SEMINAR REPORT ON

"CROSS PLATFORM APPS DEVELOPMENT USING FLUTTER"

Submitted To The Savitribai Phule Pune University, Pune

In The Partial Fulfillment Of The Requirements

SUBMITTED BY :-

NAME :- MEGHSHAM VINAYAK KAPURE.

EXAM NO :

DEPARTMENT OF COMPUTER ENGINEERING

RDTCS'S SHRI CHATTRAPATI SHIVAJIRAJECOLLEGE OF ENGINEERING

DHANGWADI, PUNE-BANGLORE HIGHWAY, PUNE 411041

SAVITRIBAI PHULE PUNE UNIVERSITY ,

Academic Year :- 2020-2021

CERTIFICATE

This is to certify that the seminar report entitles

"CROSS PLATFORM DEVELOPMENT USING FLUTTER"

SUBMITTED BY :-

NAME : MEGHSHAM VINAYAK KAPURE.

EXAM NO :

is a bonafide student of this institute and the work has been carried out by him under the supervision of **Prof. A.S. Sondkar Mam** and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, (Computer Engineering).

Prof. A.S. Sondkar	**Prof. A.S. Sondkar**
Seminar Guide	Head of Department
Computer Engineering Department	Computer Engineering Department

Dr. S. B. Patil
Principal
SCSCOE, Dhangawadi.

Date of Submission: / / 2021
Place : Dhangawadi

ACKNOWLEDGEMENTS

The satisfaction that accompanies the successful completion of this report would be incomplete without mentioning the people who made it possible, without whose constant guidance and encouragement would have made efforts go in vain. I consider myself privileged to express gratitude and respect towards all those who guided me throughout the completion of this seminar report.

Firstly, I would like to sincerely thank my guide **Prof. A. S. Sondkar & Prof. K. R. Pathak** for guiding me regarding this project. My multiple discussions with her were extremely helpful for me. It is my privilege to acknowledge the kind of guidance I received in preparation for this report. I would not be able to prepare this report without her valuable cooperation.

I am grateful to **Prof. A.S. Sondkar**, Head of the Department of Computer Engineering for giving me the support and encouragement that was necessary for the completion of this project. I wholeheartedly thank to record our sincere gratitude to the Management of this college and to my Principal, **Dr.S.B.Patil**, Principal, SCSCOE for his constant support and encouragement in the preparation of this report and for making available facilities needed to prepare this report.

Last but not the least, I wish to thank my parents for financing my studies in this college as well as for constantly encouraging us to learn engineering. Their sacrifice in providing this opportunity to learn engineering is gratefully acknowledged.

Submitted By,
Meghsham Vinayak Kapure.

ABSTRACT

Cross-platform mobile development today is full of compromise. Developers are forced to choose between either building the same app multiple times for multiple operating systems, or to accept a lowest common denominator solution that trades native speed and accuracy for portability. Flutter is an open-source SDK for creating high-performance, high- fidelity mobile apps for iOS and Android. Few important features of flutter are - Just-in-time compilation is a way of executing computer code that involves compilation during execution of a program – at run time – rather than prior to execution.

Most often, this consists of source code or more commonly bytecode translation to machine code, which is then executed directly. Ahead- of-time compilation (AOT compilation) is the act of compiling a higher-level programming language such as C or C++, or an intermediate representation such as Java bytecode or. NET Framework Common Intermediate Language (CIL) code, into a native `(system-dependent) machine code so that the resulting binary file can execute natively. Flutter's hot reload feature helps you quickly and easily experiment, build UIs, add features, and fix bugs.

Hot reload works by injecting updated source code files into the running Dart Virtual Machine (VM). After the VM updates classes with the new versions of fields and functions, the Flutter framework automatically rebuilds the widget tree, allowing you to quickly view the effects of your changes. With Flutter, we believe we have a solution that gives you the best of both worlds: hardware- accelerated graphics and UI, powered by native ARM code, targeting both popular mobile operating systems.

I. INDEX OF CONTENTS

==

II. INDEX OF FIGURES

==

LIST OF ABBREVATIONS

- SDK - Stands for Software Development kit. A toolkit that helps making the development of software applications easier. They often include debugger, compiler and software framework.

- CROSS-PLATFORM - Often refer to a product(in this case application) that can run on different platforms.

- NATIVE APPLICATION - Native in the application world refers to the development of an application that is specifically developed for an operating system (OS).

- SWIFT - A Programming language developed by Apple Inc which is used to create iOS native applications in this project.

- KOTLIN - A Programming language developed by JetBrains which is used to create Android native applications in this project.

- DART - A Programming language developed by Google which is used to create Flutter applications for IOS and android in this project.

- LLVM - Low level virtual machine. A Collection of compiler and tool chain technologies which are reusable and modular.

- LLDB - Low level debugger. A debugger that is the default debugger in XCode and part of the LLVM collection.

==

1. INTRODUCTION

1) OVERVIEW:

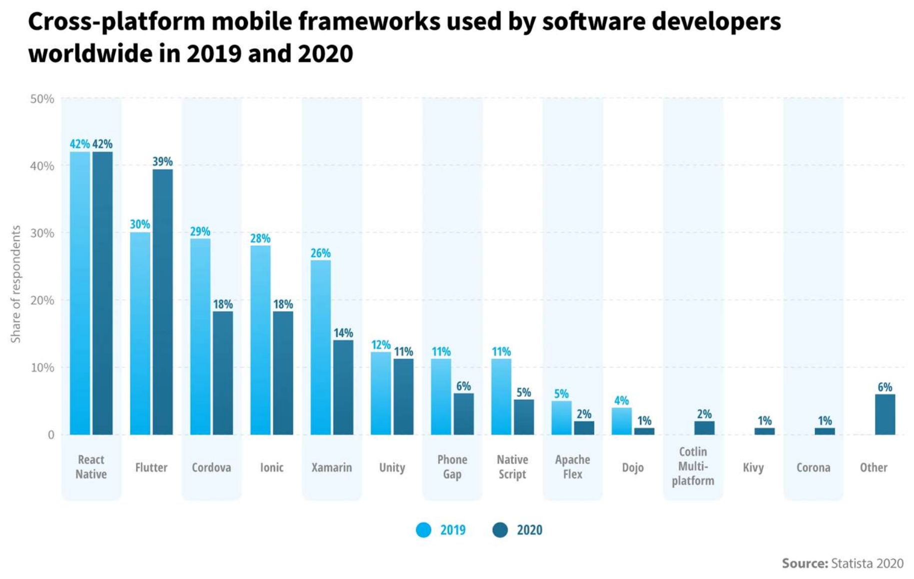
1. What is flutter ?

Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase. Flutter was introduced by Google as an open-source technology for coding and creating native apps for Android and iOS. Flutter is relatively new as it was officially presented in December 2018 as the first stable version 1.0 at the Flutter Live event. Flutter combines ease of development with performance similar to native performance while maintaining visual consistency between platforms. Flutter's programming language, Dart, was initially intended as a replacement for JavaScript. Most importantly, Flutter is open-source and completely free. At the moment, Flutter has equal popularity with React Native on both GitHub and Stack Overflow. Google uses Flutter for various Google Assistant modules and the Google Home Hub user interface. Moreover, there are already 50,000 Flutter apps available in the Google Play Store, and this number is increasing at a high rate. Alibaba Group, eBay, Groupon, and other popular e-commerce providers use Flutter as well to give their web and mobile applications uniform looks. Google also claims that, Flutter is a powerful, general-purpose, open UI toolkit for building stunning experiences on any device-embedded, mobile, desktop or beyond.

2. Why to use flutter ?

Looking at Client-Side implementations, Flutter apps are already highly optimized for speed and are known to perform better than existing cross-platform application development platforms, such as React Native or Apache Cordova. By default, Flutter apps aim to render at 60 frames per second on most devices and up to 120 frames per second on devices that support a 120 Hz refresh rate. This is made possible because of Flutter's unique rendering mechanism. Also, from development and maintenance perspective developers are not restricted to a single cross-platform mobile framework. In fact, statistics show that while the majority of developers use React Native (42%), Flutter usage in 2020 (39%) has highly grown in comparison with Flutter usage in 2019 (30%).

Let’s take a closer look at explaining why Flutter application development can be a better choice.



1. **Quick code development**

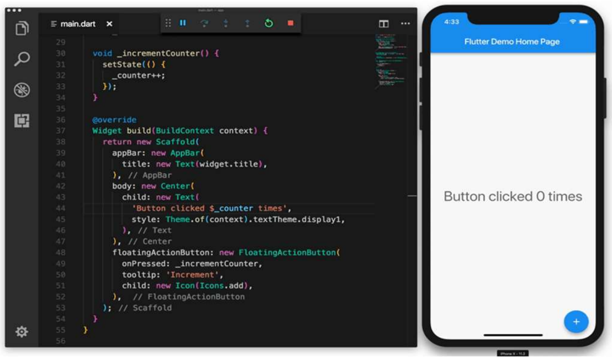
Flutter creators wanted to invent a technology with the quickest opportunity to deliver a great-performing, cross-platform mobile application. The following features allow this:

- **Widgets:**

One of the most significant benefits of Flutter is how it uses ready-made widgets. This ensures that Flutter offers a consistent model for development and design. Widgets are Google-based, so they are high code quality and perform better than other open-source frameworks. As most of them are extremely customizable, they save developers’ time like no other framework. In addition to the major layout widgets, Flutter widgets follow both the Material and Cupertino looks, which is a huge advantage. Minimal code and access to native features.

  - **Hot reload:**

  Flutter’s hot reloading helps save time while developing by letting the developer see the applied changes in real-time. This capability helps developers be significantly more efficient and productive. Flutter’s hot reload works better than competitors’ similar features. It allows the developer to pause code execution, make changes to the code, and continue the code from the same place. This greatly speeds up development and allows more experimenting.



```
  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text(widget.title),
      ), // AppBar
      body: new Center(
        child: new Text(
          'Button clicked $_counter times',
          style: Theme.of(context).textTheme.display1,
        ), // Text
      ), // Center
      floatingActionButton: new FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: new Icon(Icons.add),
      ),  // FloatingActionButton
    ); // Scaffold
  }
}
```

Fig A: Hot re-lode feature in Flutter

1. **Great UI**

   Will Larche, Software Engineer at Google, says, "Flutter's architecture is designed for building beautiful, custom UI. Flutter's main goal is to make building polished, custom app interfaces a faster, more delightful experience for designers and developers. Flutter is powerful enough to draw anything designers dream up.

