jamk.fi

Improving usability of a mobile application. Case: Jaxber

Ahmad Seerat Amini

Master's thesis
May 2021

Information and communication technologies
Master's Degree Programme in Information and communication technologies
Full Stack Software Development

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Description

Author(s)	Type of publication	Date
Amini, Ahmad Seerat	Master's thesis	May 2021
		Language of publication: English
	Number of pages 68	Permission for web publication: x

Title of publication
Improving usability of a mobile application. Case: Jaxber

Degree programme
Master's Degree Programme in Information and communication technologies, Full Stack Software Development

Supervisor(s)
Manninen, Pasi & Huotari, Jouni

Assigned by
Nestronite Oy

Abstract

The purpose of this thesis was to evaluate the current version of Jaxber created by Nestronite Oy, and introduce another alternative technology that would have improved the usability of this mobile application.

Jaxber is a cloud-based mobile application that is available free of charge for the public use and it allows the users to connect to a predefined set of campaigns which may consist of multiple challenges in the form of video, audio, pictures, texts and qualitative and quantitative surveys. Jaxber has been used by hundreds of students in many countries such as France, Germany, Spain, Hungary, Slovenia, Poland and Finland.

Currently the Jaxber is developed by PhoneGap technology and is available for Android, iOS and Windows phone. However, the latest version of Jaxber which is built using PhoneGap framework has some imperfections that intend to be fixed using Flutter technology.

Flutter is an open-source technology that assists the developers to accelerate the development process of the mobile app, at the moment Flutter is widely used for mobile application development, but it might also conquer the desktops and web as well. Applications that are developed by Flutter can be complied to be used in almost any platform like Android or iOS.

To improve the quality of Jaxber in general, a new technology "Flutter" is introduced as replacement to PhoneGap and a new prototype of Jaxber is developed to help the future development of the application.

Keywords/tags (subjects)
Software development, UX/UI, Flutter, Firebase, Jaxber, Nestronite Oy

Miscellaneous (Confidential information)

Contents

Figures

Terminology

API

API stands for Application Programming Interface, which is an intermediary peace of software that enables the software systems to communicate with each other.

CSS

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

CRUD

In computer programming, create, read, update, and delete are the four basic operations of persistent storage.

Dart

Dart is a programming language which is invented by google in year 2011, it is main purpose is to design and develop application for web and mobile.

Firebase

Firebase is Google's mobile application development platform that helps you build, improve, and grow your app.

HTML

The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser.

JavaScript

JavaScript is a scripting or programming language that allows you to implement complex features on web pages.

JSON

JSON is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays.

UI

In the industrial design field of human–computer interaction, a user interface is the space where interactions between humans and machines occur.

VM

Virtual Machine (VM) is a mechanism in computing, virtual machines are based on computer architecture and provide functionality of a physical computer.

1 Introduction

1.1 Research background and motivation

Nestronite Oy is a small startup company founded in 2011. The company spon up from the Digile Cloud Software -program that received funding from Tekes - the Finnish Funding Agency for Technology and Innovation. Currently Nestronite is owned by four persons. Nestronite is a expansive company which is aiming for the international market.

The company is involved in consultant services and data analytics. For the time of writing this thesis the company's main product is called Jaxber, which is a mobile cloud-based solution to capture and share insights, feedback & knowledge. The author is studying the Jaxber current state, define it is limitation and backdraws and suggest solution for better usability and user satifaction.

1.2 Research aim and objectives

When it comes to mobile application development, it is always a cumbersome job to make it work perfectly on almost every device available; since there are a multiple product from different producers and each works with different programming language and operating systems. Tunalı & Erdoğan (2015) state that for mobile app development, there are several alternative approaches that even software vendors usually have confusion about. Some popular approaches are:

- Mobile Responsive Web App Development
- Native App (Platform-Based Native) Development
- Hybrid App Development
- Cross-Platform Native App Development

Commonly it is advisable to use the native technologies for developing a mobile application, in such manner, the application will feel more natural and the performance level will be optimal, but at the same time, building platform specific application would mean extra resource consumption, since targeting multiple platform using native technologies would require disparate development process

(Fentaw, 2020), this would also make the application deployment and update inconsistent across the platform in a way that if there is a functionality change, the same changes must be updated in all platform's code. to overcome these issues, more companies are utilizing cross-platform mobile application development techniques to minimize the development cost, as in cross-platform mobile application development a single codebase will be written, and then it will be built for specific platform.

Jaxber, which is a multi-platform app also benefits from the cross-platform mobile application development technology, although many cross-platform frameworks and tools exists in the market nowadays, PhoneGap has been used to develop the current version of Jaxber.

The aim of this research was to assess the existing version of Jaxber intending to explore what makes it difficult to use the app and what (if any) technical limitations exist. And suggest new solution for future development.

1.3 Research method

The primary goal of this thesis is to explore the available cross-platform techniques of the time for building mobile application development faster that would also provide the users a more native like look and feel, hence this thesis is focusing in answering the following questions:

What is the best possible technology for the cross-platform application development?

How the identified solution could be used to improve Jaxber?

This would not only help the Jaxber app to be improved but also assist other developers and companies in choosing the right solution for their mobile application development.

To answer to these questions, the previous research that has been conducted on Jaxber case has been reviewed, to identify the imperfections and also explore the current technological stack of Jaxber, for instance what methods are used in development process and how the data is fetched.

To discover the weakness and strengths of cross-platform technologies, the information is gathered through online search engine such as Google Scholar, academia journals and papers, books, thesis, and official documentation for specific platform available on their websites. Statistical trend data is also extracted from prestigious online databases like Google trends and StackOverFlow insights. Furthermore, to get indebt technology specific knowledge some online course and video is used, for instance Udemy.

2 Usability

While designing an application, only using your imagination and trying your best is not good enough, since users have different interpretations of the user interface and overall functionality and performance, the application will be much better and well-designed if you consider how the users would use it to accomplish their task (Nielsen, 1993).

This section briefly overviews the concept of usability of mobile applications.

2.1 Overview

Usability could be considered as a subset of the whole system acceptability, which is the question of whether the system is good enough to fulfill most of the expectations and requirement of the users (Nielsen, 1993).

There are many models and techniques available that are used to evaluate the usability of mobile applications or any software system in general. Some of them are briefly introduced in the following:

2.1.1 Nielsen's model

Nielsen's model of usability suggests five attributes (Nielsen, 1993):

- *Efficiency*: The number of resources used in connection to the precision and completeness with which users accomplish a task.
- *Satisfaction*: Feeling of comfort and positive attitudes towards the use of the products.

- *Learnability*: The process of user adaptation with the mobile application in relation to functionality usage.
- *Memorability*: The mobile application should be easy to return to after some time of not having used it and without having to go through manuals again.
- *Errors*: The mobile application should not produce many errors, and the user must be able to not to lose or at least minimize the loss of any information if any error happen.

2.1.2 PACMAD

The PACMAD (People at the Centre of Mobile Application Development) usability model intends to overcome some of the imperfections of the existing model in context of applying it in mobile applications. However, this model is based on existing theories but it is cut to fit the applications on mobile devices. (see Figure 1.)

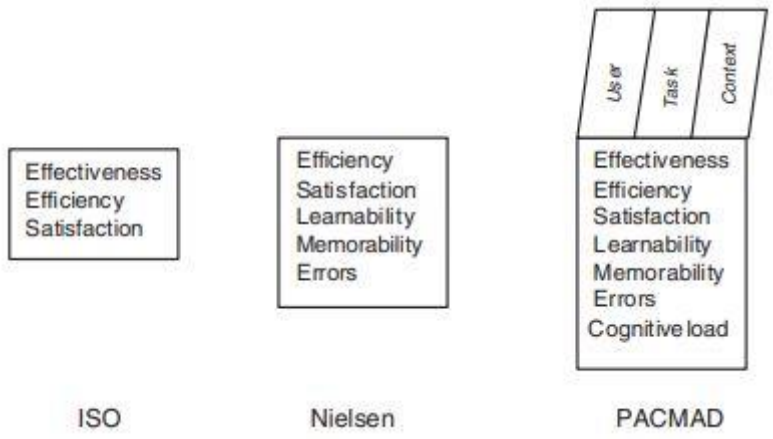


Figure 1. Comparison of PACMAD alongside Nielsen's and ISO's usability models

The PACMAD usability model introduces cognitive load which is a very critical aspect in mobile applications, in combination of IOS's and Nielsen's models.

User, Task and Context of use are the three factors which are identified by PACMAD usability for mobile application. These are the three important factors which should be considered when designing a usable mobile application according to PACMAD. In addition to these, the model also highlights seven attributes (*Effectiveness, Efficiency, Satisfaction, Learnability, Memorability, Errors and Cognitive load*) to assist in defining metrics that can be used to accommodate the usability of an application (Harrison et al., 2013).

2.1.3 Condos, et al

According to Condos, el al model the mobile applications can be evaluated based on a set of usability dimensions (Zahra et al., 2017). These measures consist of: information architecture, contents, navigation, error prevention, input rate, presentation and menu visualization.

2.1.4 Goal Question Metric (GQM)

The Goal Question Metric (GQM) method assumes that the organization should describe the goals for itself and its projects first (Basili et al., 1994). Then the goals must be traced to the data that were intended to define those goals, and lastly provide a framework to interpret the data with concentration of the proposed goals. Hence having a clear informational need for the organization is crucial, so that these needs for information could be quantified.

NASA Goddard Space Flight Center originally defined this approach to evaluate defects in a set of projects (Basili et al., 1994). Despite the fact that this approach was primarily using a certain project in a specific environment to define and evaluate goals for it, but its use has evolved to cover a greater settings or context.

The Goal Question Metric approach produces a measurement system specification to target a specific group of problems and a set of rules for the interpretation of the measurement data. The outcome measurement model consists of three levels:

- Conceptual level (Goal): depending on viewpoint and considering a specific environment, a goal is defined for numerous reasons with respect to models of quality, objects of measurement could be, Products, Processes or Resources.
- Operational level (Question): questions make sure to characterize the object of measurement (product, process and resource) considering the selected quality issue.
- Quantitative level (Metric): to provide quantitative answer to the proposed questions, a set of data is associated with each question, these data could be objective or subjective.

The hierarchical structure of the Goal Question Metric is visible in Figure 2.

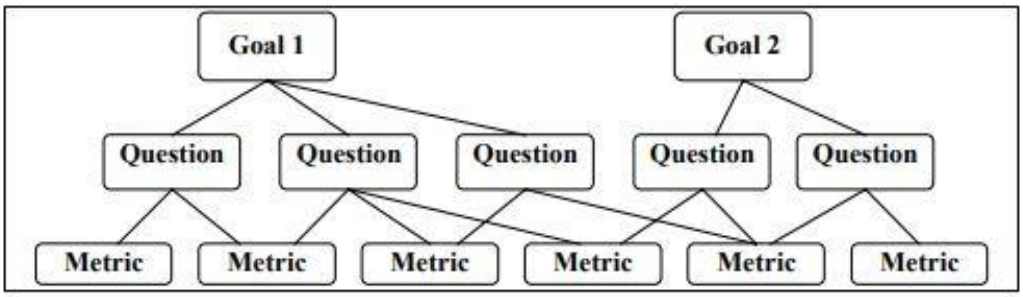


Figure 2. GQM model structure

2.1.5 mGQM

The mGQM is generally designed to support the evaluation of mobile application usability, but the model may not be applied directly to some mobile applications depending on the nature and features of specific applications (Zahra et al., 2017). The ISO 9241-11 usability measurement (Effectiveness, Efficiency and Satisfaction) was used as base in developing the mGQM model.

2.1.6 Coursaris and Kim

The main objective of proposed usability measurement model which was introduced by Coursaris and Kim (2006) after examining 100 empirical literature is to assist the

researchers to inspect and identify which usability dimensions should be considered to quantify the usability of mobile applications (Zahra et al., 2017).

Coursaris & kim (2006) proposes a framework for identifying and addressing numerous contextual mobile usability dimensions. Moreover, since it mainly focuses on usability, and offers distinct recommendations on the implementation of any interface with reasonable impact. (See Figure 3.)

