Maintenance)**: include user experience monitoring, changes, and improvements of the application.

Although, factors such as, application size and budget, clarity and specificity of requirements, size of development team and competence of each team member, changes in requirements, project time span, determine whether or not these lifecycle steps should strictly be followed, they are generally used as standard lifecycle steps in application development[4,5]. The execution of these lifecycle steps may vary from organization to organization or even from team to team in an organization. In general, application development lifecycle steps can be accomplished by using most common application development methodologies such as waterfall development, agile development, and rapid application development (RAD).

2.1 Waterfall Development

Believed to have originated from manufacturing and construction industries, waterfall software development breaks down project activities into progressive logical phases. It allows teams to break down a project into understandable and reasonable sequences of phases with clearly set specifications. Furthermore, the waterfall method assumes that requirements can be completed and approved in the requirements phase mentioned above. Once the requirements phase is completed, the project strictly goes to the next phase without leaving any chance to revise previous phases based on fresh insights. Since the whole project is divided sequentially, there may not be a chance of starting work that belongs to the next phase or any later phase until the current phase is completed and approved. After properly generating and documenting models and business logic that will be used in the application in the Analysis phase, the execution of the process falls downhill to the Design phase. The Design phase largely emphasizes on the design of technical requirements such as hardware and software architectures, UI mockups and databases for the smooth flow of project phases. During the Implementation phase of the waterfall method, the actual application code is written based on the guidelines of approved requirements and design specifications. As the project flows down to the Verification or Testing phase, the project is checked against approved specifications. Alternatively, it is common practice to ignore this phase to roll out the project to the customer and start the final phase of the waterfall method, the Maintenance phase. During the Maintenance phase, the application is being used by the customer. Since project phases flow downhill in the waterfall method, pitfalls that arise from improper requirements determination, design mistakes, changes in requirements and other possible bugs are revised at this phase. Although the waterfall method shows rigidity in flow of project phases, there are pros and cons related to following this software development method [6,7,8].

Some of the advantages of using the Waterfall development approach include:

- Since Waterfall assumes clarity and specificity of requirements in the requirement phase, full project scope is known at the early stage of the work.
- Designing is not complicated because deliverables are clearly defined and documented at the beginning of the project lifecycle.
- Since requirements remain unchanged throughout the process, each lifecycle has clearly set deliverables to review and approve.
- Clear project timeline can be set, and end results can be predicted in terms of project scope and budget.
- Project progress can be measured because project scope is known in advance.

On the other hand, the following are some of the disadvantages of choosing Waterfall as a software development methodology:

- Rigidity in adopting new insights and requirements in stages other than the requirement phase might lead to expensive re-engineering of the application at the end.
- Lack of design adaptability across different stages of the project lifecycle.
- Due to the strict step-by-step process of the waterfall development approach, it is hard to acquire customer feedback and apply changes into the application prior to the end of the project.
- Delayed Testing phase of the lifecycle often leads to prolonged and expensive Maintenance phase in the project.
- High uncertainty in customer satisfaction by the product, as all deliverables are based on pre documented specifications and customers might not see the product before the end.

2.2 Rapid Application Development (RAD)

Rapid development is one of the software development categories which was formulated to address limitations of following traditional software development methods. These drawbacks include complexity, rigidity, prolonged development time, exclusion of the client during development process and adaptability. This methodology divides the application into smaller components or modules, which encourages requirement changes and client involvement throughout the development process. Each of the application’s modules can be prototyped and delivered separately as part of the complete application. Since Rapid development emphasizes the competences and experience of team members, it allows rapid development of the application from design phase to completion, under the constraints of relatively low development cost. RAD focuses on rapid development of the functionalities of each component or unit by compressing different development lifecycle phases into short iterations. RAD follows strict delivery deadlines for project modules, hence features are highly prioritized, and requirements might be omitted to fit deadlines.

RAD mainly focuses on producing working prototypes as quickly as possible and applying continuous iterations to refine the application. The lifecycle of RAD includes stages such as requirements, design, implementation, and deployment. The design stage highly emphasizes on prototyping of functionalities. In RAD implementation stage is given higher focus which might merge or omit the design phase. RAD methodology encourages client involvement at every stage of the development phases; therefore, client has full knowledge of the product before the end [9,10,11].

The following are among the great advantages RAD methodology brings compared to the methodologies prior to its arrival:

- Due to the higher emphasis RDA puts on the competence and experiences of team members, there is a decrease in time necessary to obtain the final product.
- Project cost reduction is viable because of fast project development.
- The risk of product failure or dissatisfaction of the client by the final product is diminished by involving the client at every stage of development.
- Hence RAD is based on working prototypes, it allows clients to try out the product at various stages of the development process.
- Flexible and adaptable to any requirement changes.
- It encourages repetitive reviews and constructive feedback.
- Changes in requirements of each application module can be immediately applied.
- Each application unit can be delivered and tested separately.
- Since application is divided into modules, it facilitates reusability of components.

- Units integration is applied from the beginning which solves massive integration issues at the end.

Although RAD brings several advantages to application development, the following are among the risks to implementing RAD methodology:

- As RAD follows quick development, significant requirements might be omitted because of prioritization.
- Due to the strict delivery deadline, the product might have less features than intended.
- Requires experienced and highly competent team members.
- It highly favors applications that can be modularized
- Because of higher client involvement and frequent feedback, requirements may not be converged.

2.3 Agile Development

Agile Software development methodologies have evolved around the mid-1990s due to the incapacity of other traditional methods to address various application development problems such as lack of adaptability and responsiveness to new insights in requirements, cost or time overruns, lack of dynamicity in development, lack of client involvement in the development process etc.

Unlike other software development methods, agile development approach is dynamic, adoptive, and organic. It emphasizes incremental delivery, continuous planning and learning, and team collaboration in an iterative way. Agile methodology is a dynamic process used for creating viable applications which pass through multiple iterations before its final. Agile encourages continuous feedback and reiteration of the development process before the complete functional components are delivered. It uses iteration time frames called sprints. Each sprint has execution duration with a list of requirements which are usually planned at the beginning of the sprint. At each sprint requirements are prioritized based on business values set by the client. Unfinished requirements from the current sprint are reprioritized in the next sprint [8,12,14].

According to the "Manifesto for Agile Software Development" [13,15], the following are core focal points of Agile development:

- Agile values individuals and interactions over processes and tools. Continuous communication among team members builds close relationships which boosts team spirit and helps work to excel faster.
- It values working software over extensive documentation. The objective of the agile team is to deliver tested working functionalities frequently.
- It highly values client collaboration over contract negotiation. Close corporations of the client with the developer team reduces the risks of requirements and feature non-fulfillments.
- Agile values responding to changes in requirements over following a plan. In agile, the team is prepared to adopt new insights in requirements during the development lifecycle.

Agile consists of multiple methodologies such as Extreme programming (XP), Scrum, Lean Development, Kanban, and others. Scrum is among the widely used agile development practices that is based on iterative sprints. It brings an adaptive application development where the scrum team works as a unit to finalize functionality within the specified sprint time frame. Scrum methodology composes three main roles such as product owner, scrum master and team of developers. The product owner is responsible for identification and prioritization of requirements and measures the overall project success. The scrum master serves as a bridge between the development team and product owner to help resolve pitfalls during the sprint. The developers' team is responsible for building the application based on follow ups from product owner and scrum master.

The scrum process starts with the product owner listing priority requirements in the product backlog. The developer team together with the product owner select highest priority specifications that can be done within the sprint timeframe. At the end of each sprint, there are demonstrations of completed features, pitfalls are discussed, and feedbacks are included to the product backlog to be considered on the upcoming sprint. The following Figure 1. demonstrates the scrum process [14].

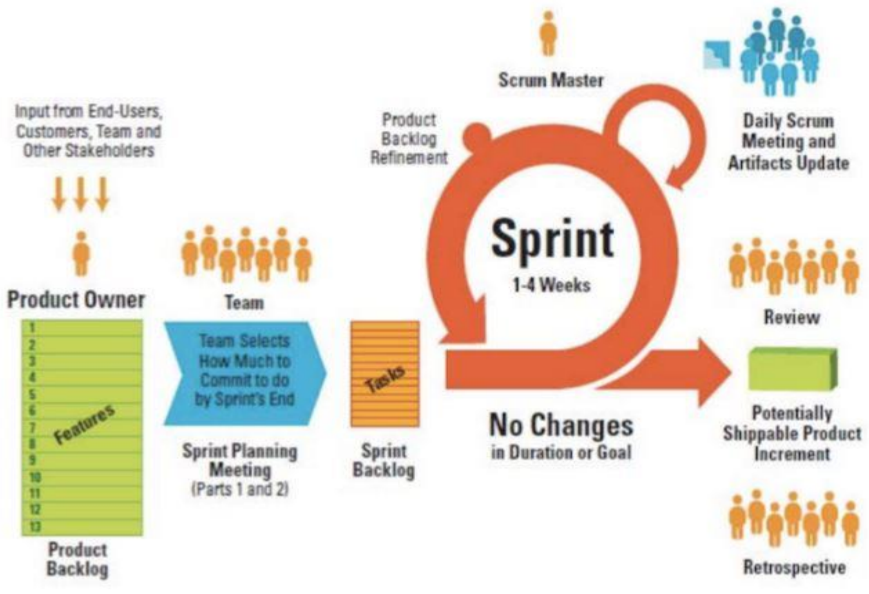

Scrum Master
Input from End-Users, Customers, Team and Other Stakeholders
Product Backlog Refinement
Daily Scrum Meeting and Artifacts Update
Sprint
1-4 Weeks
Product Owner
Team
Review
Team Selects How Much to Commit to do by Sprint's End
Tasks
Features
Sprint Planning Meeting (Parts 1 and 2)
Sprint Backlog
No Changes in Duration or Goal
Potentially Shippable Product Increment
Product Backlog
Retrospective

Figure 1 Scrum Process [14].

The above SCRUM process consists of the following five steps.

1. The product owner together with other team members and client lists requirements for the new application. The list is called *product backlog*.

2. The scrum team defines the different *sprints* and their timeframe. The team then allocates the implementation of the requirements to the different sprints.

3.After the sprint starts. The team focuses on implementing requirements within the allocated sprint time frame created in step 2. By the end of each sprint there is a *sprint review* with the product owner. The sprint review mainly focuses on discussing what was accomplished or not accomplished during the sprint and pass decisions on reprioritizing items from the backlog based on the review.

4. After the compilation of all sprints, there is a potential *final product* ready for users. And the product at this stage is ready for potential release.

5. After each release, the SCRUM team gathers for *retrospective*. At this stage team members discuss the overall process for development, trying to find pitfalls which could be improved for

the following development process and strong development skills which could be carried to the next development process.

Some of the advantages of following agile methodologies are mentioned as follows:

- It allows client involvement and availability to add new insights throughout the project life span.
- Brings a strong sense of satisfaction to the client due to high involvement in product development.
- Hence the client is involved throughout the project, agile resolves transparency issues.
- Costs and deliverables can be easily predicted.
- New insights and changes in requirements are encouraged which can be prioritized or prioritized for the upcoming sprint.
- It allows client to prioritize features and requirements based on their business values.
- Agile brings quality driven development by breaking down the project to multiple units and prioritizing features based on business values.
- It allows pitfalls to be discussed and feedback to be added to the next iteration.

On the other hand, the following are some of the drawbacks to face during following agile methodology:

- Agile might not work best when some of the development team are not open to communication and are not completely dedicated to the project.
- Very high client involvement might lead to requirement divergency, more sprints and overall project cost.
- Due to the higher nature of communication and collaboration among team members, agile might not efficiently work if team members are not located in the same physical space.
- Due to continuous prioritization, some items from the backlog might not be completed within the specified sprint timeframe.
- Agile favors large and modular projects.

3 MOBILE APPLICATION DEVELOPMENT APPROACHES

Nowadays mobile applications are directly involved in our day to day lives. There are ranges of mobile applications we use to facilitate our daily activities such as mobile payment, online shopping, transportation applications, text, and video messaging applications to mention few. There exist millions of apps available for us in different mobile application stores such as Google play Store and App store. The ever-increasing mobile application usage is driving businesses and solo developers to invest their time and money targeting these users. Such strong demand has led businesses and developers to find new ways of mobile application development to target massive amounts of users with less development cost for applications.

The diverse nature of mobile platforms has been the main constraint for businesses and developers to find new ways of mobile application development rather than following cost inefficient platform-specific mobile application development. Furthermore, mobile application development needs frequent improvements and adaptations to meet the fast-changing usability challenges. Handheld device manufacturers increasingly adapt new designs which range from changing device screen sizes to improvements in device capabilities, hence a mobile application must adapt to all these new changes to increase its usage. Although there exist millions of mobile applications in different applications markets, they can generally be classified into three categories as platform-specific applications, responsive or adaptive web applications and cross-platform native applications [19,20].

3.1 Platform-specific native applications

Platform specific or Native application is a software developed to a targeted platform based on the programming language that specific platform supports. It is developed using platform specific SDK and frameworks and its existence is tied to that specific platform. For instance, an iOS application is developed with Objective C or Swift using the iOS SDK and the APIs provided by Apple and uses platform provided elements for rendering the application UI. Native applications are sometimes called embedded applications to signify that these kinds of applications ensure an in-depth integration with the mobile operating system. Applications developed by using platform specific languages allow accessing device specific capabilities such as video and audio capabilities, Global Positioning System (GPS), native calendars etc. Native applications could provide the best performance possible and development is supported by platform friendly integrated development environments (IDEs). These IDEs are designed to provide best development experiences such as debugging the application and contain tools that help perform memory and performance analysis.

The two most widely used application platforms are Google's Android and Apple's iOS. Applications native to android can be written by using Java or Kotlin programming languages, on the most widely used development IDE, Android Studio. Android Studio provides ranges of benefits to developers. It helps developers to be more productive during the development process, by providing built-in support for Google services, such as Firebase Cloud Messaging, Google application engine. It eases application development for different devices including smartphones, wearables, and Android TV. Android Studio also provides the ability to download an android application package (.apk) file and publish it to Google play store related to the developer's account.

Whereas, applications native to iOS can be written using objective C or Swift programming languages, on the primarily used IDE, XCode. Like Android, XCode provides a range of development benefits to developers including the ability to publish application packages to the Apple’s app store. Other existing mobile platforms and their primary development technologies are shown in Table 2 [16,17].

Table 2 Some alternative mobile platforms

Platform	**Developer**	**Development language**
Windows Phone	Microsoft	.NET
MeeGo	Nokia	C++ / Qt, QML, Python, Web technologies
Bada	Samsung	C++
Symbian	Nokia / Accenture	C++ / Qt, Java, Python, Web technologies
Blackberry OS	RIM	Java (J2ME)

Open webOS	HP, Community	C/C++, Web technologies
Firefox OS	Mozilla, community	Web Technologies (JavaScript, HTML5)

Native applications developed for specific platforms are mainly available for downloads from a platform dedicated system, such as Apple Store and Google Play Store. Figure 2 demonstrates native application development.

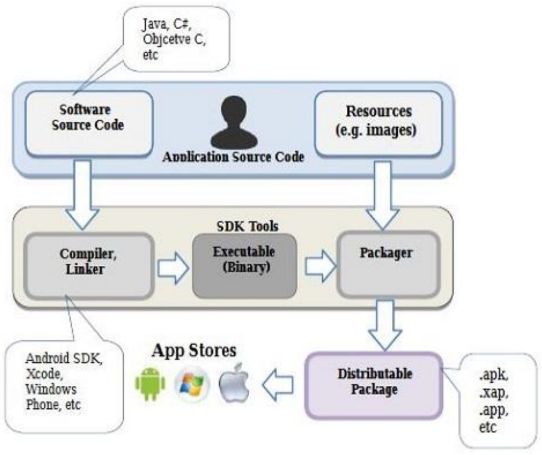


Figure 2 Native application development (reprinted from [18]).

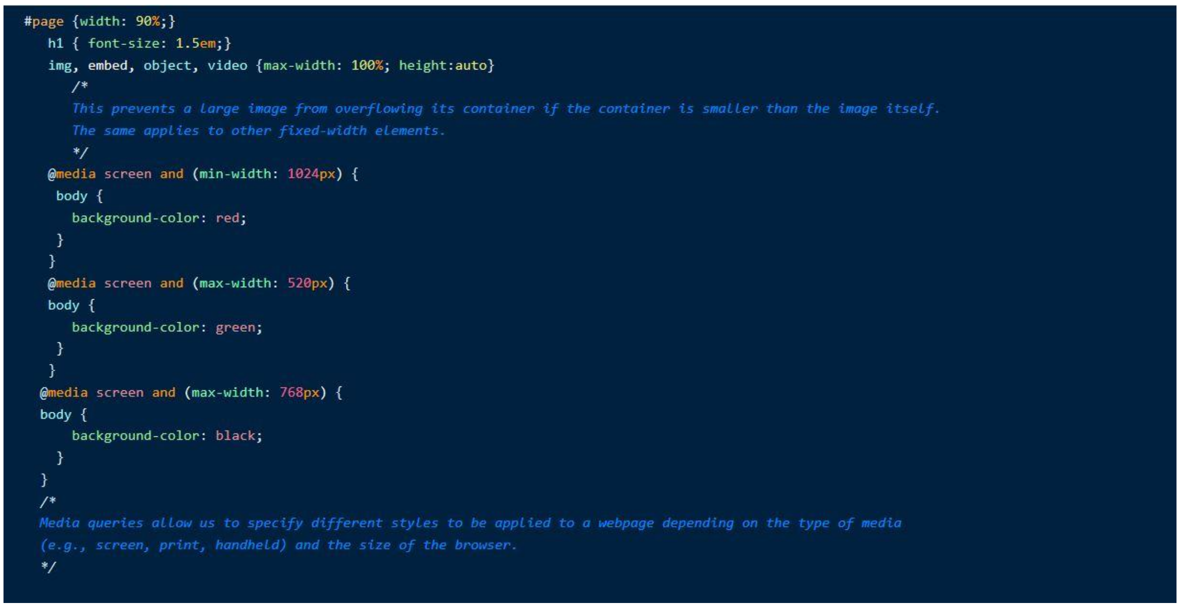
Although platform-specific mobile applications provide higher performance and utilization of native device capabilities, their variety make development and maintenance costs higher. Platform-specific application code cannot be reutilized for other platforms, development needs dedicated resources and relevant platform-specific competency.

3.2 Responsive web applications

In recent years, emerging web technologies allow businesses and developers to develop responsive web applications that work seamlessly regardless of the type and screen size of a device. Traditionally businesses used to follow pixel-perfect design approaches to develop their web applications. Pixel-perfect web design is performed by creating mock-ups of a web page using tools like Photoshop. Developers then program the design to fit a standard web browser. Often these pixel-perfect designed web applications would not fit to browser screens sizes that are not primarily considered in the design. Pixel-perfect design approaches became a bottleneck as the variety of screen sizes grew. Hence, a responsive web design approach has emerged to solve this problem [21].

Responsive web applications that run on mobile platforms depend on web technology standards and browser support of mobile platforms. The most widely used web application architecture divides the whole application into three layers such as presentation, logic, and data layers. The presentation layer is solely responsible for rendering the user interface. Whereas the logic layer contains different application logics such as server-side programs and connections to the data layer for data storage and retrieval. The data layer on the other hand is responsible for persisting data. Although the logic layer can be implemented using different programming languages (Java, Python, Nodejs etc.), the presentation layer is mainly dependent on web technologies such as JavaScript, HTML and CSS. It is mostly on the presentation layer that the responsiveness of a web application is implemented.

Generally responsive web applications try to fit different screen sizes by using web design techniques such as, flexible grid-based layout, expandable images, and media queries. Flexible grids are mainly designed by using relative percentage unit CSS styles rather than absolute pixel unit styles. Media query is a web designing technique which defines rules to include CSS properties and values when a certain condition fulfills. It helps to apply different CSS styles in response to media type and screen sizes. Expandable images can be achieved by setting the width property of an image (100%) and the height property of an image to auto. This way the image will be responsive, and it can easily scale up and down according to the screen size. The code snippet in Figure 3 illustrates how responsiveness can be applied to web applications.



```
#page {width: 90%;}
   h1 { font-size: 1.5em;}
   img, embed, object, video {max-width: 100%; height:auto}
      /*
      This prevents a large image from overflowing its container if the container is smaller than the image itself.
      The same applies to other fixed-width elements.
      */
   @media screen and (min-width: 1024px) {
    body {
      background-color: red;
    }
   }
   @media screen and (max-width: 520px) {
   body {
      background-color: green;
    }
   }
  @media screen and (max-width: 768px) {
  body {
      background-color: black;
    }
  }
  /*
  Media queries allow us to specify different styles to be applied to a webpage depending on the type of media
  (e.g., screen, print, handheld) and the size of the browser.
  */
```

Figure 3 Sample code for adding responsiveness to a web application.

Among the main advantages of responsive web applications is the ability to reutilize code. They require less platform-specific adaptations and they can easily be deployed to run on browsers regardless of the platform. On the contrary, responsive web apps are less attractive because they do not give native app-like feels, client-server interaction has higher latency than native apps, they have limitations on accessing native device capabilities. Moreover, they pose security risks since code is executed through a browser [21,22,23].

3.3 Cross-platform applications

In the world today, there exist a variety of smartphones and tablets which run different platforms. These platforms have their own specific programming language, set of APIs and application development and distribution environments. Developing an application specifically targeting an individual platform requires dedicated resources with platform specific development competency. A business which wants to target multiple platforms should therefore dedicate multiple resources with platform specific competence for the same application. This results in increasing development cost and time for the application. Diversity of handheld device platforms has long been a bottleneck for businesses to target as many application users as possible with viable application development and maintenance cost. Issues like this have given a rise to the cross-platform applications which can be deployed and run on different platforms by reutilizing a single application code base.

Cross-platform applications are generally developed by using platform independent frameworks which provide development APIs in various programming languages. The application utilizes the provided APIs to map UI elements, to persist data and to wire business logics. This application code is then compiled to a native code by using a technique called cross-compilation. Cross-compilations is a way of transforming an application code into platform-specific executable code with the help of a cross-compiler. This executable code is then deployed and executed natively on the targeted platform.

Generally cross-platform applications which are not responsive-web applications can be classified as hybrid web applications, interpreted applications and cross-compiled applications [16,17,24,25].

- **Hybrid web application**: hybrid web application brings together web technologies and native development approaches. The approach utilizes the browser engine in the device and synchronizes the HTML content in the native web containers such as, WebView in android, UIWebView in iOS, platforms. These web containers have access to platform-specific functionalities through APIs. Hybrid applications reutilize code for various platforms and provide access to native device capabilities, however user experiences might be unpleasant due to lack of usage of native UI components and slowness of the app related to loading web containers. Unlike responsive web applications which are accessed via web browsers, hybrid applications are distributed through app stores.
- **Interpreted application**: This kind of applications might use common programming languages to develop application user interface, however code must be interpreted to an equivalent platform native code at runtime to access platform specific APIs. Access to platform specific APIs allow interpreted applications to use native UI elements as a result user experience is generally better than hybrid web applications. On the contrary, there are performance issues due to code conversion during runtime.
- **Cross-compiled application**: For this kind of application, code is compiled to platform specific, high-performance native code with the help of cross compiler. Cross-compiled

applications can provide native feels and performances because they have access to all native UI elements and APIs. However, different platform specific tweaks and configurations might be necessary to access device capabilities such as device audio and video, native calendar applications, device information etc. Cross-compiled applications can be developed with the help of cross-platform mobile application frameworks such as React Native, Flutter etc. The characteristics of React Native and Flutter are discussed in subsequent sections.

When it comes to development, cross-platform applications can be developed by using different tools. These tools can be generally grouped as libraries, frameworks, and platforms, depending on the functionality they provide [24].

- **Library**: usually optimized, self-contained collection of implementations which provide specific functionality such as graphics or UI. A library provides a well-defined interface to invoke functionalities through independent programs. Libraries should coexist with other libraries to make a fully functional mobile application.
- **Framework**: a collection of libraries, software components and architectural guidelines which provide generic functionality. Generally, frameworks contain different libraries, configuration files and APIs for building a fully functional mobile application.
- **Platform**: an integrated development environment which composed a set of frameworks, tools and services for building and distributing mobile applications. platforms usually offer functional tools for debugging, configuring, documenting, and automating an application.

3.3.1 React Native

React Native is an open source framework for developing cross-platform natively compiled mobile applications in JavaScript by using the React JavaScript Library. React Native is based on ReactJS, a JavaScript library for building user interfaces. React was originally developed by Facebook to build user interfaces of web applications. It uses JavaScript XML (JSX), a syntax for embedding XML within JavaScript, to write UI elements of an application. React has since matured with the contributions from different organizations and developers' community to be able to support mobile application development using React Native.

By using React Native one can build a mobile application which has native feels and views, and access platform-specific UI elements from JavaScript code. Unlike hybrid mobile applications, which mainly uses WebView to package application code into platform-specific code, React Native compiles JavaScript application code into platform-specific executable code that can utilize native APIs and UI components. The main building block in a React Native application is the component. Component is the declarative definition of some view in the UI. React Native components are modular which makes them suitable for testing and reutilization of code. React framework provides a representation of the HTML Document Object Model (DOM) in memory, called Virtual DOM. The Virtual DOM allows react to make all necessary changes and computations to reconciles it with real DOM. This way react avoids re-rendering the whole application and applies efficient minimal changes targeting only those HTML DOM elements which have differences with the Virtual DOM.

React Native compiles Virtual DOM UI elements to equivalent platform-specific UI elements during compile-time.

Table 3. shows some of the basic UI elements for web, React Native and their equivalent for the two mainly used mobile platforms namely Android and iOS [26, 28,29].

Table 3 Common UI elements on different platforms.

Web (ReactJS)	**React Native**	**Android**	**iOS**
<input>	<TextInput>	EditText	UITextField
<p>	<Text>	TextView	UITextView
<div>	<View>	Android.view	UIView

Components in React Native

Components are the main building pieces of React Native application. A component in React Native might require other components to function or can stand alone and represent part of the application. Components in React Native can be grouped as presentational and container components. Presentational components are also widely known as stateless components. On that other hand, container components are also called stateful components. Although these components have their own characteristics, their main difference is that presentational components do not hook into any lifecycle methods and have no state of their own. Data flows to the presentation components via react properties (props). Therefore, presentational components are purely used for presenting a content to the user [29,30].