   - **Beautiful, custom design**.

     The powerful thing of Flutter is Skia, the open-source, high-performance graphics engine used by Adobe, Chrome, and Amazon Kindle. Flutter allows users to develop applications with custom designs, which will look equally good on iOS and Android devices. Applications developed on Flutter — unlike its competitors — have no risk that there will UI failures when updating the software.

Fig B: UI design in Flutter

- **UI customization potential**:

  A huge advantage of Flutter is the ability to customize everything you see on screen, regardless of the complexity of the element. The amount of effort required is fundamentally lower from that required on native platforms' development software.

- **Same business logic and UI**:

  Possible for Flutter sharing the UI and business logic on Android and Apple devices allow developers to achieve a seamless experience regardless of the OS. This is primarily important for brands with a unique and outstanding corporate style. Flutter doesn't need any platform-specific UI components to render its UI. The one and only thing Flutter needs to show the application UI is a canvas to draw on. It can be illustrated the following way:

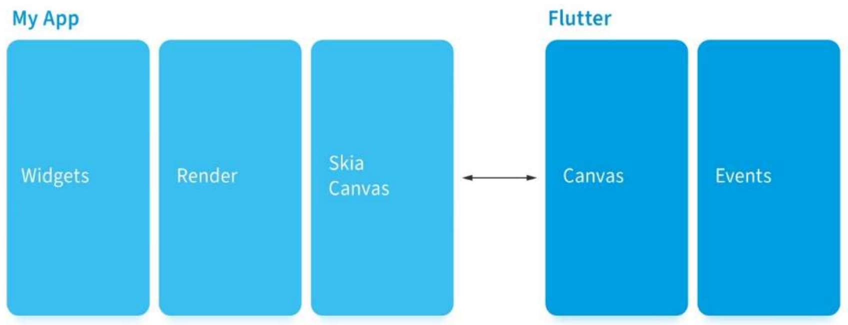


Fig C: UI design in Flutter

2. <u>**How we can implement flutter cross-platform application in real world problems ?**</u>

As we know, flutter provide rapid and easy implementation of mobile applications. Cross-platform development is the ability to build and deliver apps that can run across multiple device platforms. There are many cross-platform app development platforms currently available — Xamarin by Microsoft, React Native by Facebook and Angular by Google. All are available on the market before Flutter. The thing is, which platform should startups select to make the most of a mobile-dominating world?

In my opinion, Flutter is a promising alternative. For the time being, Flutter has swiftly acquired the top position in the list of cross-platform development. Just three years from the initial release, this Google-backed open-source platform has a vast community that consists of over 2 million developers. Flutter is developed by Google - one of the big four, so your maintenance will be supported better in the long term than other frameworks.

You could refer to the below chart for close comparison:

Now let's go through some of the critical attributes of why startups should consider Flutter:

- Fast development
- Intuitive and rich user experience with its own rendering engine.
- Close to native app performance.
- Ideal for MVP app development.
- Flutter is open-source, so you do not have to worry about the budget.
- Fast App Development with Hot Reload.
- Easy integration with the existing app, which means you can integrate new UI in the old app.
- Straightforward Integration with Firebase.

2) MOTIVATION

Motivation behind this experiment to learn cross platform application development is to develop a fully independent skillset that can help to get exposure to emerging technology to build android as well as iOS application for small scale business. For me flutter is killing two sparrows with one arrow. Choose flutter for this seminar was not an easy choice as flutter is still in its very initial stages and community for flutter development is growing up at it pace. But never the less flutter is more rapidly growing than anything in Software Development. Flutter as its own YouTube to guide you in this. It is much faster and easier to learn as there are a lot of resources. If you are new to android or iOS development and you have knowledge about any other language like Java, C, Python etc. then you should go for Flutter. You can build apps for various platform via single code base specifically when it comes to Android and iOS applications. Secondly, there's no need to get separate IDEs and separate person for each platform. You can just do it yourself. Considering this on a bigger scale, it is much easier to manage apps for Android and iOS from single code instead of managing the Android one with another code and iOS one with another plus managing the cost for each platform is another headache. If Flutter's advantages can provide all the capabilities that an application needs, I would choose it in a heartbeat. That is because, aside from the few areas where Flutter is currently lacking, all other aspects of it are ones that I have come to admire and rely on in developing and launching the five different Flutter applications in the past year:

- Flutter's ease of getting started
- The speed gains that can be had from using a framework that was designed from the ground up to focus on speed of development
- The rich ecosystem of readily available Dart libraries and third-party packages
- Flutter's different but extremely productive way of developing applications using the everything is a widget approach.

3. LITERATURE SURVEY

The literature study for this paper was executed to give the reader a better understanding of the cross-platform and native applications. Studying how Swift/iOS, Kotlin/Android and Dart/Flutter handles development and compiling. It contains earlier studies on cross-platform and native comparisons.

The peer reviewed publications that were used for this paper comes from the 9 databases: Google Scholar, Diva, IEEE and BTH Summon. The following keywords and sentences were used:

- Cross-platform vs native
- Google Flutter
- Cross-platform performance
- Native
- Cross-platform tools
- Mobile application performance

Another search word that was used in the search for material on performance was" PhoneGap performance". By looking at the earlier related work, the majority of studies found on cross-platform comparisons uses PhoneGap which makes it a relevant search word for finding comparative studies. The tools official pages were used for primary information about documentation and code standards. Online technology articles were used to find general information on differences between the ways of creating applications. Multiple articles were used to ensure a strengthening of the information that came from online articles. With Flutter being fairly new, it was hard to find a good number of publications. This is why most of the studies and articles specifically about Flutter are taken from non-peer reviewed publications. However, the priority was still put on finding peer reviewed publications in the first place. Information on cross platforms versus native and books about Flutter was taken from the earlier mentioned databases.

Snowball sampling was used on the first scholarly database results that showed up from searching with the search words. All of the paper types that was found in the scholarly databases were used to conduct the literature study.

4. SYSTEM DESIGN

In this report an experiment was carried out to answer RQ1 and RQ2. This section presents the experiment and how it was prepared, its goals and the process itself. It is done to give the reader a better understanding of how the experiment was executed and planned. Due to outside influence, the method had to be changed significantly from the prior method of this thesis work. In summary, promised software that were to be used in the project was not delivered. Therefore, adjustments had to be made that changed the method and execution significantly. This will be further analyzed in the analyses section of the report. Battery consumption is an important metric for mobile applications CPU usage. The measurement of battery consumption was not possible due to the measuring tools needing a connection through USB cord which made the mobile phones charge automatically while performing the tests. Four applications were made with Flutter(which creates two applications), Kotlin and Swift. The applications were created to look like each other and with code that followed according to each documentation.

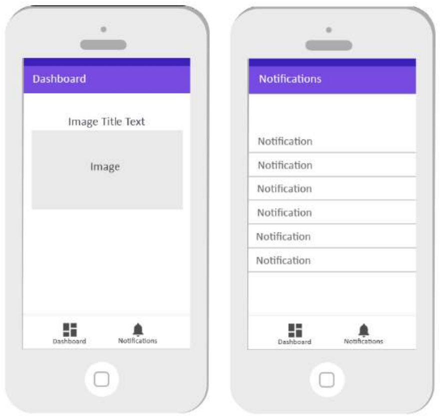


Figure E: Example application layout appearance

A relatively simple layout was chosen because of the time constraint of the project and to be able to handle complex code. The application layout consisted of a navigation bar and two page views "Dashboard" and "Notifications". Dashboard were labelled with a picture of a salad and had the headline "Spring Salad Recipe" while notifications had a simple list of labels with the word "Notification". Each view had an app bar at the top with the view name. For this system to be able to represent a real-life system, the UI and ow of the application were inspired by applications such as: Tasty application and the notification history application. Navigation bars exist in most of the applications today that has content and is not dependant on the number of items that exist in the bar itself, but the actual function and look of the navigation bar. The example application was inspired from the "Tasty" application where the same look of the navigation bar exists as well as the recipe part of the example application. A similar list view to the one that is used in the example application design, can be found in the " notification history" application.

A. <u>SYSTEM ARCHITECTURE:</u>

- A brief history of mobile app development:

Mobile app development is a relatively recent field of endeavor. Third-party developers have been able to build mobile apps for less than a decade, so it is no surprise that tools are still evolving.

- The Platform SDKs:

The Apple iOS SDK was released in 2008 and the Google Android SDK in 2009. These two SDKs were based on different languages: Objective-C and Java, respectively.

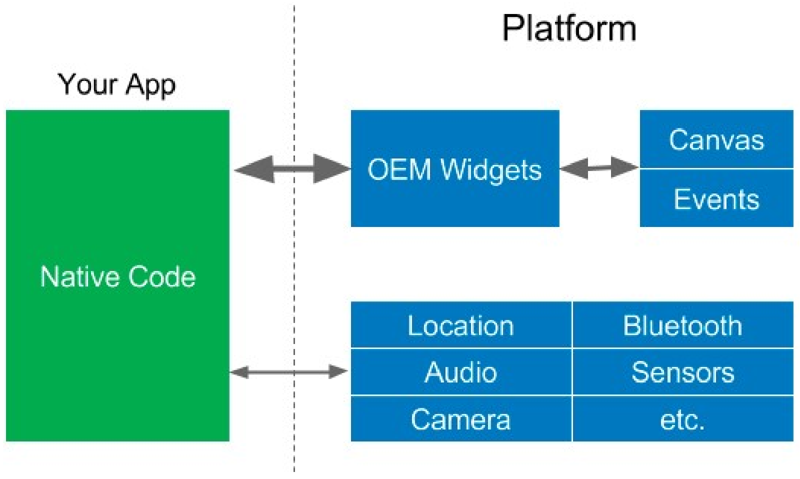


Fig :1 Native System Architecture

Your app talks to the platform to create widgets, or access services like the camera. The widgets are rendered to a screen canvas, and events are passed back to the widgets. This is a simple architecture, but you pretty much have to create separate apps for each platform because the widgets are different, not to mention the native languages.