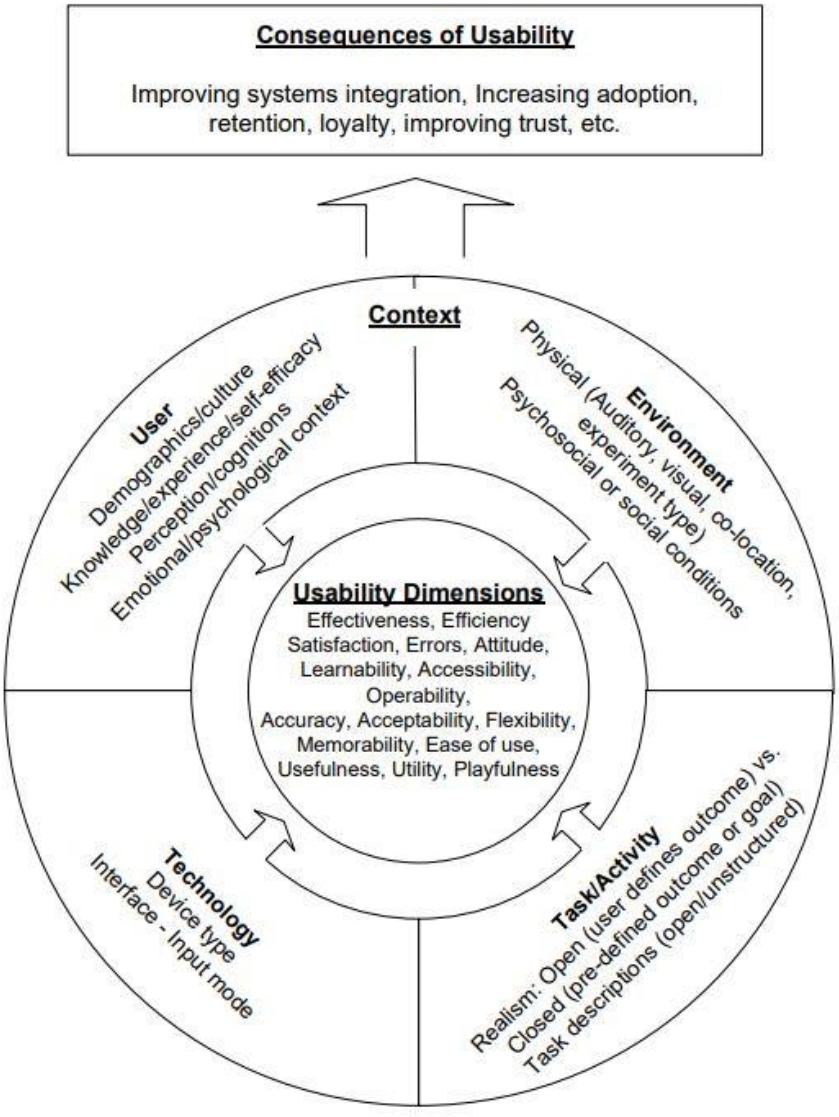


Figure 3. a suggested usability framework

2.1.7 Tan, Ronkko and Gencel

Tan et al. (2013) marks nine attributes and twenty-seven associated sub-attributes measurement guidelines for an adequate evaluation of mobile application usability (See Figure 4.)

SUB-ATTRIBUTES	ATTRIBUTES (with # of Questions identified)								
	Efficiency (77)	Effectiveness (48)	Satisfaction (98)	Productivity (5)	Learnability (52)	Safety (16)	Accessibility (22)	Generalizability (32)	Understandability (18)
Time behavior	•			•					
Resource utilization	•			•					
Users' assessment		•							
Experts 'assessment		•							
Operability	•		•				•		
Minimal action	•		•		•		•		
Feedback	•	•						•	
Minimal memory load	•		•		•		•	•	
Flexibility		•	•				•	•	
Quality of outcome		•							
Navigability	•	•			•		•	•	
Preference			•						
Users' attitudes/perceptions			•						
Memorability		•			•				
Likeability			•						
Fault tolerance						•			
Security						•			
Privacy						•			
Accuracy		•				•			
User Guidance			•		•		•	•	
Consistency		•			•		•	•	
Completeness		•							
Attractiveness			•					•	
Self-descriptiveness					•		•	•	
Simplicity					•		•	•	
Controllability							•	•	
Readability							•	•	

Figure 4. Proposed taxonomy for usability and UX attributes

Even though this model is designed for a specific purpose, the introduced metrics and relationship between selected usability measures are not comprehensive enough, as a result this may prevent to usability practitioner to adapt and apply the model quickly in mobile applications. In particular mobile companies can benefit the most out of this model to study and conclude the collected data on the user experience and usability of their products (Zahra et al., 2017).

2.2 Jaxber

Jaxber is invented by a Finnish company called Nestronite Oy, it is a cloud-based mobile application that enables the user to join a set of predefined campaigns which consist of multiple challenges in the form of video, audio, pictures, texts and qualitative and quantitative surveys (Montero Terán, 2020).

On the technical perspective Jaxber consist of three main components (Kontio, 2015). The actual mobile application which is available for public and easily installable to most of the devices. The API a key feature of Jaxber since it provides the connectivity between the app, web administrative portal and background services. And the web based administrative portal: which is an interface used by Jaxber team itself as well as the paying customer of Jaxber. This portal enables the admins to manage (create and edit) the campaigns or the user's privilege for instance. Moreover, the paying customer can use the portal to manage their own campaigns and the collected feedback.

The Jaxber mobile application is compatible with most of the available devices as it has been developed using PhoneGap technology. REST API and the administrative portal use their own technology stack (Kontio, 2015).

Jaxber scopes a variety of use-cases like service providers, the world of academia, and public institutions (Montero Terán, 2020). The application can also be used as a collective learning diary the purpose of which could be to connect students and teachers to interact about specific topic. The teachers could use the Jaxber web administrative portal to create campaigns so the students could share their comments in either one of textual, audio, picture, or video format.

To support the above-mentioned point, here an illustration of Jaxber use is going to be provided based on the author's personal experience. In the mandatory course offered by JAMK university of applied science "*YTSP0400 User-centred Design*" the teacher had created a campaign through which all of the students were submitting feedbacks and assignments. (See Figure 5.)

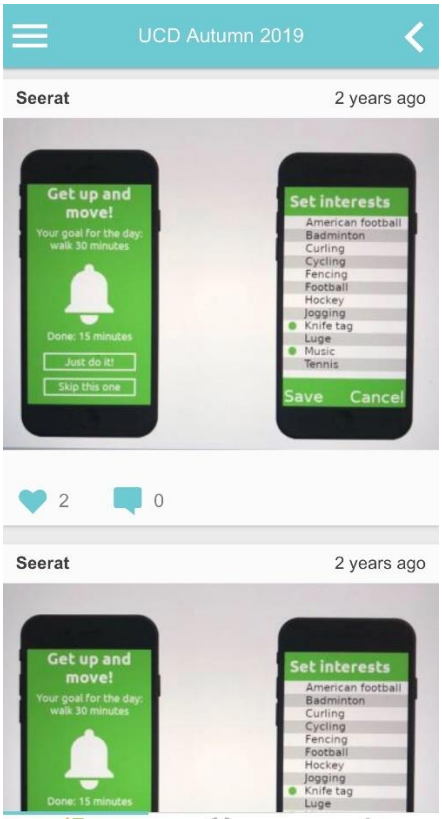


Figure 5. an example of Jaxber use from student's perspective

It's worth to mention that the Jaxber mobile app can be downloaded free of charge in both Android and iOS platforms, in order to provide equal access to students (Montero Terán, 2020).

Just like me Jaxber has been used by hundreds of students in many countries such as France, Germany, Spain, Hungary, Slovenia, Poland and Finland over the past years (Montero Terán, 2020). The students are not only encouraged to submit assignments or learning diaries but also to put the application general functionality in test like video capturing, user interface and design and give feedback openly.

2.3 Jaxber's needs for improvement

As every application encounter plenty of challenges Jaxber is not an exception too.

Montero Terán (2020), has conducted a valuable research by evaluating the Jaxber user-experience. She has used the survey method for qualitative and quantitative data collection, the survey is done using the bipolar questionnaire (close-ended question), (See Figure 6.)

Quantitative Part	Qualitative Part
Please rate the enjoyability of Jaxber app. Where (1) means not-enjoyable (2) mostly not-enjoyable, (3) almost enjoyable, (4) mostly enjoyable, and (5) enjoyable	Explain the reasons behind your rating of the enjoyability of Jaxber app.

Figure 6. Example of question used in Jaxber survey

The respondents were asked to rate a specific UX property from 1 to 5 which indicates their level of satisfaction.

Moreover, Montero Terán (2020) has also used a set of secondary data from the years 2017 and 2018 provided by the company which owns the Jaxber app.

Montero Terán (2020) qualitative and quantitative analysis of Jaxber user-experience allows us to infer about the challenges that the current version of the Jaxber has.

Quantitative results indicates that collaboratives, Efficiency, User-Friendliness, and Enjoyability has not changed over time. While usefulness, productiveness, and novelty seem to be changed. Usefulness and Novelty were perceived worse in 2018 compering to 2017, on the other hand productiveness received higher score. (See Figure 7.)

	ANOVA		Kruskal-Wallis Test	
	Significance	Decision	Significance	Decision
Usefulness	.003	Reject H0	0.003	Reject H0
Collaborativeness	.086	Retain H0	0.082	Retain H0
Productiveness	.000	Reject H0	0.001	Reject H0
Novelty	.000	Reject H0	0.001	Reject H0
Efficiency	.809	Retain H0	0.739	Retain H0
User-Friendliness	.072	Retain H0	0.058	Retain H0
Enjoyability	.749	Retain H0	0.482	Retain H0

Figure 7. Summary results

Qualitative results: sentiment analysis overall indicates that a majority of participants 48.10% has given positive feedback about Jaxber, whereas only 35.82% opinions were negative. Figure 8. demonstrates the summary of responses from sentiment analysis.

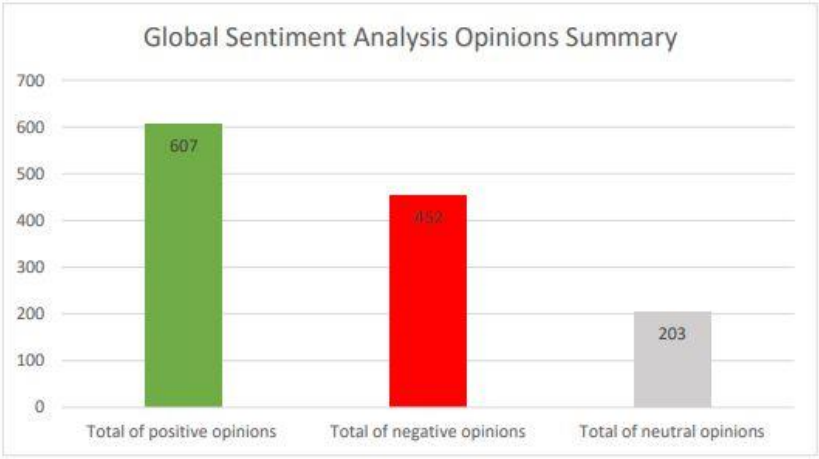


Figure 8. Summary of responses

Although the result analysis of Montero Terán (2020) research indicates a positive vibe regarding Jaxber overall user-experience, there are still directions for improvement as 35.82% were not satisfied with the quality of Jaxber application in many ways, for example user-friendliness (Montero Terán, 2020).

2.4 New improvement and suggestions

As the previous research indicates some imperfections are still visible in Jaxber app, for example user-friendliness (Montero Terán, 2020). The company which owns the Jaxber has already taken the initiatives to design a new version considering their users feedback over years. I have used the provided design by permission of the company as a base to develop the next version with Flutter technology. (See Figure 9.)

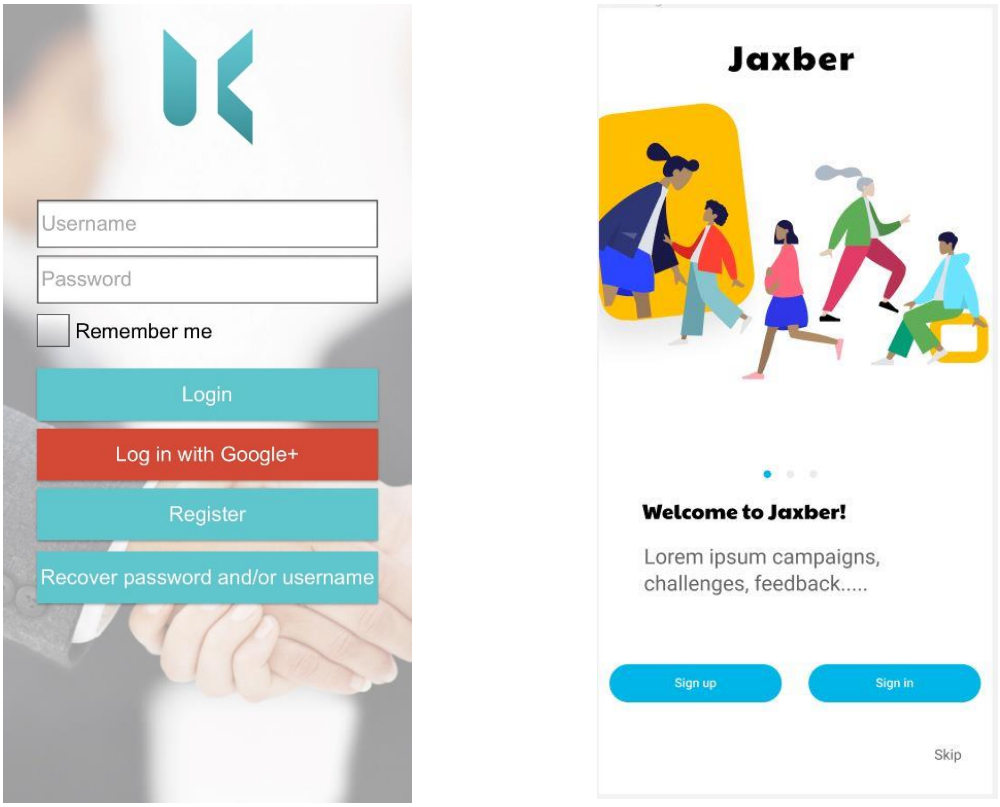


Figure 9. Visual comparison, in the left old design, in the right new version of Jaxber.

3 Mobile Application development techniques

3.1 Mobile native application development

Huy & VanThanh. (2012) states, "*A mobile native application or native app is an application specifically developed to execute on a specific device platform and machine firmware, and cannot be used for other device platform without modifications.*".