The following are some of the characteristics of presentation components:

- Are concerned with presenting data.
- May hold both presentational and container components in them.
- Have no dependencies on the rest of the application such as data stores.
- Receive data and callbacks via react props.
- Do not have their own data state, rather might have UI state.
- Are highly reusable.

On the contrary, container or stateful components are linked with data stores and actions, which might be generated from these components, influence the state of data. The following are some of the characteristics of container components:

- Are concerned with functioning the application.
- May also hold other presentational or container components.
- Are sources of data and behavior to other container or presentational components.
- Are usually stateful and serve as data sources.
- Have a direct link to the application data state.

State management in React Native

State in a React Native application can be defined as a set of values that a component consumes and manages. States can be used to create and manage data throughout React Native components. A component state is a JavaScript object which is declared when a component is created. A state within a component can be updated using React provided function called *setState*. Recent React versions made it possible for developers to define their own state updating functions with the help of React hooks, however the basic idea of updating a state has not changed. Although states are mainly used to handle application data in container components, it is also possible to handle data for both presentation and container components via React props. Props are passed at component creation as parameters. Unlike states, props cannot be updated within the component itself. When the state of a component changes, react re-renders the component and all other components which inherit that specific state as props [28,29]. Table 4. clarifies some behaviors of State and Props in React or React Native components.

Table 4 State vs. Props [28]

State	**Props**
Internal data	external data
Mutable	Immutable
Created in the component	Inherited from a parent
Can only be updated in the component	Can be updated by parent component
Can be passed down as props	Can be passed down as props
Can set default values inside Component	Can set default values inside Component

Although component states and props can handle data for simple React Native applications, it is hard to handle data precisely and deliberately for more complex systems. Among the very popular third-party libraries React Native uses is Redux. As described by the library creators, Redux is "a predictable state container for JavaScript applications." It is a global state object used as a single source of data in an application and it is passed as props into React Native components. Redux can contain all application states in one place called Redux Store. By the time a certain data is changed in the Redux state, the application receives the new data as props. The three main pillars of Redux Library are Reducers, Store and Actions [28,31].

- **Reducers**. Reducer is a function that returns an object. They create a global state when combined with other reducers. Reducers specify how the application’s state changes in response to actions sent to the store. Reducers can be considered as data stores.
- **Store.** Redux Store is an object which wires actions and reducers together. Moreover, the store holds the application state and allows access and updates to it. Furthermore, the store allows listeners to subscribe and unsubscribe.

- **Actions**. Actions are functions which update reducers by returning an object which sends data to the store. Actions are the only way to mutate data in Redux stores.

Figure 4. illustrates how Redux handles and manages state in React Native applications. The core process is explained as:

1. The only way to manipulate state in Redux is by dispatching an action. Actions are messages passed to the store to facilitate updates in data.
2. After receiving an action, reducer then updates the Redux state. An application can have multiple and combined reducers.
3. when the Redux store receives an update, it passes it to all React Native components which are connected to it and re-rendering happens afterwards. A component can keep looking at store updates by subscribing to it.

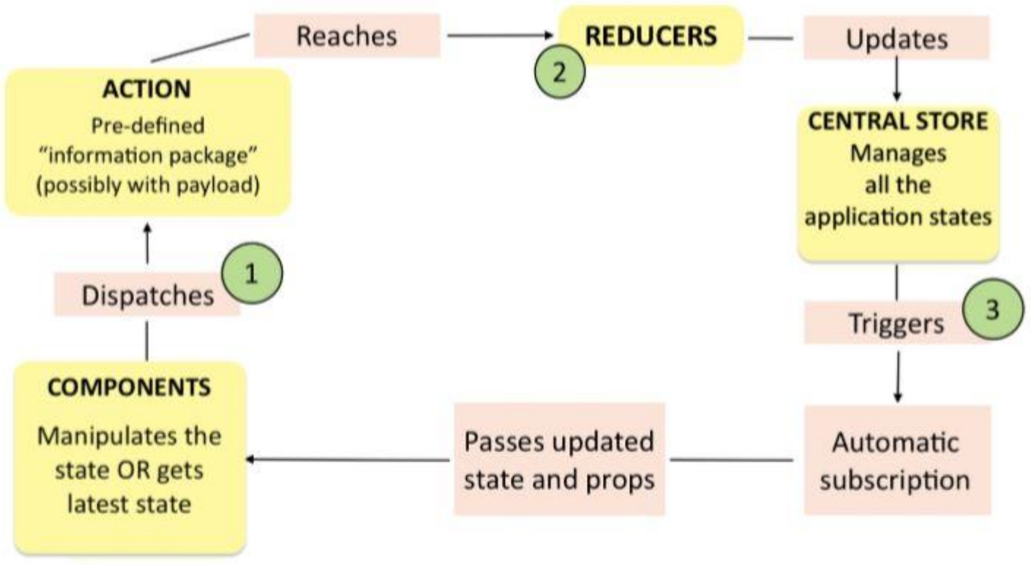


Figure 4 An overview of Redux State Management [31].

Although React Native helps to build cross-platform mobile applications, it significantly depends on third-party libraries when it comes to accomplishing some functionality such as complex state management and navigation. React Native is shipped with basic sets of pre-built components, hence development has higher dependency on third-party libraries. It also requires knowledge of web technologies using React Library. Since React Native is an abstraction on top of platform specific APIs, new changes in native APIs might not be available until they are adopted by React Native or other third-party libraries. Despite these challenges, React Native is constantly improving and adopting new changes. Overall, React Native makes it possible to develop cross-platform applications which have high integration with native platform APIs and has performance and feels closer to native apps.

3.3.2 Flutter

Written in C, C++, and Dart, Flutter is an open-source cross-platform mobile application development SDK developed by Google. It is primarily used for developing applications for mobile platforms such as Android and iOS. In addition to that Flutter can be used for developing Web and desktop applications from a single codebase. Flutter provides a set of fully customizable widgets for building native user interfaces. Among the rich set of widgets Flutter provides are, Material Design library and Cupertino widgets for building UIs and rich motion APIs. These widgets provide ready and customizable functionalities for building up native user interfaces for platforms like iOS. Like React Native, Flutter also provides hot reload, the ability to keep the app running and quickly reflect any code changes edited at runtime, without losing state on the emulators or any hardware for iOS and Android. Flutter ships with a rendering engine, ready-made widgets and developments tools that help accelerate application development. It has a consistent unified Object model, the Widget, which unifies views, view controllers, layouts, and other properties [33,34].

Widgets in Flutter

A widget is a re-utilizable piece of code which describes how an application's UI is constructed. Widgets are used as the fundamental building blocks of Flutter applications. Widgets define UIs by their state and buildup application UI by making a tree-like structure called Widget tree. whenever there is a change in a widget's state, Flutter re-renders the user interface by applying changes in the widget tree since last rendering. Re-rendering happens efficiently because it targets only those with changes from the widget tree. Flutter comes with many built-in widgets which can be easily customized to enhance quick application development. A Widget might be composed of many small, single purpose widgets that combine to produce meaningful and powerful effects, meaning widgets can be nested. They can also have their own properties like styling elements. Widgets are Dart class which extend a proper base class. Everything in Flutter applications is a Widget, a widget can be, a structural element like button or menu, a stylistic element like font, an aspect of layout like margin, etc. Widgets can generally be classified as Stateless and Stateful widgets depending on whether they manage any state. Stateless widgets can be created by extending the *StatelessWidget* base class from Flutter [32]. Below are some of the characteristics of StatelessWidgets [35]:

- StatelessWidgets do not change their appearance and properties throughout their lifetime.
- Extend from *StatelessWidget* class.
- Can be created by overriding the *build()* method.
- Have immutable properties.
- Useful when the UI element is not exposed to dynamic changes.
- Cannot be modified until and unless initialized again.
- Have no internal state of their own.

whereas Stateful widgets are created by extending *StatefulWidget* base class. Following are some of the properties of StatefulWidgets [36]:

- StatefulWidgets change their properties during run-time.
- Extend from *StatefulWidget* class.
- Can be created by overriding the *createState()* method.
- Have mutable properties.
- Useful when the UI element is exposed to dynamic changes
- Have a state of their own.
- Can be modified without reinitialization

State Management in Flutter

A State in Flutter can be defined as a set of values that can be accessed when the widget is built and might be updated or changed throughout the lifetime of the application. Flutter provides *setState()* function for StatefulWidgets that can be used to update the State object. The function allows setting the properties of the State object which triggers re-rendering of the UI. Since Flutter is declarative, it reflects the current state of the applications when building the UI. Conceptually in Flutter, States can be classified as Ephemeral State and App State. Ephemeral States are those states which are scoped to a specific widget. They can be implemented using *State* and *setState()*. They are also called UI or local states to signify that they are defined in a widget itself and other parts of the widget tree rarely need to access these kinds of states. Since Ephemeral States are local to a widget, they can be easily managed and modified by using *setState()* function. On the other hand, App States are those States which could be shared across different widgets of the application. App states are also called global or shared states to signify that they have a global scope in the application. App states are more complex than Ephemeral States, hence it is not recommended to use the simple *setState()* method only to manage and modify them [37,38]. Figure 5. illustrates the difference between Ephemeral and App states briefed above.

Although it is sufficient to use *State* and *setState()* to manage all the state in a simple application, Flutter development team along with a community of contributors provides a range of options for managing state for more complex applications.

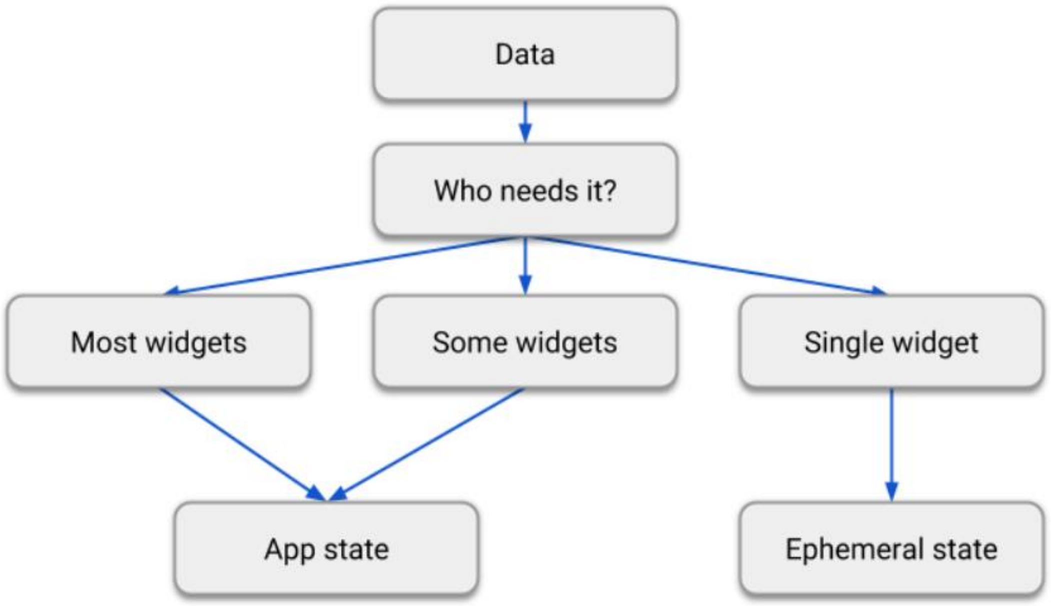


Figure 5 Ephemeral State vs. App State [reprinted from 38].

Among the range of options of state management approaches in Flutter are ScopedModel, Business Logic Component (BLoC) and Redux.

- **ScopedModel:** is a third-party library that provides a set of utilities to facilitate flow of data from a parent widget to its descendants. It also rebuilds all child widgets that use the model whenever the model they consume faces any update. The library provides *Model* class which can be extended during creation of own models and listen to models for changes. It also comes with a *ScopedModel* widget that can wrap the Model created and make it available to all descendent widgets. Moreover, it has a *ScopedModelDescendant* widget which can be used to find a specific ScopedModel in the widget tree. *ScopedModelDescendant* widget will automatically rebuild whenever there is an update in the Model. Although ScopedModel is useful to separate application business logic from presentation logic, it might be challenging to know when to call notifiers to avoid unnecessary updates in more complex Models [39,40].
- **BLoC**: a pattern created by Google to separate business logic from the presentation layer embracing the asynchronous nature of UI elements. BLoC is platform and environment independent which rely on the use of *Streams* for input (*Sink*) and output (*Stream*) data as shown in Figure 6.

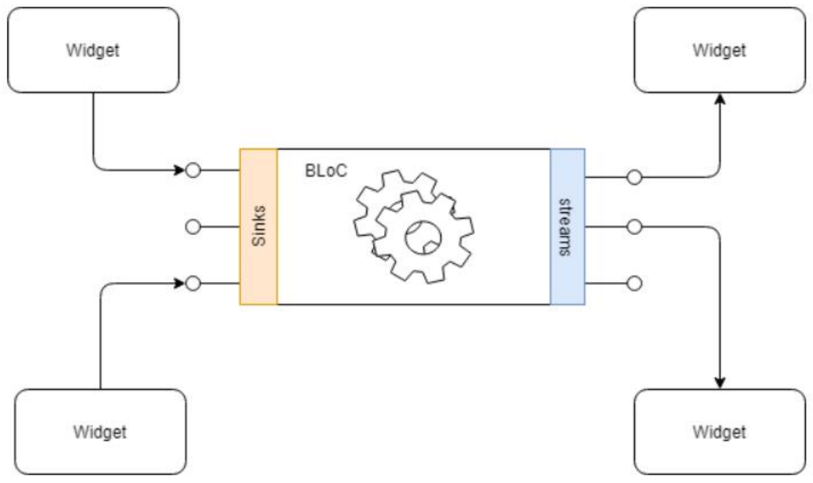


Figure 6 Overview of BLoC pattern [reprinted from 41].

As Figure 6 illustrates, BLoC exposes *Sink* APIs to describe asynchronous inputs to a widget and Widgets send events to the BLoC via *Sinks*. It also exposes *Stream* APIs to describe asynchronous output from a widget and widgets get notified by the BLoC via *Streams*. BLoC provides a *StreamBuilder* widget which can be used to manage streams of data that can maintain automatic subscription of streams and redraw of the widget tree.

Although BLoC is flexible and can be used independent of platform or environment, Bigger applications with many business logics might result in producing many BLoCs that are hard to manage and it could be challenging to properly inject a BLoC that is accessed by several UI widgets. Hence, BLoC is recommended to specially manage local states in complex applications [39,41].

- **Redux**: Redux provides a similar approach to that of BLoC. Any event that is created by user interaction will dispatch an action, which could mutate the inner state of the widget. The core concept of Redux is described in the previous section 3.3.1. Redux comes with tools favorable to managing global state. In Flutter, Redux provides a *StoreProvider* base widget that can pass a given redux store to all its descendants which request it. It also exposes *StoreBuilder*, a descendant widget that accesses data from *StoreProvider* and passes it to a widget. Another important descendant widget Redux provides is *StoreConnector* that gets data from the nearest *StoreProvider* and passes it to a widget *builder* function. Widget automatically re-renders when the Store emits any change event [39,42].

Flutter provides a range of options for managing state of the whole application and a specific widget. The choice relies on the need of state to be visible and accessible by other different widgets in the application and the complexity of the application itself. Table 5. summaries the visibility of states when we follow different state management techniques in Flutter application.

Table 5 State availability when using different state management approaches.

State	**ScopedModel**	**BLoC**	**Redux**
Local scope	maybe	yes	no
Global scope	no	maybe	yes

To sum up, mobile applications can be developed by using different available approaches depending on the need for accessibility and the variety of platforms that the application needs to target. Table 6. describes comparison of some mobile application properties when using different available approaches.

Table 6 Comparison of mobile apps based on widely used approaches.

Application Type	**Platform-specific**	**Responsive Web**	**Cross-platform**
Programming language	Java/Kotlin (Android), Objective C/Swift (iOS)	JavaScript, HTML, CSS	React Native (JSX), Flutter (Dart)
Executable	binary (. apk, .ipa)	JavaScript, HTML, CSS	binary
Accessibility	Appstore, marketplace	via browser, hosted on web servers	Appstore, marketplace
Cross-platform	device specific	cross-platform	cross-platform
Device API	fully accessible	limited accessibility	fully accessible
App speed	very fast	medium	fast
Development or maintenance cost	expensive	relatively low	reasonable
Update /release	with mandatory approval	instant	with mandatory approval

3.3.3 React Native Vs Flutter

Most cross-platform mobile application development frameworks cannot provide seamless user experiences in every aspect. Hence React Native and Flutter come with their own pros and cons. Let us assume a couple of cases to compare what development options each framework provides for solving a certain problem.

Audio and Video

React Native: React native does not ship with core components that work with platform Audio and Video APIs. However, the community provides supporting libraries for Audio and Video. *react-native-sound* is among popular libraries that provide components for playing sound clips on iOS, Android, and Windows. It wraps *AVAudioPlayer* native API on iOS which supports acc, aiff, mps, wa, and other audio files. On Android, the module wraps *android.media.MediaPlayer* to support a range of audio files. On the other hand, playing and recording video is possible by using *react-*

native-video, a third-party library with platform specific configuration for iOS and Android [60,61].

Flutter: The Flutter core team does not provide plugins for sound, however, there exist several work arounds by using third party libraries. Among the popular libraries is *flutter_sound* which supports most audio file formats in both platforms with some platform specific configuration. Although it is still under development, Flutter core team provides a *video_player* plugin for displaying inline video with other flutter widgets on Android and iOS. The team also provides *video_player_platform_interface* a common platform interface for the *video_player* plugin [62,63,64].

3D graphics

React Native: Although React Native does not come with core components to support 3D rendering, there exist some work around by using third party libraries such as *react-native-gl-model-view*, *expo-three*, and others. react-native-gl-model-view allows to display and animate 3D objects using native bridges for iOS and Android. expo-three is a wrapper library for using popular three.js, 3D animation cross-browser library, for React Native. It uses WebGL (Web Graphics Library) for rendering interactive 3D graphics [56,57].

Flutter: According to the official Flutter documentation, Flutter does not support 3D via OpenGL (Open Graphics Library) or similar libraries and Flutter focuses on 2D rendering. However, the documentation mentioned there is a long-term plan to expose an optimized 3D API [55]. Although the fact is that Flutter doesn't support real 3D rendering yet, the *Texture* widget, and *flutter_3d_obj* third party library provide a work around for displaying 3D images by hooking into OpenGL on the platform [58,59].

Access to files

React Native: React Native does not have file access APIs as core components, however, there exist third party libraries which provide work arounds. Among very popular file support React Native libraries is *react-native-fs* which support native filesystem access for Android and iOS. Platform specific configurations are required for setup permission for accessing filesystem [65].

Flutter: Flutter core team provides *path_provider* plugin for interacting with commonly used locations on the Android and iOS filesystems [66].

From the above problems and the most popular solutions developers used to address the problems using React Native and Flutter, both frameworks do not provide complete APIs to interact with platform native APIs that enable native capabilities. Although several factors like framework size can be mentioned for not including most native capabilities and for keeping a framework lightweight with only basic native functionality, React Native seems to heavily depend on third party libraries for solving the above-mentioned example problems. On the other hand, the Flutter core team provides plugins that do not ship with the framework for solving the above-mentioned problems in addition to other third-party plugins that can be used. Table 7 shows a comparison between React Native and Flutter based on most common criteria.

Table 7 React Native Vs Flutter [50,67,58].

	React Native	Flutter
Programming Language	JavaScript (most popular among web developers and easy to adopt to React Native)	Dart (rarely used and less developer community)
Architecture	Flux (uses JavaScript bridge to communicate with native APIs)	SKia (often does not require bridge to communicate with native APIs but bigger in size)
Installation	Via package manager (NPM, HomeBrew)	Binary Download from source
API (UI and beyond)	Less core components	Rich in Widgets
Developer productivity	Depends on JavaScript skill	Depends on Dart skill
Community support	Very high	Rapidly growing
Testing support	relies on third party for integration or UI testing	rich set of packages for integration or UI testing
Build & release automation support	iOS deployment manual from Xcode	command line interface for deployment (Android, iOS)

DevOps and CI/CD support	not so easy CI/CD setup and relies on third party	rich command line interface for easy CI/CD setup

Moreover Table 8. shows some popular applications from different application categories that are crafted using React Native and Flutter. The table discusses the applications and their most common use cases to provide a broader insight of what can be implemented by using these frameworks.

Table 8 Flutter and React Native example Apps and use-cases [75,76]

Category	Example Apps	Use-case
social networking	in10 (*Flutter*) Facebook (*React Native)*	*in10*: create events, add participants from device contacts, show event location on maps, send event notification, track time of arrival of a participant, share event happenings via other social media, etc. *Facebook*: create content, share content, add events, react to posts, attend events, live stream, etc.
Photo & video	PostMuse (*Flutter*) Instagram (*React Native)*	*PostMuse:* personalized Instagram stories and posts creator, Instagram integrated, contains plenty of fonts, free images photo frames, emojis and other photo manipulation tools. *Instagram:* photo and video sharing, comment & like to post by friends, share to groups, react to posts using different emojis, etc.
Health & Fitness	Reflectly (*Flutter*) Gyroscope (*React Native)*	*Reflectly:* Artificial intelligence based self-structuring and reflection app, mental health companion, integrated to device calendar, graphs and charts to present progress, AI based recommendations, group exercises, rates, and share. *Gyroscope:* health tracker app, health Score & grades, daily and weekly reports on exercises, invite and compete with friends, integrated with Apple

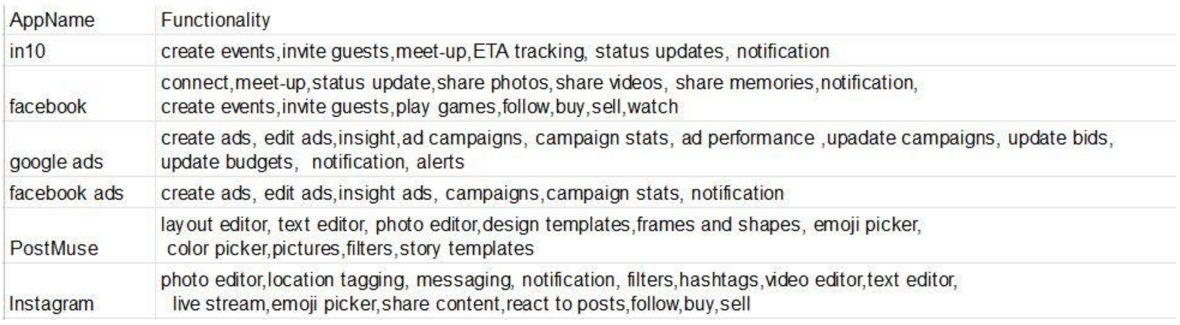
		watch, Mood tracker, sleep AI tracker, places tracker.
Shopping	Xianyu (Alibaba) (*Flutter*) Walmart (*React Native)*	Xianyu: e-commerce, buy, sell Walmart: e-commerce, buy
Music	Topline (*Flutter*) Sound cloud (*React Native)*	*Topline:* Record and save song ideas, play, edit, convert (formats), share. *Sound cloud:* audio-sharing, upload audio, share audio, follow, listen to audio, comment, and respond.
Travel	Flydirekt (*Flutter*) Townske (*React Native)*	*Flydirekt:* check for flights, load estimation, Weather forecasts *Townske:* share and discover cities, recommend cities, rate cities, like cities, bookmark travel guide.
Business	Google Ads (*Flutter*) Facebook Ads (*React Native)*	*Google Ads:* online advertising platform by Google *Facebook Ads:* online advertising platform by Facebook.

As seen from Table 8, businesses can implement competitive and similarly functional apps which can fall under the same category by using React Native and Flutter. It shows the capability of both frameworks for different categories of mobile applications.

3.3.4 Clustering Mobile Applications

With the boom in mobile applications, it is necessary to categorize applications based on some variables to provide easy of choice for users. Most mobile application marketplaces tag application categories based on the theme developers provide during application release. However, there have been criticisms of this theme-based approach of categorization that most app stores use, for failing to group apps according to the features they exhibit. This kind of manual assessment of broad themes might label applications far from their intended functionality and lead users to download wrong applications. There has been research that suggests a way of improving app categorizations based on the features description the applications provide [77]. Hence this section of the thesis reinforces the suggested feature based mobile apps clustering by using sample mobile applications.

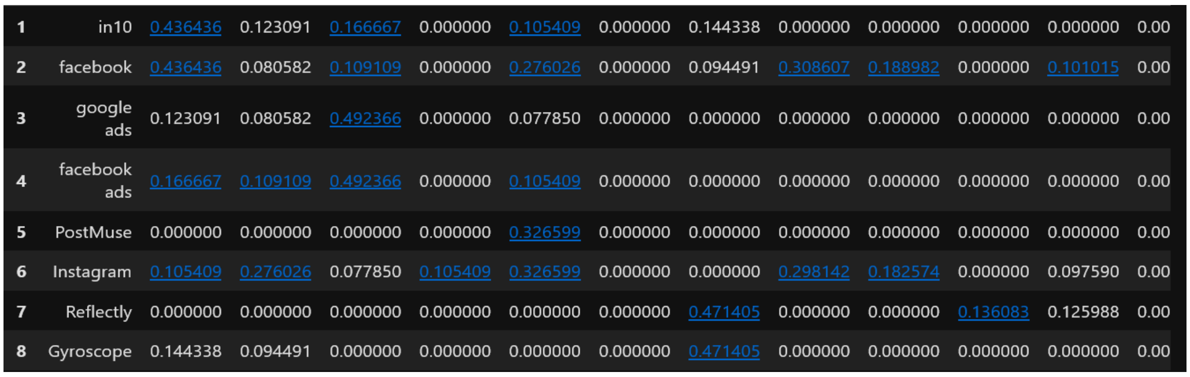
The selection of samples focuses on apps that are built by using React Native or Flutter frameworks that are used in section 3.3.3. The features of the mobile applications are extracted from their respective descriptions in app stores and are manually processed for analysis as shown in Figure 7. The whole data set is presented in Appendix B.