- WebViews:

The first cross-platform frameworks were based on JavaScript and WebViews. Examples include a family of related frameworks: PhoneGap, Apache Cordova, Ionic, and others. Before Apple released their iOS SDK, they encouraged third party developers to build webapps for the iPhone, so building cross-platform apps using web technologies was an obvious step. Your app creates HTML and displays it in a WebView on the platform. Note that it is difficult for languages like JavaScript to talk directly to native code (like the services) so they go through a "bridge" that does context switches between the JavaScript realm and the native realm. Because platform services are typically not called all that often, this did not cause too many performance problems.

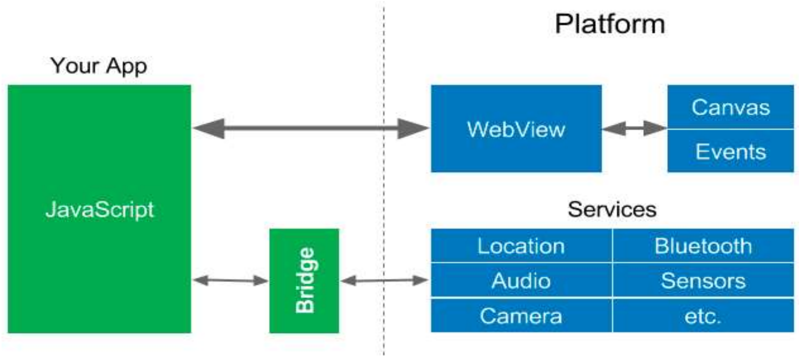


Fig : 2 WebView System Architecture

- Reactive Views:

Reactive web frameworks like ReactJS (and others) have become popular, mainly because they simplify the creation of web views through the use of programming patterns borrowed from reactive programming. In 2015, React Native was created to bring the many benefits of reactive-style views to mobile apps.React Native is very popular (and deserves to be), but because the JavaScript realm accesses the platform widgets in the native realm, it has to go through the bridge for those as well. Widgets are typically accessed quite frequently (up to 60 times a second during animations, transitions, or when the user "swipes" something on the screen with their finger) so this can cause performance problems.

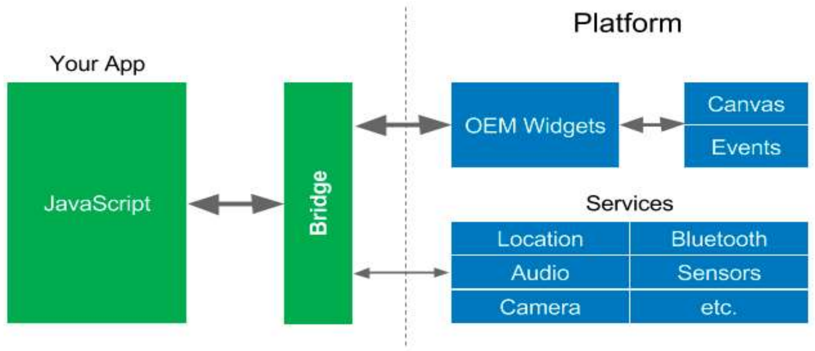


Fig 3 : Reactive Views System

B. DATA FLOW DIAGRAMS

The goal with this experiment was to out the difference between the Flutter and the native applications in terms of run time CPU usage. Another area that was studied, was the amount of code that each development environment required to create the desired looks and functionality of the applications. This was a side-to-side measurement and gave an insight of how the native code bases compared to the Flutter code base.

- Execution:

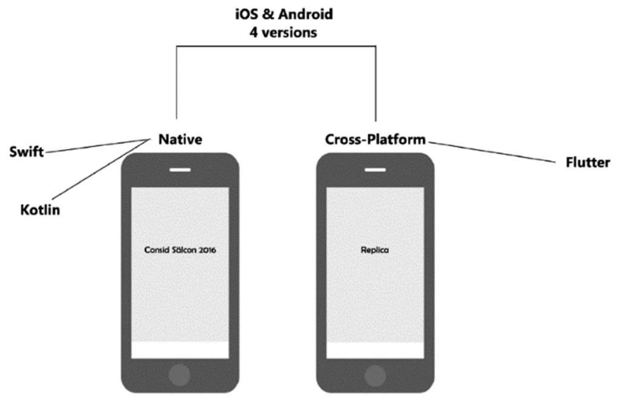


Figure 4: Graphic representation of the applications and their technology

Four applications in Flutter, Kotlin and swift were created and built. Each development environment was setup accordingly to their documentation:

- Flutter
- Android - Android Studio
- iOS – X code

The CPU usage was measured three times manually per application build and the highest, lowest and mean values were written down. Values that were higher than 0 was chosen for the lowest, as the CPU showed 0 when the applications. Were passive. The tools that were used to measure were: Android studio profiler (Android) and XCode instruments (iOS). Because the output from the measurements instruments only showed as graphs, sampling was used to determine mean value and ease measurement. Standard deviation was then calculated for each set of values. The collected values and mean of each test case were gathered in a summarized table. The author created the applications and measured development time without prior experience with application development. The first prototypes were used for the experiment. Code complexity was measured by the author through the code review. Devices that were used for the experiment were one iPhone 7 and one Android Samsung S7. These mobiles were chosen to match each other's performance for more precise results and because of easy accessibility.

Exact device specifications that were used for this experiment can be found in Appendix A under "Device specifications". To create a base that would align to each language's standards, terminal commands and templates were used to generate a code base that contained a navigation and two connected views. This was done to follow the documentation as much as possible. Package and tools can be found in Appendix A under "Tools and packages". Default margin was used when positioning parts such as images and lists. This means that the recommended margin was used for each development environment. The navigation route below was followed manually when measuring the CPU usage. It was done to generate a good amount of data of the applications functionality and to ensure all the test measurements to follow the same guidelines.

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

Preparation of the Devices for testing. The devices were prepared by doing the following before the measures:

- Fully charging the battery
- Activating Flight mode
- Clearing the processes by restarting the phone
- Restarting the application before each measurement

The hypothesis is that Flutter performs equal to or better than a native based application in this particular study. This theory is based upon the earlier study made by Connick where he performed an experiment where Flutter was compared to native android, Xamarin forms, native iOS and react native which showed that Flutter had a lower CPU usage.

- **Survey:**

  To answer RQ3, a survey was created and sent out to people who worked or had been educated in the software industry. Two of the applications android native and android utter were compared without the participants knowing which one was android native or android Flutter.

- **Preparation:**

  Preparing for the survey, the earlier created android applications in the experiment was used for each participant to analyse and compare. The following setup was done to identify each application for later use in the results:
  - App A - Android native
  - App B - Flutter Application

- **Survey Goal:**

  The survey goal was to and out how or if Flutter feels and looks different from a native application. This was done to see how much a user notices the differences and what impact it has for them. Results from this survey were then used together with the results of the experiment study on performance to conclude on the differences between native and Flutter.

- **Target group:**

  The target group for this survey will mainly consist of employees at Consider, students and teachers at Blekinge Institute of Technology. These people have a high probability to have knowledge about web techniques and a basic design knowledge as well as using many different applications daily.

5. METHODOLOGY USED IN SEMINAR

This section will give an insight in the backgrounds of Flutter, Swift and Kotlin, as well as the development and building environments that were used in the experiment. It will provide support for RQ1 and RQ3. The goal of this literature review, is to help the reader and author understand how each language and environment works and what differentiate them in terms of functioning, management of looks and development.

To find the results for the three research questions and the under-laying support for this paper, three different methods were carried out. A literature study was conducted to give an understanding to the author and the reader how Flutter works. It contains information on how the three tools and languages that was used in the experiment, works in development. The research that was found in the literature study acted as a support for methods used in the experiment.

To answer RQ2, an experiment was used. The experiment included four applications that were made in Dart with Flutter, Kotlin with Android Studio and Swift with XCode. Two of them were made of the single Flutter and Dart code base. It was carried out to find out how Flutter compares in the Ares of CPU usage compared to native applications.

RQ1 was answered by studying the authors code review for the development of the experiment applications. It was done after the development to ensure that the code bases were complete before examining. Development time was measured by taking the time that each of the applications took from that of generating the starting layout to compiling and running the final version. Measurements of lines, files, size and application size were taken from the finished projects and applications.

A survey was created and sent out to answer RQ3. It had questions revolving look and feel between the two earlier created android applications. This was done to see if the survey takers could notice any difference between them and to see how much difference it would make for them as users.

6. NATIVE MOBILE APPLICATIONS

The meaning of native in the field of mobile applications refers to applications that are built to run on a specification platform or OS. There are numerous languages that can be used for building native mobile applications and a few of examples of these are:

- Kotlin
- Java
- Swift

Mainly, developers have been focusing on the native building of applications because of the customization's that is enabled for the native devices such as camera access.

❖ Native implementation of UI:

One thing that is associated with native mobile applications, is that there are more fluid looking animations and easier integration with the mobile technology. Native applications inherit the targeted platforms looks and apply it to their appearance, this is called Native UI.

This allows the native applications to behave and look more accordingly to the "native" system to give the user a more relatable experience to the mobile platform OS itself.

7. CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT

Cross-platforming refers to a product or software that can be used on another platform than what it was developed for. In app development the principle of cross-platforming is to build and maintain only one code base, which is the alluring part of using it since it saves time in development compared to native that is limited to only one platform per code base. Examples of cross-platform frameworks/tools are Flutter, React Native and Ionic.

A. FLUTTER:

Flutter is a mobile SDK and UI tool that is developed by Google, based on the programming language Dart and officially released in 2018. The developer's goal is to deliver applications that comes as close to the performance and look and feel of native applications.