Native applications are often called embedded applications due to the fact that these types of application have profound integration with the mobile operating system. They are developed using platform specific SDKs and frameworks and are knitted with that specific platform, for example, applications that are build using Objective C or Swift with the help of iOS SDK are qualified only for Apple devices likewise applications developed by Google's Android are specific to Android enabled devices (Fentaw, 2020).

Native or platform-specific mobile applications accommodate best performance and adopt the device capabilities in best possible manner, but since these applications are not capable of being utilized for other platform, the development and maintenance comes with higher costs.

3.2 Cross-platform mobile application development

In today's technological era where there are thousand to millions of devices, each might use a distinctive set of technology; produced by multiple manufacturers, and the software producer / vender has absolutely no control over the end user's choice of device. In practical the software manufacturer would not want to limit their app to be applicable with only one single device or platform, this would mean the software producer has to satisfy this requirement and make the software product available for as much users as possible and the app should behave similarly across all the platforms. In theory this sounds very straightforward and easy but in reality, it is always a tiresome and complex job to build an application which can be used in most of these platforms seamlessly. A wide range of factors might affect the complexity of

the app such as rapidly evolving standards and limitations inflicted by mobile devices (screen size, input method, display capabilities, etc.) (Hartmann et al., 2011)

To overcome this challenge one way would be to build a native app for each device, native applications are those which are specific to a respective platform, for example iOS or Android (Kontio, 2015). However native applications provide many advantages such as: best performance, full access to the hardware and giving better feelings to the users in general. but this approach could also be massively expensive since multiple human forces such software developers who must have a good grasp of that particular technology must be hired, different set of tools would be needed to accomplish the goals. Multiple versions of the app must be built and released for each particular platform, having multiple versions would mean having multiple codebases, and if any new update is to be implemented in the app, all of the codebases must be updated, this in turn could break the software release consistency, as the app might perform well in one platform but not all.

This approach would be perfect for larger companies who might have a larger capacity, a good amount of budget for instance, and time interval is not very tide.

An alternative way, which is getting very fashionable nowadays is to use one single technology and after some tweaks and magic done, the software can be used in almost all of the mobile devices and platforms. Kontio (2015) defines this solution as single codebase approach. In single codebase method the developer writes the code with one principle and then this source code can be compiled to be installed on multiple platforms.

As technologies evolves there are many tools and frameworks available to pave the path and tackle the cross-platform app development challenges without compromising the quality, user experience and performance (Hartmann et al., 2011) some are listed below:

- Cross-compilation: this method splits the build environment and the target environment. The framework implements a platform-independent API by using a programming such as JavaScript or Java. The APIs are used to build the app, the code is then compiled to platform specific applications. As the produced app with this method acts like a native app hence this gives the app

to have the freedom to unlock the device specific potentials like integrated camera and sensors.

- Another method uses a virtual machine, these frameworks not only provide the APIs but also provides a mechanism to execute the code on the fly and plays an intermediate between the application and the mobile operating system. In this method the performance level is reduced due the runtime interpretation.
- Mobile web application: yet another progressively famous technique is to build the application as a mobile web application which will be viewed on device normal browser. In this approach standard web technologies such as HTML, CSS and JavaScript are used. This possibility is feasible nowadays since the HTML 5 and CSS 3 are more advanced.

As cross-platform application development seems to become a feasible solution for building application for multiple platforms, many companies and individuals induces tools and frameworks to facilitated and boost up the development process, though there are great number of tools for this purpose, introducing all these technologies is out of context of this research but some of the related tools are in-listed in next.

3.3 Cross-Platform tools

3.3.1 PhoneGap

PhoneGap is an open-source framework that allows to create mobile applications with the help of standard web technologies such as HTML, CSS and JavaScript. (Kontio, 2015), however PhoneGap seemed to be a practical solution for the cross-platform mobile app development but an article on the "*excellent webworld*" website (Panchal, 2020) provides an evidence that this technology is no more in use, and it is time to look for alternatives.

PhoneGap has been used as development platform for the previous version of the application.

3.3.2 React-Native

React Native is an open-source framework that uses JavaScript technology like React JavaScript Library to build natively compiled applications for mobile. ReactJS is primitively developed by Facebook to facilitate the web applications with user interfaces (Fentaw, 2020), ReactJS use a mechanism called JavaScript XML (JSX) which is basically the concept of injecting XML within JavaScript.

Because React Native uses JavaScript and access the platform-specific UI element this gives the native feels to the users, contrastingly in other techniques like hybrid mobile applications, which uses WebView.

3.3.3 Flutter

The aim of this section is to present the basics of Flutter technology and its development process.

3.3.4 Introduction to Flutter

Likewise above mention tools Flutter is also an open-source technology that enables the developers to easily build application for mobile devices (in the future for desktops and web) applications created by Flutter can easily be deployed on almost every hardware or mobile devices. Flutter is a product of Google and the initial stable version was released in 2016 for public use. Applications developed by Flutter can be installed on most of the currently available operating systems in the market, such as Android and iOS. Figure 10. demonstrate the process of how Flutter works.

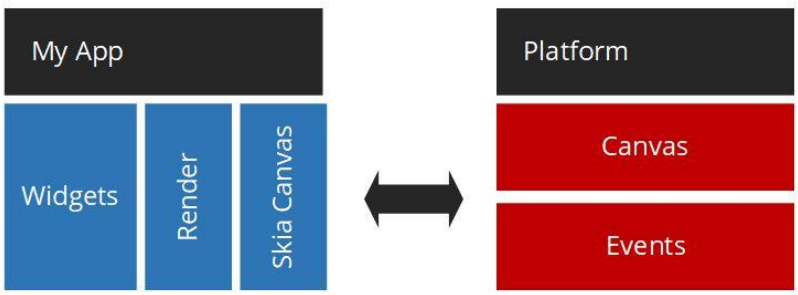


Figure 10. Flutter's compile process

However, there are many alternatives to the Flutter nowadays, Flutter uses its own rendering engine rather than its competitors that rely on utilizing web views or device's OEM. This feature makes the Flutter to stand out from other available technologies. If we look into the Flutter's high-level structure (Adam, 2018), it consists of:

- Platform layer, Flutter provides a platform specific shell that hosts the Dart VM and gives access to native platform APIs.
- next is Engine layer which is built with C/C++ and provides the Dart Runtime, it runs inside the respective platform Shell.
- lastly; The Flutter framework, with which the developer is mostly interacting.

(See Figure 11.)

Figure 11. Flutter nutshell

3.3.5 Dart

Flutter uses Dart programing language. Dart is a programming language invented by Google. Dart is also extensively used in-house by Google [8]. Since Dart was primarily developed to take over JavaScript. Hence, it utilizes most of the essential aspects of (ES7) or JavaScript's next standard, noticeable examples would be "async" and "await".

Dart is using a Java-like syntax, this makes it easier for the developers who comes from other programming language background and are not familiar with JavaScript.

3.3.6 Flutter widgets

Flutter is equipped with entirely customizable widgets that will assist the developer to create native like user interfaces. Flutter encompasses a prosperous set of widgets including the amazing Material Design library (for Android) and Cupertino (for iOS), smooth natural scrolling (Freitas, 2019). A Flutter widget could be as simple as a "*Text*" widget or a "*Center*" widget to align the other widget in the center of the screen for instance. (See Figure 12 & Figure 13.).

main.dart

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        backgroundColor: Colors.white,
        body: Center(
          child: Text(
            'Hello, world!',
            textDirection: TextDirection.ltr,
          ), // Text
        )), // Center, Scaffold
    ), // MaterialApp
  );
}
```

Figure 12. Text widget example

Figure 13. Text widget example output

A widget is a set of predefined code that can be re-utilized to describe how an UI is build (Fentaw, 2020). In every Flutter application widget defines the principal building blocks of the app. All the widgets are defined by their state and are stacked together in a tree like architecture to build up the application's user interface, to Flutter this structure is called Widget tree. Flutter re-renders the user interface in case a change happens in the widget's state. This re-rendering takes effect very masterfully due to the fact that it spots only those in which the changes happened and then update them.

Flutter's build-in widgets can effortlessly be personalized to amplify a rapid application development. The developer can easily combine multiple small widgets to build a single purpose one in other word the widgets can be nested inside another widget.

In simple term Flutter widgets are naturally extended re-usable Dart classes, similar to Components in ReactJS, in fact Flutter is all about widgets, A widget can be any collection of type of visual, structural, platform and interactive. To better understand and easy to use Flutter categorize the widgets some are mentioned bellow:

- Basics: Appbar, Colum, Container, Scaffold
- Text: Text, RichText and DefaultTextStyle
- Input: Autocomplete, Form, FormField and RawKeyboardListner
- Scrolling: CustomScrollView, DraggableScrollableSheet, GridView, ListView, NestedScrollView, Scrollbar, Scrollable and more
- Styling: Theme, Padding and MediaQuery
- Painting and effects: Opacity, BackDropFilter, ClipPath, CustomPaint and more
- Material: which is an implementation of the Material Design guideline

3.3.7 Widget classification

Flutter widget is are commonly divided into two types, dependent on the state usage of the widget, if the state of the widget changes over time it is called Stateful widget otherwise Stateless. Some of the characteristics of Stateless widget are (Fentaw, 2020):

- The appearance and properties of the widget never changes
- It extends the StatelesWidget class
- Can be created by overriding the *build*() method
- Properties are immutable
- Is very handy when UI element is not having dynamic changes
- It can never be altered, unless re-initialized
- No internal state is available

On the other hand, Stateful widget are dynamic change friendly and these widgets are extended from StatefulWidget base class, some of the characteristics are defined bellow (Fentaw, 2020):

- The appearance and properties can be changed over time
- Derived from StatefulWidget class
- Can be created by overriding the *createState()* method
- Properties are mutable
- Is very handy when UI is supposed to change, very flexible to dynamic chages
- Internal state is available

- Can be altered without re-initialization

3.3.8 Flutter Layout

In Flutter almost everything can be considered as a widget, like Buttons, Icons, images and more, including the layout and things that are invisible such as rows, columns, and grids which are used to organize the other widgets (Layout in Flutter, 2021).

Flutter categories the layout widgets to Single-child layout, Multi-child layout and Sliver widgets. Single-child layout widgets are those which houses a single widget like Align, AspectRation, center and etc. (See Figure 14.). In contrast Multi-child layout can accommodate multiple child widgets in them, GridView, Column, Row, Flow, IndexedStack, ListView and more for instance. and a Sliver is a fragment of scrollable area, slivers are used to achieve custom scrolling effects like elastic scrolling.

Figure 14. A simplest example of Single-child layout "*center*" that makes the "*Text*" widget fit in the center of the screen.

Flutter's build in features for layout and organizing widgets is remarkable, for example, when a layout is too large to fit a device and the layout goes off screen, Flutter manages the situation by providing a widget like "*Expanded*" to automatically fill the available space (See Figure 15.).

```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(
      child: Image.asset('images/pic1.jpg'),
    ), // Expanded
  ],
); // Row
```

Figure 15. Sizing widgets using "*Expanded*"

At the present time, application responsiveness is one of the key concerns, since the mobile applications are obliged to look and feel the same in a variety of devices. Flutter provides two basic approaches to make the app to self-adapt with different screen size and orientation. The first approach is to use the "*LayoutBuilder*" class and the second is to utilize the "*MediaQuery.of()*" method in the build functions alongside other handy widgets and classes such as AspectRation, CustomerSingleChildLayout, CustomMultiChildLayout, FittedBox and more.

3.3.9 Navigation

Flutter provides standalone widget "Navigator" to control the smooth navigation between the pages by stacking the child widget, the most recently visited pages will be on top of the others (See Figure 16 & Figure 17.), conversely, React Native framework relays on its community to implement navigation, "React Native Navigation (RNN)" could be a good example (Wu, 2018).

```
body: Center(
  child: RaisedButton(
    onPressed: () {
      // Navigate to the second screen using a named route.
      Navigator.pushNamed(context, '/second');
    },
    child: Text('Launch screen'),
  ), // RaisedButton
), // Center
```

Figure 16. Flutter navigation - open the second page.

```
body: Center(
  child: RaisedButton(
    onPressed: () {
      // Navigate back to the first screen by popping the current route
      // off the stack.
      Navigator.pop(context);
    },
    child: Text('Go back!'),
  ), // RaisedButton
), // Center
```

Figure 17. Flutter navigation - go back to first page.

3.3.10 Animations

Flutter comes with built-in animation widget to assist in making attractive user interfaces, there are two main types of animations (Drawing-based animations and Code-based animations) that could be included in the applications (Flutter Animation, 2021). Code-based animations are widget focused, and are rooted in standard layout and style primitives like rows, columns, colors, or text styles. Code-based animations are tended to enhance the widget appearance or transition. Drawing-based animations are like game characters that involve transformations and would be challenging to achieve purely with code. The Flutter animation system provides the "*animation library*" which depends only on core Dart libraries.

3.3.11 Form validation

In software development context, the process of validating if the entered values match the set of predefined rules is considered as Form validation, for instance checking for the correct email format or phone number. (Faust, 2020). Flutter provides build-in tools and functionality for validation. For instance, "*Form Field*" widget will validate the input through "*validator*" function (See Figure 18.). But in contrast React Native depends on libraries provided by community.