AppName	Functionality
in10	create events,invite guests,meet-up,ETA tracking, status updates, notification
facebook	connect,meet-up,status update,share photos,share videos, share memories,notification, create events,invite guests,play games,follow,buy,sell,watch
google ads	create ads, edit ads,insight,ad campaigns, campaign stats, ad performance ,upadate campaigns, update bids, update budgets, notification, alerts
facebook ads	create ads, edit ads,insight ads, campaigns,campaign stats, notification
PostMuse	layout editor, text editor, photo editor,design templates,frames and shapes, emoji picker, color picker,pictures,filters,story templates
Instagram	photo editor,location tagging, messaging, notification, filters,hashtags,video editor,text editor, live stream,emoji picker,share content,react to posts,follow,buy,sell

Figure 7 Sample data set showing Apps and their functionality.

After extracting the features, a set of unique features is constructed as a *corpus*. The set of features are then converted to a vector for each app and the value of each element in the vector represents the count of each element in the respective app's features. At this stage, the vector contains only *1s* and *0s* which represent the presence and absence of a feature from the corpus, respectively. To further prepare the data set for analysis, the vector is transformed by calculating the *cosine distance* of features of an app to the rest of the apps' features. The cosine distance is used to relate the similarity of features between the mobile applications. At this stage, the data is well processed, and it is used for clustering of the apps. Figure 8 shows the sample of the final data set, and the whole vector used for clustering is presented in Appendix C.

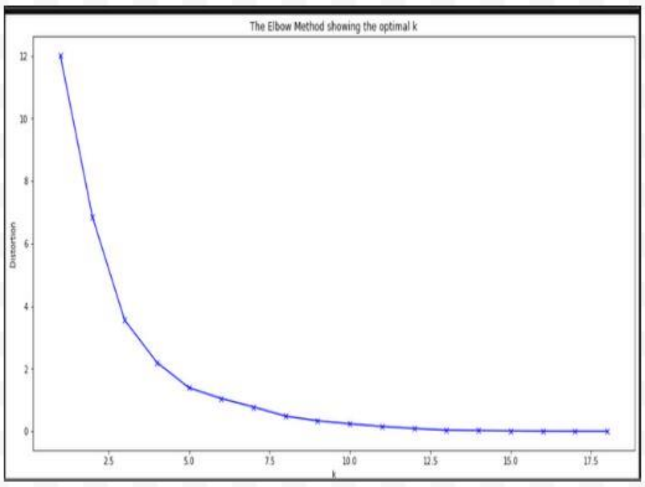


1	in10	0.436436	0.123091	0.166667	0.000000	0.105409	0.000000	0.144338	0.000000	0.000000	0.000000	0.000000	0.00
2	facebook	0.436436	0.080582	0.109109	0.000000	0.276026	0.000000	0.094491	0.308607	0.188982	0.000000	0.101015	0.00
3	google ads	0.123091	0.080582	0.492366	0.000000	0.077850	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
4	facebook ads	0.166667	0.109109	0.492366	0.000000	0.105409	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
5	PostMuse	0.000000	0.000000	0.000000	0.000000	0.326599	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
6	Instagram	0.105409	0.276026	0.077850	0.105409	0.326599	0.000000	0.000000	0.298142	0.182574	0.000000	0.097590	0.00
7	Reflectly	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.471405	0.000000	0.000000	0.136083	0.125988	0.00
8	Gyroscope	0.144338	0.094491	0.000000	0.000000	0.000000	0.000000	0.471405	0.000000	0.000000	0.000000	0.000000	0.00

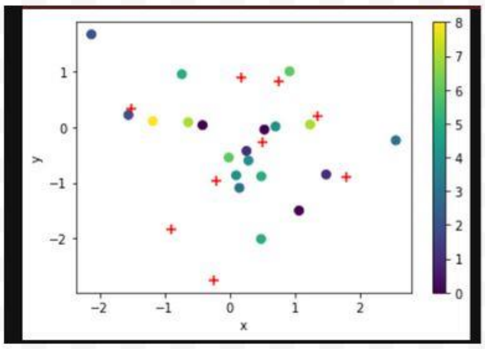
Figure 8 Sample data showing cosine distance between apps' features.

By using the final data set presented in Appendix C, *Scikit-learn* machine learning library is used to perform *K-means clustering* technique for clustering the apps based on their claimed features. The optimal number of clusters for the clustering is estimated by using the widely used approach *elbow-method*. Based on the approach the optimal number of clusters is estimated to be at a point where the curve starts to flatten. Hence, the number of clusters *k* is chosen to be nine (*k=9*) as shown in Figure 9a. After determining the number of clusters, clustering is performed on the data set. The result is presented in Figure 9b, which shows the apps being clustered into nine groups based on their features. Moreover, Figure 9c. shows scattered plots of the clusters with their respected cluster centroids.

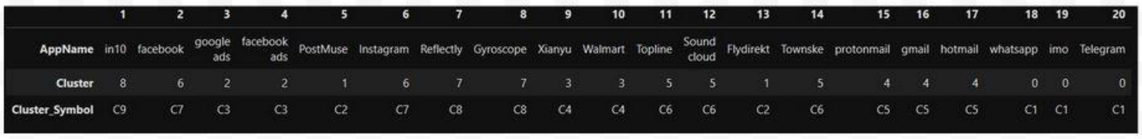
Figure 9b. shows samples of instant messaging applications such as *WhatsApp*, *imo* and *telegram* being clustered as *C1*. In the app stores these applications are categorized as broad *Social Networking* applications which made them fall in the same category as *Facebook* and *in10* like applications. *C2* contains *PostMuse* and *Flydirekt* apps showing closer similarity based on their features, whereas in the app stores these apps are categorized as *photography* and *travel*, respectively. *C3* contains a cluster of *Google ads* and *Facebook ads* which have closer intended functionalities. In Addition to that *C4* contains more related e-commerce applications *Walmart* and *Xianyu*. Another cluster *C5* contains email communication applications such *Protonmail*, *Gmail* and *Hotmail*. *C6* clusters *Sound cloud*, *Topline* and *Townske* apps together. However, the app *Townske* is categorized as *travel and local* in the app store. The clustering put *Facebook* and *Instagram* together as *C7* which reflect more similarity in features between these apps. *C6* clusters health related applications *Reflectly* and *Gyroscope* together.



a



c



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
AppName	in10	facebook	google ads	facebook ads	PostMuse	Instagram	Reflectly	Gyroscope	Xianyu	Walmart	Topline	Sound cloud	Flydirekt	Townske	protonmail	gmail	hotmail	whatsapp	imo	Telegram
Cluster	8	6	2	2	1	6	7	7	3	3	5	5	1	5	4	4	4	0	0	0
Cluster_Symbol	C9	C7	C3	C3	C2	C7	C8	C8	C4	C4	C6	C6	C2	C6	C5	C5	C5	C1	C1	C1

b

Figure 9 Plots and result of K-means clustering.

On the other hand, *in10* is an outlier which is clustered as *C9*. This app has less related features to other apps in the data set. Although app features were manually extracted and refined, in general this sample analysis shows the apps being clustered primarily on similarities of features.

Moreover, an interesting insight could be observed at clusters *C7* and *C2*. *C7* clusters *Facebook* and *Instagram* applications together, both of which are developed by using React Native framework. The features representing *Facebook* and *Instagram* in the sample clustering include video and photo sharing, instant reaction to posts, videos and pictures, instant status updates, instant messaging, browser through contents to gain some knowledge about a context, among others. Besides faster network connection, these types of features require an application to be faster. The fastness of an application might come from an optimized way of handling those functionalities by the framework and the light weightiness of the framework itself. On the other hand, *C2* clusters *PoseMuse* and *Flydirekt* together, both of which are developed using the Flutter framework. The features representing these two applications in the sample clustering include searching or filtering of an item such as (editor, story template, frame, flight, weather forecast) from the list of items and use it either to gain knowledge of a context or to create and modify content. These kinds of applications require a framework which provides an efficient way of search and filtering of an item from list of items.

4 IMPLEMENTATIONS

This section of the thesis presents the implementation of global COVID-19 [44] pandemic tracking mobile application. Although the medical explanation of the pandemic is beyond the scope of this thesis, the application is developed mainly because there are free public REST API providers on the topic. The mobile application is developed by using the two most widely used cross-platform mobile application frameworks React Native and Flutter. The application contains two navigation screens namely "NCOV19 SUMMARY" as landing page and "COUNTRY STATS" as detailed page. The landing page "NCOV19 SUMMARY" displays global number of confirmed cases, deaths, and recovery from the pandemic. In addition to that, it contains external links "MYTH BUSTERS" and "DONATE", which lead to WHO (World Health Organization) website pages for clarification of myth and contribution to the pandemic, respectively. On the other hand, the detailed page "COUNTRY STATS" provides a list of countries with related statistics on the pandemic and the ability to search for a specific country from the list of countries.

The application developed by the two frameworks consumes the same REST API data sources and it follows the same design principle for both frameworks to avoid indiscriminate performance comparisons in the following chapter. Moreover, Visual studio code editor [47], is used for writing and editing code. Unfortunately, the application will not be published to Google play store or Apple’s app store due to their application policies on COVID-19. However, application source code from both frameworks can be requested from related project repositories and the application is guaranteed to work so long as the API providers continue to server data [45, 46].

4.1 React Native mobile application

This section details the React Native implementation of the two features of the mobile application: NCOV19 SUMMARY and COUNTRY STATS. NCOV19 SUMMARY feature presents some useful global statistics on COVID-19 pandemic and links to the WHO website. Pressing an item other than links leads to the COUNTRY STATS page where specific country statistics on the pandemic is presented. On the other hand, pressing on a link from the NCOV19 SUMMARY page opens a related WebView from the WHO website. For the matter of presentation, the implementation is divided into UI, Navigation and Data and State. The UI part mainly focuses on how the application UI is constructed in React Native. Whereas the Navigation part discusses what makes navigating from one screen to another screen possible. Furthermore, the Data and state part presents the way of fetching data from REST API and making it available for presentation in related views. The whole application features are presented in Appendix A.

UI

In React Native applications UIs are constructed by using different UI components provided by either React Native library or by third party libraries. Like every other UI framework React Native provides some basic set of UI components such as View, Text, Image, etc. Apart from React Native, third party libraries such as React Native Elements provide a range of alternative UI components. These UI components will then be mapped to platform specific UI elements during compile time. Figure 10 shows some React Native equivalent UI elements in different platforms.

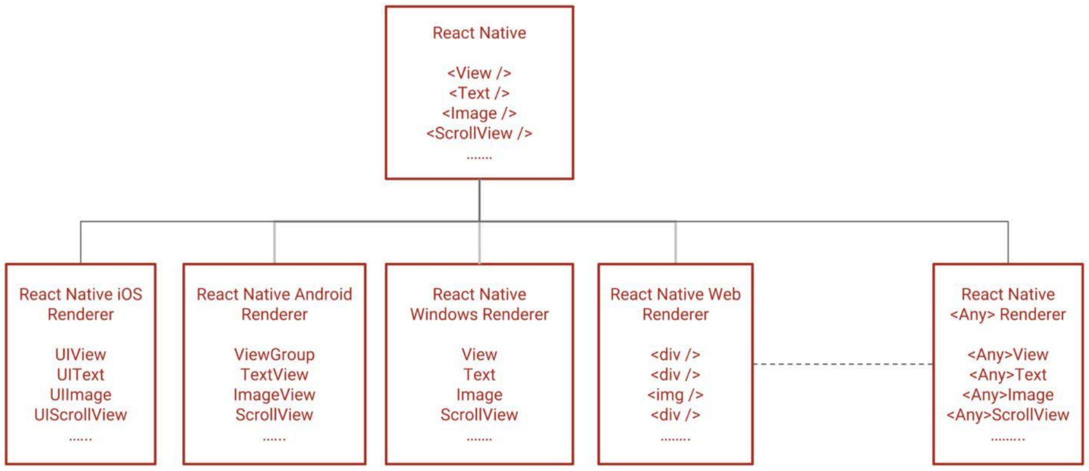


Figure 10 React Native equivalent UI elements in different platforms [48].

The React Native part of the mobile application is built by using some of the core UI components provided by React Native. Figure 11 shows different UI components used to construct NCOV19 SUMMARY screen for the mobile application.

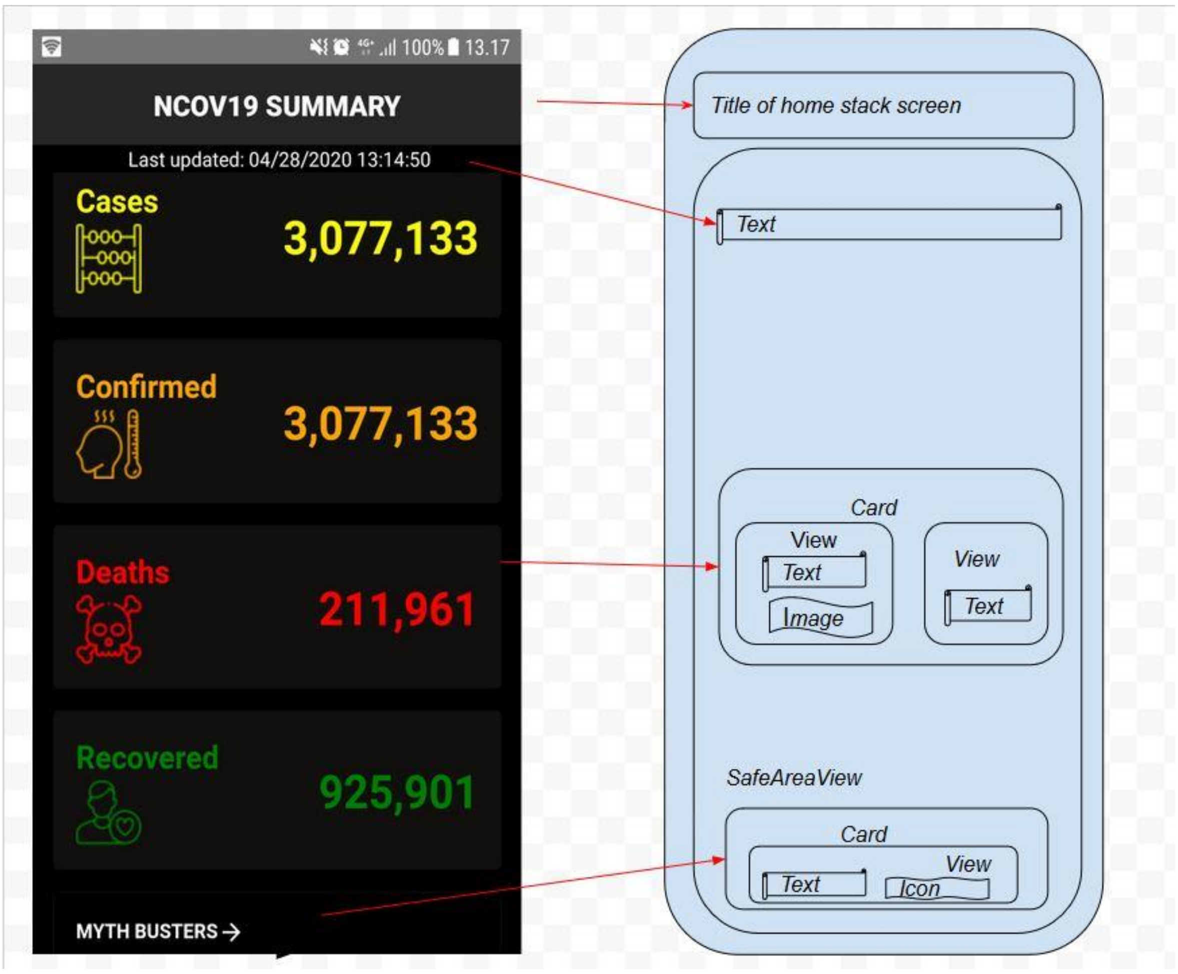
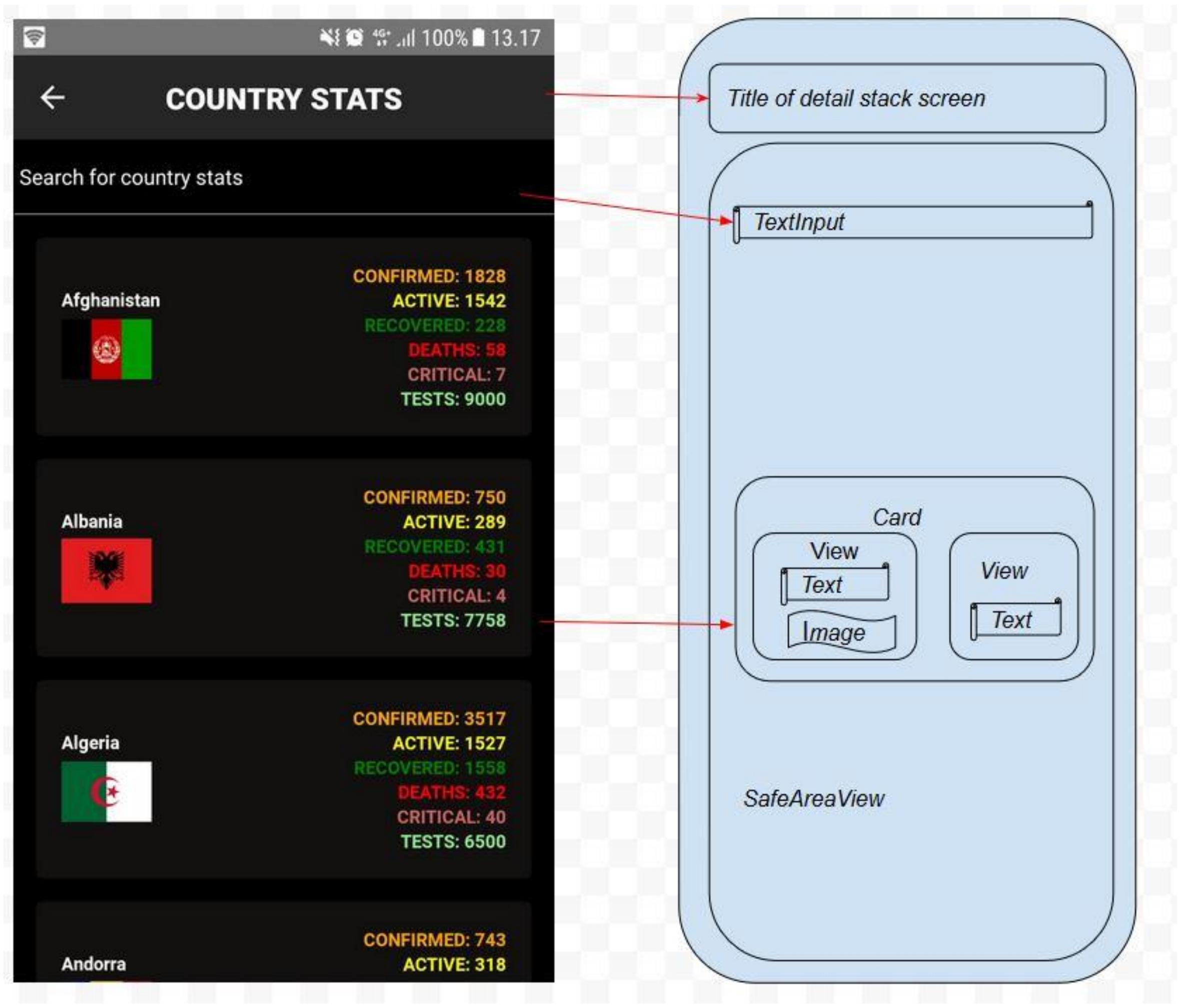


Figure 11 Some UI components used to construct NCOV19 SUMMARY screen.

As shown in Figure 11, the title in each screen is the value of the title attribute on each stack screen which is a *Text* UI element. The entire body of NCOV19 SUMMARY screen is a *SafeAreaView* component which contains *Text* and *FlatList* components as children. Each item in the *FlatList* is presented as a *Card*. Each *Card* contains a *ListItem* child component which makes it render left and right items. The left item uses *Text* and *Image* components to render the text and related image, respectively. The COUNTRY STATS screen of the application is constructed in a similar fashion by using *SafeAreaView* container component which renders its children *TextInput* and *FlatList* components. *TextInput* component made possible for users to type text input to search for specific country statistics. Whereas the *FlatList* renders a *ListItem* of each country and related statistics as a *Card*. Figure 12 shows some UI components used for building the COUNTRY STATS screen.


13.17

COUNTRY STATS

Search for country stats

Afghanistan
CONFIRMED: 1828
ACTIVE: 1542
RECOVERED: 228
DEATHS: 58
CRITICAL: 7
TESTS: 9000

Albania
CONFIRMED: 750
ACTIVE: 289
RECOVERED: 431
DEATHS: 30
CRITICAL: 4
TESTS: 7758

Algeria
CONFIRMED: 3517
ACTIVE: 1527
RECOVERED: 1558
DEATHS: 432
CRITICAL: 40
TESTS: 6500

Andorra
CONFIRMED: 743
ACTIVE: 318

Figure 12 Some UI components used to construct COUNTRY STATS screen.

Navigation

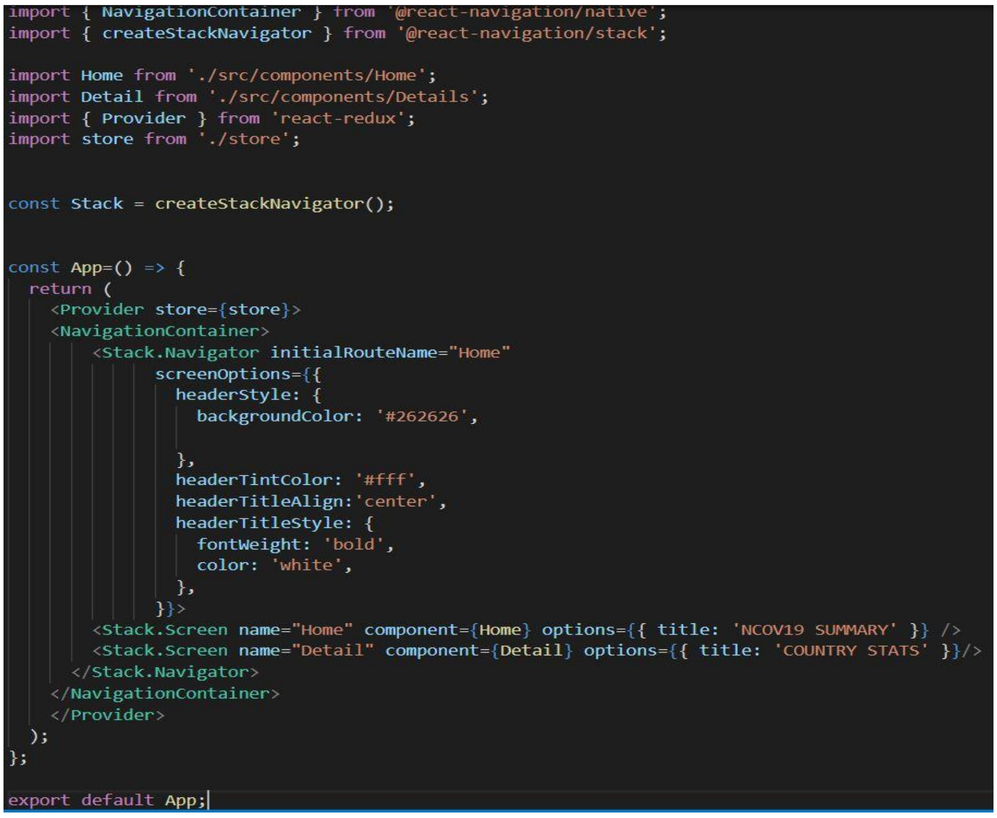
Mobile apps are usually made up of multiple screens. Managing the presentation of, and transition between these screens is handled by what is known as a navigator. In React Native applications this can be achieved by using the most widely used library called React Navigation. React Navigation provides a cross-platform React Native application with the ability to present stack and tab navigation patterns on both Android and iOS platforms.

The implementation of the React Native application for this thesis uses React Navigation (version 5.x) for creating a navigator and facilitating smooth transition between available screens. According to the library creators React Navigation is easy to use routing and navigation for React

Native applications by providing customizable navigation components which could enable platform-specific look-and-feel with smooth animations and gestures [49].

For the React Navigation to work properly, all dependence libraries must be installed. After installing the dependencies, the whole app is wrapped in a *NavigationContainer* component which is imported from React Navigation. The *NavigationContainer* component is essential in managing the application’s navigation tree and contains the navigation state.