B. DART:

Dart is a programming language created by Google. It is a client-optimised language that can be used on multiple platforms. It has C-style syntax and is object oriented using classes. Dart can be compiled into both JavaScript and
native based code.

- **Flutter/Dart UI management:**

  Flutter uses widgets as the main concept within in the code. Widget is the nickname for every component part that is built in Flutter. This could mean a box or a text that is referred to as a widget. A noticeable part of the widgets is that they are created by the Flutter developers to look native and developers are able to fully customize these to their liking. Flutter uses a material components library by default which lets the developer use components that are ready from the beginning, which is a concept web technology often work with Flutter has a development functionality called hot reloading which means that when changes are made, they are injected into the dart Virtual Machine and Flutter rebuilds the structure and let you view the changes that was made faster.

- **Flutter compiling:**

  The way Flutter works when running on Android is by compiling the developers written Dart code to native with the help of AOT compiling. These together with x86 libraries that were created when compiling the dart code, are put into a runner and built as an APK. Flutter runs similar on iOS system, but instead the Dart code compiles into ARM library and is later put as a runner and built as an ". ipa." Flutter themselves suggest the whole process is similar to how game engines works and that is how the developers explain it on their website.

- **Use and popularity:**

  In a paper written by Mehdi Satei 2019, there is a section where the author discusses which programming languages and frameworks that are popular in the industry. Satei presents statistics from 2019 where Dart is on the rise of most wanted languages while Flutter is top 3 on most wanted frameworks.

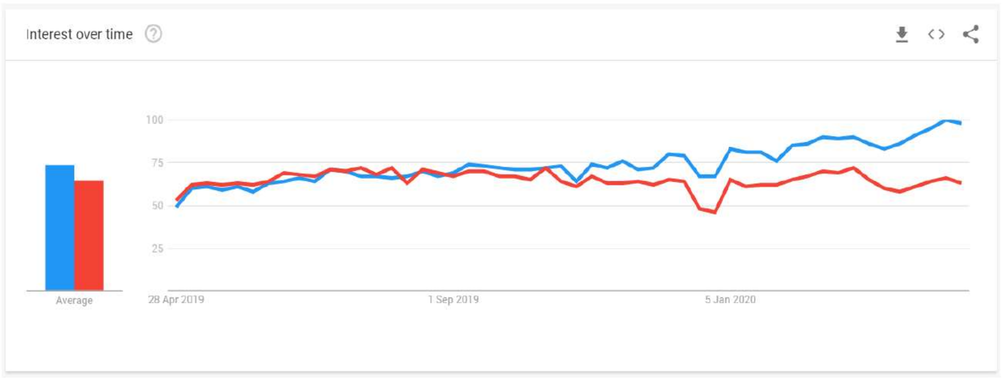


Figure 5: Graph of React Native and Flutter interest of search from Google trends (React Native is red and Flutter is blue)

Looking at the Google trends chart for Flutter in comparison to React Native, there is a difference starting in trends for searching which means that more people are taking interest in Flutter development. Flutter is however, not used much when looking at the Stack Overflow survey of 2019 where Flutter is only used by 3.4% of all users and not even on the list for the professional developers use. 1.9% of the users liked Dart while for the professional developers use, Dart was not on the list. Both Flutter and Dart is however very highly ranked in the "Most loved" categories as 3rd and 12th.In the Stack Overflow survey of 2020, Flutters use statistics has increased from 3.4% to 7.2% in the overall statistics, but stays off the list as before in the professional only statistics. Flutter stays in the 3rd position as the most loved framework/library/tool for 2020.Dart is still off the list of the professional use of Dart, but has increased from 1.9% to 4.0% since 2019. For the 2020 Stack Overflow survey, it has become the 7th most loved programming language.

C. KOTLIN:

Kotlin is a programming language that is made by JetBrains. It is open-source software statically typed language. The result when doing so would be a compiling error. Kotlin is fully compatible with Java and supported on the Android platform for creating Android applications.

- **Kotlin compiling:**

  According to the documentation, Kotlin compiles into compatible bytecode for Java when targeted on the JVM. Kotlin is a versatile programming language that aims to be able to function on multiple platforms. If targeted to native, Kotlin will go through the LLVM to produce specification code for the platform in question.

- **Kotlin/Android UI management:**

  When designing an UI in Android together with Kotlin, the developer is inclined to use the Android view group system. The layout and UI elements can either be declared in XML or in code at run time. Drag and drop is available for the non-programmable layout creation. Both of these options need to have the UI parts connect to the code and will need to be initiated upon creation in the on Create method for the class layout. The objects in the layout is referred to as viewGroups and views.

- **Swift:**

  Swift is a programming language created by Apple Inc 2014. It was made to be able to work on multiple of platforms and aims to be a replacement to the C-languages. Swift is managed as a group of projects where it is split into: swift compiler, standard library, core libraries, LLDB library, swift package manager and XCode playground support.

- **Swift compiling:**

  Swift code is compiled down to machine code with the help of seven level steps in the compiler. The parser is run at the beginning to check if there are any grammatical errors and show warnings. After the parser, a semantic analysis is carried out to see if it's safe to compile the code without errors. Clang importer then imports clang modules that can be referred from the analyser. This is followed by something called a SIL generation and SIL guaranteed transformations. The first SIL runs another analysis on the code to improve optimization. The second SIL do data ow diagnostics to check so there are no uninitialized variables and results in a canonical SIL, meaning that it is a SIL that exists after the optimization and analyses has been done.

- **XCode UI management:**

When developing a UI in XCode, the Interface Builder is used to create storyboards. Scenes are used in storyboard to represent what is seen and what is happening on the device screen. Segues connects the scenes and holds upholds their relation. Scene objects can be dragged and dropped to create a new item to view.

- **Results:**

This section displays the results that were discovered through the experiment and literature review. Pictures of the final applications are found below.

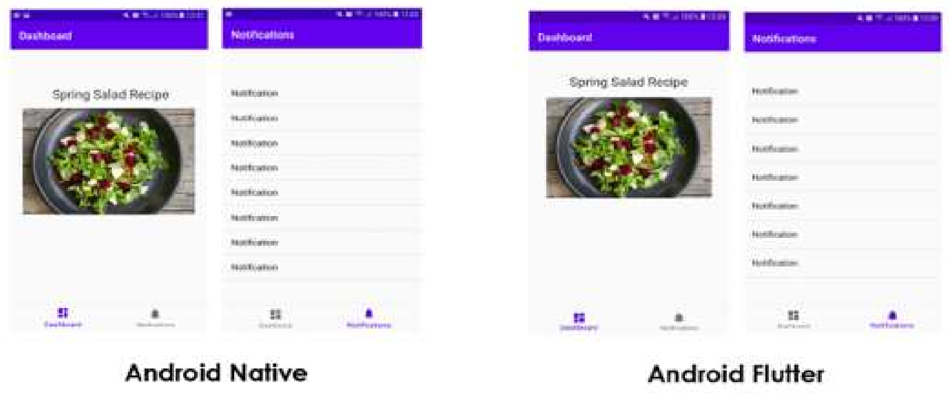


Fig 6 : Images of final android native and android flutter applications

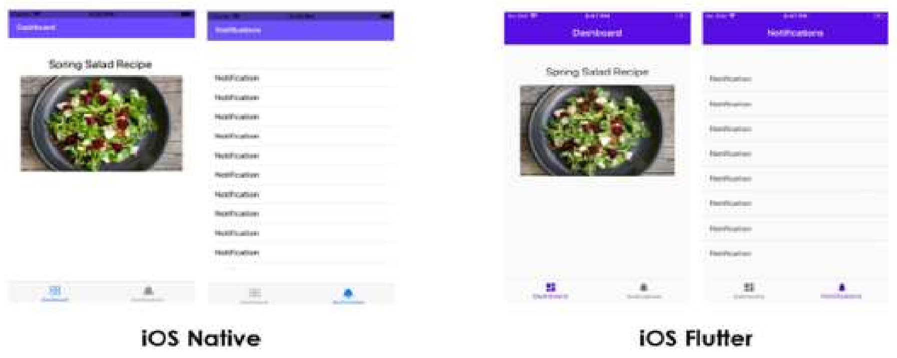
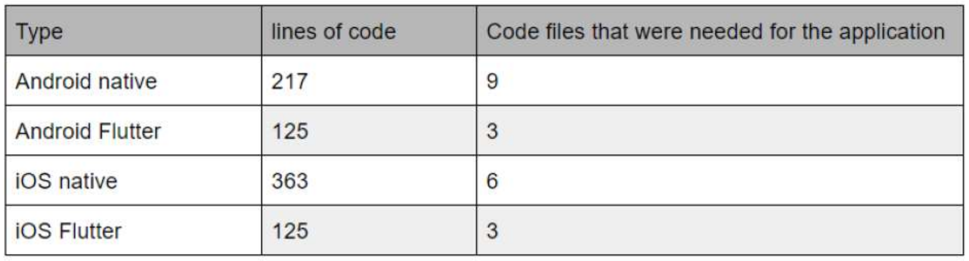


Figure 7 : Images of final iOS native and iOS Flutter application

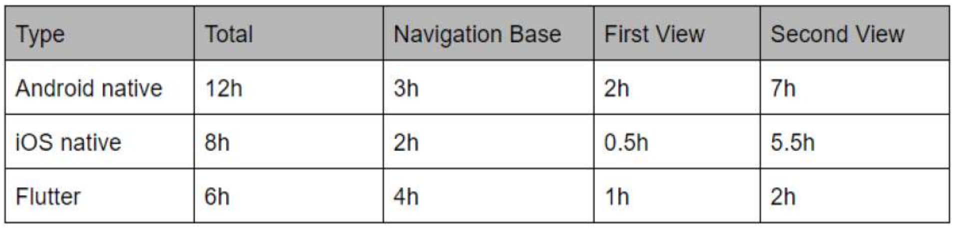
8. RESULTS

❖ **RQ1: How does Flutter differentiate in code size and how complex code is needed in comparison to native built applications?**



Type	lines of code	Code files that were needed for the application
Android native	217	9
Android Flutter	125	3
iOS native	363	6
iOS Flutter	125	3

Figure 8: Application lines of code and file count



Type	Total	Navigation Base	First View	Second View
Android native	12h	3h	2h	7h
iOS native	8h	2h	0.5h	5.5h
Flutter	6h	4h	1h	2h

Figure 9: Development time of each code base

Looking at the results as a whole, Flutter wins the majority of most categories in the development area. There are however some differences that are interesting to take note of when comparing Flutter to native builds.