```
body: Center(
  child: TextFormField(
    // The validator receives the text that the user has entered.
    validator: (value) {
      if (value == null || value.isEmpty) {
        return 'Please enter some text';
      }
      return null;
    },
  ), // TextFormField
), // Center
```

Figure 18. Form validation snippet.

3.3.12 Internationalization

If the application expects users from different who might speak another language, the app needs to be internationalized. This would mean the app must be capable of showing localize values like text and layouts for each language (Flutter internationalization, 2021). Flutter provides "*Flutter_localizations*" package which supports 78 languages (including Finnish language) as of November 2020.

3.3.13 State management

In Flutter and any other frameworks, a group of accessible values used by widgets at any given time frame like while the widget is built or modified is called state (Fentaw, 2020). To update or modify the state object of in a StatefulWidget, Flutter has a specific function called *setState*(), this will enable the re-rendering of the IU.

Theoretically, Flutter consists of Ephemeral state and App state (Fentaw, 2020., Differentiate between ephemeral state and app state. 2021). Ephemeral state also known as UI state or local state, is tied to a specific single widget, in these kinds of states, the need for other state management such as Redux is not necessary, being a StatefulWidget is good enough, *setState()* can easily modify the Ephemeral states. App state is kind of a shared and available to all widgets throughout the application, for instance the state that you want to be accessible between session. User preferences, login details, notification systems, in e-commerce application the shopping cart are good examples of App state.

The usage of the state in Flutter app purely depends on the level of complexity and your own personal choice. (See Figure 19.)

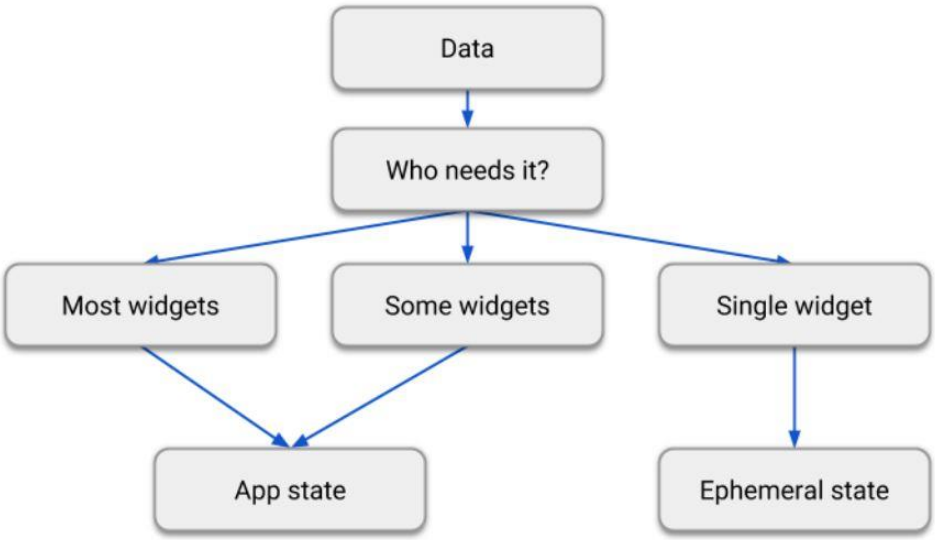


Figure 19. Flutter state and the different between Ephemeral and App states.

Finally, yet importantly, however you can use the Flutter's build in capabilities to manage state, but it is recommended to use other state management techniques such as ScopedModel or Redux for the complex applications.

3.4 Comparison of cross-platform technologies

In this section different cross-platform mobile application development is compared.

3.4.1 PhoneGap, React Native and Flutter

However, PhoneGap was a good initiative before the invention of other techniques but still observing the statical data from google trends for instance (See Figure 20.) the usage of PhoneGap has been very low in comparison with others (in this case React Native and Flutter)

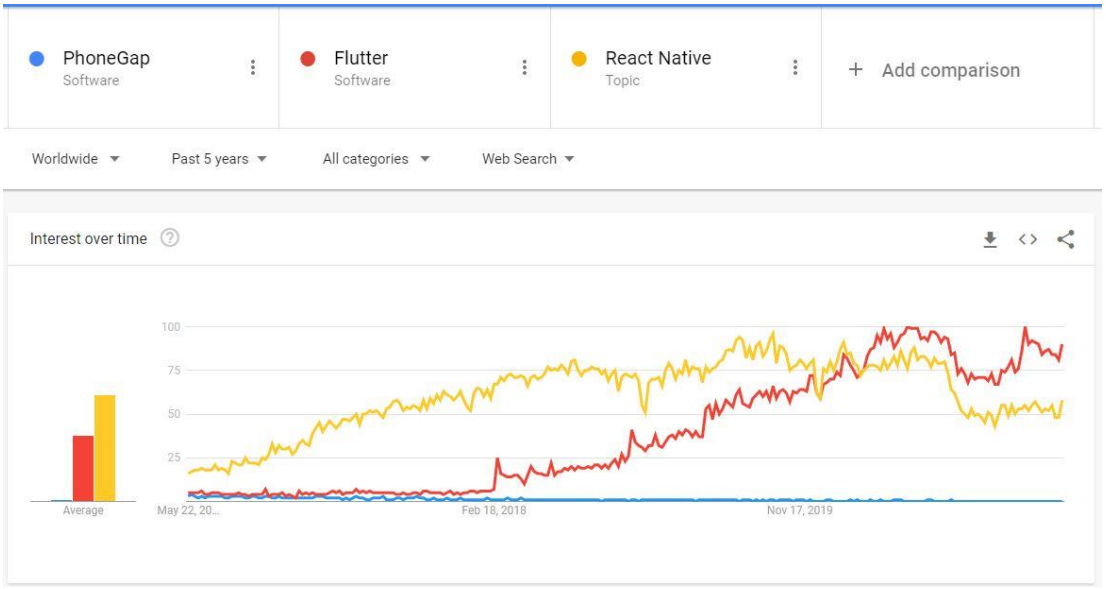


Figure 20. PhoneGap, React Native and Flutter comparison for the past 5 years

Moreover, since Adobe the company behind the PhoneGap and PhoneGap Build has shuttered its investment in the Apache Cordova. PhoneGap was closed in October 1, 2020 (Panchal, 2020).

Now that PhoneGap is out of market, the author intends to compare React Native and Flutter in short. In general, the statical data from stackoverflow indicates that Flutter seems to be used widely and the usage shows a notable increase comparative to React Native, (See Figure 21.)

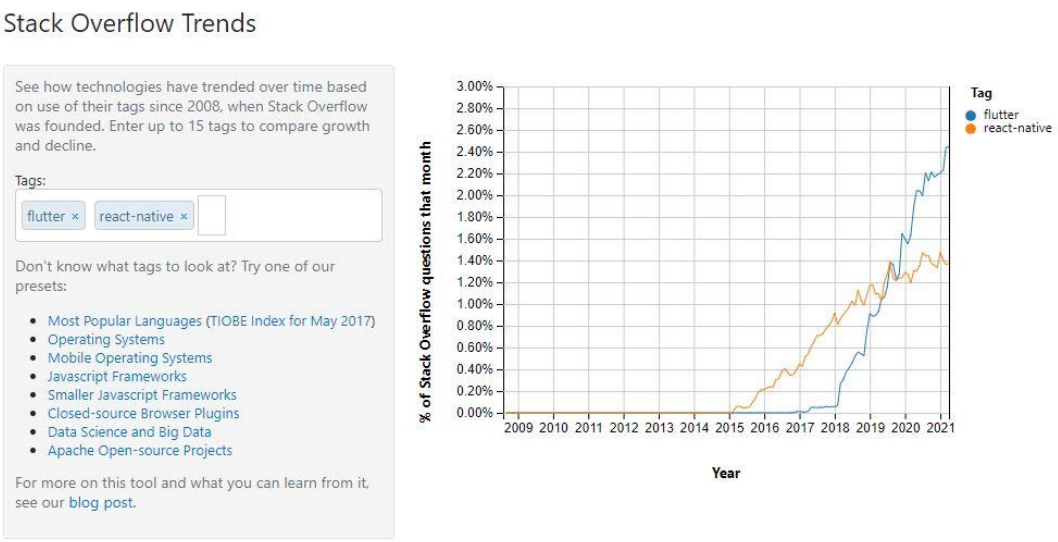


Figure 21. trends React Native and Flutter

Meanwhile the statistical data from google trends also indicates that Flutter has started to gain more popularity than React Native, (See Figure 22.)

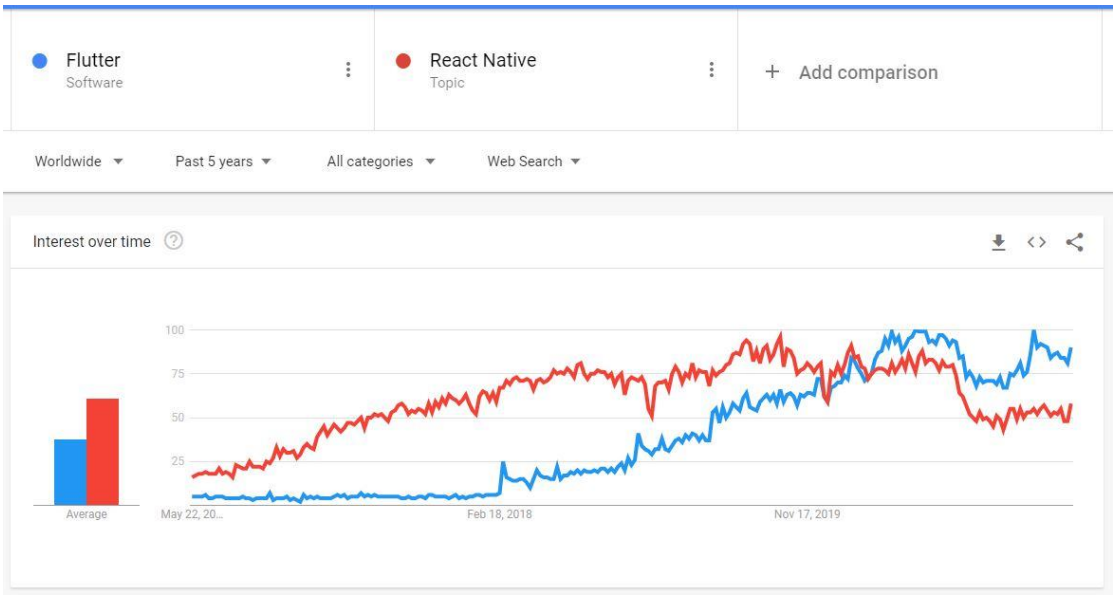


Figure 22. trends between React Native and Flutter based on google trends data.

Apart from the statistical data only, some of the pros and cons of the two technologies are listed and couple of cases are compared to find which framework provides better solution for given problem.

3.4.2 Audio and Video

Both React Native and Flutter core does not seem to provide core component for the audio and video that interact directly with the platform Audio and Video APIs (Fentaw, 2020). React Native community provides supporting libraries for the purpose, *native-sound* is considered one of the popular libraries, *react-native-audio-recorder-player* could be another good example. On the other hand, Flutter also relies on third party libraries, for example *flutter_sound* which is capable of playing a majority of audio files, Flutter core team also provides the *video_player* plugin for inline video display and *video_player_platform_interface* a plugin for *video_player* plugin (Fentaw, 2020) ,19]

3.4.3 File access

React Native does not seems to provide any core component to access the files, but despite that there many supporting third party libraries that helps to achieve the goal, *react-native-fs* for instance, provides the ability to gain filesystem access in iOS and Android, however this would also require platform specific configurations (Fentaw, 2020).

But Flutter core team seems to be taking care of the matter, *path_provider* plugin assists in interacting with generally used locations with iOS and Android (Fentaw, 2020)

Considering the above-mentioned points, both frameworks seem to not ship all the necessary APIs to enable the app to interact with platform native APIs, React Native gives the impression of being steadily dependent to the third-party libraries, in contrast, Flutter core team seems to taking care of providing certain plugins in order to triumph over the problem.

Thus, based on the evaluation of multiple techniques, Flutter seems to have the qualities of a wining choice for the Jaxber prove of concept or to be implemented in the upcoming version, unless the future development of technologies changes the direction.

3.4.4 Performance

Generally, performance refers to how the application affects users experience, but technically, performance encompasses the visual performance, RAM, CPU and other resource management, input-output speed and more (Tran, 2020).

As attested by Flutter team, Flutter intends and has acquired the optimal performance or native performance accuracy. Applications built with Flutter, with proper development optimization kept in practice, the performance will reach 60 frames per second (fps) or 120 fps performance on devices which are capable of 120Hz updates (Tran, 2020).

Hjort (2020) defines the performance level of React Native as "*Good*" and "*Close to native*" for Flutter (See Figure 23.). Since React Native uses the interpreted approach,

this means that React Native translated the JavaScript code to native language at runtime, which is a shortcoming for the performance, but in contrast, Flutter compiles a fully native application for specific platform, this would utilize nearly same performance as native application.

	React Native	Flutter
Release	2015	2017
Created by	Facebook	Google
Language	JavaScript	Dart
Supported platforms	Android, iOS	Android, iOS
Open source	Yes	Yes
Cross-platform approach	Interpreted	Cross-compiled