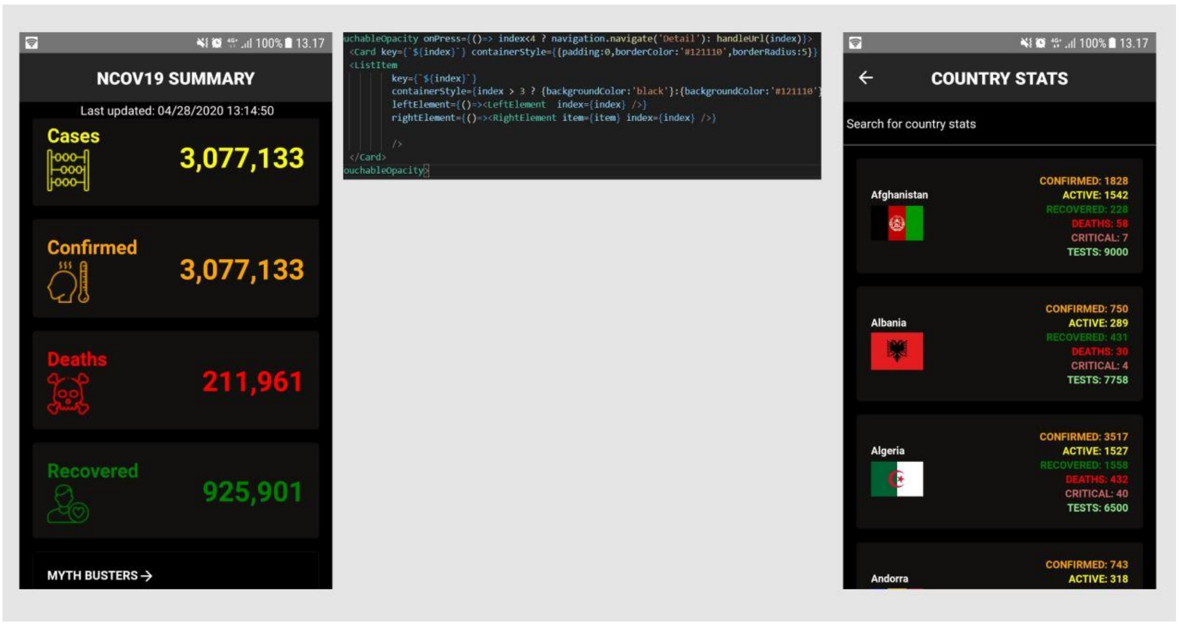
Another useful function used is *createStackNavigator*, which returns *Screen* and *Navigator* React components used for configuring the navigator. *createStackNavigator* provides a way for the app to transition between screens where each new screen is placed on top of the created stack. Stack Navigator in React Native is configured to have iOS and Android look and feel [49]. Figure 13 illustrates code snippets from the React Native application which shows how navigator was configured in the root of the application.



```
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

import Home from './src/components/Home';
import Detail from './src/components/Details';
import { Provider } from 'react-redux';
import store from './store';


const Stack = createStackNavigator();


const App=() => {
  return (
    <Provider store={store}>
    <NavigationContainer>
        <Stack.Navigator initialRouteName="Home"
              screenOptions={{
                headerStyle: {
                  backgroundColor: '#262626',

                },
                headerTintColor: '#fff',
                headerTitleAlign:'center',
                headerTitleStyle: {
                  fontWeight: 'bold',
                  color: 'white',
                },
              }}>
        <Stack.Screen name="Home" component={Home} options={{ title: 'NCOV19 SUMMARY' }} />
        <Stack.Screen name="Detail" component={Detail} options={{ title: 'COUNTRY STATS' }}/>
      </Stack.Navigator>
    </NavigationContainer>
    </Provider>
  );
};

export default App;
```

Figure 13 Configuration of navigator at the root of the application.

As seen in Figure 13, the body of the application's root component (App.js) is wrapped by the *NavigationContainer* which will manage the navigation tree. Furthermore, Stack navigator is created with two screens **Home** and **Detail,** which represents the Home and Detail components of the application. Creating stack navigation is possible by the *createStackNavigation* from React Navigation library. We then provide the navigator’s initial route name which represents a component to navigate to when the application launches. Each screen in the stack is given a title which will appear in the navigation bar when the screen is mounted. After the configuration of the Navigator, the *navigation* state object is globally available for all application components. It is used to trigger navigating from NCOV19 SUMMARY screen to COUNTRY STATS screen. Figure 14 shows how the *navigation* state is used to trigger navigating between two screens.



```
uchableOpacity onPress={()=> index<4 ? navigation.navigate('Detail'): handleUrl(index)}>
 <Card key={`${index}`} containerStyle={{padding:0,borderColor:'#121110',borderRadius:5}}
 <ListItem
         key={`${index}`}
         containerStyle={index > 3 ? {backgroundColor:'black'}:{backgroundColor:'#121110'}
         leftElement={()=><LeftElement  index={index} />}
         rightElement={()=><RightElement item={item} index={index} />}

         />
 </Card>
ouchableOpacity>
```

Afghanistan
CONFIRMED: 1828
ACTIVE: 1542
RECOVERED: 228
DEATHS: 58
CRITICAL: 7
TESTS: 9000

Albania
CONFIRMED: 750
ACTIVE: 289
RECOVERED: 431
DEATHS: 30
CRITICAL: 4
TESTS: 7758

Algeria
CONFIRMED: 3517
ACTIVE: 1527
RECOVERED: 1558
DEATHS: 432
CRITICAL: 40
TESTS: 6500

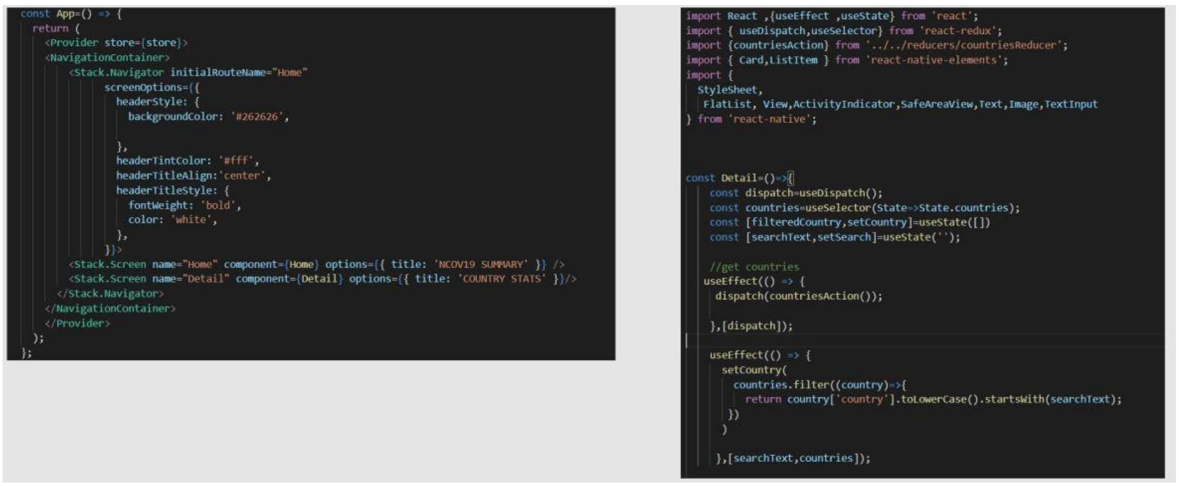
Andorra
CONFIRMED: 743
ACTIVE: 318

Figure 14 Code snippet for navigating between screens

As seen in Figure 14, pressing a *Card* UI element from the NCOV19 SUMMARY screen will navigate to the COUNTRY STATS screen. This is possible by passing the desired screen name to the *navigate* function from the *navigation* state object that is globally available after configuring the navigator in the root of the application.

Data and State

Data for the React Native application is fetched from free COVID-19 public REST APIs provided by WHO [43,3]. The React Native application was designed to use the Redux library for controlling data flow and managing State. The detailed explanation for the usage of the Redux library in React Native applications is discussed in section 3.3.1. The Redux store which contains the combinations of reducers that hold different states is then made available to the application by wrapping the root of the application with *Provider* component from react-redux. To get a specific state from the store, the application uses the *useSelector* hook provided by redux library. In addition to that, the application uses *useDispatch* hook to dispatch relevant action from a component to a reducer. The reducer then updates a state related to the dispatched action. Figure 15. illustrates code snippets of how the *Provider* is used to make redux *Store* available throughout all the components in the application and how countries data was consumed from the store with related action being dispatched to update the initial countries state with the help of *useEffect* react hook.



```
const App=() => {
  return (
    <Provider store={store}>
    <NavigationContainer>
        <Stack.Navigator initialRouteName="Home"
              screenOptions={{
                headerStyle: {
                  backgroundColor: '#262626',

                },
                headerTintColor: '#fff',
                headerTitleAlign:'center',
                headerTitleStyle: {
                  fontWeight: 'bold',
                  color: 'white',
                },
              }}>
        <Stack.Screen name="Home" component={Home} options={{ title: 'NCOV19 SUMMARY' }} />
        <Stack.Screen name="Detail" component={Detail} options={{ title: 'COUNTRY STATS' }}/>
      </Stack.Navigator>
    </NavigationContainer>
    </Provider>
  );
};
```

```
import React ,{useEffect ,useState} from 'react';
import { useDispatch,useSelector} from 'react-redux';
import {countriesAction} from '../../reducers/countriesReducer';
import { Card,ListItem } from 'react-native-elements';
import {
  StyleSheet,
   FlatList, View,ActivityIndicator,SafeAreaView,Text,Image,TextInput
} from 'react-native';



const Detail=()=>{
    const dispatch=useDispatch();
    const countries=useSelector(State=>State.countries);
    const [filteredCountry,setCountry]=useState([])
    const [searchText,setSearch]=useState('');

    //get countries
   useEffect(() => {
     dispatch(countriesAction());

    },[dispatch]);

    useEffect(() => {
      setCountry(
        countries.filter((country)=>{
          return country['country'].toLowerCase().startsWith(searchText);
        })
      )

     },[searchText,countries]);
```

Figure 15 Provider and redux hooks in action.

4.2 Flutter mobile application

This section presents the Flutter implementation of the mobile application. All application features that are implemented in React Native are also implemented using Flutter. Apart from having similar features, both implementations use the same REST APIs as data sources and the UI elements are made as equivalent as possible to avoid irrational comparison in the following chapter.

Unlike React Native which depends on components, the Flutter application uses Widgets to construct the application UI. Widgets can be considered as the equivalent of React Native components, which are primarily used for building up user interfaces. Flutter provides a rich set of widgets as core packages, whereas React Native ships with some basic core UI components and the React Native community provides alternative libraries as external packages that can be downloaded for use.

UI

Flutter categorizes layout widgets for constructing UIs as Single-child layout widgets and Multi-child layout widgets. Single-child layout widgets are those widgets which can accommodate a single widget within themselves. Whereas Multi-child layout widgets are those which can contain a list of other child widgets [51]. Like the React Native implementation, the Flutter implementation of the mobile application consists of two UI screens namely, NCOV19 SUMMARY and COUNTRY STATS.

As shown in Figure 16, the entire screen for NCOV19 SUMMARY is scaffolded by using Flutter's *Scaffold* widget, which contains *AppBar* and *ListView* widgets as children. The *AppBar* widget accommodates the *Text* widget as a child that makes the title of the screen appear on the top. On the other hand, the *ListView* widget contains *Text*, *Card* and *Container* widgets as children. The *Text* is responsible for displaying 'Last updated' date and time as a text. The *Card* widget that contains *Text* and *Image* widgets as children is responsible for displaying useful statistics such as global number of confirmed cases and thumbnail images as a *ListView* item. Moreover, the *Container* widget which composes *Text* and *Icon* widgets a child is responsible for displaying related text and icon for links to the WHO page.

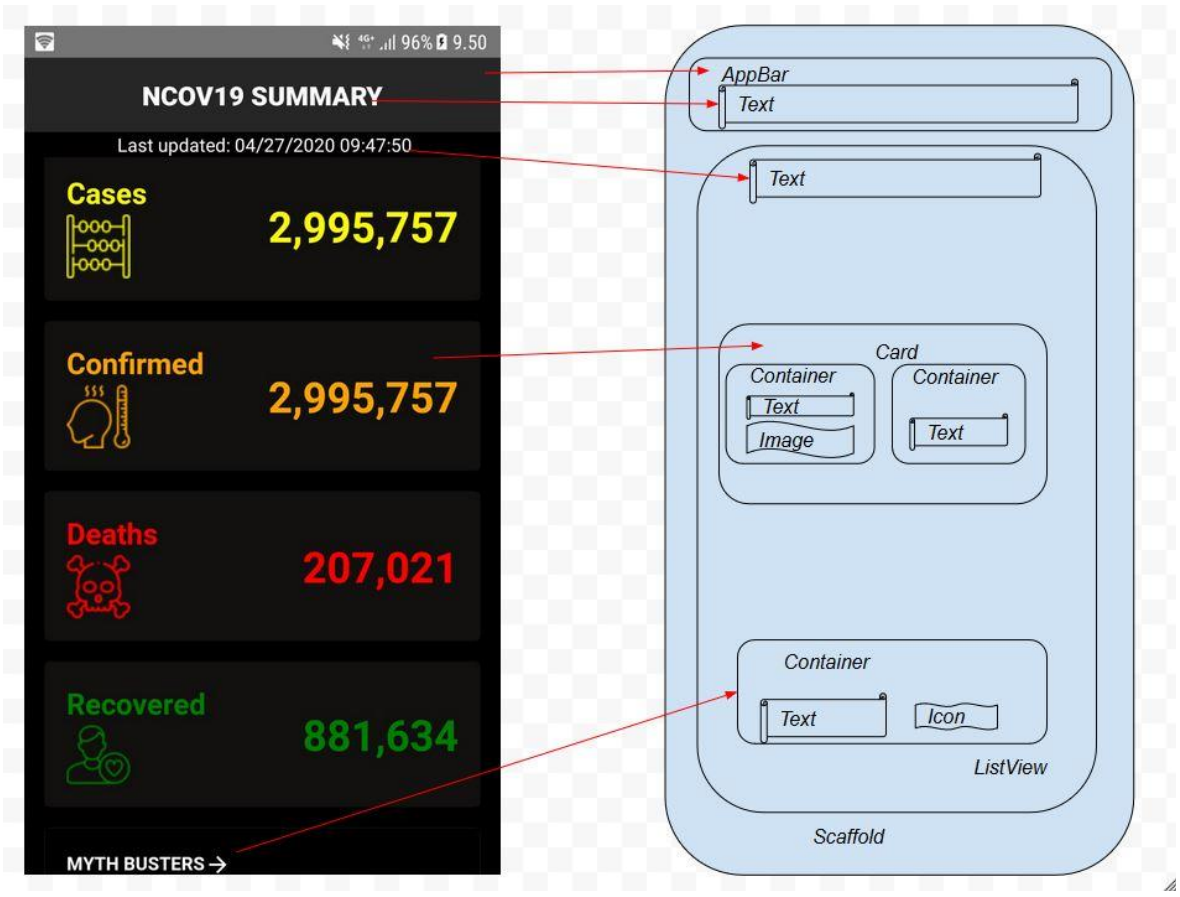


Figure 16 NCOV19 SUMMARY screen and related Flutter widgets for the UI.

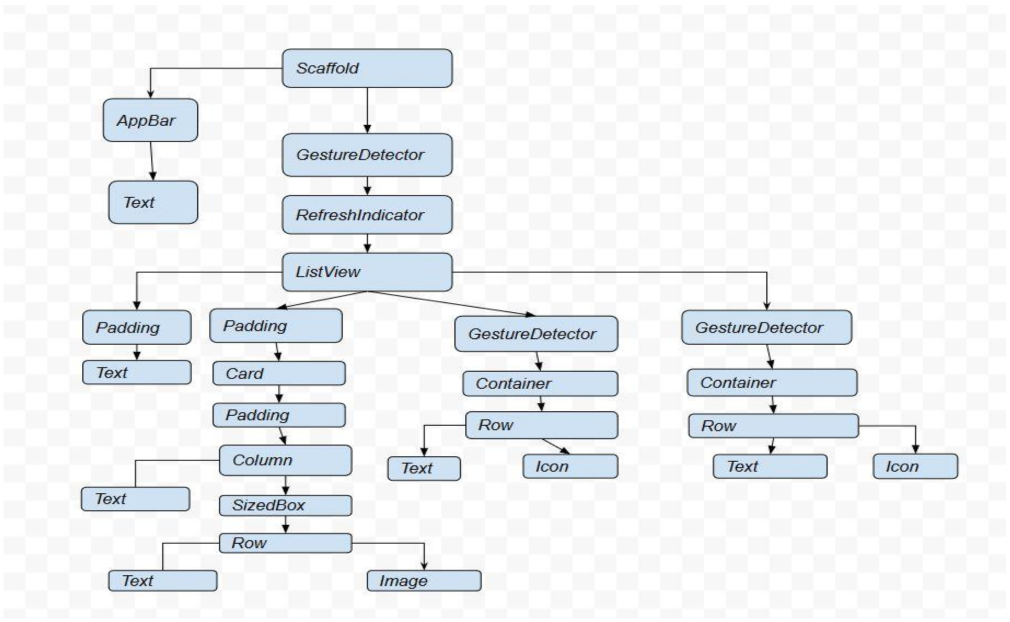
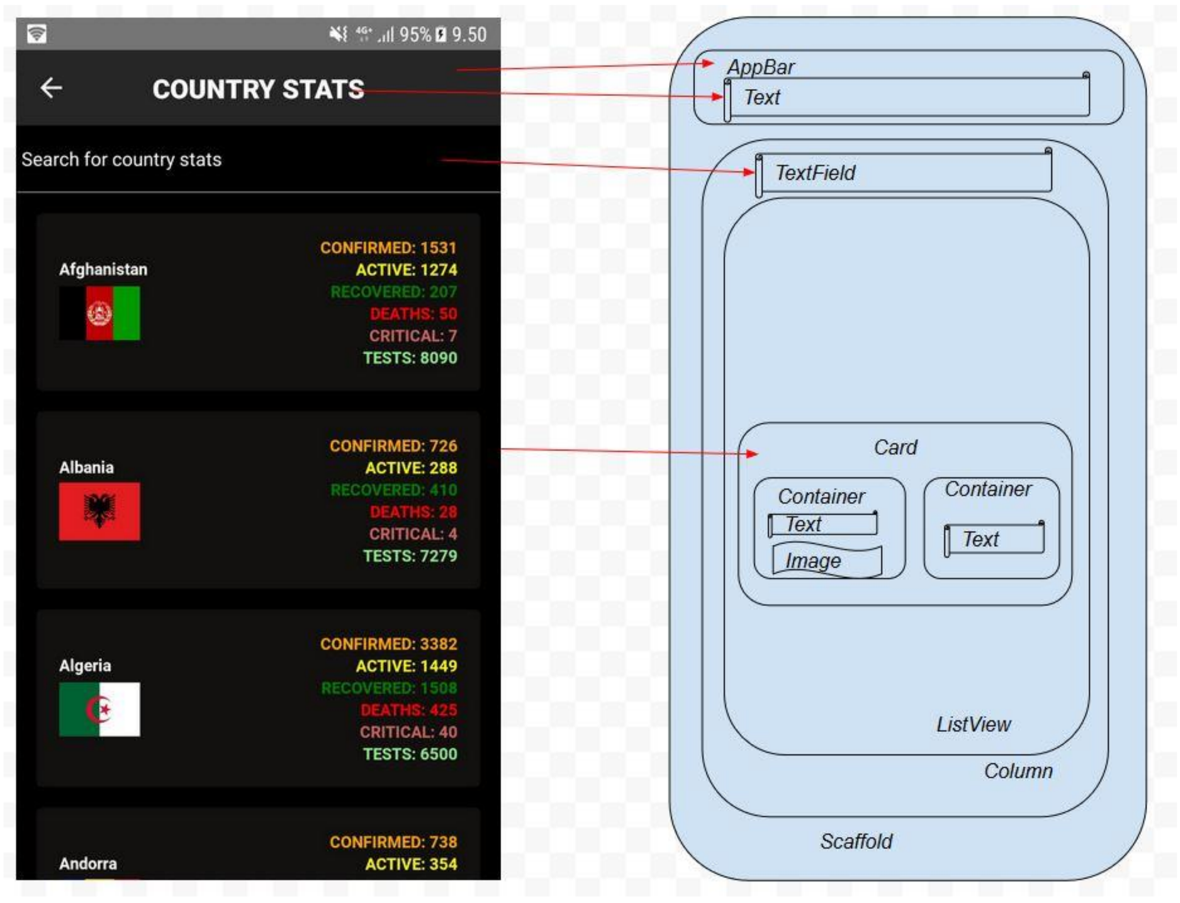


Figure 17 Widget tree for NCOV19 SUMMARY screen.

Although Figure 16. contains main UI widgets which make up the specified screen, there are other widgets used to wrap up the main UI widgets for the purpose of styling the UI and adding necessary interactivity. As shown in Figure 17, widgets like *Padding*, *Column*, *Row* and *SizedBox* are used for styling and arranging contents. On the other hand, *GestureDetector* and *RefreshIndicator* widgets are used to add Tap and Pull to refresh events to the content, respectively.

The second screen, COUNTRY STATS, is constructed in a similar fashion as the first one. One important interaction added to this screen is the ability to search for a specific country's statistics by writing the country name in the *TextField* widget.


95% 9.50

COUNTRY STATS

Search for country stats

Afghanistan
CONFIRMED: 1531
ACTIVE: 1274
RECOVERED: 207
DEATHS: 50
CRITICAL: 7
TESTS: 8090

Albania
CONFIRMED: 726
ACTIVE: 288
RECOVERED: 410
DEATHS: 28
CRITICAL: 4
TESTS: 7279

Algeria
CONFIRMED: 3382
ACTIVE: 1449
RECOVERED: 1508
DEATHS: 425
CRITICAL: 40
TESTS: 6500

Andorra
CONFIRMED: 738
ACTIVE: 354

Figure 18 COUNTRY STATS screen and related Flutter widgets for the UI.

The *TextField* widget has attributes to detect change of text in the search box and related action will be triggered to update the state (countries list data) with the match filtered country from the list of countries. Figure 18. illustrates COUNTRY STATS screen and main UI widgets used for building it up. Apart from the main UI widgets used for constructing the specified screen, other styling and interaction widgets are used for aligning content and adding necessary interactive events to the continent. Figure 19 presents the whole Widget tree for the COUNTRY STATS screen.

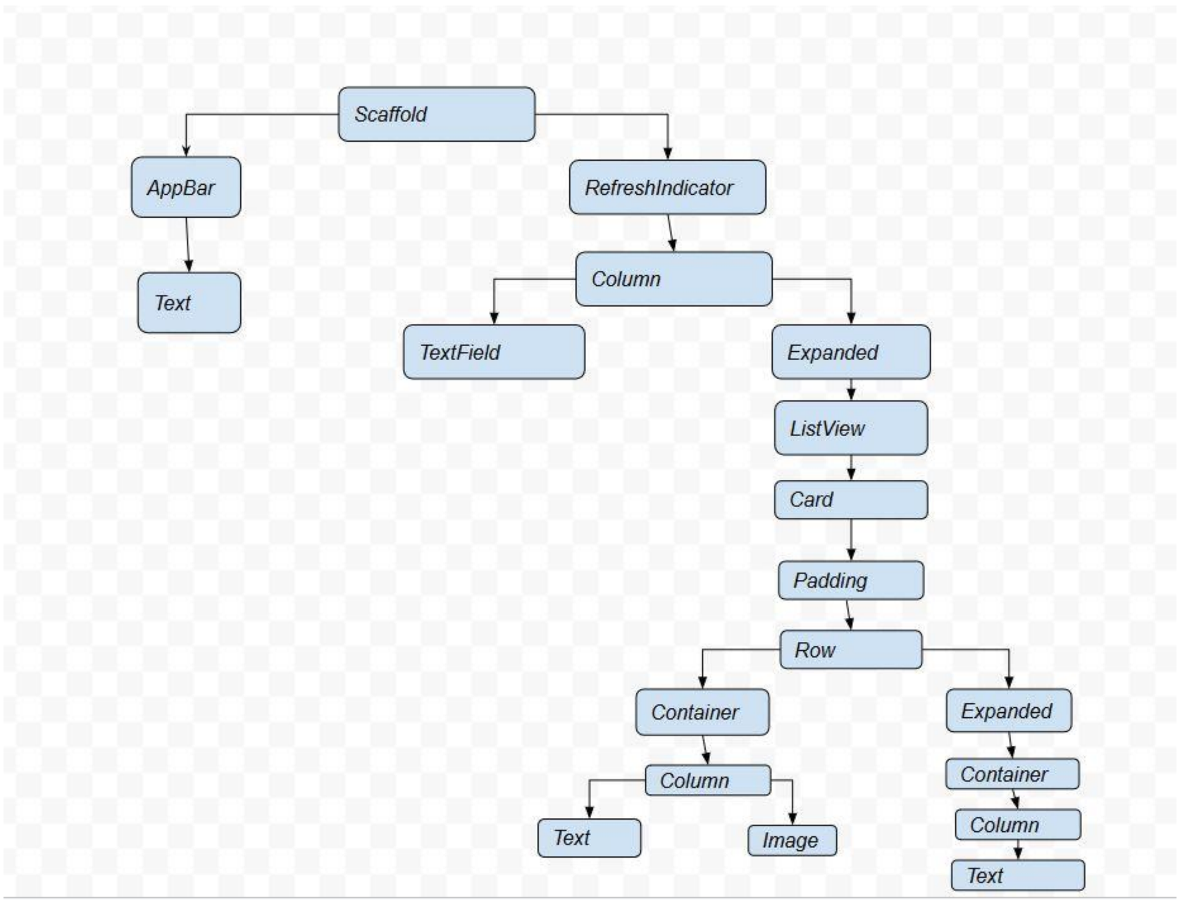
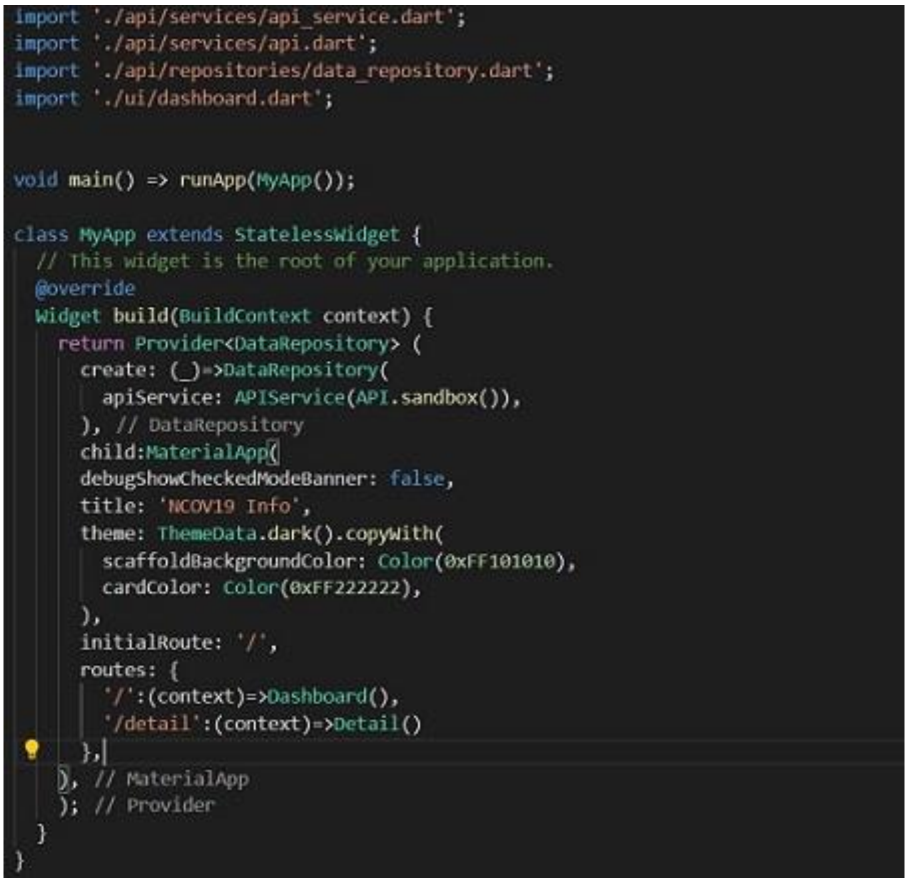


Figure 19 Widget tree for COUNTRY STATS screen.