- **Code Size:**

  As seen in figure 8, Flutter had the lowest amount of code lines and files that were needed in order to create the application. Native iOS had the lowest size of project files and app size but a significantly larger amount of code lines than the other builds. The native android had the greatest number of files created and required lower amount of code lines than the iOS native.

- **Code Complexity:**

  A part of answering Q3 is comparing the code complexity of the development code of each of the applications. The part of the application that was chosen for this was the creating of the notification list. It was chosen in particular because it contained the most code that was written and because the other view only featured an image and a title. The code that is shown in this section is only a part of the application code bases.

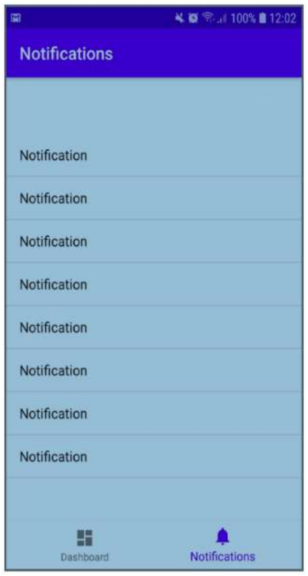
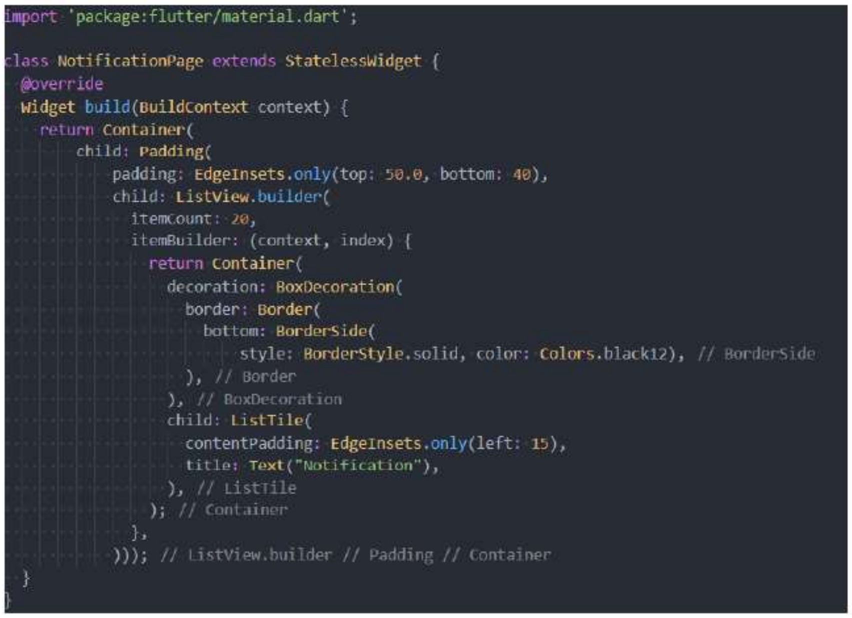


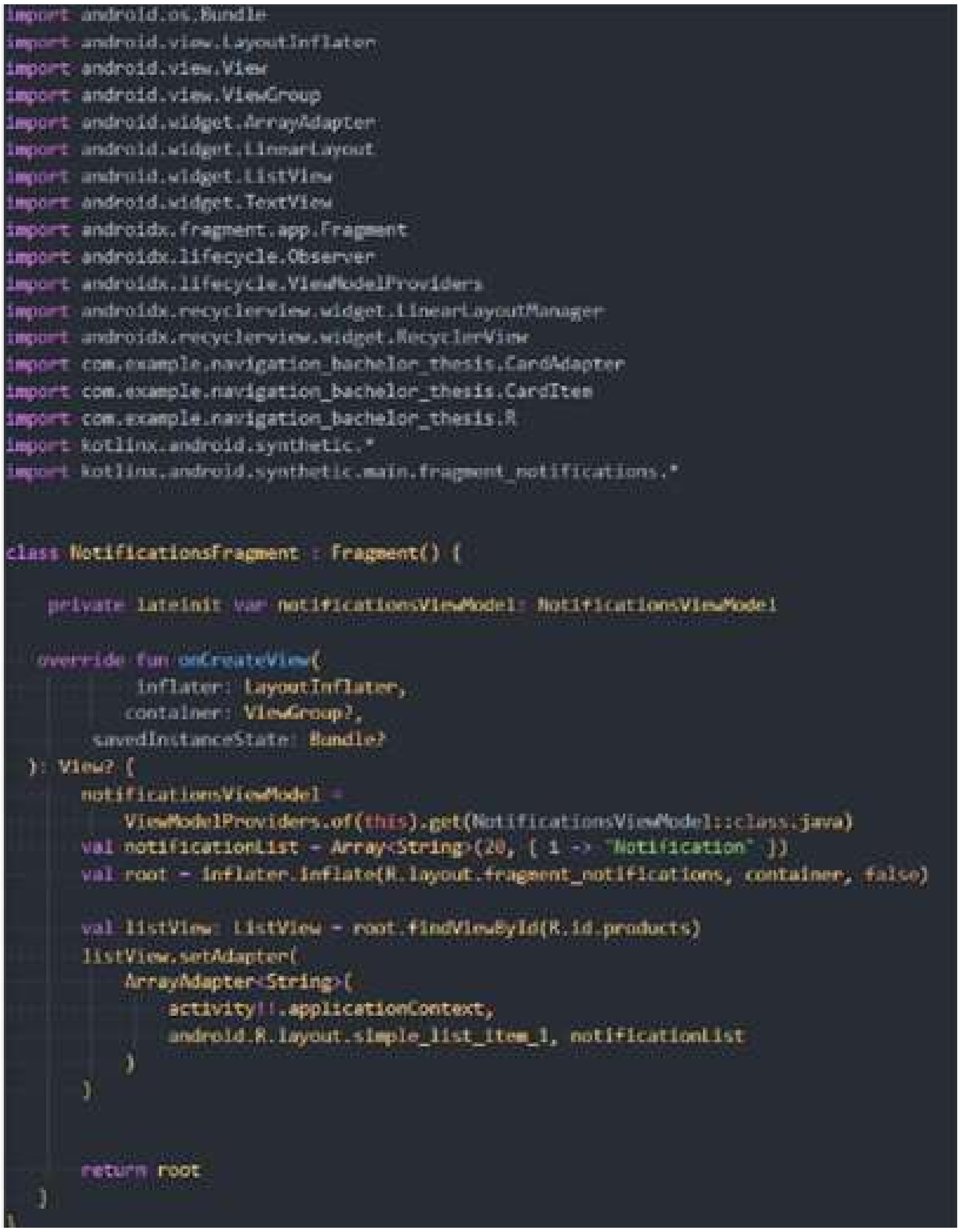
Figure 10: Picture showing notification list that was chosen for the code complexity part



```dart
import 'package:flutter/material.dart';

class NotificationPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
        child: Padding(
            padding: EdgeInsets.only(top: 50.0, bottom: 40),
            child: ListView.builder(
              itemCount: 20,
              itemBuilder: (context, index) {
                return Container(
                  decoration: BoxDecoration(
                    border: Border(
                      bottom: BorderSide(
                          style: BorderStyle.solid, color: Colors.black12), // BorderSide
                    ), // Border
                  ), // BoxDecoration
                  child: ListTile(
                    contentPadding: EdgeInsets.only(left: 15),
                    title: Text("Notification"),
                  ), // ListTile
                ); // Container
              },
            ))); // ListView.builder // Padding // Container
  }
}
```

Figure 11: Flutter code snippet showing the creation of notification

Figure 11 shows how the Flutter code creates the visual layout and functional code in the same code. In the code image, there is a child parent relationship for the widget elements. The code shows that there is a widget that is built to return nested widgets. The child to the main containers Padding, shows the creation of a ListView Flutter widget. This view returns a ListView with 20 items and 20 container items.



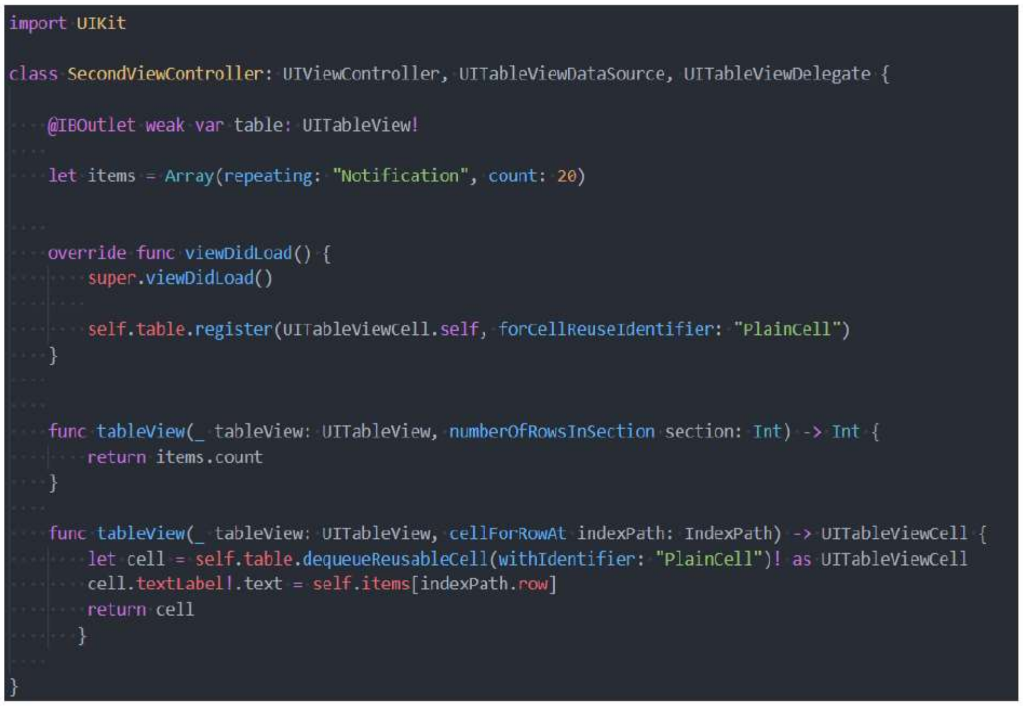
```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.LinearLayout
import android.widget.ListView
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProviders
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.navigation_bachelor_thesis.CardAdapter
import com.example.navigation_bachelor_thesis.CardItem
import com.example.navigation_bachelor_thesis.R
import kotlinx.android.synthetic.*
import kotlinx.android.synthetic.main.fragment_notifications.*


class NotificationsFragment : Fragment() {

    private lateinit var notificationsViewModel: NotificationsViewModel

   override fun onCreateView(
            inflater: LayoutInflater,
           container: ViewGroup?,
        savedInstanceState: Bundle?
  ): View? {
        notificationsViewModel =
            ViewModelProviders.of(this).get(NotificationsViewModel::class.java)
        val notificationList = Array<String>(20, { i -> "Notification" })
        val root = inflater.inflate(R.layout.fragment_notifications, container, false)

        val listView: ListView = root.findViewById(R.id.products)
        listView.setAdapter(
            ArrayAdapter<String>(
                activity!!.applicationContext,
                android.R.layout.simple_list_item_1, notificationList
            )
        )


        return root
   }
}
```