Performance	Good	Close to native
UI components	Native	Proprietary

Figure 23. React Native and Flutter comparison

4 Implementation

In this section, the details of actual implantation and development of Jaxber using Flutter is explained.

4.1.1 Development

Current version of Jaxber uses PhoneGap technology as development tool, section 4 that compares different techniques for cross-platform mobile development, and hence, PhoneGap is no more in use (Panchal, 2020) and is not an applicable choice any further, as a replacement, Flutter seems to be the best alternative to use in development of future generation of Jaxber. Therefore, it has been decided to use Flutter and Firebase to develop the next version.

It is to be noted that Jaxber is a commercial product and the software source code is considered as confidential information to the company, hence it cannot be shared in this research. This section represents the application development process excluding any internal technical details of it.

4.1.2 Development environment set up

This section provides details and steps to setup development environment, for the convenience a windows 10 based laptop was used as development platform, Flutter has no limitation regarding the OS level platform during the development process, however, to build the application for iOS devices a Mac OS is required to use the xcode capabilities and build the application so it could be deployed in apple store.

4.1.3 Editor

However, there are many editors that can be used for Flutter and Dart development, Flutter official documentation (Flutter installation instruction, 2021), recommends the Android Studio, IntelliJ, Visual Studio Code and Emacs, all of the mentioned editors are great but the comparison of the editors is out of scope of this research. as matter of personal choice, Android Studio version 4.1 is used as the editor.

The installation process of Android studio is quite straightforward, you can obtain the latest version from android studio official website (Android Studio Official website, 2021). As any windows application follow the installation wizard and you are done. For details, please refer to Android studio official documentation. After successful install the main window looks similar to Figure 24.

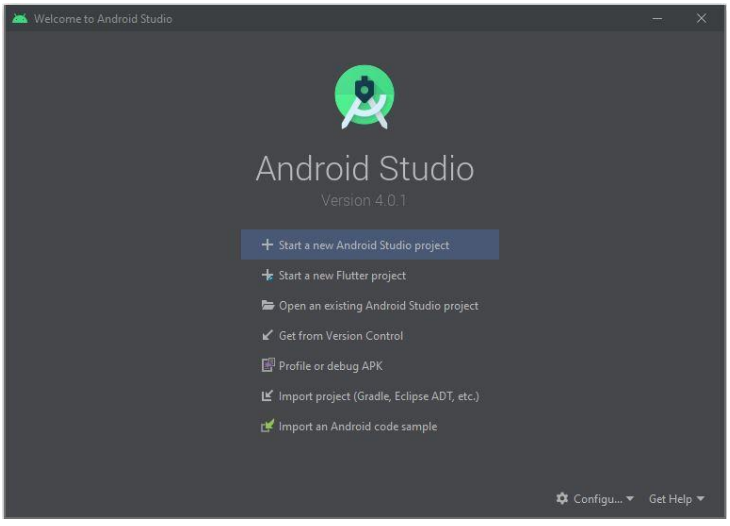


Figure 24. Android studio version 4.0.1 welcome screen

4.1.4 Terminal

Although windows 10 build-in command prompt and PowerShell are good enough to accomplish the required task for Flutter development and there is no need for a third-party terminal system. But due to personal interest, the cmder which is a portable console emulator for windows (Cmder official website, 2021) has been used.

4.1.5 Flutter installation

Before installation of Flutter SDK, the minimum system requirements are checked. The flutter documentation suggests the following (Flutter installation instruction, 2021):

- Operating system: Windows 7 SP1 or later
- Disk Space: 1.64 GB
- Tools: such as Windows PowerShell 5.0 and Git for Windows 2.x

Once the minimum requirements are fulfilled, download the Flutter SDK from flutter official website, extract the zip file in a desired directory (*in this case C:\src_dev*), alternatively, if you do not wish to install a fixed version, you can get the source code from Flutter repo on GitHub "*C:\src>git clone https://github.com/flutter/flutter.git -b stable*". The final step would be to edit the environment variables to be able to run Flutter commands from terminal.

To validate if Flutter requires any platform dependencies, open a terminal and run Flutter doctor, this will check the system and reports the status. (See Figure 25.)

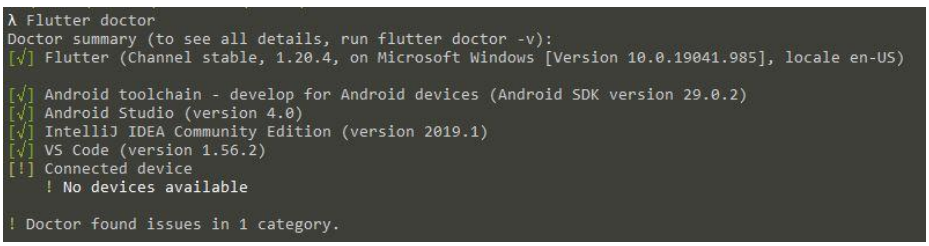


Figure 25. Flutter doctor result

In addition, Flutter and Dart plugins must be installed for the android studio, the install defers depending on OS. In windows machine navigate to "*File > Settings > Plugins*", Select Marketplace, then Flutter plugin and install

4.1.6 Creating Flutter application

After successful installation of all required tools, to create the Flutter application open the android studio and from the welcome screen (See Figure 24.). Choose the "*Start a new Flutter project*" option this will open the "*New Flutter Project*" window (See Figure 26.). Depending on the nature of the project select one of the provided templates (Flutter Application, Flutter Plugin, Flutter Package or Flutter Module)

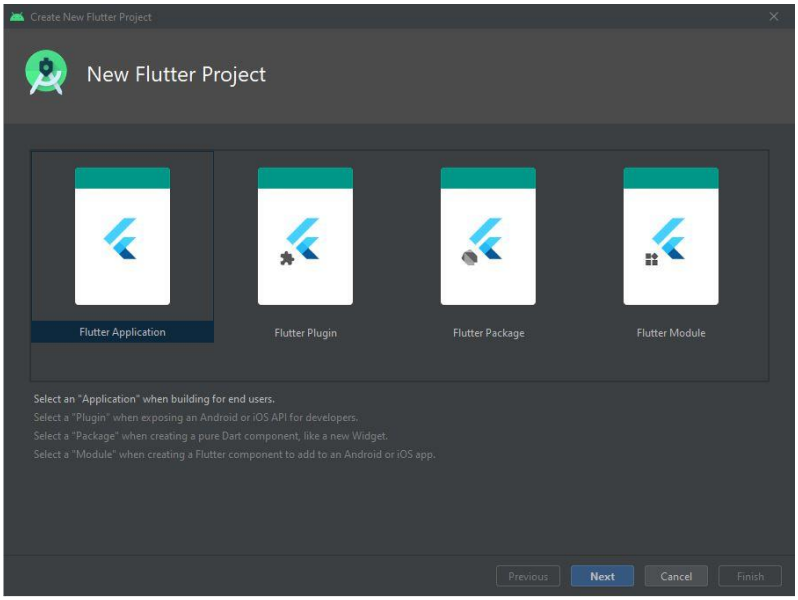


Figure 26. New Flutter Project window - android studio

For the purpose of this research, the "*Flutter Application*" will be selected, as the ultimate goal is to build an application for the end user. Once the project type is selected, pressing the next button will redirect to the project configuration window (See Figure 27.). Choose an appropriate project name "lower_case_with_underscores" (Dart package name convention, 2021).

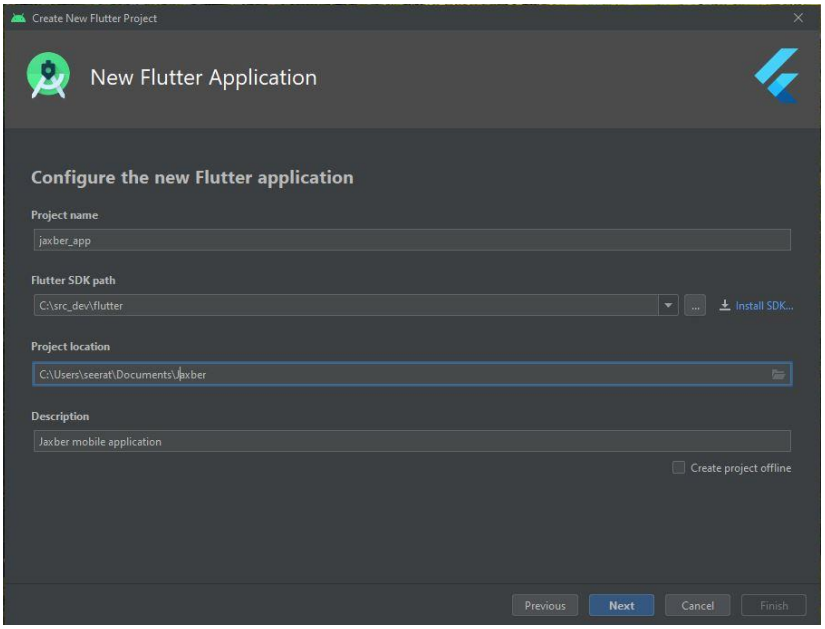


Figure 27. project configuration window - android studio

Next Android studio should automatically detect the Flutter SDK and set its location, assuming it has already been installed (See section 5.1.4), in case Android studio is complaining about Flutter SDK, there are possibilities of either install it or if the Flutter SDK is already installed but not picked up by Android Studio, the Flutter SDK path selector window (See Figure 28.) will assist to locate and choose the path to the installation directory.

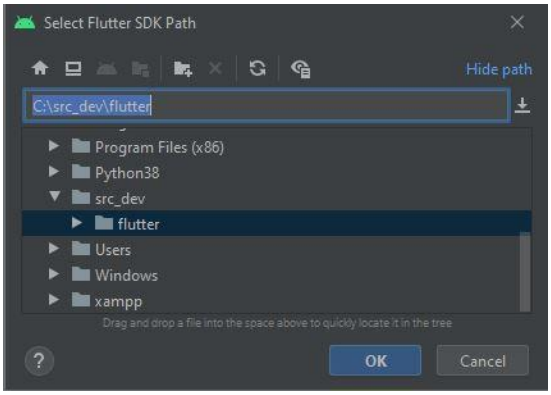


Figure 28. Select Flutter SDK path window - Android studio

After the Flutter SDK is successfully selected, define the project location and a proper description for the project, the final step would be to give a proper package name for the project (See Figure 29.), the name must follow the “com.domain.appname” pattern, this will uniquely identify the application on the device and app store such as Google Play Store (Application ID, 2021).

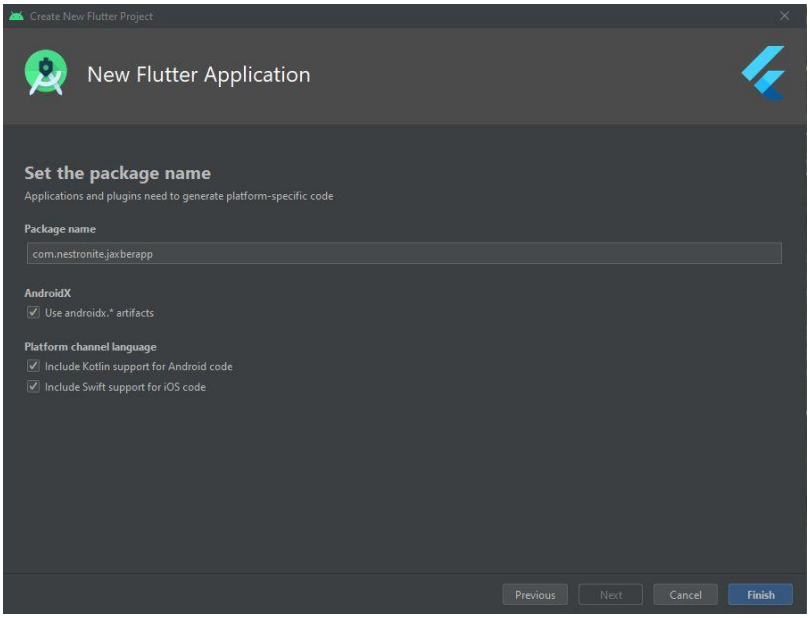


Figure 29. Package name - Android Studio.

4.1.7 Flutter project structure

Flutter creates a set of predefined folders, some of which are purely used by Flutter and the developer do not need to touch them, the only folder that matters the most and is constantly modified is the “*lib*” directory, however the developer is not limited to default directory, and will have the authority to add more directories and files, for instance a directory to store the assets (images and etc.), config folder, and for better management a sub-folder for custom widgets. (See Figure 30.).

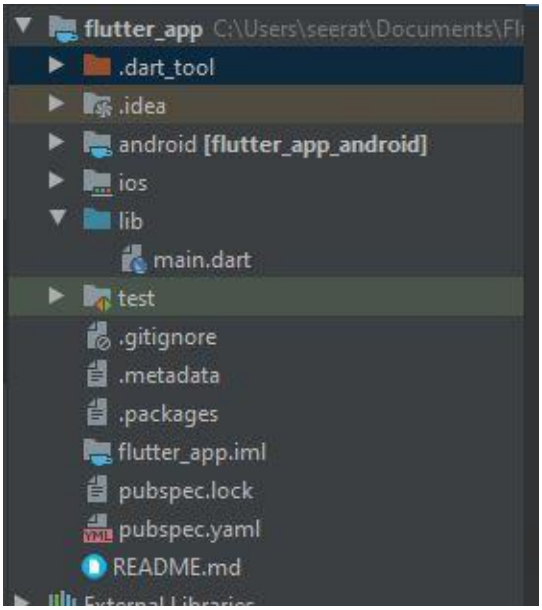


Figure 30. Basic Flutter directory structure.

4.1.8 Running Flutter app

In order to observe the created application result, there are options to either run it in a virtual environment (Android Virtual Device or iOS simulator) or a physical mobile device. Android Virtual Device (AVD) is a configuration containing a hardware profile, system image, storage area, skin and other properties, and it simulate the nuts and bolts of a physical device such as Android phone or tablet. Likewise, iOS simulator is in reach within Xcode in Mac computers only, and it provides a simulation for apple products such as iPhone or iPad.