As seen in the widget tree, widgets like *Column*, *Expanded*, *Padding* and *Row* are used as wrappers for styling contents. whereas *RefreshIndicator* widget is used for adding pull to refresh interactivity for fetching updated data from the source.

Navigation

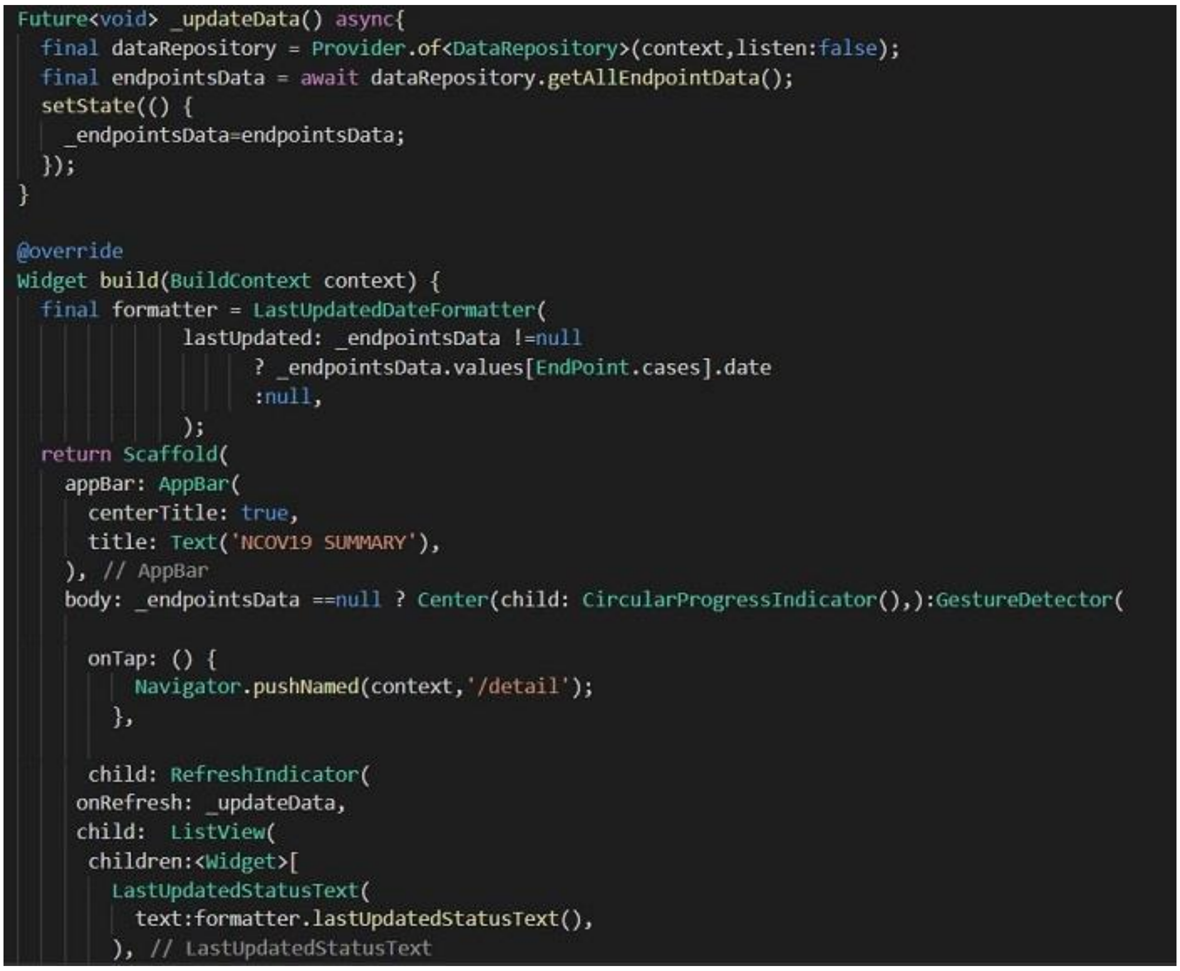
Like the React Native implementation, the Flutter implementation of the application also contains configuration of *routes* and the ability to navigate from one screen to another screen. In Flutter *screens* and *pages* are referred to as routes. The equivalent of routes in native android is called an *Activity*, whereas in iOS it is called *ViewController*. As everything in Flutter application is composed of widgets, a route in Flutter is another widget. Routes in Flutter application must be configured in the root of the application with default *initialRoute* that refers to the first screen to load when the application starts and with other routes and related screens [52]. Figure 20 shows code snippets of how routes were configured in the Flutter implementation of the mobile application.



```
import './api/services/api_service.dart';
import './api/services/api.dart';
import './api/repositories/data_repository.dart';
import './ui/dashboard.dart';


void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return Provider<DataRepository> (
      create: (_)=>DataRepository(
        apiService: APIService(API.sandbox()),
      ), // DataRepository
      child:MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'NCOV19 Info',
      theme: ThemeData.dark().copyWith(
        scaffoldBackgroundColor: Color(0xFF101010),
        cardColor: Color(0xFF222222),
      ),
      initialRoute: '/',
      routes: {
        '/':(context)=>Dashboard(),
        '/detail':(context)=>Detail()
      },
    ), // MaterialApp
    ); // Provider
  }
}
```

Figure 20 Routes configuration in the Flutter implementation of the app.

Flutter provides another important widget called *Navigator*. Navigator widget manages a set of child widgets with a stack discipline. Hence, the navigator manages a stack of *Route* objects and provides methods such as *Navigator.push*, *Navigator.pushNamed* and *Navigator.pop* for managing the stack and the ability to switch visibility between screens. *Navigator.push* or *Navigator.pushNamed* methods are used to put a navigation context to the top of the stack. whereas, *Navigator.pop* method is used to remove context from the stack [53]. Figure 21, a code snippet shows how navigating from the NCOV19 SUMMARY to the COUNTRY STATS screen page is made possible.

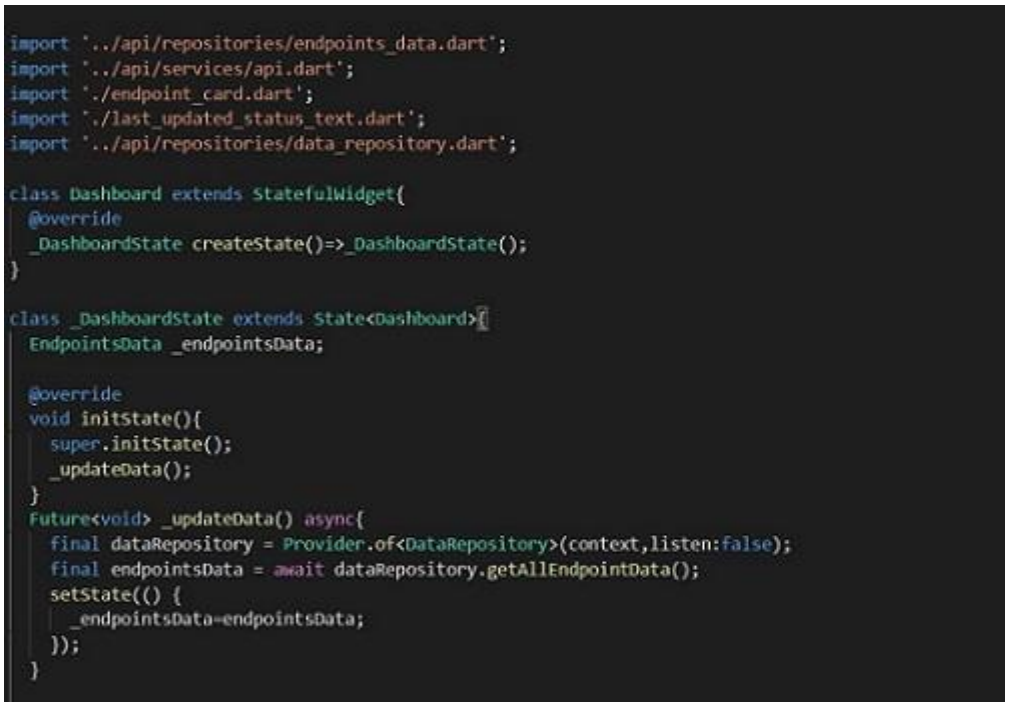


```
Future<void> _updateData() async{
  final dataRepository = Provider.of<DataRepository>(context,listen:false);
  final endpointsData = await dataRepository.getAllEndpointData();
  setState(() {
    _endpointsData=endpointsData;
  });
}

@override
Widget build(BuildContext context) {
  final formatter = LastUpdatedDateFormatter(
            lastUpdated: _endpointsData !=null
                  ? _endpointsData.values[EndPoint.cases].date
                  :null,
            );
  return Scaffold(
    appBar: AppBar(
      centerTitle: true,
      title: Text('NCOV19 SUMMARY'),
    ), // AppBar
    body: _endpointsData ==null ? Center(child: CircularProgressIndicator(),):GestureDetector(

      onTap: () {
        Navigator.pushNamed(context,'/detail');
      },

      child: RefreshIndicator(
     onRefresh: _updateData,
     child:  ListView(
      children:<Widget>[
        LastUpdatedStatusText(
          text:formatter.lastUpdatedStatusText(),
        ), // LastUpdatedStatusText
```

Figure 21 Usage of *Navigator.pushNamed* for navigation.

Data and State

As discussed in section 3.3.2, there are different approaches of managing state in application crafted using Flutter. The Flutter implementation of the application uses the *Provider* Flutter widget for wrapping context and consuming data that is fetched from the REST API. *Provider* is simple to use state management widget which guarantees unidirectional data flow with a possibility to override or update a value or state. *Provider* widget contains methods such as *Consumer* and *of*, which can listen to the data source for any data change and rebuild the UI according to the implementations. Calling the *Consumer* method on the *Provider* will rebuild a widget apart from consuming data from the source. Whereas calling the method *of* on *Provider* with the *listen* flag to *false* will not rebuild the widget but can still be able to access data from the source [54]. Figure 22. shows code snippets of how the *Provider* widget is implemented to access data from the source.



```
import '../api/repositories/endpoints_data.dart';
import '../api/services/api.dart';
import './endpoint_card.dart';
import './last_updated_status_text.dart';
import '../api/repositories/data_repository.dart';

class Dashboard extends StatefulWidget{
  @override
  _DashboardState createState()=>_DashboardState();
}

class _DashboardState extends State<Dashboard>{
  EndpointsData _endpointsData;

  @override
  void initState(){
    super.initState();
    _updateData();
  }
  Future<void> _updateData() async{
    final dataRepository = Provider.of<DataRepository>(context,listen:false);
    final endpointsData = await dataRepository.getAllEndpointData();
    setState(() {
      _endpointsData=endpointsData;
    });
  }
```

Figure 22 Configuration of *Provider* widget in the application.

5 PERFORMANCE COMPARISON

This chapter details a performance comparison between the mobile application implemented by using React Native and Flutter. As discussed in the previous chapter, both applications are made as similar as possible to avoid indiscriminate comparisons. However, there exist slight differences on UI element usages, text dimensions and other stylings which have no significant effect on the performance comparison between the applications.

The aim of this chapter is to address the second objective of the thesis, which is to find out if an application implemented by using React Native and Flutter has any performance differences by evaluating CPU (Central Processing Unit), GPU (Graphics Processing Unit) and MEMORY usage of the application. The comparison is limited to iOS and Android platforms. The chapter first discusses the performance analysis when both React Native and Flutter implementations of the application are run on an iOS device, followed by when the same applications are run on an Android device. To simulate the realistic use of the applications, both applications are run on a real device with 4G data connection. The android device used to run the applications is Samsung SM-A510F, whereas the iOS device used to run the applications is iPhone 7 (13.4.1).

The comparison is done by taking peak measurements of MEMORY, CPU and GPU when a user is experiencing all the functionalities available on NCOV19 SUMMARY and COUNTRY STATS screens. Five different tests are run to take measurements and the arithmetic mean of these tests is taken as representative measurement. The following are four main activities when measurements are taken:

- Scrolling cards: This test focuses on scrolling through NCOV19 SUMMARY screen cards.
- Opening webview: a test for taping on the MYTH BUSTERS link and opening WHO website.
- Rendering listview: focuses on rendering lists of countries on COUNTRY STATS screens, which also includes network requests to REST API and loading images from URL.
- Filtering a list: it tests for filtering for specific country statistics from a list of countries.

5.1 iOS application

For the iOS application, the performance analysis is done by the help of Apple's "Instruments" tool, which is an application performance analyzer and visualizer, integrated in Xcode IDE [70]. Instruments contain a set of tools which can be used for checking the performance of iOS applications. Among the variety of tools instruments provide, "Time profiler tool" is used for analyzing CPU usage of the applications. The GPU, the number of frames rendered per second, is counted by the help of "Core Animation Tool". Moreover the "Allocations Tool" is used to analyze the memory usage of an iOS application in MiB (mebibyte). Figure 23 illustrates samples of Instruments tools in action.

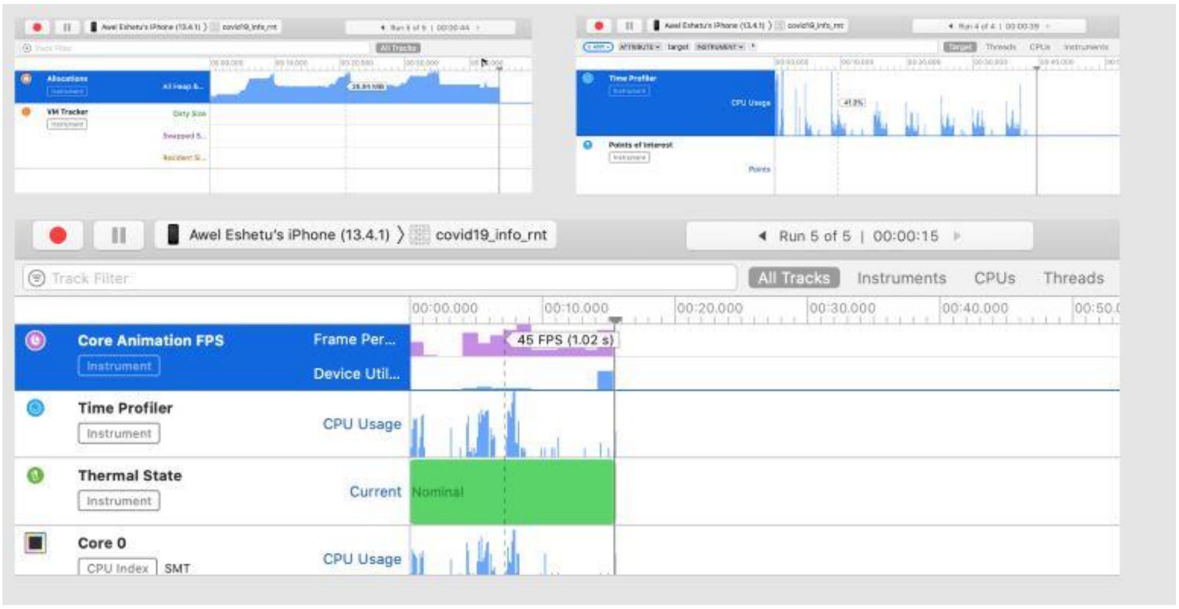


Figure 23 Sample Instruments tools in action.

5.1.1 CPU

Figure 24 shows the CPU usage of the Flutter mobile application compared to that of the React Native application. The measurements are taken by using the "Time profiler tool" from Apple's Instruments. The representative measurement for each framework is taken from the arithmetic mean of five test runs.

The outcome of the CPU usage tests are as follows:

- *Scrolling cards:* The React Native iOS application uses 8.6 % more CUP than the Flutter one.
- *Opening webview:* As seen from the graph, the React Native application consumes 17 % more CPU than the Flutter application.
- *Rendering listview*: Rendering list of countries in the React Native application uses 79.6 % more CPU than rendering list of countries in Flutter application.
- *Filtering:* Filtering for a specific country's statistics uses 87.4 % more CPU in React Native than in Flutter.

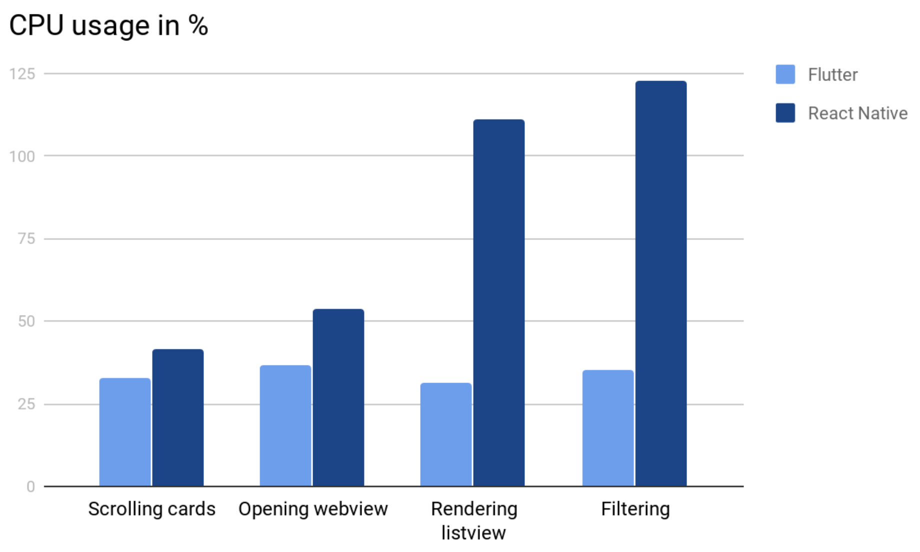


Figure 24 CPU usage bar chart, Flutter Vs React Native of the iOS application.

5.1.2 Memory Usage

The memory usage analysis for the iOS application is done by using an Instruments tool called the "Allocations Tool". Unlike Android, Apple measures memory usage by mebibyte (MiB). Like the CPU usage analysis, the arithmetic mean of five tests is taken as the representative measurement. Figure 25 illustrates the Memory usage of the Flutter mobile application compared to that of the React Native application.

The outcome of the Memory usage tests are as follows:

- *Scrolling cards:* As seen from the graph the Flutter application uses 41.32 MiB memory than React Native.
- *Opening webview:* Opening the webview in Flutter uses 40.04 MiB more memory than React Native.
- *Rendering listview:* Rendering list of countries in Flutter uses 38.8 MiB more memory than doing the same thing by using React Native.
- *Filtering:* Filtering for specific country's information from the list of countries in Flutter consumes 19.31 MiB more memory than React Native.

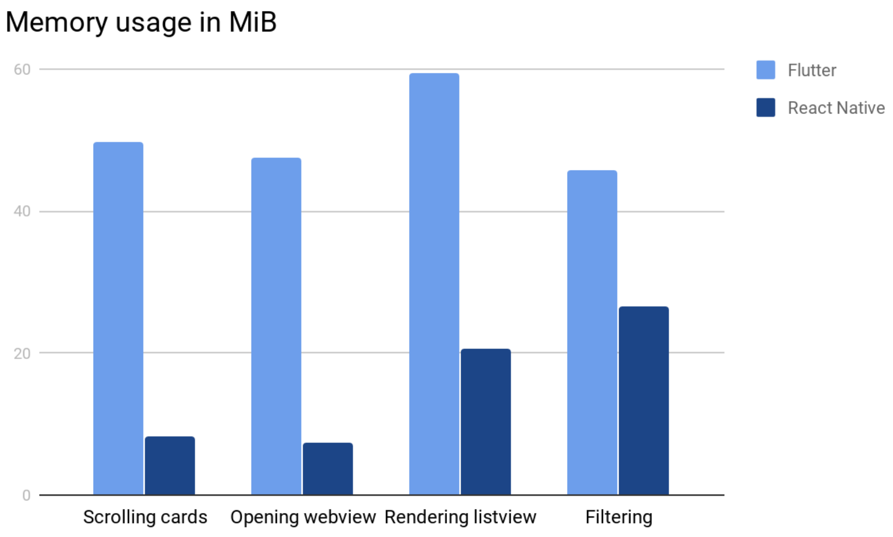


Figure 25 Memory usage bar chart, Flutter Vs React Native of the iOS application.

5.1.3 GPU

Among the great tools Apple's Instruments provide is the "Core Animation Tool". Core Animation tool measures rendering and animation of views in an application by using FPS (frames per second). Figure 26 shows the GPU usage of the Flutter iOS application compared to that of the React Native application. Like the CPU and Memory usages analysis, the arithmetic mean of five tests is taken as the representative GPU measurement.

The result of the GPU usage tests are as follows:

- *Scrolling cards:* Scrolling items of cards in Flutter renders 1.2 more frames per second than React Native.
- *Opening webview:* Opening links in a webview in Flutter renders 0.2 more frames per second than React Native.
- *Rendering listview:* React Native renders 13 more frames per second than Flutter when rendering list of countries.
- *Filtering:* During filtering for country statistics from list of countries, Flutter renders 8.4 frames per second than React Native.

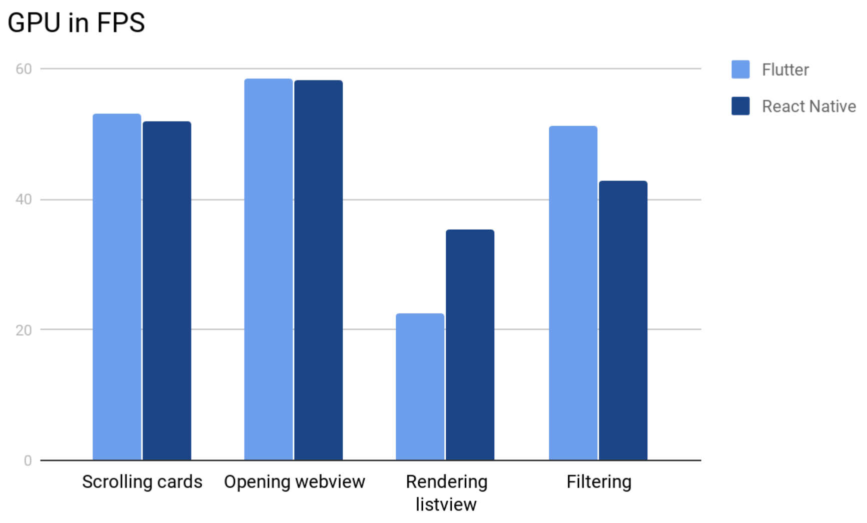


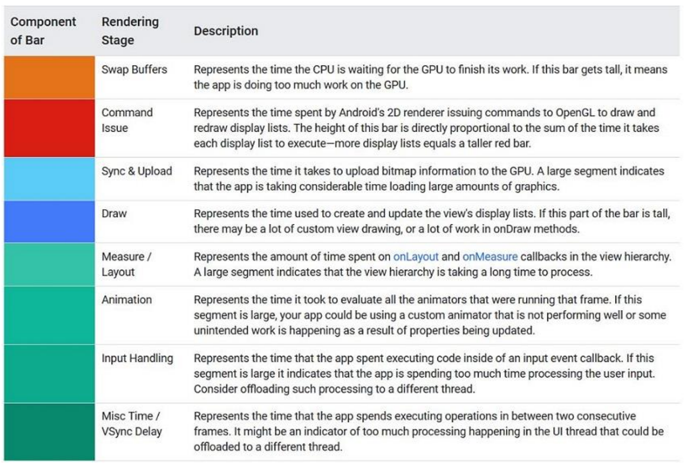
Figure 26 GPU usage bar chart, Flutter Vs React Native of the iOS application.

5.2 Android application

Performance analysis of an Android application is done with the help of tools that are available on Android Studio. Among the useful tools which are built-in to Android Studio is Android profiler. The Profiler contains a set of advanced profiling tools which help in diagnosing android application performances. By using the Profiler one can get useful information about CPU, Memory, Network and Energy usages of an android application. Real-time information is displayed to the Android studio profiler once the user has connected to an Android device with

USB debugging enabled. When the user browses through the app, Android profile will be able to detect the process and display related CPU, Memory, Network and Energy usages of that specific process.

Although Android studio provides many useful tools, it does not offer a tool for monitoring the GPU usage of an app. Hence for monitoring GPU usage, “GPU rendering tool” is used. GPU rendering tool is a built-in tool that is available in any Android smartphone. When activated, the tool displays a visual representation of how much time it takes to render the frames of a UI window relative to a benchmark of 16ms (millisecond) per frame as a scrolling histogram [69]. Figure 27 provides descriptions of each segment of a vertical bar in the profiler output when using a device running Android 6.0 and higher.