Figure 12: Android native code snippet showing the creation of notification list

In figure 12, the android Kotlin application code declares the view model and import necessary items for the code. Everything happens in the onCreateView function, which inflates the XML layout for the notification list and creates an array with the same strings "Notification" that it injects as data to the already existing ListView XML target with the id "products". The android native code has a lot of environment specific variables that a developer has to take in for consideration.

As shown in figure 13, the iOS swift fetches the table from the corresponding story file and creates an array within, which it registers a tablecell(item) with the name "PlainCell". The TableView function checks the number of items in the created array "items" and returns a cell to the TableView with the text from the array. All this happens in the class for the specific controller in which the notification view exists.

This code is relatively short for its purpose and it is easily structured for a beginner in mobile development. There are language specific parts of the code but otherwise, it could probably be understood by someone who have knowledge in other languages.



```swift
import UIKit

class SecondViewController: UIViewController, UITableViewDataSource, UITableViewDelegate {

    @IBOutlet weak var table: UITableView!

    let items = Array(repeating: "Notification", count: 20)


    override func viewDidLoad() {
        super.viewDidLoad()

        self.table.register(UITableViewCell.self, forCellReuseIdentifier: "PlainCell")
    }


    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return items.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = self.table.dequeueReusableCell(withIdentifier: "PlainCell")! as UITableViewCell
        cell.textLabel!.text = self.items[indexPath.row]
        return cell
        }

}
```

Figure 13 : Image of iOS code

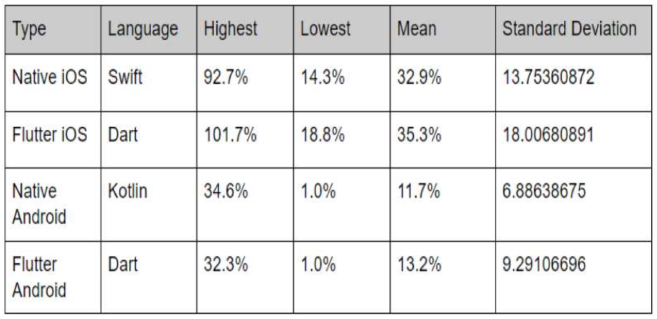
- **Development time:**

  In figure 9 Android native had the longest developing time in total which was 12 hours, followed by the iOS native with 8 hours and lastly the Flutter which had the shortest of 6 hours. In the development of both native applications, the use of drag and drop can be utilized for faster development. This is useful until a certain point when the dropped elements need to be connected with code which was harder to do than generate the layout through direct code. This makes it easier to develop because the fact that the layout is always visible without the need of building it. Corresponding development function in Flutter is the hot reload function which lets you build the application and be able to reload it based on new features added.

❖ **RQ2: How well does Flutter perform in run time CPU usage compared to native applications on Android and iOS?**

Using both the native and utter builds for running CPU usage tests, they showed different results in terms of platform. Both the iOS applications showed higher CPU usage in the beginning as seen in figure 14.

Figure 14 shows the Android versions differentiated little. The Android Flutter application had lower max CPU usage but the native android had a lower mean value and both had the same lowest value. The android applications overall, had a high CPU performance in the beginning but not as high as the iOS applications.



Type	Language	Highest	Lowest	Mean	Standard Deviation
Native iOS	Swift	92.7%	14.3%	32.9%	13.75360872
Flutter iOS	Dart	101.7%	18.8%	35.3%	18.00680891
Native Android	Kotlin	34.6%	1.0%	11.7%	6.88638675
Flutter Android	Dart	32.3%	1.0%	13.2%	9.29106696

Figure 14: Table showing summarized results of CPU measurement. Individual results for each development platform can be found in Appendix D

❖ **RQ3: How much does Flutter and android native differentiate in terms of look and feel for the application users?**

The survey was active during the time period 04/03-2020 to 17/04-2020 and was sent out to people that mainly works or are educated in the IT industry. There was a total of 39 people that answered the survey during the time period it was available. In the survey, the application A was the native and application B was the Flutter application. A full view of the questions and a spreadsheet of all the answers can be found in appendix C. Experience with the operating systems:

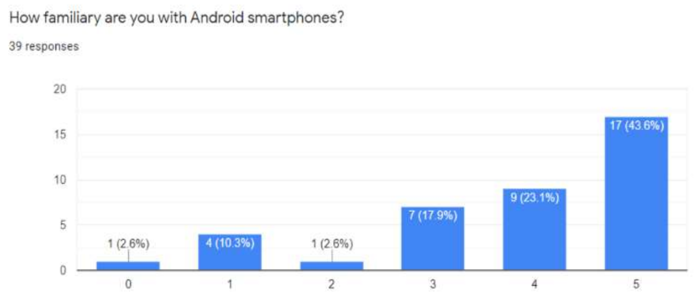


Figure 15: Diagram showing survey takers knowledge of android

In Figure 15, only one of the survey takers had never used an android nor owned one. Most of the takers had 3-5 in experience with android applications beforehand. This meant most of the users were familiar with the android which was an advantage since the pictures and animations were android.

- **Application looks, animation and behaviour:**

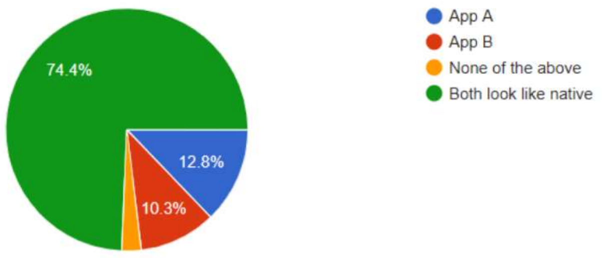
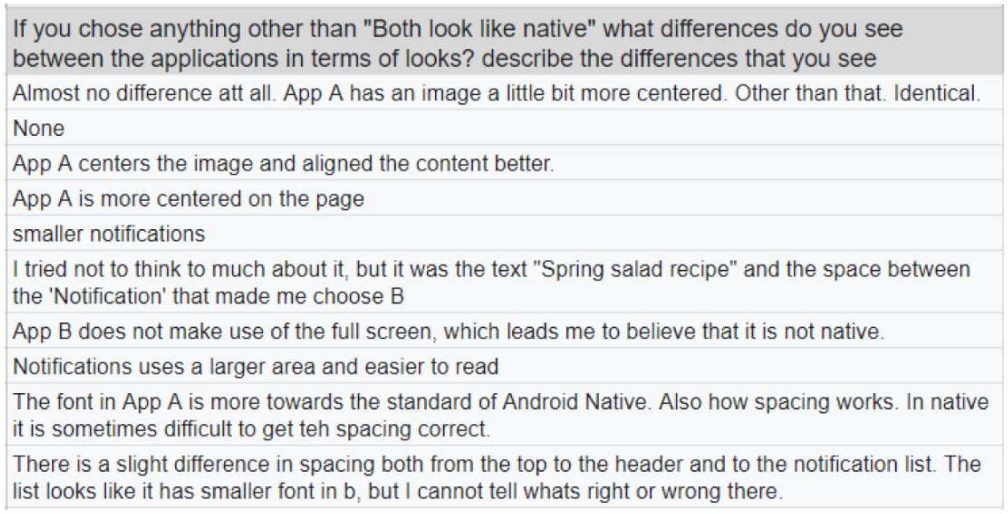


Figure 16: Diagram showing answers for looks of application A and B

In figure 16, 74.4% answered that they thought both applications looked like native, 12.8% answered that app A looked more native, 10.3% answered that app B looked more native and 2.6% answered that none of the applications were native. The ones that answered an alternative that was not "both look like native", had to answer a follow-up question about the differences they spotted which is showed in figure.



If you chose anything other than "Both look like native" what differences do you see between the applications in terms of looks? describe the differences that you see
Almost no difference att all. App A has an image a little bit more centered. Other than that. Identical.
None
App A centers the image and aligned the content better.
App A is more centered on the page
smaller notifications
I tried not to think to much about it, but it was the text "Spring salad recipe" and the space between the 'Notification' that made me choose B
App B does not make use of the full screen, which leads me to believe that it is not native.
Notifications uses a larger area and easier to read
The font in App A is more towards the standard of Android Native. Also how spacing works. In native it is sometimes difficult to get teh spacing correct.
There is a slight difference in spacing both from the top to the header and to the notification list. The list looks like it has smaller font in b, but I cannot tell whats right or wrong there.