Android Virtual Device can be easily created through AVD manager in Android Studio, open the AVD manager from "*tools > AVD manager*", then click on "*Create Virtual Device*" (See Figure 31.).

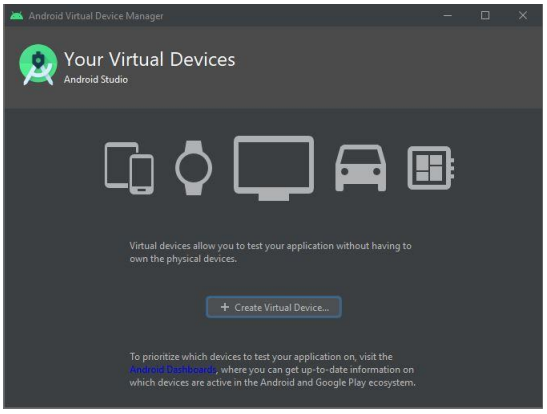


Figure 31. Android Virtual Device manger.

Next select a device definition from the available list, in this case "*Nexus 6*" has been selected (See Figure 32.).

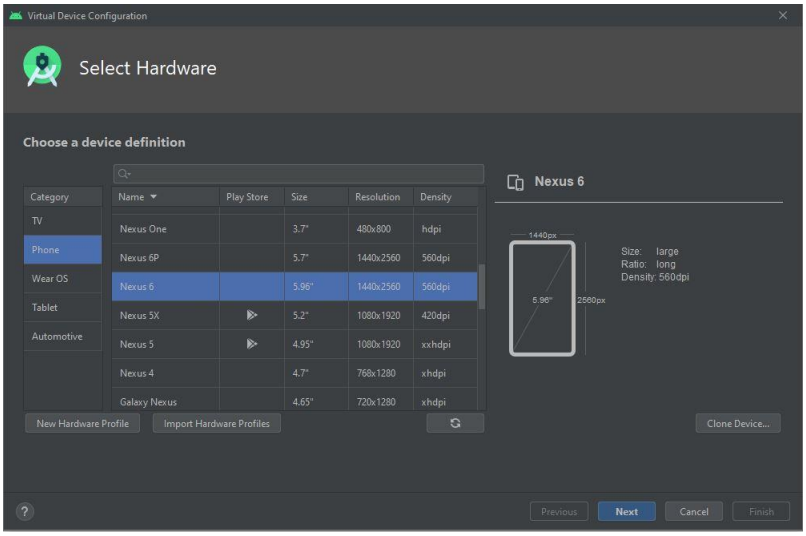


Figure 32. Device selection for AVD.

Once the device is selected, click next and select the system image that targets a specific Android version according to the test requirement (See Figure 33.).

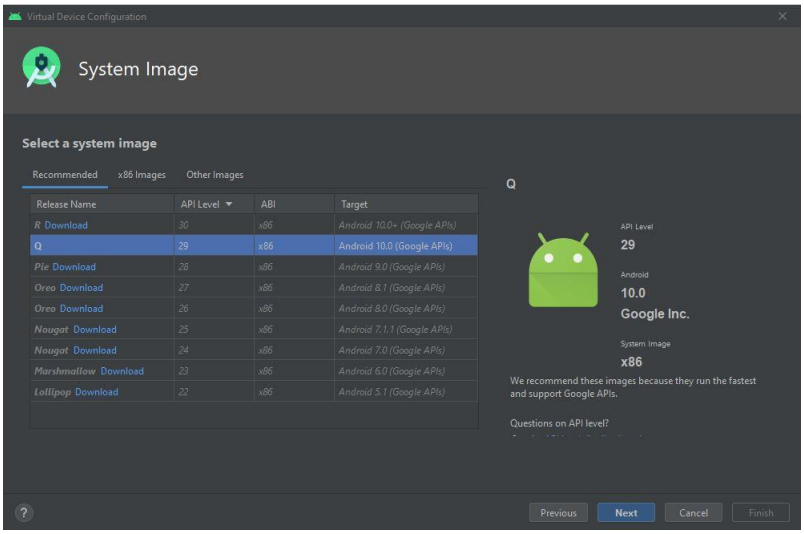


Figure 33. System image selector for AVD.

The final step is to choose the emulated performance, and it is recommended to select a hardware graphic than Software (See Figure 34.). The AVD should be installed, and can be activated from "*Tools > AVD Manager*" under the Actions click on the play button.

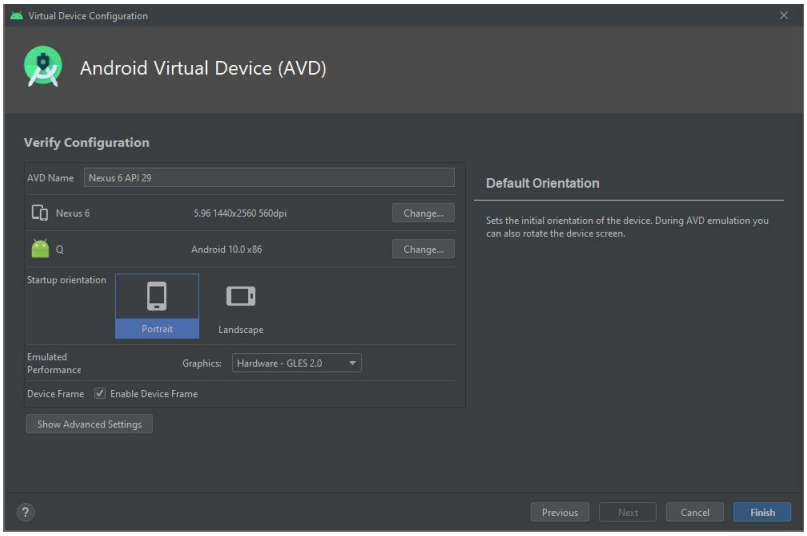


Figure 34. AVD configuration

iOS simulator will be pre-installed by xcode on mac computers, it can be opened either from finder or alternatively running the "*open -a Simulater*" from a terminal will activate the simulator.

Using physical phone is one of the other possibilities to test the app while developing, the steps are to simply connect the phone with the development machine, point to be noted that the "*Developer options*" and "*USB Debugging*" must be enabled in the Android device. Running the Flutter doctor in the terminal can confirm if the device is successfully connected. The connected device can be selected in Android Studio toolbar (Flutter device selector) and by clicking on Run (play button) the app will be automatically install and open on the device.

4.1.9 Firebase

The original Jaxber implements a REST API to communicate with database, in this research, the application is not using the original REST API. Instead, Firebase is used to store the data. In order to examine if Firebase could be a possible solution to replace the custom REST API in the future development.

Firebase can be considered as web application platform. Firebase is widely used by developers to build high-quality applications (Khawas & Shah, 2018). Unlike the traditional database system which uses SQL structure and sets of queries are used to apply CRUD operations like insert, update and delete. Firebase is NoSQL.

Firebase provides a set of services, for instance in this research "*Firebase Auth*" is used to manage the user's signup and login functionality, boots up the development process Firebase also provide a set of other services like Firebase Analytics, Firebase Cloud Messaging (FCM), Firebase Auth, Real-time Database, Firebase Storage, Firebase Test Lab for Android, Firebase Crash Reporting and Firebase Notifications.

4.1.10 Firebase Setup

Although Firebase is designed to help developers build apps faster, but getting started and configuring it could be a complex task to some extent, in this section the process of creating a project in Firebase and connecting it to the Flutter application is described.

To begin with Firebase, you need to create a project, any normal google account can be used to login to firebase portal, after successful login, from the Firebase Console (to get to this page click the link “Go to console” at the right-top corner of the Firebase home page), click “*Add project*”, you will be prompted to provide a proper project a name and an optional Project ID, click continue, optionally configure the Google Analytics (point to be noted that Google Analytics requires an account to be created). If Google Analytics is not select the finally step would be to click “create project” button (See Figure 35 & Figure 36.).

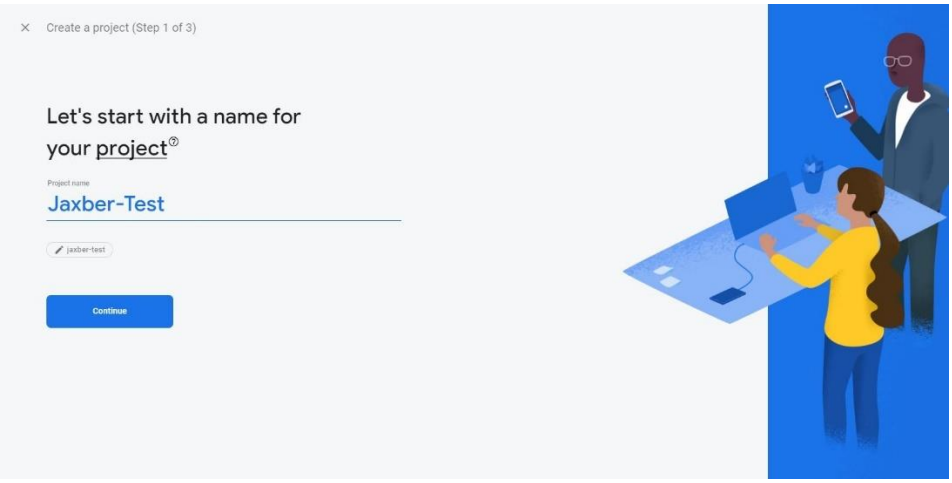


Figure 35. Firebase create project step one

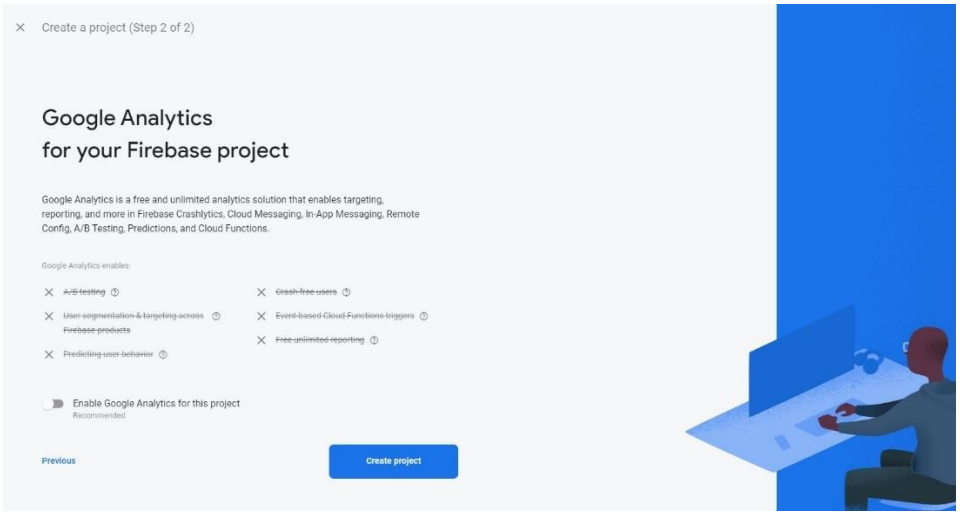


Figure 36. Firebase create project step two

After the project is added, next step is to register your app with Firebase and provide the connectivity with Flutter, in the Firebase project overview, click one of the icons (“*Android*” or “*iOS*”), in the open screen (as depicted in Figure 37.)

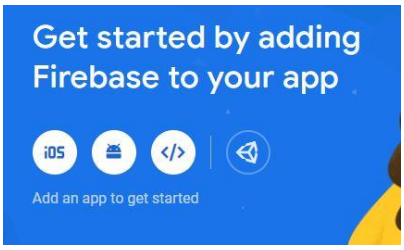


Figure 37. Adding Firebase to the app

While the page is opened (refer to Figure 38.). Add the bundle ID or Android package name, this in Flutter project is referred as “applicationId” and can be found inside “*android > app > build.gradle*” file. An optional nickname and debug signing certificate

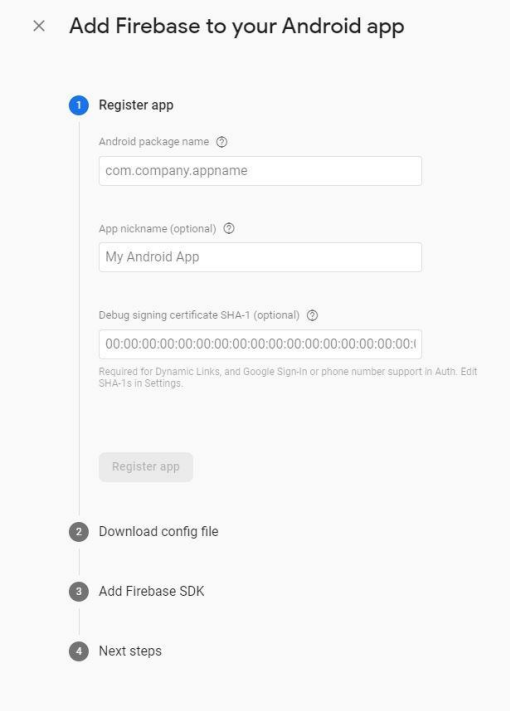


Figure 38. Adding Firebase to app

Next, the Firebase configuration file must be added into the project, download the GoogleService-Info.plist (See Figure 39.). This Firebase configuration file needs to be placed inside "*android* > *app*" directory in the project.

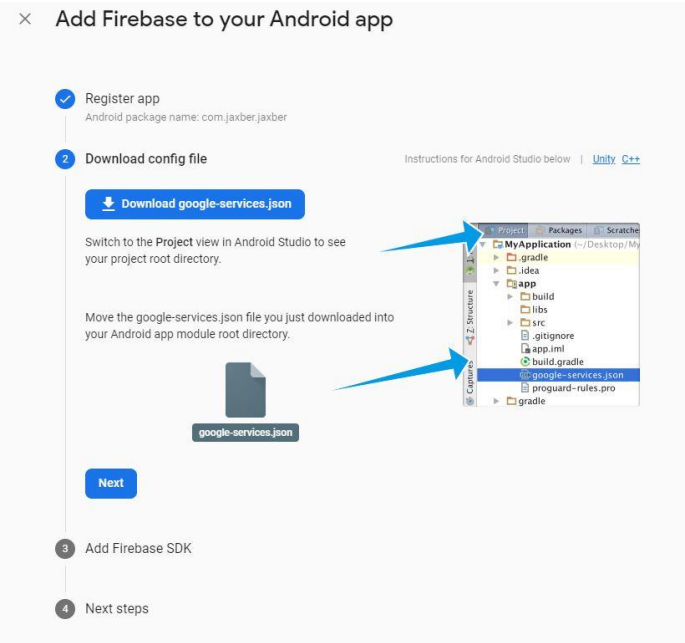


Figure 39. download the google service info

The next step is to enable Firebase services in the Android app, this can be done by adding rules to include the Google Services Gradle plugin to “android/build.gradle” (See Figure 40.). And add the “*apply plugin: 'com.google.gms.google-services'*” at the bottom of “android/app/build.gradle” file.

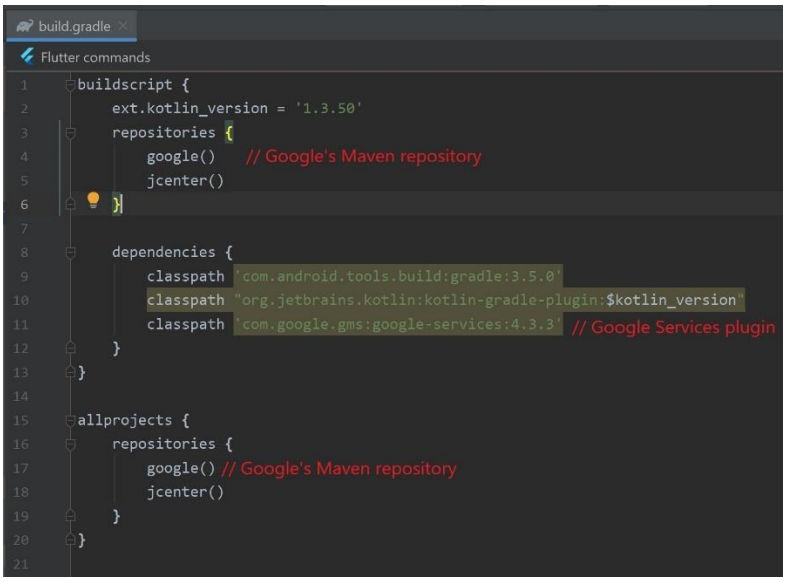


Figure 40. gradle configuration file

The Final step is to add Firebase plugins “*firebase_core*” for instance, in the pubspec.yaml and from a terminal run “flutter packages get”.

4.1.11 Firebase Authentication

One of the Firebase product or feature is the Firebase Authentication which provides backend services and easy-to-use SDKs for user authentication using passwords, phone numbers, federated identity like Google, Facebook and More (Firebase Authentication, 2021). In order to allow Flutter application to utilize this feature, the Sign-in method must be enabled from Firebase console, in the project overview page, click on the “Authentication” link, afterwards select the “Sing-in method” tab and enable any of the method Email/Password for instance, from the Sign-in providers list according to the project’s requirement (See Figure 41.). In the same

page the “Users” tab will contain a list of all registered users and their details such as Created and Signed In dates.

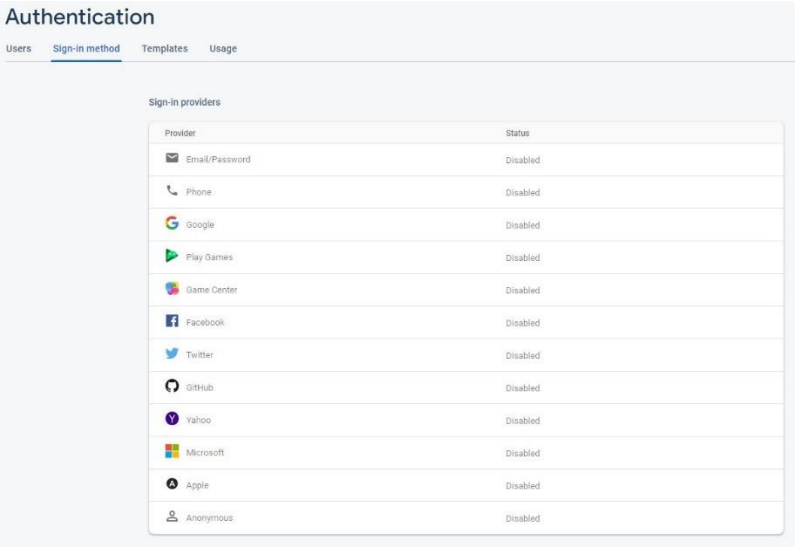


Figure 41. Sign-in providers list - Firebase Authentication

4.1.12 Firebase Authentication in Flutter

To utilize the Firebase authentication within Flutter app, first step is to add the dependency in the project, this could be done through “*pubspec.yaml*” file (See Figure 42.). Then download it by running the “*flutter pub get*” command from project root directory.

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: "^0.7.0" # Add this line
  firebase_auth: "^0.20.1" # Add this line
```