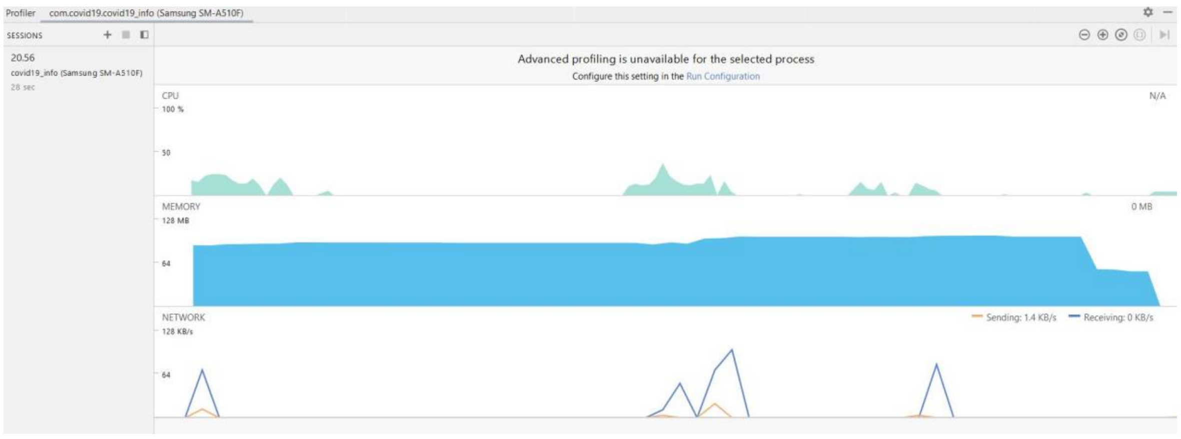


Component of Bar	Rendering Stage	Description
🟧	Swap Buffers	Represents the time the CPU is waiting for the GPU to finish its work. If this bar gets tall, it means the app is doing too much work on the GPU.
🟥	Command Issue	Represents the time spent by Android's 2D renderer issuing commands to OpenGL to draw and redraw display lists. The height of this bar is directly proportional to the sum of the time it takes each display list to execute—more display lists equals a taller red bar.
🟦	Sync & Upload	Represents the time it takes to upload bitmap information to the GPU. A large segment indicates that the app is taking considerable time loading large amounts of graphics.
🟦	Draw	Represents the time used to create and update the view's display lists. If this part of the bar is tall, there may be a lot of custom view drawing, or a lot of work in onDraw methods.
🟩	Measure / Layout	Represents the amount of time spent on onLayout and onMeasure callbacks in the view hierarchy. A large segment indicates that the view hierarchy is taking a long time to process.
🟩	Animation	Represents the time it took to evaluate all the animators that were running that frame. If this segment is large, your app could be using a custom animator that is not performing well or some unintended work is happening as a result of properties being updated.
🟩	Input Handling	Represents the time that the app spent executing code inside of an input event callback. If this segment is large it indicates that the app is spending too much time processing the user input. Consider offloading such processing to a different thread.
🟩	Misc Time / VSync Delay	Represents the time that the app spends executing operations in between two consecutive frames. It might be an indicator of too much processing happening in the UI thread that could be offloaded to a different thread.

Figure 27 GPU profiler component bars in Android 6.0 and higher (reprinted from [69]).

The GPU Rendering tool displays a bar graph for each visible application. The height of each vertical bar, which represents a frame, is the amount of time the frame took to render in milliseconds. The horizontal green line drawn per each process is the threshold of 16 milliseconds. Every vertical bar which is below this line indicates an achievement of 60 frames per second GPU rendering [69]. Figure 28 shows Profile GPU Rendering graph with a threshold line. On the other hand, Figure 29 shows a sample of Android studio profiler while profiling CPU, MEMORY and NETWORK of the application implemented using Flutter.

Figure 28 Sample Profile GPU Rendering graph [31].



Profiler
com.covid19.covid19_info (Samsung SM-A510F)
SESSIONS
20:56
covid19_info (Samsung SM-A510F)
28 sec
Advanced profiling is unavailable for the selected process
Configure this setting in the Run Configuration
CPU
100 %
50
N/A
MEMORY
128 MB
64
0 MB
NETWORK
128 KB/s
64
Sending: 1.4 KB/s
Receiving: 0 KB/s

Figure 29 Sample Android Studio Profiler in action.

5.2.1 CPU

Figure 30 illustrates the percentage CPU usage of the Flutter mobile application compared to that of the React Native application. The representative measurement for each framework is taken from the arithmetic mean of five test runs. The outcome of the CPU usage test are as follows:

- *Scrolling Cards*: as seen from the graph, the Flutter application uses 9.4% more CPU than the React Native when scrolling through cards of items.
- *Opening webview*: Opening up webview in the Flutter application uses 9.6% more CPU than the React Native.

- *Rendering listview*: The React Native application uses 4.5% more CPU than the Flutter application when rendering a list of countries.
- *Filtering:* Filtering for a specific country data from a list of countries in the React Native application uses 14.8% more CPU than a similar action in Flutter application.

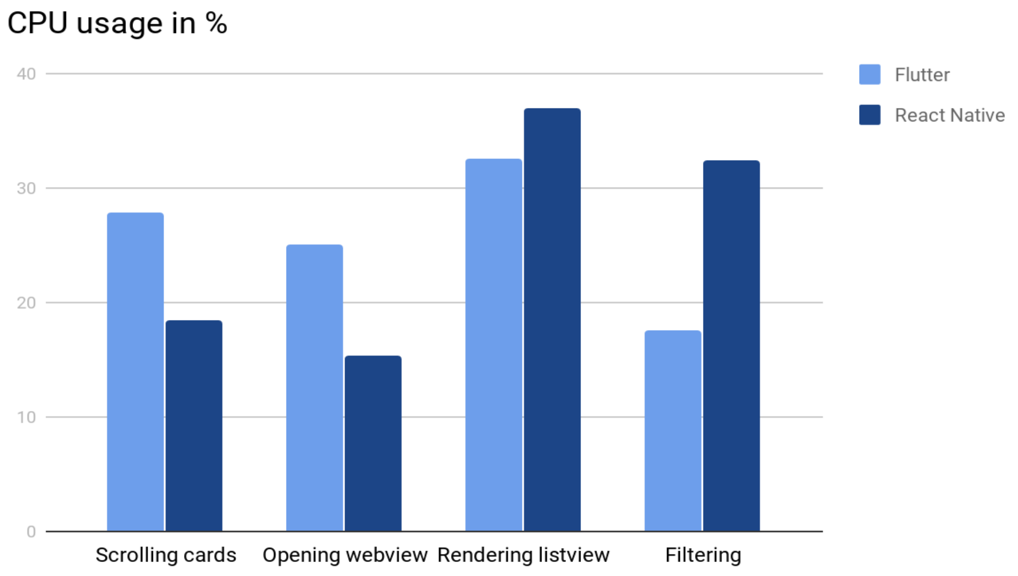


Figure 30 CPU Usage bar chart, React Native Vs Flutter of the android application.

5.2.2 Memory Usage

Figure 31 demonstrates the memory usage measurements of the Flutter and React Native mobile applications. Android uses MB (MegaByte) as the unit of measurement for memory consumption. Similarly, to the CPU usage, the representative measurement is taken from the arithmetic mean of five test runs. The result of the memory usage test are as follows:

- *Scrolling cards*: The Flutter application uses 1.9 MB more memory than the React Native application when scrolling through cards of items.
- *Opening webview*: The React Native application uses 20.14 MB more memory than the Flutter application when loading a webview.
- *Rendering listview:* as seen from the graph rendering list of countries in Flutter uses 2.8 MB more memory than React Native.

- *Filtering*: Filtering a country from a list of countries in React Native uses 2.06 MB more memory than a similar action in Flutter.

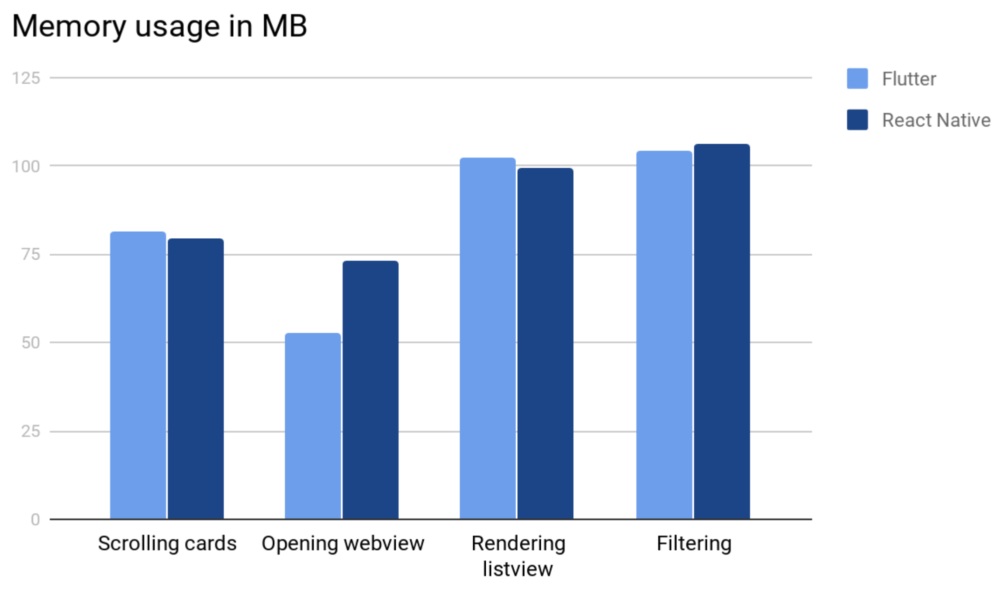


Figure 31 Memory usage bar chart, React Native Vs Flutter of the android application.

5.2.3 GPU

As mentioned in the preceding section, Android studio profiler tool is used for the performance analysis of the android applications developed using React Native and Flutter. However, the profiler does not provide a tool for analyzing the GPU usage, for that reason the Android device “GPU rendering tool” was activated from the settings of the test device Samsung SM-A510F. The "GPU rendering tool" only provides the visual representation of GPU usage by drawing vertical bars on the device screen along with a benchmark of green horizontal line which represents 60 frames per second. The vertical bars drawn on the screen have different colors and the representation of each color is mentioned on Figure 27. Any vertical bar above the benchmark indicates that the application is rendering less than 60 frames per second. Moreover, the higher the density of those bars, the lower the application is performing [69].

Unfortunately, the GPU usage of a Flutter android application cannot be detected using "GPU rendering tool", for that matter flutter provided an alternative DevTools. According to the Flutter documentation DevTools is a set of performance and debugging tools for Dart and Flutter which is still under active development [71,72]. Hence it is not advisable to rely on DevTools for GPU usage analysis and this section only focuses on the GPU usage of the React Native android application. Figure 32 illustrates the GPU usage of different activities of the React Native Android application as measured by the device's "GPU rendering tool". The outcomes of the tool are discussed as follows:

- *Scrolling cards*: The orange bars represent the time the CPU is waiting for the GPU to finish its work. These bars stand above the benchmark line, indicating that the app is doing too much work on the GPU. On the other hand, the red bars represent the time spent by the Android's 2D renderer issuing commands to OpenGL to draw and redraw lists of cards. The height of these bars indicates that there are more display lists. In general, most bars still stand below the benchmark, indicating the app performing well.
- *Opening webview:* As seen from Figure 32, the tallest bars for Opening webviews are Misc Time/VSync Delay, indicating that the app spends more time on executing operations between two consecutive frames which in this case is opening an external webview from the application. Moreover, the Animation rendering stage still stands above the benchmark green line, indicating the time it took to evaluate all the animators during the transition to the webview. Despite those two most significant bars that stand above the benchmark green line, other rendering stages remain below the green line.
- *Rendering listview*: During rendering the list of countries, most rendering stage bars are visible above the green benchmark line, which indicates list of countries are rendered at less than 60 frames per second. The reason behind that is, in the React Native application *FlatList* is used to render a list of countries. React Native *FlatList* has known performance issues when rendering large sets of data. Although that was intentionally used for indiscriminate performance comparison purposes with the Flutter application, third party high performance listview handler library *RecyclerListView* could have been used for rendering large sets of data. According to the library creators, *RecyclerListView* uses "cell recycling". Which is reusing views that are no not visible on the screen to render items rather than creating new views for every item to be rendered. In other words,

RecyclerListView could be able to render an infinite list of items in a more efficient way by reutilizing views [73].

- *Filtering:* As seen from Figure 32, Filtering for country data still renders at less than 60 frames per second, which has direct relation to the *FlatList* view. The implementation for filtering depends on the size of the data set and under the hood, React Native *FlatList* is used to render the item after the filtering is done. However, the vertical bars for the filtering action are still shorter when compared to rendering the list of all countries, indicating that *FlatList* performs better with a smaller set of data.

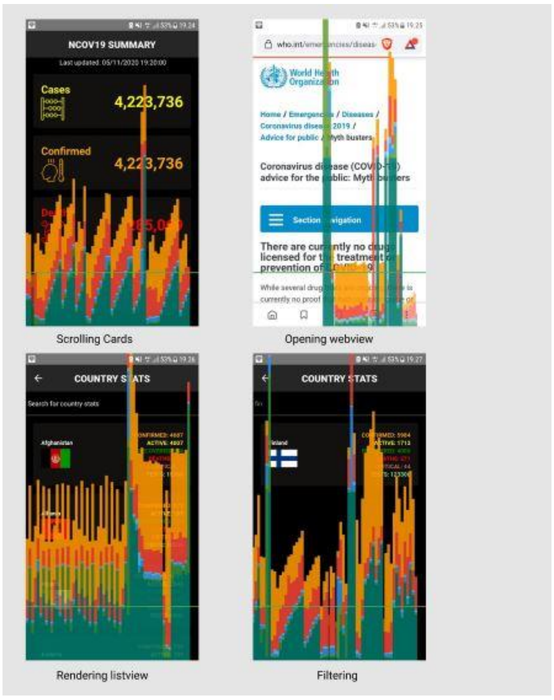


Figure 32 GPU usage of different parts of the React Native Android application.

6 Summary

This chapter contains sections which detail the analysis of performance comparison and framework choice guideline. The “Framework choice guideline” section aims at helping classes companies to select the right cross-platform mobile application development framework to develop clusters of mobile applications that are presented in sections 3.3.4 based on the performance analysis that is presented in the previous chapter.

6.1 Analysis of performance results

The preceding chapter detailed extensive performance tests of the Flutter and React Native applications on both iOS and Android platforms. Although both the Flutter and React Native applications share the same codebase for iOS and Android platforms, the tools used to check performances in iOS and Android platforms vary. The goal of this section is to discuss related findings from the performance comparison and possible reason / reasons why one application performs better than the other one on the described platforms.

As described in section 5.1 the performance comparison between the Flutter and React Native iOS application is done by using different tools from Apple's Instruments suite. The "Time Profiler tool" is used for measuring CPU usage. As seen from the result of the CPU usage measurements (Figure 24), the React Native iOS application consistently uses more CPU than the Flutter application in all the test cases. The main reason behind this result is that React Native uses JavaScript bridge to communicate with platform native modules, and this communication creates heavy usage on the CPU. On the other hand, Flutter does not require a bridge to communicate with platform native modules, hence it does not consume extra CPU for communicating with platform native modules. Another important tool used during the performance analysis is the "Allocations Tool", which measures memory usage. The result of the memory usages of the Flutter and React Native applications is presented in Figure 25. As seen from the result, React Native performs better in memory management that Flutter. A contribution to this result might be that the React Native application is implemented to use Redux state management library, whereas the Flutter one uses a simple Provider package. In addition to CPU and memory measuring tools, "Core Animation Tool" is used to measure the GPU usage. The result of which is presented in Figure 26. The GPU usage graph presents an interesting mix of results during different tests. It can generally be said that during *Scrolling cards* and *Opening webview* tests, the two applications show nearly the same results. On the other hand, React Native shows better GPU usage during *Rendering listview* test. Whereas Flutter results in better GPU usage during *Filtering* test. In general Flutter performs slightly better than React Native in GPU usage considering the overall result from the test cases.

The performance comparison between Flutter and React Native Android applications is done by using the Android Profiler tool from Android Studio as described in section 5.2. As seen from Figure 30, the CPU usage of the Android applications presents mixed results. Flutter consumes relatively more CPU than React Native during *Scrolling cards* and *Opening webview* tests. On the other hand, React Native uses significantly more CPU than Flutter during *Rendering listview* and *Filtering* tests. As explained above the CPU usage of the React Native application could be related to the JavaScript bridge that is used to communicate to the native modules. On the other hand, loading images and web pages might have contributed to the slightly more CPU usage by the Flutter application during *Scrolling cards* and *Opening webview* tests. In general, considering all the four tests conducted, the first two CPU tests (*Scrolling cards and Opening webview*) roll in favor of React Native whereas the later tests (*Rendering listview* and *Filtering*) roll in favor of Flutter. Hence it can be concluded as both Flutter and React Native manage to use CPU equally efficiently depending on the tests. In addition to the CPU usage, Android Profiler also provides the memory usage of an Android application. The memory usage result of the React Native and Flutter Android applications is presented in Figure 31. Again, the Android applications memory usage chart presents mixed results compared to that of iOS memory usage chart. The Flutter Android application clearly uses memory more efficiently than the React Native one during *Opening webview* and *Filtering* tests. One contribution to React Native using slightly more memory could be in the React Native application third-party library is used for Linking and opening external web pages and third-party libraries usually have more dependencies to load, hence they consume more memory. In addition to that the filter function of JavaScript is slow when it comes to filtering items from bigger lists which might have added to more memory consumption by the React Native application. whereas the React Native Android application shows slightly better memory usage than the Flutter one during *Scrolling cards* and *Rendering listview* tests. A contribution to React Native being more memory efficient than Flutter during the later tests might be due to the efficient way of using Redux state management library as explained in the iOS comparison. In general, it can be considered as both frameworks manage memory equally efficiently since the Flutter android application uses memory more efficiently during *Opening webview* and *Filtering tests*, whereas the React Native android application manages memory more efficiently during *Scrolling cards* and *Rendering listview* tests. As described in section 5.2.3, the “GPU rendering tool” from the Android devices is not yet compatible with Flutter applications, hence GPU usage comparison is not done with the Android applications. The summary of all performance analysis charts is presented in Figure 33.

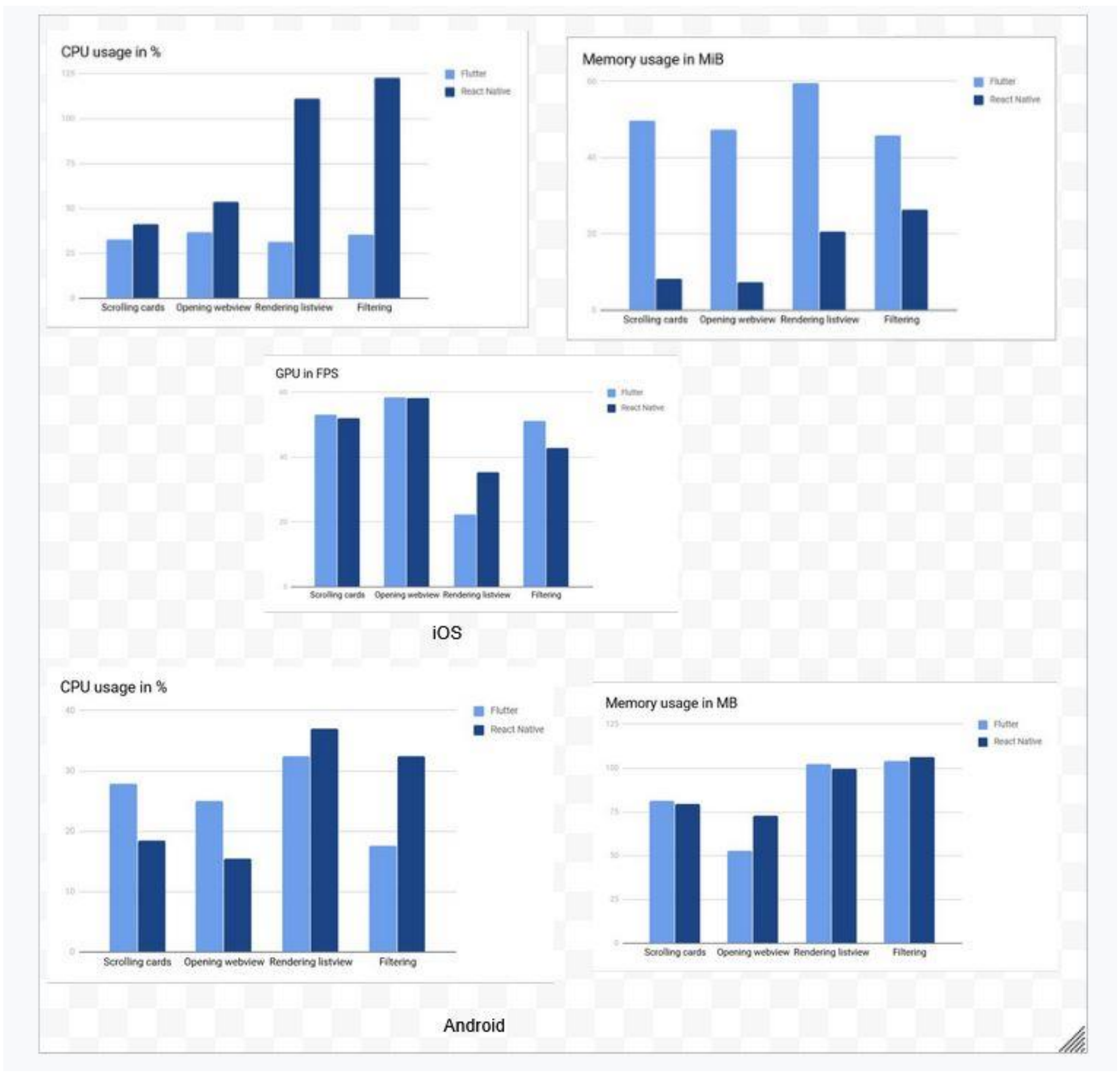
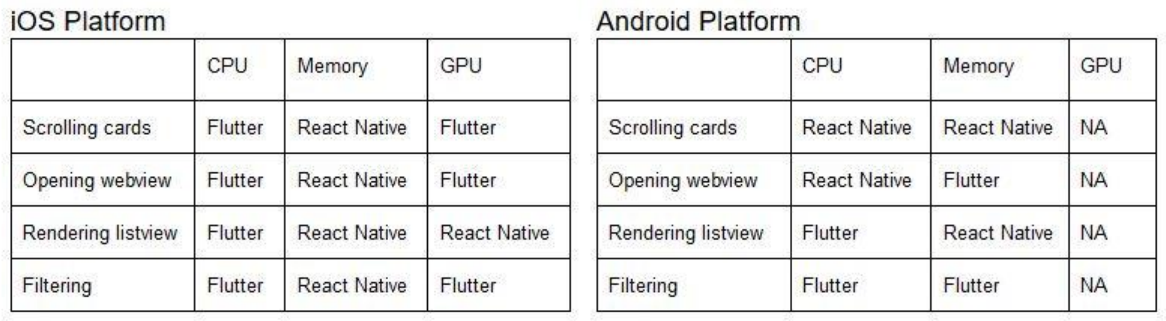


Figure 33 Summary of performance analysis charts.

The above discussion shows that there are also platform specific result differences for a test case. Figure 34 shows platform specific performance summary for each test case.



iOS Platform

	CPU	Memory	GPU
Scrolling cards	Flutter	React Native	Flutter
Opening webview	Flutter	React Native	Flutter
Rendering listview	Flutter	React Native	React Native
Filtering	Flutter	React Native	Flutter

Android Platform

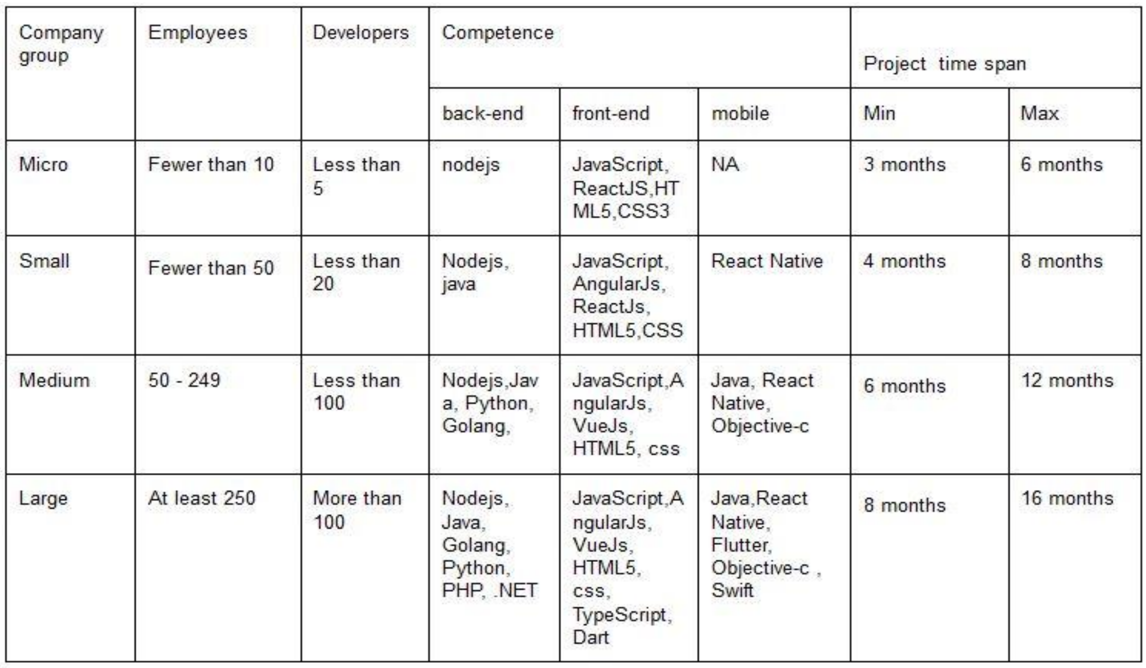
	CPU	Memory	GPU
Scrolling cards	React Native	React Native	NA
Opening webview	React Native	Flutter	NA
Rendering listview	Flutter	React Native	NA
Filtering	Flutter	Flutter	NA

Figure 34 Platform specific performance summary.

Considering the selected test cases and the arithmetic mean representative measurement taken from five different tests per each test case, Figure 34 shows that on iOS platform Flutter manages to use CPU and GPU more efficiently than that of React Native. On the Other hand, React Native uses less memory than Flutter on iOS platform. Moreover, considering the Android platform summary, 50% of the test results are in favor of React Native when it comes to efficient CPU and Memory usages. Although the GPU usage comparison is not applicable in the Android platform, considering the other two metrics we can say that both frameworks use CPU and Memory equally efficiently.