Figure 17: Individual answers for looks of application A and B

Figure 18 shows a graph on the difference in looks between the two applications. The question displayed a range of 0-5 where 0 was "no difference at all" to 5 "Major difference e.g., they don't resemble at all". The majority answered around 0-2 and 2 people answered 3 which shows that the majority of people deemed the difference to be small.

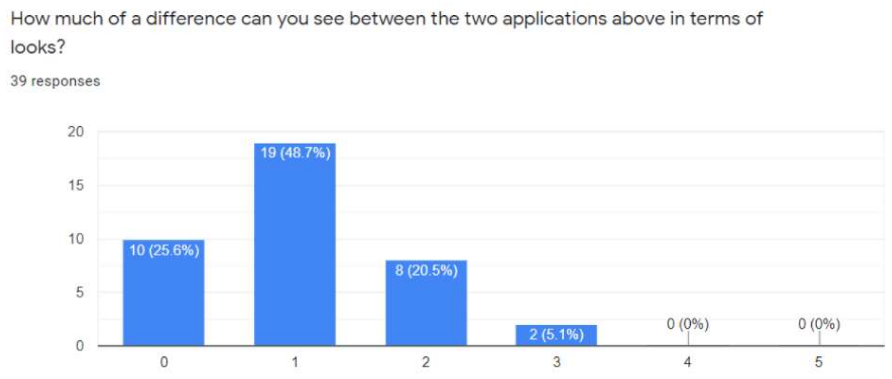


Figure 18: Answers for measured difference of application looks

Figure 19 show that the majority found both looked like native and around a quarter of the takers thought that App A behaved more native and around 12% thought that app B behaved more like native. Comparing these answers to the first question that only regarded looks, the results show a more uncertainty of which one that behaved more native.

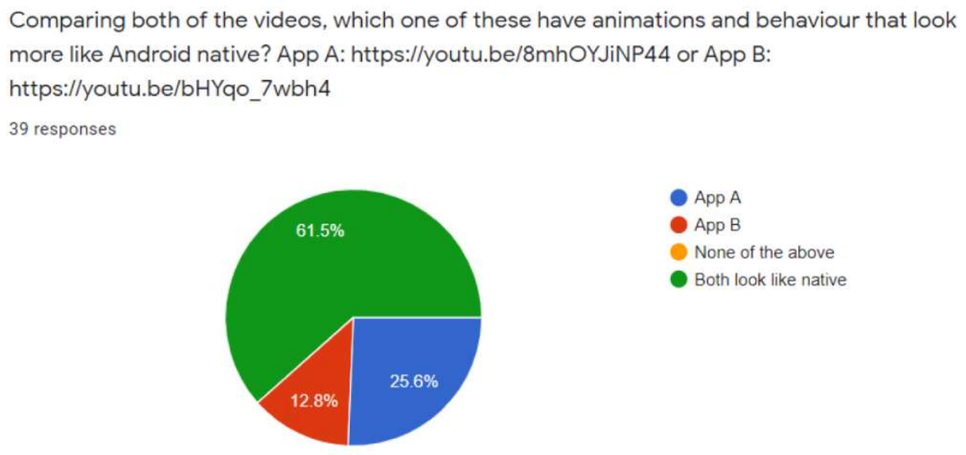


Figure 19: Answers for animations and native behaviour

As seen in figure 20, there were many survey takers that felt like the behaviour of the list was different. Focus was mostly on the list animations and colour difference. One survey taker answered that the App B had more lag which corresponded with another answer that said that App A was more fluid looking.

Below fig. shows how much difference survey takers thought there was between the two applications in terms of behaviour and animations. This question's goal was to give a measurement in how much difference there can be. The majority of the answers were 0-1 of the 0-5 range that was given which were the same type of range like the question that appeared before about looks.

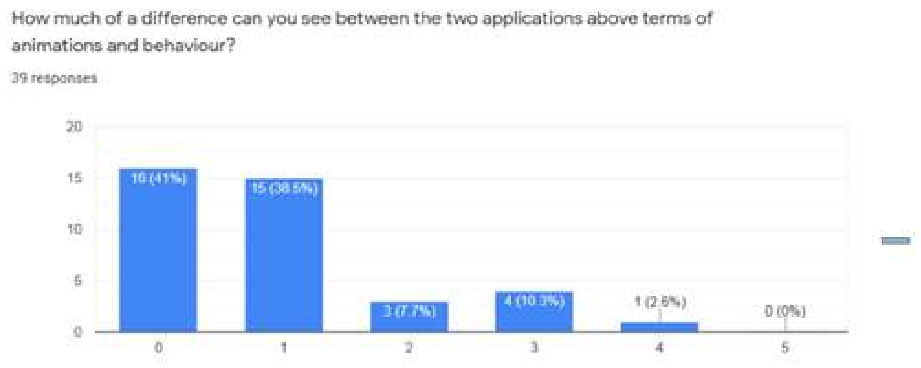


Figure 21: Answers for measure difference in animations and native behaviour

9. ANALYSIS

❖ **RQ1: How does Flutter differentiate in code size and how complex code is needed in comparison to native built applications?**

These results were derived from the method that is found in 5.2, where there is a description of how the applications were developed. Collaboration with a third-party developer, Consider, was initiated. They were to provide professionally developed applications for the experiment. This had eliminated the bias that rises with self-developed applications. The alternative was tried during the course of the work, but the collaboration with the third party failed and the method described had to be used. The author had no prior experience in working with the three tool kits, this created an even playing field for each code base. Between each of the applications, Flutter was the easiest to work with, followed by Swift and last of all android. What made Flutter faster to work with and generated less code when having no prior experiences, was the material components. This is a library of widgets which comes ready to use and are meant to have the native look. However, the nesting that Flutter uses to mount its elements can easily turn into a mess for the developer event though there are auto generated labels that marks out the ending bracket of each widget element to make it easier to spot.

The thing that made the android native code harder to work with was how the android environment and Kotlin code handles the connection to the separate UI layout. There were many android specific parts of the code in Kotlin that made the code more difficult for a beginner to handle. iOS/Swift had a similar approach to both Flutter and android/Kotlin by connecting the UI layout as the android but having an easier styling system like Flutter. Code complexity do not differentiate much between the three applications but has a few areas that are noticeable. This can be due to the application being small and not be comparable to if the applications were bigger. The fact that Flutter is a cross-platform and contains the code base for both iOS and android build is a fact that has to be taken in consideration. The answer to the question based on the results would be that Flutter do keep a shorter code base from a neutral project perspective but has a bigger project size, which means that there is a bigger number of revolving files that is not pure code that holds up the project. This adds up to the considerable fact that Flutter holds a structure for two builds.

There is not much more complicated code that is needed to build a Flutter application compared to a native (both operating systems), there are mostly a difference in the split and connections of the visual and the code parts which is easier in Flutter for the most part. Code complexity was measured by the author's review. This made the RQ1 results inadequate and additional methods of measuring code complexity would be needed to draw reliable conclusions. Examples of these methods would be to have multiple people develop the same applications or measuring the code by a complexity algorithm.

- **RQ2: How well does Flutter perform in run time CPU usage compared to native applications on Android and iOS?**

There was only one other study found on performance comparison with Flutter which was a study in an article which was said to be published but the actual paper could not be found. Beyond this paper, many others were found on comparing cross-platform tools and some native platform. The hypothesis for this question and experiment was that Flutter would perform equally or better than native. This was based on the earlier research that was made before the experiment. The testing for CPU difference in the application showed that there was not much difference in CPU sage between both native OS builds and Flutter. This can be a result of using small built applications for the experiment and the experiment could produce different results if carried out with larger applications. Flutter did not perform as good as previously thought in the hypothesis. Flutter iOS and native iOS showed a higher overall CPU usage than their android counterparts. This could have been the difference in measuring tools that was used or the tools direct measurement on application compared to overall CPU performance.

Compared to the hypothesis, the results corresponded well. The hypothesis was that Flutter would perform the same or better and the results showed that Flutter performed closely to the native builds. Comparing the results against the other studies that is found under related work, Flutter do seem to show equal CPU performance results to native, like its predecessors and competitors. The bigger differences would be located in other areas than performance. When measuring CPU usage for the experiment, 3 measurements per application were averaged. To increase the validity of the results, more measurements could have been taken in order to eliminate the influence of any one measurement of the end results. There may exist considerable differences between the measurements due to the human factor being involved in the experiment (i.e., user input). Automating (i.e., scripting) the execution of the experiment would have removed this effect.

- **RQ3: How much does Flutter and android native differentiate in terms of look and feel for the application users?**

As shown in the results for the first question of the survey, the majority had used or had owned an android at least once. This was positive because android phones were used in the examples of the survey. The results that revolve around looks show that even though most of the people thought that both looked like native, there was a slightly bigger percentage that thought app A looked more native over app B and a small amount that found none of the applications as native. The survey takers found no difference to a small difference between the applications. This shows that Flutter can mimic the looks of a native application without affecting the users too much. An ideal addition to the survey, would have been to use the same application in iOS to compliment the survey that only used the android applications, but this had to be omitted due to time constraints and delayed access to equipment. Not using full screen entirely

- Standard native font and native spacing

An issue that could be a problem regarding the images and videos that were used in the survey is that differences in padding between the applications are different. This could have created confusion around the amount of notification bars in the list. However, looking at the individual answers regarding looks, results show that the survey takers notice the differences in padding but do not mention the possibility of different numbers of notifications. Looking at the other part of the survey that revolves around animation and behaviour, there were similar answers to the first question regarding looks. There was a majority that found that both behaved like native while the others chose either app A or app B. Picks for App A were however around twice the amount of as the App B pick which indicated that there was a more uncertainty with the behaviour and animation compared to the looks portion. The individual answers showed that the most noticeable differences were these:

- Smoother/ fluid animations
- Touch animation on click

10.CONCLUSION

Flutter is a useful toolkit that enables easy ways of creating new applications. It has gotten more and more popular recently and is talked about in the application development industry as a possible replacement of React Native and how it can be compared with native applications. The summery for this is that Flutter can perform up to par with a native application for the type of application size that was tested. To verify these results and determine that Flutter can keep up with native, further testing needs to be carried out. There are multiple other metrics that are needed in order to fully display how Flutter compares to native applications in performance. When it came down to code size, the Flutter application had a low amount of code needed (125 lines) compared to iOS (363 lines) and android (217 lines). Flutter do not split its functionality code and visual code which lowers the amount of code and files. The Flutter code had similarities to web languages and often only required replicating to create new parts. The complexity of code that is needed for utter, is simpler than the code that was needed for the native builds.