Figure 42. pubspec config file.

Once the dependency is installed, it must be imported into the Dart code (import 'package:firebase_auth/firebase_auth.dart';). To integrate secure authentication in your application, most often it is necessary to be aware of authentication state of the user if logged in or not (FlutterFire, 2021). Firebase Auth uses “stream” to allow the

app to subscribe in real-time state, all needs to be done is to call the "authStateChanges()" method on "FirebaseAuth" instance (See Figure 43.).

```
FirebaseAuth.instance
  .authStateChanges()
  .listen((User user) {
    if (user == null) {
      print('User is currently signed out!');
    } else {
      print('User is signed in!');
    }
  });
```

Figure 43. Firebase auth state change.

Firebase is equipped with many numbers of ways to sign users into the application, for example, password authentication, phone authentication and even anonymous users. Here the "email and password" method is examined, to sing-in with email and password a call to "signInWithEmailAndPassword()" method is sufficient. (See Figure 44.)

```
// Sign in with email & password
Future signInWithEmailAndPassword(String email, String password) async {
  try {
    AuthResult result = await _auth.signInWithEmailAndPassword(
        email: email, password: password);
    FirebaseUser user = result.user;
    return _userFromFirebaseUser(user);
  } catch (e) {
    print(e.toString());
    return null;
  }
}
```

Figure 44. an example code snippet for email and password sign-in - Flutter + Firebase

4.1.13 Cloud Firestore

Similar to other database Cloud Firestore (Firestore database) is a document database that stores data in tree-like structure, it can be thought of as a JavaScript object containing key-value pairs called fields (Thota et al, 2020). Firebase services can be started with a “free plan” but scaling up to “Pay as you go” is easy and manageable at any time. Firebase also provides a functionality to store and retrieve user-generate files such as images, audio and video without writing any server-side code.

Since Jaxber allows the users to upload a ton of images, audio and video files through the campaigns, this Firebase functionally would assist the process to perform as satisfactorily as possible.

5 Jaxber application

In contrast to the original Jaxber project, which consist of three components, REST API, Web administrative and mobile app itself, in this research the work was limited to mobile app only and Firebase is used for user authentication and data storage. In this section the proposed user interface is explain briefly.

5.1.1 Onboarding

Unlike the original Jaxber, in which the user was redirected to login page after the app was opened, but in this version the onboarding view as shown in Figure 45. is added to give the user an opportunity to decide what to do next, for example if the user is not yet registered, by pressing on the "*Sign up*" button can easily register. Or pressing the "*Sign in*" as expected the user will be directed to the login view. Moreover, a "*Skip*" button is also added, this will allow the user to go the campaigns view and join any of the public campaigns that does not require the user to be logged-in or in other word, anonymous users are allowed to give feedback.

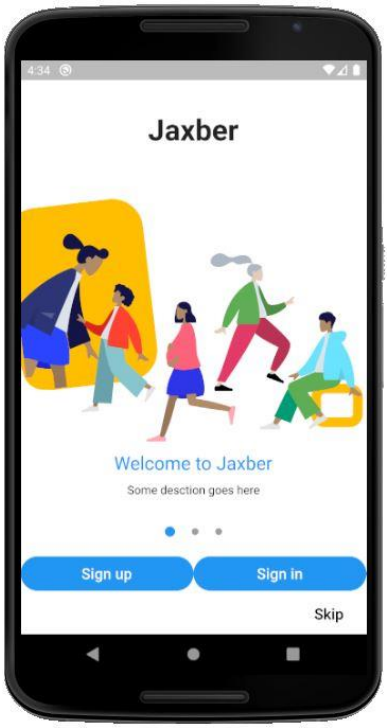


Figure 45. Jaxber onboarding view

This onboarding view also comes with a sliding functionality containing three separate pages each with different information, the slides changes automatically after five seconds.

5.1.2 Signup

If the user decided to not to give feedback anonymously and does not already have an account in Jaxber, they can reach this page by pressing the “sign-up” button from onboarding view, as depicted in Figure 46, by providing a valid email and password the user can easily register to the Jaxber app, and after the registration is done, the user will automatically be redirected to the campaigns page.

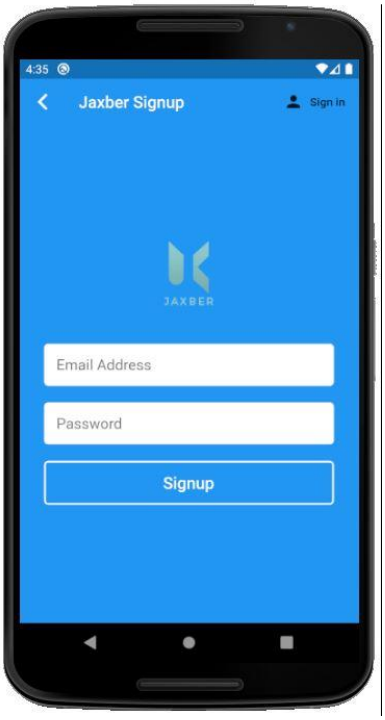


Figure 46. Jaxber sign-up view

5.1.3 Login

If the user is already registered, by pressing the “*Sign in*” button from Onboarding view, can access this view as shown in Figure 47, and if the provided credential is

valid and the authentication process is successful, the user will be redirected to the campaigns view.

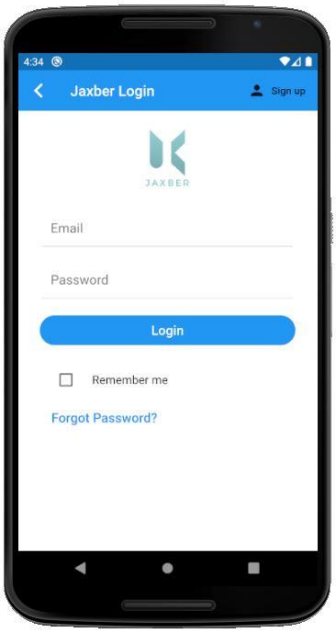


Figure 47. Jaxber Sign in view

5.1.4 Campaigns

This view can be considered as main view in Jaxber, the campaigns view contains a list all the available campaigns as shown in Figure 48, these campaigns could be created by admins using the Jaxber web administrative portal. The user then can interact with these campaigns to provide feedback using the possible methods like textual, audio or video.

Figure 48. Jaxber campaign's view

5.2 Testing

Not much time was spent in application testing, despite, some basic testing from a user point of view, most of the bugs were detected by developer at the time of coding, obviously some of the bugs might have been left undetected.

The app was executed in four Android Virtual Devices (Nexus 10, Nexus 6, Pixel 3a XL and Pixel C tablet) and one physical device (Samsung Galaxy J5) to test the user interface alignment and performance. See the Appendix 1 for the Figures.

According to Flutter, automated testing must be performed in all mobile application developed by Flutter to ensure the correctness of app performs before its published. Flutter categories these automated testing into three: unit test, widget test and integration test. A unit test examines a single function, method or class in order to verify the correctness of logic under a variety of conditions. A widget test is done to make sure that the widget's UI looks and interacts as expected. And an integration test is performed to attest that all widgets and services work together properly.

In this research no proper test automation mechanism was in place, a test automation could have been utilized to detect certain number of bugs during the development, the absence of proper testing was the pressing shortcomings for this research.

5.3 Deployment

The application is not officially published to any of the app stores (like google play or apple store), since the proposed solution is considered as a prototype and is not fully market ready, the app might contain some bugs left unintentionally, the source code and an installable build version is shared in GitHub repository. And the build version was used for testing purposes.