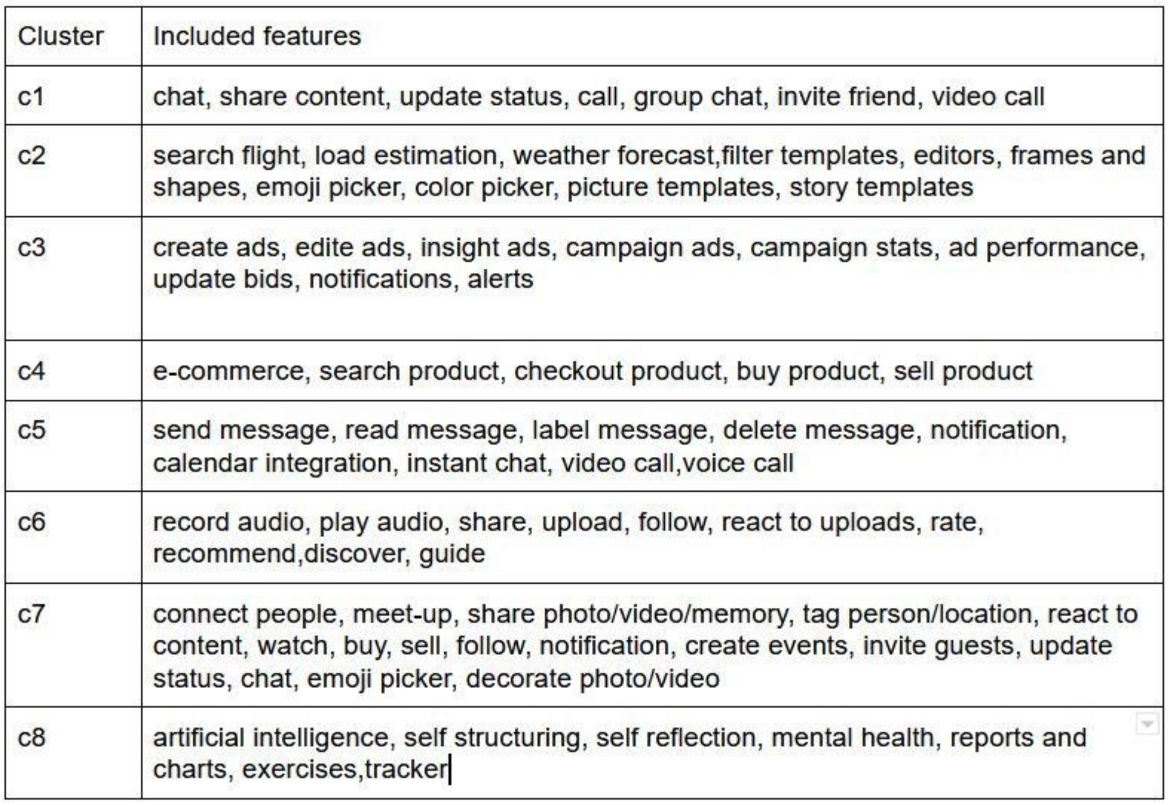
6.2 Framework choice guideline

This section of the thesis provides guideline for choosing cross-platform mobile application development framework from the available choices to develop cluster of features that are discussed in section 3.3.4 for umbrella classes of companies that are presented in Figure 35. Each class of company contains attributes such as size of the company, number of employees and available developers, competence of developers and ideal project time span per size and budget of the company which have significance in the decision of choosing a framework for a mobile application development project [80,81]. The clusters of mobile applications excluding the outlier and the set of features they include is presented in Figure 36.



Company group	Employees	Developers	Competence			Project time span	
			back-end	front-end	mobile	Min	Max
Micro	Fewer than 10	Less than 5	nodejs	JavaScript, ReactJS,HTML5,CSS3	NA	3 months	6 months
Small	Fewer than 50	Less than 20	Nodejs, java	JavaScript, AngularJs, ReactJs, HTML5,CSS	React Native	4 months	8 months
Medium	50 - 249	Less than 100	Nodejs,Java, Python, Golang,	JavaScript,AngularJs, VueJs, HTML5, css	Java, React Native, Objective-c	6 months	12 months
Large	At least 250	More than 100	Nodejs, Java, Golang, Python, PHP, .NET	JavaScript,AngularJs, VueJs, HTML5, css, TypeScript, Dart	Java,React Native, Flutter, Objective-c , Swift	8 months	16 months

Figure 35 Umbrella class of companies and related variables



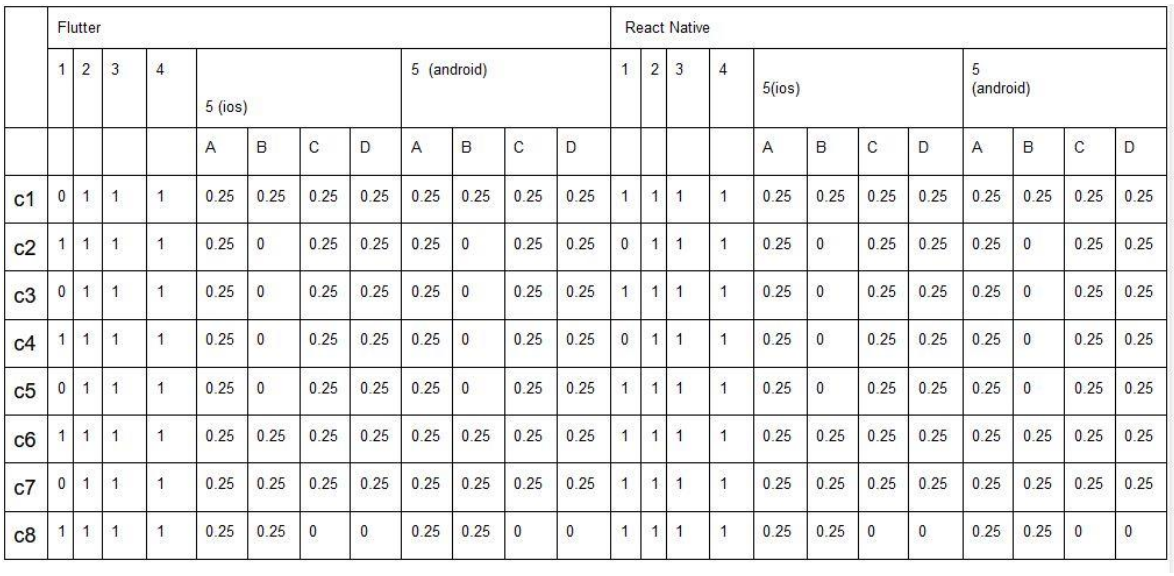
Cluster	Included features
c1	chat, share content, update status, call, group chat, invite friend, video call
c2	search flight, load estimation, weather forecast,filter templates, editors, frames and shapes, emoji picker, color picker, picture templates, story templates
c3	create ads, edite ads, insight ads, campaign ads, campaign stats, ad performance, update bids, notifications, alerts
c4	e-commerce, search product, checkout product, buy product, sell product
c5	send message, read message, label message, delete message, notification, calendar integration, instant chat, video call,voice call
c6	record audio, play audio, share, upload, follow, react to uploads, rate, recommend,discover, guide
c7	connect people, meet-up, share photo/video/memory, tag person/location, react to content, watch, buy, sell, follow, notification, create events, invite guests, update status, chat, emoji picker, decorate photo/video
c8	artificial intelligence, self structuring, self reflection, mental health, reports and charts, exercises,tracker

Figure 36 Clusters of apps and included features

The essence of this section is to provide an aggregated suggestion of a framework for each cluster of features per each company class based on the following five criterions for decision.

1. Suitability of framework to support majority of cluster features
2. Availability of direct competence to framework
3. Learning curve based on related competence
4. Learning curve based on project time span
5. Features related to testbed application

The above-mentioned criterions are then weighted for each framework and cluster per company class as shown in Figure 37 for sample large company group. Similar data for the other groups of companies is presented in Appendix D.



	Flutter												React Native											
	1	2	3	4	5 (ios)				5 (android)				1	2	3	4	5(ios)				5 (android)			
					A	B	C	D	A	B	C	D					A	B	C	D	A	B	C	D
c1	0	1	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	1	1	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
c2	1	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25	0	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25
c3	0	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25	1	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25
c4	1	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25	0	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25
c5	0	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25	1	1	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25
c6	1	1	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	1	1	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
c7	0	1	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	1	1	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
c8	1	1	1	1	0.25	0.25	0	0	0.25	0.25	0	0	1	1	1	1	0.25	0.25	0	0	0.25	0.25	0	0

Key:

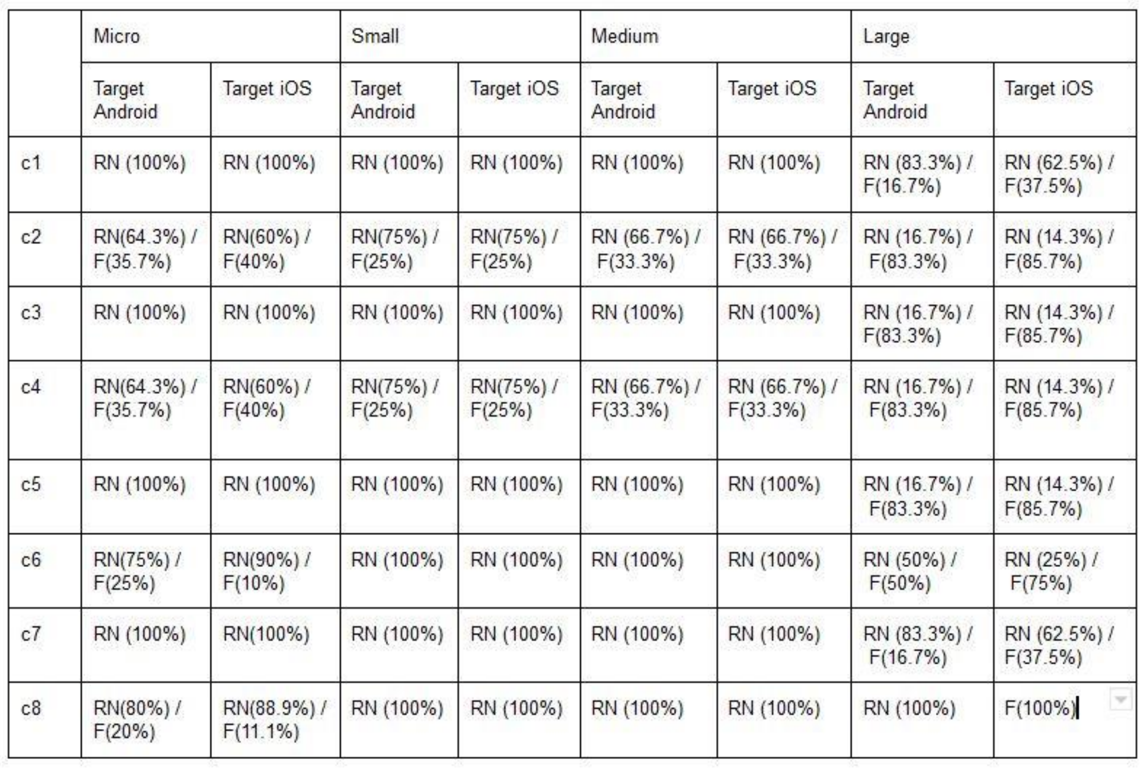
1= Suitability of a framework to support majority of cluster features
2= Availability of direct competence related to framework
3= Learning curve based on related competence
4= Learning curve based on project time span
5= Features related to testbed application
A= Scrolling cards / scrolling
B= Opening WebView / external link
C= Rendering Listview / listview
D= Filtering / search for content

Figure 37 Weighting of variables for large company class

The weighting of criterion one depends on the clustering analysis that is done in section 3.3.4. On the other hand, weighting of criterions two to four depends on variables that are presented in Figure 35. Criterion five is used to include platform specific decision. It is based on the platform specific comparison of testbed features that is presented in Figure 34. If the availability of direct competence is only to one of the frameworks, then further platform specific weighting is not applicable because it is logical to capitalize on the available direct competence for choosing a framework.

The aggregated percentage suggestion of a framework for a cluster of features for a class of company is calculated by omitting values that are not applicable (NA) or tie to both frameworks from the supper set of decision criterions. Tie values are omitted because they are less significant in percentage calculation. If platform specific weighting is not applicable for a cluster of features, then the aggregate percentage suggestion is calculated by summing the weight of all applicable decision criterions for that specific framework and divide it with the sum of the weight of all applicable decision criterions from the supper set of decision criterions.

On the other hand, if further platform specific weighting is applicable, the aggregated percentage suggestion calculation includes platform specific decision criterions based on platform specific comparison result presented in Figure 34. That is, for iOS platform, the weights of decision criterions A, B and D will be add to the Flutter framework, whereas weight of decision criterion C will be added to React Native framework. For Android platform, decision criterion B and C are tie to both frameworks, hence they are omitted from calculation. On the other hand, weight of decision criterion A will be added to React Native on Android platform and weight of decision criterion D will be added to Flutter in the aggregated percentage calculation. The suggested aggregated percentage suggestion of a framework for cluster of features for each company class is presented in Figure 38.



	Micro		Small		Medium		Large	
	Target Android	Target iOS	Target Android	Target iOS	Target Android	Target iOS	Target Android	Target iOS
c1	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (83.3%) / F(16.7%)	RN (62.5%) / F(37.5%)
c2	RN(64.3%) / F(35.7%)	RN(60%) / F(40%)	RN(75%) / F(25%)	RN(75%) / F(25%)	RN (66.7%) / F(33.3%)	RN (66.7%) / F(33.3%)	RN (16.7%) / F(83.3%)	RN (14.3%) / F(85.7%)
c3	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (16.7%) / F(83.3%)	RN (14.3%) / F(85.7%)
c4	RN(64.3%) / F(35.7%)	RN(60%) / F(40%)	RN(75%) / F(25%)	RN(75%) / F(25%)	RN (66.7%) / F(33.3%)	RN (66.7%) / F(33.3%)	RN (16.7%) / F(83.3%)	RN (14.3%) / F(85.7%)
c5	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (16.7%) / F(83.3%)	RN (14.3%) / F(85.7%)
c6	RN(75%) / F(25%)	RN(90%) / F(10%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (50%) / F(50%)	RN (25%) / F(75%)
c7	RN (100%)	RN(100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (83.3%) / F(16.7%)	RN (62.5%) / F(37.5%)
c8	RN(80%) / F(20%)	RN(88.9%) / F(11.1%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	RN (100%)	F(100%)

Key:

F= Flutter

RN= React Native

Figure 38 Aggregated percentage suggestion of framework for cluster of features

7 CONCLUSIONS

In this thesis, we have compared the Flutter and React Native cross platform application development frameworks on the bases of performance metrics. The comparison was done by developing COVID-19 tracking mobile applications using the Flutter and React Native frameworks for Android and iOS platforms. The aim of the performance comparison was to provide answers to the research question defined in the first chapter (section 1.1). Apart from the performance comparison, categories of application development in general and mobile application development approaches were discussed to give a broader insight of application development. Moreover, small scenarios were presented to show popular development approaches for a particular use case by using both frameworks. Clustering of mobile apps was also done by using features of the apps that are extracted from their app store descriptions.

The result of the performance comparisons indicated that developers need to pay special emphasis on certain performance metrics when developing applications using React Native and Flutter frameworks. One of the main advantages of using React Native is its strong developer community which provides many alternative solutions for different use cases. Since React Native uses JavaScript, it is easy to pick up specifically for developers with strong web development background. For companies with skilled web developers, choosing React Native as their cross platform mobile application development framework would help them reuse competence and reduce development cost. On the other hand, the fact that React Native uses JavaScript bridges to communicate with platform native modules has performance effects. Moreover, React Native demands extensive platform specific stylings to make the UI work seamlessly on certain platforms.

For companies with enough budget and resources, Flutter could be a great choice for cross platform mobile application development framework. The fact that Flutter uses Dart programming language, which is not widely used, has a negative impact on its developer community. Although the Flutter creators state that they took inspiration from React Native [74], learning Flutter might take more time for developers who are not familiar with Dart programming language. Despite its steep learning curve, Flutter is rich in UI widgets which work seamlessly on different platforms.

As explained in section 1.4, the testbed application implemented for performance comparisons only includes a subset of features from the wide range of features that industry scale applications might contain, due to time constraints. Hence it would be far-fetched to conclude this analysis as a generalized result. However, based on the selected metrics for comparison and the summary of performance comparisons in Figure 34, It is clearly visible that both frameworks have their own strongest sides. In Fact, apart from the GPU results on iOS where Flutter dominates, the cumulative result based on other metrics on both platforms could be considered a tie. Therefore, companies could alternatively exploit the benefits of using these frameworks depending on availability of competence, time, and budget.

Although the overall performance analysis on testbed application using some selected features does not run in favor of one of the frameworks, clustering of sample mobile applications in section 3.3.4 provide a great insight to make feature-based recommendation of the frameworks. Depending on the analysis done in section 3.3.4, it might be favorable to use React Native framework for applications that could contains features such as instant messaging and commenting, video and picture sharing , instant status updates and notifications, instant reaction to contents, among other related features. On the other hand, it might be favorable to use Flutter framework for applications that contain extensive search and filtering features.

However, Flutter is still under great development and choosing it as the next cross platform mobile application development framework for a company depends on the availability of widgets to support the implementation of requirements. In general, both Flutter and React Native are under continuous improvements and soon both frameworks might support loads more near native functionalities than they support now. Probably a thesis like this can be used as a guideline to make a comprehensive and in-depth performance analysis of the frameworks on industry scale applications as a future work.

BIBLIOGRAPHY

[1] Ericsson. Ericsson Mobility Report November 2019 [Internet]. 2019 [cited 2020 Feb 15] p. 36. Available from: https://www.ericsson.com/4acd7e/assets/local/mobility-report/documents/2019/emr-november-2019.pdf

[2] Boulos MNK, Wheeler S, Tavares C, Jones R. How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from eCAALYX. BioMedical Engineering OnLine. 2011 Apr 5;10(1):24.

[3] Nubentos. The best APIs for Health [Internet]. Nubentos. [cited 2020 Oct 4]. Available from: https://www.nubentos.com/en/home/

[4] IBM Corporation. Designing and developing applications for z/OS. 2006, 2010;90.

[5] Sally Cornett. Application Development: Definition & Types [Internet]. Study.com. [cited 2020 Feb 29]. Available from: https://study.com/academy/lesson/application-development-definition-types.html

[6] Douglas Hughey. The Traditional Waterfall Approach [Internet]. [cited 2020 Mar 2]. Available from: http://www.umsl.edu/~hugheyd/is6840/waterfall.html

[7] Conrad Weisert, Information Disciplines, Inc. There’s no such thing as the Waterfall Approach! (and there never was) [Internet]. 2003 [cited 2020 Mar 2]. Available from: http://www.idinews.com/waterfall.html

[8] Mary Lotz. Waterfall vs. Agile: Which Methodology is Right for Your Project? [Internet]. Segue Technologies. 2018 [cited 2020 Mar 3]. Available from: https://www.seguetech.com/waterfall-vs-agile-methodology/

[9] Geambaşu CV, Jianu I, Jianu I, Gavrilă A. INFLUENCE FACTORS FOR THE CHOICE OF A SOFTWARE DEVELOPMENT METHODOLOGY. 2011;10(4):16.

[10] Gerber A, Alberts R, Alta van der M. Practical Implications of Rapid Development Methodologies. Computer Science and IT Education Conference. 2007;13.

[11] Daud NMN, Bakar NAAA, Rusli HM. Implementing rapid application development (RAD) methodology in developing practical training application system. In: 2010 International Symposium on Information Technology [Internet]. Kuala Lumpur, Malaysia: IEEE; 2010 [cited 2020 March 7]. p. 1664–7. Available from: http://ieeexplore.ieee.org/document/5561634/

[12] Lee G, Xia W. Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility. MIS Quarterly. 2010;34(1):87.

[13] Pekka Abrahamsson, Outi Salo & Jussi Ronkainen,Juhani Warsta. Agile software development methods Review and analysis. Espoo 2002 [cited 2020 March 8]. Available from: https://www.vtt.fi/inf/pdf/publications/2002/P478.pdf

[14] Northern C, Mayfield K, Benito R, Casagni M. Handbook for Implementing Agile in Department of Defense Information Technology Acquisition: [Internet]. Fort Belvoir, VA: Defense Technical Information Center; 2010 Dec [cited 2020 Mar 8]. Available from: http://www.dtic.mil/docs/citations/ADA546756

[15] Manifesto for Agile Software Development [Internet]. 2001 [cited 2020 Mar 8]. Available from: http://agilemanifesto.org/

[16] Delia L, Galdamez N, Thomas P, Corbalan L, Pesado P. Multi-platform mobile application development analysis. In: 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS) [Internet]. Athens, Greece: IEEE; 2015 [cited 2020 Mar 16]. p. 181–6. Available from: http://ieeexplore.ieee.org/document/7128878/

[17] Charkaoui S, Adraoui Z, Benlahmar EH. Cross-platform mobile development approaches. In: 2014 Third IEEE International Colloquium in Information Science and Technology (CIST) [Internet]. Tetouan, Morocco: IEEE; 2014 [cited 2020 Mar 16]. p. 188–91. Available from: http://ieeexplore.ieee.org/document/7016616/

[18] Lachgar M, Abdali A. Survey of mobile development approaches. Journées Doctorales en Systèmes d’Information, Réseaux et Télécommunication (JDSIRT’ 2015). 2015;5.

[19] Nagesh A, Caicedo CE. Cross-Platform Mobile Application Development. ITERA 2012, The 10th Annual Conference on Telecommunications and Information Technology, 30th March-1 April 2012, Indianapolis Indiana. 2012;10.

[20] Ghandi L, Silva C, Martinez D, Gualotuna T. Mobile application development process: A practical experience. In: 2017 12th Iberian Conference on Information Systems and Technologies (CISTI) [Internet]. Lisbon, Portugal: IEEE; 2017 [cited 2020 Mar 18]. p. 1–6. Available from: http://ieeexplore.ieee.org/document/7975825/

[21] Harnegie MP. The Library Mobile Experience: Practices and User Expectations. *Library Technology Reports: Expert Guides to Library Systems and Services* by Bohyun Kim: (2013). 49(6):1–40. Chicago: ALA Techsource, ISSN: 0024-2586. Journal of Hospital Librarianship. 2014 Jul 3;14(3):328–9.

[22] Shahzad F. Modern and Responsive Mobile-enabled Web Applications. Procedia Computer Science. 2017;110:410–5

[23] Laine M, Shestakov D, Litvinova E, Vuorimaa P. Toward Unified Web Application Development. IT Prof. 2011 Sep;13(5):30–6.

[24] Gustavo Hartmann, G Stead, and A DeGani. *Cross-platform mobile development.* Tribal, Lincoln House, The Paddocks, Tech. Rep, (March): 1–18, 2011 [cited 2020 March 23]. Available from: https://wss.apan.org/jko/mole/SharedDocuments/Cross-PlatformMobileDevelopment.pdf.

[25] Latif M, Lakhrissi Y, Nfaoui EH, Es-Sbai N. Cross platform approach for mobile application development: A survey. In: 2016 International Conference on Information Technology for Organizations Development (IT4OD) [Internet]. Fez, Morocco: IEEE; 2016 [cited 2020 Mar 24]. p. 1–5. Available from: http://ieeexplore.ieee.org/document/7479278/

[26] https://reactnative.dev/docs [cited 2020 March 25]

[27] A short Story about React Native [Internet]. JobNinja Blog. 2018 [cited 2020 Mar 25]. Available from: https://jobninja.com/blog/short-story-react-native/

[28] Dabit N. React Native in action: developing iOS and Android apps with JavaScript. Shelter Island, NY: Manning Publications; 2019. 293 p

[29] Masiello E, Friedmann J. Mastering React Native leverage frontend development skills to build impressive iOS and Android applications with Native React [Internet]. Birmingham, UK: Packt Publishing; 2017 [cited 2020 Mar 22]. Available from: http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=1452261

[30] Abramov D. Presentational and Container Components [Internet]. Medium. 2019 [cited 2020 Mar 23]. Available from: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

[31] Spallino A. Native versus hybrid mobile application development for professional membership services. 2018 Jul 30;70.

[32] Dawid S. Basics of Flutter Widgets [Internet]. CodeJourney.net. 2020 [cited 2020 Mar 26]. Available from: https://www.codejourney.net/2020/01/basics-of-flutter-widgets/

[33] Technical overview [Internet]. Flutter. [cited 2020 Mar 29]. Available from: https://flutter.dev/docs/resources/technical-overview

[34] Freitas E. Flutter Succinctly. Syncfusion, Inc. 2019;129. [cited 2020 March 27]. Available from: http://ebooks.syncfusion.com/downloads/flutter-succinctly/flutter-succinctly.pdf

[35] StatelessWidget class - widgets library - Dart API [Internet]. [cited 2020 Mar 28]. Available from: https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

[36] StatefulWidget class - widgets library - Dart API [Internet]. [cited 2020 Mar 28]. Available from: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

[37] Flutter State Management - Javatpoint [Internet]. www.javatpoint.com. [cited 2020 Mar 30]. Available from: https://www.javatpoint.com/flutter-state-management

[38] Differentiate between ephemeral state and app state [Internet]. Flutter. [cited 2020 Mar 30]. Available from: https://flutter.dev/docs/development/data-and-backend/state-mgmt/ephemeral-vs-app

[39] Coca J. Let me help you to understand and choose a state management solution for your app [Internet]. Medium. 2018 [cited 2020 Mar 30]. Available from: https://medium.com/flutter-community/let-me-help-you-to-understand-and-choose-a-state-management-solution-for-your-app-9ffeac834ee3

[40] scoped_model | Flutter Package [Internet]. Dart packages. 2018 [cited 2020 Mar 30]. Available from: https://pub.dev/packages/scoped_model

[41] Flutter - Reactive Programming - Streams - BLoC [Internet]. 2018 [cited 2020 Mar 30]. Available from: https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/

[42] flutter_redux | Flutter Package [Internet]. Dart packages. 2019 [cited 2020 Mar 30]. Available from: https://pub.dev/packages/flutter_redux

[43] coronavirus open API. [cited 2020 April 17].