Appearance wise, Flutter and native seem to differentiate little to a majority of users. It is able to mimic the native looks and behaviour to a certain point. The biggest difference showed to be around the behaviour and animations of the application where the difference was noticeable on items such as list, menu, default spacing and font. Animations are however something that can be added according their documentation although it requires more of the developer and application as it is not something that is accompanied by the default in Flutter. To conclude the answers and ideas of Flutter, it is a tool with a promising feature if the community continue to grow in the direction that it is right now. The line to drawn when to choose Flutter over two separate native builds, can be chosen at the development of smaller to medium applications which are more flexible. Considering that Flutters strong side is being a cross-platform solution, Flutter still performs good on a single application base if compared to native applications. Flutter may not beat native for developing applications at this point but the results show good potential for the future although further studies need to be done in these areas to conclude safer answers.

A. VALIDITY THREATS

During the experiment, results showed a high CPU usage for both the iOS compiled applications. This can have been caused from the way XCode were set on measuring application performance. The readings showed more processes than the application but it was unclear if they counted as a part of the readings since only the application was chosen to be read in the settings. Another validity threat is that there might have been insufficient knowledge when creating all the applications. This could cause the results in code complexity to be faulty and could have created difference between the applications. A validity threat regarding the video used in the survey is that it might not be equal quality for every survey taker. This is due to the fact that the video was uploaded on YouTube and the quality is dependent on the internet speed of the user watching it. An alternative to having the YouTube videos, would have been to conduct at live survey with a testing group. This was not possible due to the ongoing situation with Covid-19 and the government regulations. This could be misinterpreted because of the difference in padding between utter and the native applications. As earlier stated in the analysis and conclusion, the measurement of development and code complexity is only measured by myself. The results regarding code complexity can therefore not be counted as a full reaction of the reality. Because the experiment was carried out manually, the performance measurement result may be faulty.

B. FUTURE WORK

This paper focuses on the aspects in how Flutter made applications can be used compared to a native application. A big part of Flutter is that its constantly evolves for example adding new widgets and behaviour. The next step for studying Flutter would be to test more of its code and look into how it can be used and developed. For example, Flutter has material implemented which would add to the native look and feel of the applications. These widget features are fully able to be customized according to the Flutter developers and it would be interesting to see how far it could be taken in development. Another interesting aspect would be to test the metrics with heavier data for the applications. Consider had mentioned that they would be interested in seeing how Flutter would work in a full system, meaning that they would want to know if it could work as for example back end as well as for a web page front end. A paper written in 2013 mentioned in the related work section, contains 38 a study that compares native and cross-platform applications with different loads of data, which showed that native performed better with heavier data.

11.REFERENCES AND RESOURCES

- **Case Studies:**
  - A Comparison of Performance and Looks Between Flutter and Native Applications by Matilda Olsson Faculty of Computing, Blekinge Institute of Technology SE-371 79 Karlskrona Sweden on June 13th, 2020
  - O. Axelsson and F. Carlstr om, \Evaluation targeting react native in comparison to native mobile development," Ergonomics and Aerosol Technology, LUP student papers, 2016, p. 105. [Online]. Available: URl : https : //lup.lub.lu.se/student-papers/search/publication/8886469
  - M. R.-S. Guerra, \Cross-platform development frameworks for the developmentof hybrid mobile applications: Implementations and comparative analysis," ESCUELA SUPERIOR DE INGENIERIA GRADO EN INGENIER IA INFORMATICA, hdl, 2018, p. 86. [Online]. Available: http: //hdl.handle.net/10498/20951

- **Referred Books:**
  - Beginning App Development with Flutter by Rap Payne
  - Flutter Complete Reference by Alberto Miola
  - Beginning Flutter: A Hands-On Guide to App Development by Marco L. Napoli

- **Websites:**
  - https://flutter.dev/
  - https://flutterx.com/

- **Other Resource**
  - https://github.com/flutter/flutter
  - https://stackoverflow.com/tags/flutter
  - https://www.javatpoint.com/flutter-stack
  - Official Flutter YouTube channel

12.APPENDIX

- **Device Specifications**

Operating System	Chipset	CPU	GPU	Memory
Android 8.0.0 (Oreo)	Exynos 8890 Octa (14 nm)	Octa-core (4x2.3 GHz Mongoose 4x1.6 GHz Cortex-A53)	Mali-T880 MP12	4 GB RAM
iOS 13.4	Apple A10 Fusion (16 nm)	Quad-core (2.34 GHz 2x Hurricane + 2x Zephyr)	PowerVR Series7XT Plus (six-core graphics)	3 GB RAM

- **Tools and packages**
  - Xcode 11: SwiftUI version that came with Xcode 11
  - Android studio 3.6.1:
    - ✓ Kotlin 1.3.70
    - ✓ Flutter 1.12.13+hot_x.8
    - ✓ dart 2.7.0
    - ✓ font awesome utter: 8.0.1
    - ✓ cupertino icons: 0.1.2
    - ✓ hexcolor: 1.0.1x.8

- **Code base:**

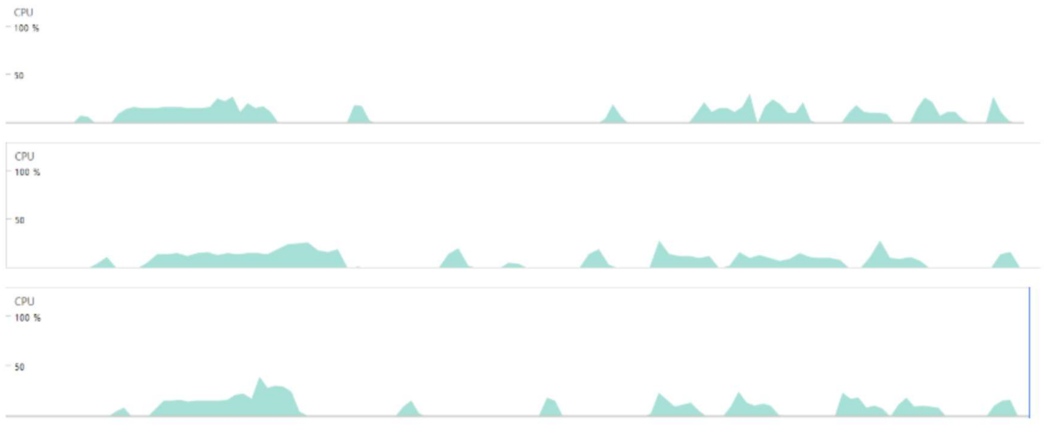
  Code base for the three applications classes can be downloaded here:
  https://drive.google.com/file/d/1sFCSRbfbmi9nG40jzieSSxpn_j8QpjcG/view?usp=sharing

- **Detailed measurements for each environment version :**

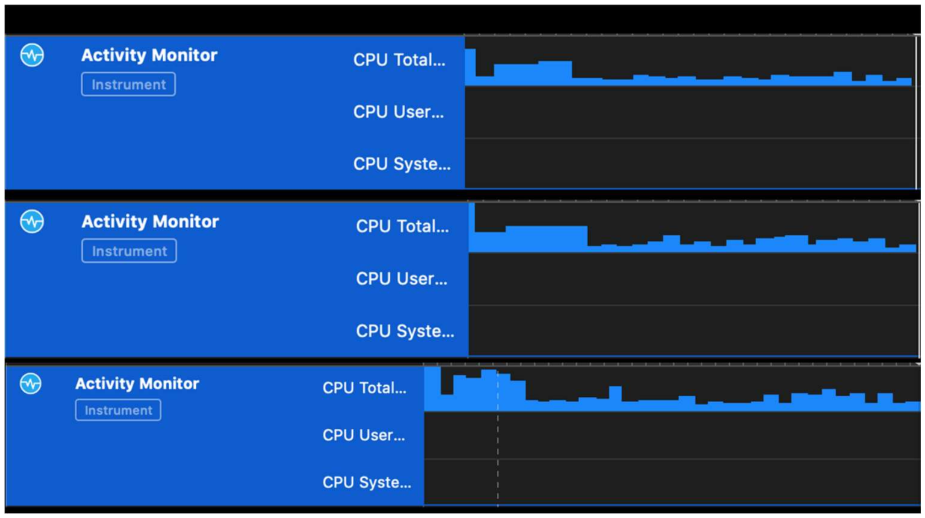
Measure(Flutter iOS)	**Highest**	**Lowest**	**mean**	**Standard Deviation**	**set of values (Sample values)**
M1	89.6%	19.0%	33.0%	18.39126604	68.7, 20.5, 26.7, 28.4, 34.9, 19
M2	115.0%	19.9%	40.0%	25.52022074	85.1, 26, 30.3, 22.7, 55.9, 20.5
M3	100.6%	17.5%	32.9%	10.10893994	17.5, 45.6, 26.1, 32.1, 40.3, 36.2
Average	101.7%	18.8%	35.3%	18.00680891	

Measure(Flutter Android)	Highest	Lowest	mean	Standard Deviation	set of values (Sample values)
M1	31.0%	1.0%	13.8%	7.80811544	0,10, 17,18,16,22
M2	26.0%	1.0%	10.5%	9.81325634	0, 6, 27, 3, 15, 12
M3	40.0%	1.0%	15.5%	10.25182911	0,10,16,17,19,31
Average	32.3%	1.0%	13.2%	9.29106696	

- **Android Flutter graphs:**



- **iOS Flutter graphs:**


Activity Monitor
Instrument
CPU Total...
CPU User...
CPU Syste...
Activity Monitor
Instrument
CPU Total...
CPU User...
CPU Syste...
Activity Monitor
Instrument
CPU Total...
CPU User...
CPU Syste...