However, when the app is market ready and fully prepared to be release, there are few recommended checklists (Launch checklist and Localization checklist) to be thoroughly reviewed before publishing it to store such android (Launch plan checklist, 2021). Launch checklist consist of twenty-two items, the most important are: Understand the Developer Program Policies, Prepare you developer account, Plan for localization, Plan for simultaneous release and more. Also, Localization checklist includes items such as: Research target languages and locales, design your app for localization and many more.

Moreover, Flutter also recommends few steps before publishing the app like, adding a launcher icon which uniquely identify your app, Signing the app by digital signature, create an upload keystore, shrinking you code with R8, a code shrinker from Google and review the app manifest.

6 Conclusion

This study was conducted to evaluate the current Jaxber usability in general, not limited to user interface but also from the technological perspective.

In this research, a number of different available solutions are analyzed through reviewing academia papers, trends data and other valuable information resources like published books. The advantages and disadvantages for each platform are identified. The result indicates that PhoneGap has been removed from market, which means this technique is no more feasible for Jaxber development. On the other hand, statistical trends from google and stackoverflow indicates that React Native the other competitor platform has lost popularities in comparison to Flutter. Hjort (2020) states the React Native performance as “good” and “Close to native” for Flutter which could add a great value to Flutter. More to that React Native heavily depends on its community libraries to perform a functionality for instance file access.

In contrast, even though Flutter is a younger technology in comparison to other mentioned, it provides many build-in functionalities and polished widgets that would boost the development process as well as responsive UI. Considering the information provided in 3.3.3. Flutter could be determined as best possible solution for cross-platform mobile application development.

Currently Jaxber uses a custom-made REST API to communicate with back-end and database, while exploring, Firebase was identified to perfectly take over the REST API, since Firebase provides many services like Firebase Auth, Firebase Analytics, Firebase Cloud Messaging (FCM), Real-time database, Firebase Storage and more.

Since Flutter provides a numerous number of widgets, these Widget could assist Jaxber in the context of performance and native like feeling to the users, moreover Firebase could enable Jaxber to overcome the challenges that may come with the traditional REST API, for instance Firebase storage could assist in managing the images and video files, the Firebase messaging functionality would enable the Jaxber with real-time notifications.

The proposed solution could despite the fact that no deployable app was produced, but a feasible solution was found that will help the future development of Jaxber.

To sum up, Jaxber seems to be in the right direction, as a majority of public opinion is positive towards it in general, Montero Terán, (2020) study could support this statement.

Like every software project, Jaxber should also follow a progressive path for the purpose of improving, some of the internally discussed features should be implemented in the next version, features like QR code reader and invite system through campaign id sharing.

References

Adam. (2018, April 21). How Flutter Works. BuildFlutter. https://buildflutter.com/how-flutter-works/

Android Studio Official website. (2021). Google Developers. https://developer.android.com/studio

Android Virtual Device. (2021). Google Developers. https://developer.android.com/studio/run/managing-avds

Application ID. (2021). Google Developers. https://developer.android.com/studio/build/application-id

Basili, V.R., Caldiera, G., & Rombach, H.D. (1994). Goal Question Metric Paradigm. John Wiley & Sons. Inc.

Cmder official website. (2021). https://cmder.net/

Coursaris, C., & Kim, D.J. (2006). A Qualitative Review of Empirical Mobile Usability Studies. ResearchGate. https://www.researchgate.net/publication/220892707_A_Qualitative_Review_of_Empirical_Mobile_Usability_Studies

Dart package name convention. (2021). Dart Team. https://dart-lang.github.io/linter/lints/package_names.html#:~:text=Package%20names%20should%20be%20all,isn't%20a%20reserved%20word.

Differentiate between ephemeral state and app state. (2020). Flutter. https://flutter.dev/docs/development/data-and-backend/state-mgmt/ephemeral-vs-app

Faust, S. (2020). Using Google´s Flutter Framework for the Development of a Large-Scale Reference Application. [Bachelor's Thesis, Technical University Cologne]. URN. urn:nbn:de:hbz:832-epub4-14989

Fentaw, A.E. (2020). Cross platform mobile application development: a comparison study of React Native Vs Flutter. [Master's thesis, University of Jyväskylä]. JYX Digital Repository. https://jyx.jyu.fi/handle/123456789/70969

Firebase Authentication. (2021). Google Developers. https://firebase.google.com/docs/auth

Flutter Animation. (2021). Flutter Official Documentation. https://flutter.dev/docs/development/ui/animations

Flutter installation instruction. (2021). Flutter. https://flutter.dev/docs/get-started/install

Flutter internationalization. (2021). Flutter Official Documentation. https://flutter.dev/docs/development/accessibility-and-localization/internationalization

Flutter Sound. (2021). pub.dev. https://pub.dev/documentation/flutter_sound/latest/

FlutterFire. (2021). Flutter official documentation. https://firebase.flutter.dev/docs/auth/overview

Freitas, E. (2019). Flutter Succinctly. Syncfusion.

Harrison, R., Flood, D., & Duce, D. (2013). Usability of mobile applications: literature review and rationale for a new usability model. SpringOpen Journal. https://journalofinteractionscience.springeropen.com/articles/10.1186/2194-0827-1-1

Hartmann, G., Stead, G., & DeGani, A. (March 2011). Cross-platform mobile development. Tribal. https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf

Hjort, E. (2020). Evaluation of React Native and Flutter for cross-platform mobile application development. [Master's thesis, Åbo Akademi University]. The National Library of Finland. http://urn.fi/URN:NBN:fi-fe2020112392758

Huy, N.P., & VanThanh, D. (2012, December). ResearchGate. https://www.researchgate.net/publication/262233159_Evaluation_of_mobile_app_paradigms

Khawas, C., & Shah, P. (2018). Application of Firebase in Android App Development-A Study, 179, 46. https://www.researchgate.net/publication/325791990_Application_of_Firebase_in_Android_App_Development-A_Study

Kontio, Teemu. (2015) Cross-platform mobile development with PhoneGap. [Bachelor's thesis, JAMK University of Applied Sciences]. Theseus. https://www.theseus.fi/handle/10024/102950

Launch plan checklist. (2021). Google Developers. https://developer.android.com/distribute/best-practices/launch

Layout in Flutter. (2021). Flutter official documentation. https://flutter.dev/docs/development/ui/layout

Montero Terán, H.N. (2020) User-Experience based Evaluation of the Jaxber App: Towards competitive advantage through a study of User Experience. [Bachelor's thesis, JAMK University of Applied Sciences]. Theseus. https://www.theseus.fi/handle/10024/345055

Nielsen, J. (1993). Usability engineering. Morgan Kaufmann.

Panchal, M. (2020, August 13). Phonegap is Now Dead – Let's Move Towards PhoneGap Alternatives. Excellent Web World. https://www.excellentwebworld.com/phonegap-alternatives/

path_provider. (2021). pub.dev. https://pub.dev/packages/path_provider

PhoneGap vs React Native vs Flutter trends data. (2021). Google Trends. https://trends.google.com/

React Native vs Flutter trends data. (2021). Stack Overflow Trends. https://insights.stackoverflow.com/trends?tags=react-native%2Cflutter

react-native-audio-recorder-player. (2021). npm, Inc. https://www.npmjs.com/package/react-native-audio-recorder-player

react-native-fs. (2021). Github. https://github.com/itinance/react-native-fs

Sibinski, D. (2020, January 16). Basics of Flutter Widgets. Code Journey. https://www.codejourney.net/2020/01/basics-of-flutter-widgets/

Tan, J., Gencel, C., & Rönkkö, K. (2013, October). A Framework for Software Usability & User Experience Measurement in Mobile Industry. ResearchGate. https://www.researchgate.net/publication/258705485_A_Framework_for_Software_Usability_User_Experience_Measurement_in_Mobile_Industry

The Widget catalog. (2020). Flutter. https://flutter.dev/docs/development/ui/widgets

Thota, S., Ramya, C.V., Susmitha, G., Pranuthi, C.G., & Parshapu, P. (2020). Web-Based Department Focused Framework using Angular and Google Cloud Firestore, 10, 4. http://lokvigyanparishad.com/gallery/jeca-2064.29-f.pdf

Tran, T. (2020). Flutter Native Performance and Expressive UI/UX. [Bachelor's thesis, Metropolia University of Applied Sciences]. Theseus. https://www.theseus.fi/handle/10024/336980

Tunalı, V., & Erdoğan, Ş.Z. (October 2015). Comparison of Popular Cross-Platform Mobile Application Development Tools. ReserarchGate.

https://www.researchgate.net/publication/282816272_Comparison_of_Popular_Cross-Platform_Mobile_Application_Development_Tools

Video player platform interface. (2021). pub.dev. https://pub.dev/documentation/video_player_platform_interface/latest/

Video Player plugin for Flutter. (2021). pub.dev. https://pub.dev/documentation/video_player/latest/

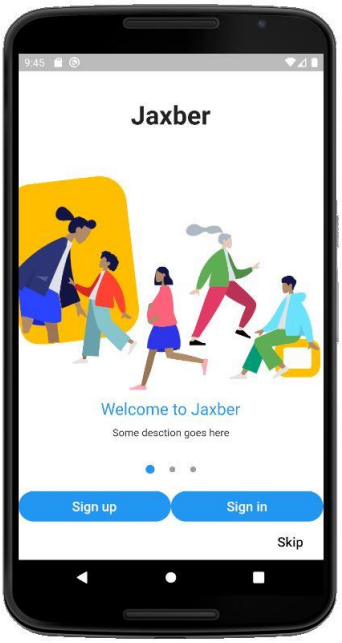
Virtual Machine. (2021). Wikipedia. https://en.wikipedia.org/wiki/Virtual_machine

Wu, W. (2018). React Native vs Flutter, cross-platform mobile application frameworks. [Bachelor's thesis, Metropolia University of Applied Sciences]. Theseus. https://www.theseus.fi/handle/10024/146232

Zahra, F., Azham, H., & Haslina, M. (2017, October 03). Usability evaluation of mobile applications; where do we stand?. AIP Conference Proceedings, https://doi.org/10.1063/1.5005389

Appendices

Appendix 1. The screen shots of Android Virtual Devices.



Name: Nexus 6

Resolution: 1440 x 2560: 560dpi

Target: Android 10.0 (Google APIs)

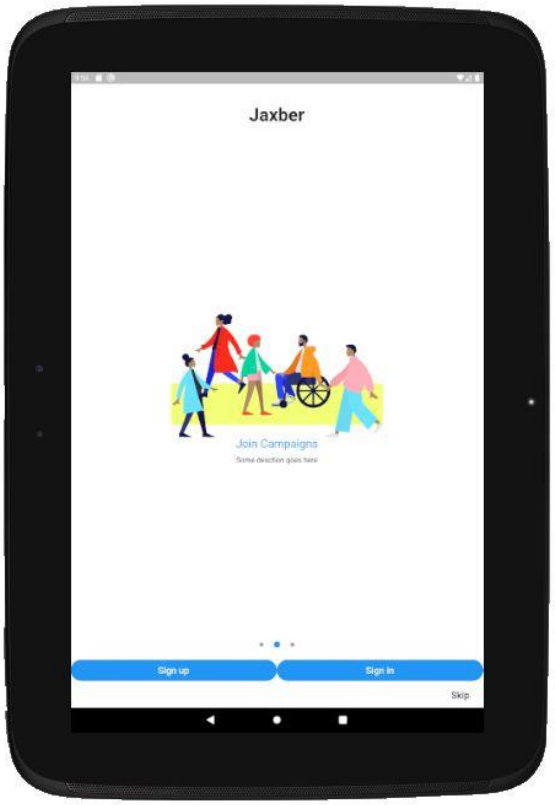
CPU/ABI: x86

Name: Pixel 3a XL

Resolution: 1080 x 2160: 400dpi

Target: Android 10.0 (Google APIs)

CPU/ABI: x86

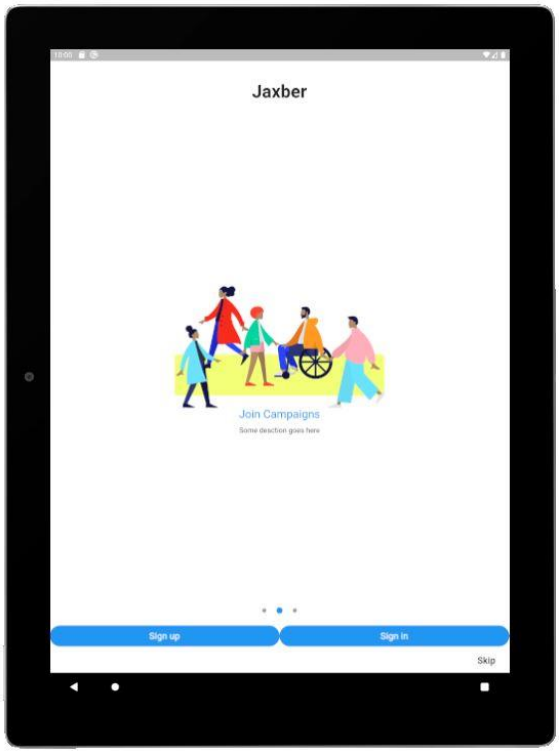


Name: Nexus 10

Resolution: 2560 x 1600: xhdpi

Target: Android 10.0 (Google APIs)

CPU/ABI: x86



Name: Pixel C tablet

Resolution: 2560 x 1800: xhdpi

Target: Android 10.0 (Google APIs)

CPU/ABI: x86