Available from: https://corona.lmao.ninja/v2/countries

[44] Coronavirus disease (COVID-19) pandemic [Internet]. World Health Organization. [cited 2020 Apr 24]. Available from: https://www.who.int/emergencies/diseases/novel-coronavirus-2019

[45] COVID-19 Flutter application source code

Available from: https://github.com/AwelEshetu/covid19_info

[46] COVID-19 React Native application source code

Available from: https://github.com/AwelEshetu/covid19_info_react_native

[47] Visual Studio Code

Available from: https://code.visualstudio.com/

[48] Shailesh. Introduction to React Native Renderers aka React Native is the Java and React Native Renderers are the JVMs of declarative UI [Internet]. Medium. 2018 [cited 2020 Apr 27]. Available from: https://medium.com/@agent_hunt/introduction-to-react-native-renderers-aka-react-native-is-the-java-and-react-native-renderers-are-828a0022f433

[49] React Navigation [Internet]. [cited 2020 Apr 28]. Available from: https://reactnavigation.org//docs/getting-started

[50] Bartosz S, Agnieszka M, Damian W. Flutter vs React Native – what to choose in 2020? [Internet]. Droids On Roids. 2019 [cited 2020 April 28]. Available from: https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2020

[51] Layout widgets [Internet]. Flutter. [cited 2020 Apr 28]. Available from: https://flutter.dev/docs/development/ui/widgets/layout

[52] Navigate to a new screen and back [Internet]. Flutter. [cited 2020 Apr 29]. Available from: https://flutter.dev/docs/cookbook/navigation/navigation-basics

[53] Navigator class - widgets library - Dart API [Internet]. Flutter. [cited 2020 Apr 30]. Available from: https://api.flutter.dev/flutter/widgets/Navigator-class.html

[54] Simple app state management [Internet]. Flutter. [cited 2020 May 1]. Available from: https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple

[55] FAQ [Internet]. Flutter. [cited 2020 May 2]. Available from: https://flutter.dev/docs/resources/faq

[56] expo-three [Internet]. GitHub. [cited 2020 May 2].

Available from: https://github.com/expo/expo-three

[57] react-native-gl-model-view [Internet]. GitHub. [cited 2020 May 2].

Available from: https://github.com/rastapasta/react-native-gl-model-view

[58] Texture class - widgets library - Dart API [Internet]. Flutter. [cited 2020 May 2]. Available from: https://api.flutter.dev/flutter/widgets/Texture-class.html

[59] flutter_3d_obj. Available from: https://github.com/hemanthrajv/flutter_3d_obj

[60] react-native-video. Available from: https://github.com/react-native-community/react-native-video

[61] react-native-sound [Internet]. GitHub. [cited 2020 May 3].

Available from: https://github.com/zmxv/react-native-sound

[62] flutter_sound - Dart API docs [Internet]. Flutter. [cited 2020 May 3]. Available from: https://pub.dev/documentation/flutter_sound/latest/

[63] video_player - Dart API docs [Internet]. Flutter. [cited 2020 May 3]. Available from: https://pub.dev/documentation/video_player/latest/

[64] video_player_platform_interface - Dart API docs [Internet]. Flutter. [cited 2020 May 3]. Available from: https://pub.dev/documentation/video_player_platform_interface/latest/

[65] react-native-fs [Internet]. GitHub. [cited 2020 May 4].

Available from: https://github.com/itinance/react-native-fs

[66] path_provider | Flutter Package [Internet]. Dart packages. [cited 2020 May 4]. Available from: https://pub.dev/packages/path_provider

[67] Shashikant J. Flutter vs React Native: A Developer's Perspective [Internet]. Nevercode. [cited 2020 May 4]. Available from: https://nevercode.io/blog/flutter-vs-react-native-a-developers-perspective/

[68] Krzysztof L. Flutter vs React Native in 2020. Is it the Future of Mobile Development? [Internet]. Monterail. 2020 [cited 2020 May 4]. Available from: https://www.monterail.com/blog/flutter-vs-react-native-mobile-development

[69] Inspect GPU rendering speed and overdraw [Internet]. Android Developers. [cited 2020 May 7]. Available from: https://developer.android.com/topic/performance/rendering/inspect-gpu-rendering

[70] Instruments Overview - Instruments Help [Internet]. Apple. [cited 2020 May 10]. Available from: https://help.apple.com/instruments/mac/current/#/dev7b09c84f5

[71] Hracek F. Performance Testing of Flutter apps [Internet]. Medium. 2019 [cited 2020 May 11]. Available from: https://medium.com/flutter/performance-testing-of-flutter-apps-df7669bb7df7

[72] DevTools [Internet]. Flutter. [cited 2020 May 11]. Available from: https://flutter.dev/docs/development/tools/devtools

[73] RecyclerListView [Internet]. GitHub. [cited 2020 May 12].

Available from: https://github.com/Flipkart/recyclerlistview

[74] Flutter for React Native developers [Internet]. Flutter. [cited 2020 May 13]. Available from: https://flutter.dev/docs/get-started/flutter-for/react-native-devs

[75] Matt W. 10 Famous Apps Built With React Native [Internet]. Blog Brainhub.eu. 2017 [cited 2020 May 17]. Available from: https://brainhub.eu/blog/react-native-apps/

[76] Agnieszka M. Top Apps Made with Flutter – 17 Stories by Developers and Business Owners [Internet]. Droids On Roids. 2020 [cited 2020 May 17]. Available from: https://www.thedroidsonroids.com/blog/apps-made-with-flutter

[77] Al-Subaihin AA, Sarro F, Black S, Capra L, Harman M, Jia Y, et al. Clustering Mobile Apps Based on Mined Textual Features. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM ’16 [Internet]. Ciudad Real, Spain: ACM Press; 2016 [cited 2020 May 29]. p. 1–10. Available from: http://dl.acm.org/citation.cfm?doid=2961111.2962600

[78] Amit M. Where Do Cross-Platform App Frameworks Stand in 2020? [Internet]. Insights - Web and Mobile Development Services and Solutions. 2019 [cited 2020 May 31]. Available from: https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/

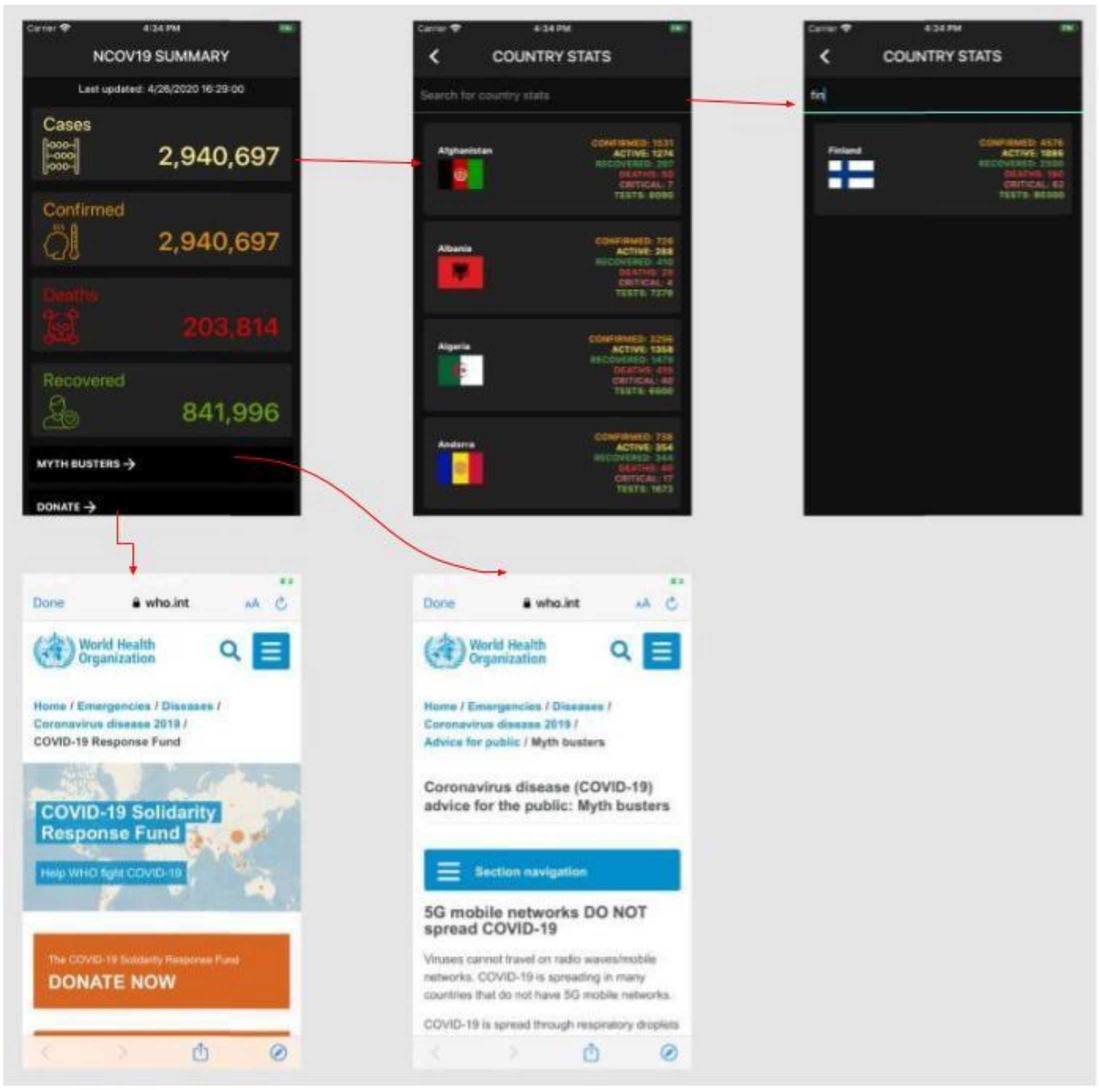
[79] Dalmasso I, Datta SK, Bonnet C, Nikaein N. Survey, comparison and evaluation of cross platform mobile application development tools. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC) [Internet]. Sardinia, Italy: IEEE; 2013 [cited 2020 May 31]. p. 323–8. Available from: http://ieeexplore.ieee.org/document/6583580/

[80] Entrepreneurship in Finland [cited 2020 June 12]. Available from: https://www.yrittajat.fi/en/about-suomen-yrittajat/entrepreneurship-finland-526261

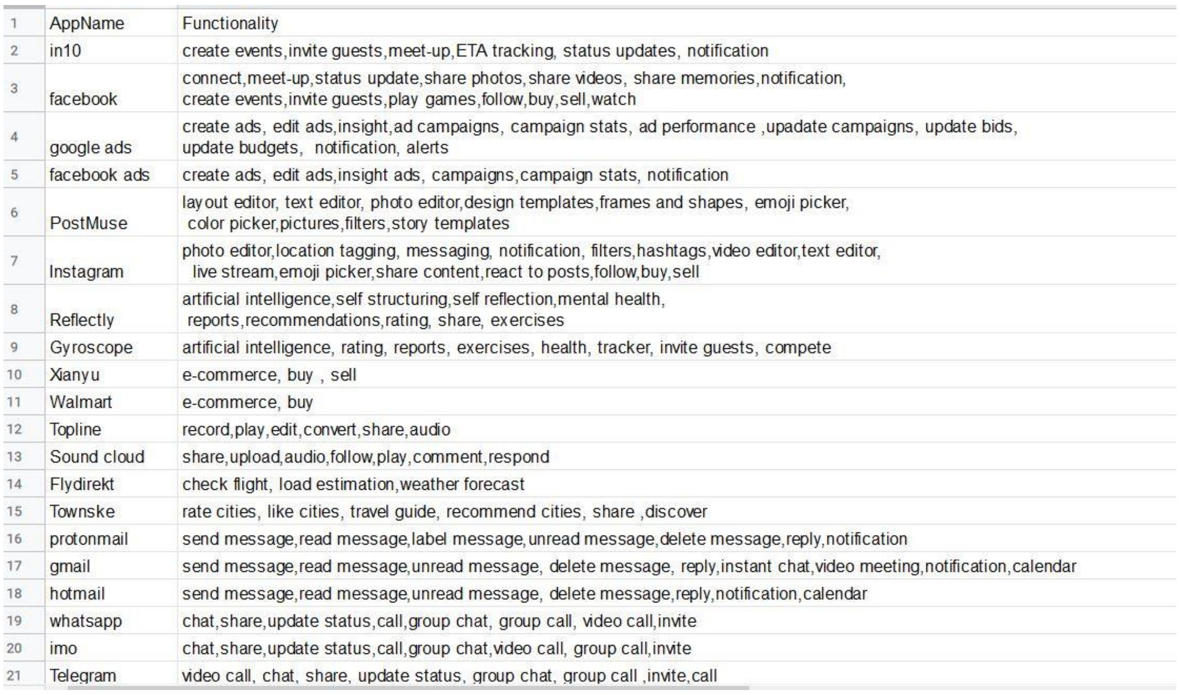
[81] 5 Examples of Tech Stacks from Top-Performing Companies [cited 2020 June 12]. Available from: https://mixpanel.com/topics/tech-stack-examples/

APPENDICES

A Whole application features

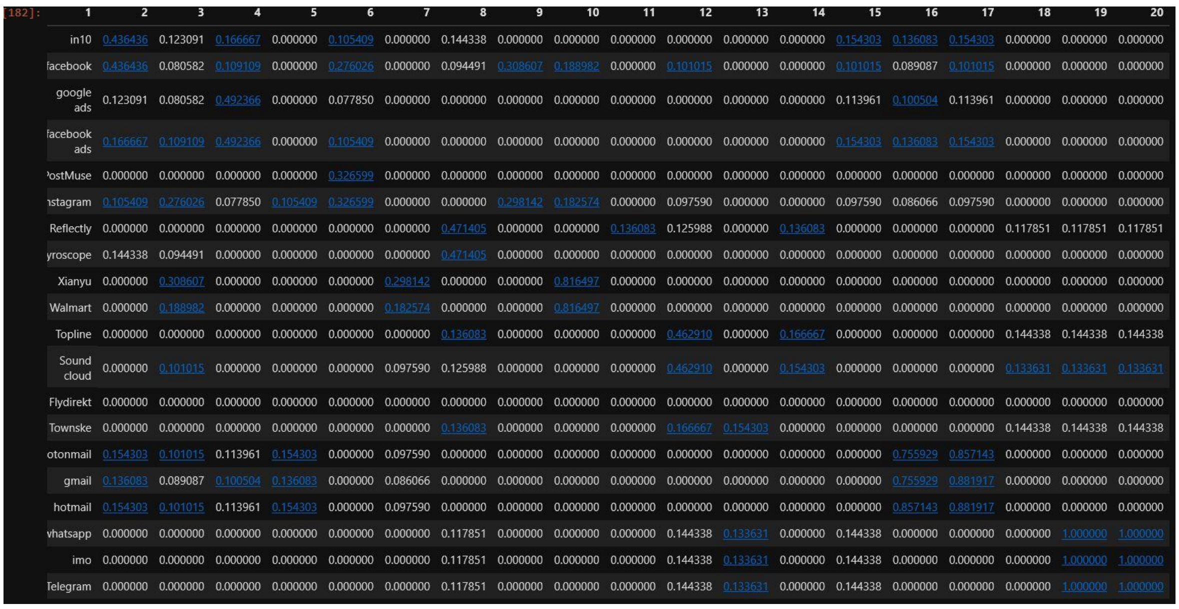

NCOV19 SUMMARY
Last updated: 4/26/2020 16:29:00
Cases 2,940,697
Confirmed 2,940,697
Deaths 203,814
Recovered 841,996
MYTH BUSTERS →
DONATE →
COUNTRY STATS
Search for country stats
Afghanistan
Albania
Algeria
Andorra
COUNTRY STATS
fin
Finland
Done who.int
World Health Organization
Home / Emergencies / Diseases / Coronavirus disease 2019 / COVID-19 Response Fund
COVID-19 Solidarity Response Fund
Help WHO fight COVID-19
The COVID-19 Solidarity Response Fund
DONATE NOW
Done who.int
World Health Organization
Home / Emergencies / Diseases / Coronavirus disease 2019 / Advice for public / Myth busters
Coronavirus disease (COVID-19) advice for the public: Myth busters
Section navigation
5G mobile networks DO NOT spread COVID-19
Viruses cannot travel on radio waves/mobile networks. COVID-19 is spreading in many countries that do not have 5G mobile networks.
COVID-19 is spread through respiratory droplets

B Data set (Apps and their functionality)



1	AppName	Functionality
2	in10	create events,invite guests,meet-up,ETA tracking, status updates, notification
3	facebook	connect,meet-up,status update,share photos,share videos, share memories,notification, create events,invite guests,play games,follow,buy,sell,watch
4	google ads	create ads, edit ads,insight,ad campaigns, campaign stats, ad performance ,upadate campaigns, update bids, update budgets, notification, alerts
5	facebook ads	create ads, edit ads,insight ads, campaigns,campaign stats, notification
6	PostMuse	layout editor, text editor, photo editor,design templates,frames and shapes, emoji picker, color picker,pictures,filters,story templates
7	Instagram	photo editor,location tagging, messaging, notification, filters,hashtags,video editor,text editor, live stream,emoji picker,share content,react to posts,follow,buy,sell
8	Reflectly	artificial intelligence,self structuring,self reflection,mental health, reports,recommendations,rating, share, exercises
9	Gyroscope	artificial intelligence, rating, reports, exercises, health, tracker, invite guests, compete
10	Xianyu	e-commerce, buy , sell
11	Walmart	e-commerce, buy
12	Topline	record,play,edit,convert,share,audio
13	Sound cloud	share,upload,audio,follow,play,comment,respond
14	Flydirekt	check flight, load estimation,weather forecast
15	Townske	rate cities, like cities, travel guide, recommend cities, share ,discover
16	protonmail	send message,read message,label message,unread message,delete message,reply,notification
17	gmail	send message,read message,unread message, delete message, reply,instant chat,video meeting,notification,calendar
18	hotmail	send message,read message,unread message, delete message,reply,notification,calendar
19	whatsapp	chat,share,update status,call,group chat, group call, video call,invite
20	imo	chat,share,update status,call,group chat,video call, group call,invite
21	Telegram	video call, chat, share, update status, group chat, group call ,invite,call

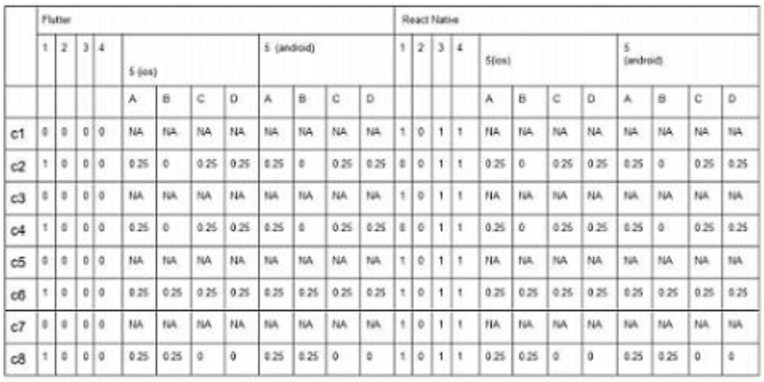
C Apps and their cosine distance vector (Final data set)



[182]:

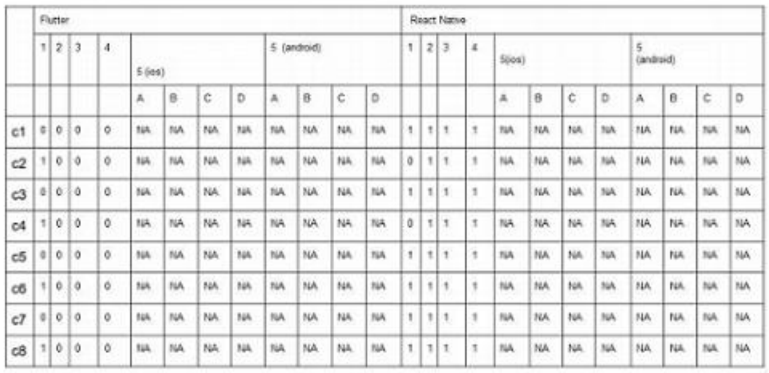
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
in10	0.436436	0.123091	0.166667	0.000000	0.105409	0.000000	0.144338	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.154303	0.136083	0.154303	0.000000	0.000000	0.000000
facebook	0.436436	0.080582	0.109109	0.000000	0.276026	0.000000	0.094491	0.308607	0.188982	0.000000	0.101015	0.000000	0.000000	0.101015	0.089087	0.101015	0.000000	0.000000	0.000000	0.000000
google ads	0.123091	0.080582	0.492366	0.000000	0.077850	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.113961	0.100504	0.113961	0.000000	0.000000	0.000000	0.000000
facebook ads	0.166667	0.109109	0.492366	0.000000	0.105409	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.154303	0.136083	0.154303	0.000000	0.000000	0.000000	0.000000
ostMuse	0.000000	0.000000	0.000000	0.000000	0.326599	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
stagram	0.105409	0.276026	0.077850	0.105409	0.326599	0.000000	0.000000	0.298142	0.182574	0.000000	0.097590	0.000000	0.000000	0.097590	0.086066	0.097590	0.000000	0.000000	0.000000	0.000000
Reflectly	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.471405	0.000000	0.000000	0.136083	0.125988	0.000000	0.136083	0.000000	0.000000	0.000000	0.117851	0.117851	0.117851	0.117851
yroscope	0.144338	0.094491	0.000000	0.000000	0.000000	0.000000	0.471405	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Xianyu	0.000000	0.308607	0.000000	0.000000	0.000000	0.298142	0.000000	0.000000	0.816497	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Walmart	0.000000	0.188982	0.000000	0.000000	0.000000	0.182574	0.000000	0.000000	0.816497	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Topline	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.136083	0.000000	0.000000	0.000000	0.462910	0.000000	0.166667	0.000000	0.000000	0.000000	0.144338	0.144338	0.144338	
Sound cloud	0.000000	0.101015	0.000000	0.000000	0.000000	0.097590	0.125988	0.000000	0.000000	0.000000	0.462910	0.000000	0.154303	0.000000	0.000000	0.000000	0.133631	0.133631	0.133631	
Flydirekt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
Townske	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.136083	0.000000	0.000000	0.000000	0.166667	0.154303	0.000000	0.000000	0.000000	0.000000	0.144338	0.144338	0.144338	
otonmail	0.154303	0.101015	0.113961	0.154303	0.000000	0.097590	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.755929	0.857143	0.000000	0.000000	0.000000	
gmail	0.136083	0.089087	0.100504	0.136083	0.000000	0.086066	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.755929	0.881917	0.000000	0.000000	0.000000	
hotmail	0.154303	0.101015	0.113961	0.154303	0.000000	0.097590	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.857143	0.881917	0.000000	0.000000	0.000000	
hatsapp	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.117851	0.000000	0.000000	0.000000	0.144338	0.133631	0.000000	0.144338	0.000000	0.000000	0.000000	1.000000	1.000000	
imo	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.117851	0.000000	0.000000	0.000000	0.144338	0.133631	0.000000	0.144338	0.000000	0.000000	0.000000	1.000000	1.000000	
Telegram	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.117851	0.000000	0.000000	0.000000	0.144338	0.133631	0.000000	0.144338	0.000000	0.000000	0.000000	1.000000	1.000000	

D Weighting of variables for micro, small and medium company classes



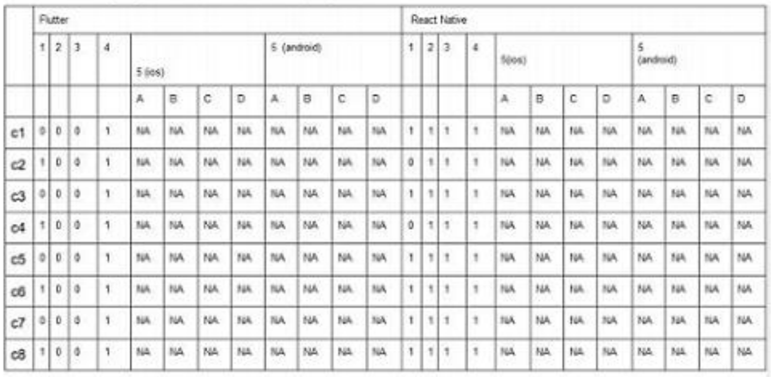
	Flutter												React Native											
	1	2	3	4	5 (ios)				5 (android)				1	2	3	4	5(ios)				5 (android)			
					A	B	C	D	A	B	C	D					A	B	C	D	A	B	C	D
c1	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	0	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c2	1	0	0	0	0.25	0	0.25	0.25	0.25	0	0.25	0.25	0	0	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25
c3	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	0	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c4	1	0	0	0	0.25	0	0.25	0.25	0.25	0	0.25	0.25	0	0	1	1	0.25	0	0.25	0.25	0.25	0	0.25	0.25
c5	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	0	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c6	1	0	0	0	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	1	0	1	1	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
c7	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	0	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c8	1	0	0	0	0.25	0.25	0	0	0.25	0.25	0	0	1	0	1	1	0.25	0.25	0	0	0.25	0.25	0	0

Weighting of variables for micro company class



	Flutter												React Native											
	1	2	3	4	5 (ios)				5 (android)				1	2	3	4	5(ios)				5 (android)			
					A	B	C	D	A	B	C	D					A	B	C	D	A	B	C	D
c1	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c2	1	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	0	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c3	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c4	1	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	0	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c5	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c6	1	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c7	0	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c8	1	0	0	0	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA

Weighting of variables for small company class



	Flutter												React Native											
	1	2	3	4	5 (ios)				5 (android)				1	2	3	4	5(ios)				5 (android)			
					A	B	C	D	A	B	C	D					A	B	C	D	A	B	C	D
c1	0	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c2	1	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	0	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c3	0	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c4	1	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	0	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c5	0	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c6	1	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c7	0	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA
c8	1	0	0	1	NA	NA	NA	NA	NA	NA	NA	NA	1	1	1	1	NA	NA	NA	NA	NA	NA	NA	NA

Weighting of variables for medium company class