Haaga-Helia
University of Applied Sciences

Electronic subscription book

Vivian Bridy

Bachelor's Thesis
Degree Programme in BITE
2020

Haaga-Helia
University of Applied Sciences

Abstract

27 April 2020

Author(s)
Vivian Bridy

Degree program
BITE

Report/thesis title
Electronic subscription book

Number of pages and appendix pages
35 + 4

This goal of this thesis is to develop an electronic subscription book using the Flutter SDK. The project is commissioned by the brass band “Echo de la Vallée” from Switzerland. The thesis consists of three parts. First, it will familiarise the reader with Flutter and cross-platform technologies. Then, it focuses on the implementation process of the application. Finally, it describes and analyses the whole process through different testing methods to prove the validity of this work.

To get this paper relevant to a broader audience, this thesis emphasizes the importance, advantages, and gain in resources cross-platform, such as Flutter could bring to smaller businesses. It also reflects on improvements that could be made to this technology while enabling new capabilities.

The main use of the thesis is twofold. First, it will grant the possibility for small companies to organise and gather money for events quickly. Second, it is an in-depth analysis of Flutter as a cross-platform SDK. It gives a clear understanding of what are the requirements needed to use it at its best and what it could give back in return.

Keywords
Flutter, iOS, Android, Cross-platform, SDK, Widgets

Table of contents

Table of figures

Abbreviations

SDK	= Software Development Kit
OOP	= Object-oriented programming
UI	= User interface
JSON	= JavaScript Object Notation
NoSQL	= Not Only SQL
SQL	= Structured Query Language
UML	= Unified Modelling Language
QR	= Quick Response
IT	= Information Technology
NFC	= Near Field Communication
API	= Application Programming Interface
VM	= Virtual Machine
IDE	= Integrated Development Environment
JIT	= Just-in-time
AOT	= Ahead-of-time
PWA	= Progressive Web Application
FPS	= Frame per Second
HTML	= HyperText Markup Language
CSS	= Cascading Style Sheets
PHP	= Hypertext Preprocessor

1 Introduction

1.1 Topic

The organisation of an event can very often be challenging. There are many factors to consider if you want to make it successful. Money constraints tend to swallow all the attention, leaving more important matters such as innovation or quality behind. This aspect is even more exacerbated when you are a small business or organisation. Not having great staff or committee will force multi-tasking, which will again push prioritisation under stressful boundaries.

This project will focus on the money issue. More precisely, I will provide some research and a real implementation prototype to help small businesses or associations to find sponsors, gather money, handle accounting during the organisation of a small event.

1.2 Commissioner

The swiss brass band “Echo de la Vallée – Val-d’Illiez” is the commissioner of this project. This society was created in 1878. First, it was only a group of friends who wanted to play music together. Then, through the years, it has professionalised itself. Nowadays, ten per cent of the players are professional being paid to play in this brass band. To have sufficient incomes to cover all the charges, the brass band has different events. There is the annual concert, the Giron (once every four years) and the Festival (once every 23 years).

1.2.1 Event organisation process

The brass band elects a committee that will be responsible for the organisation of the event. It is composed of volunteers that do this in their free time. One particular job is demanding, the secretary. He or she has to take care of the “Event book” and manage all the sponsors. An event book showcases the program of the event while having listed all the sponsors. The more significant is the event, the thicker is the event book. Some sponsors will want to reserve an entire page to put some advertising. For example, an annual concert will gather around 2’825 Euros, a Giron 56’000 Euros and a Festival 141’000 Euros.

Nowadays, the search for sponsors is conducted as follows. Each member of the brass band is given a small subscription notebook. Each one of them will use his or her friends, family or network to gather pledges or directly cash.

Generally, one month before the event, all research for sponsors are stopped. The secretary begins his or her work. Apart from gathering all the subscription notebook, he or she needs to send an invoice to all the companies or associations that pledged. It will also ask for the logo if required. Those are then sent to the printer to construct the complete event book.

1.3 Objectives

To this day, everything is handled using paper notebooks, excel sheets and external accounting software. The work is spread across multiple points, increasing the difficulty of managing it. This project aims to provide a simple, on measure solution for small organisations such as the commissioner. It would take the form of a full-stack application that could handle everything from sponsors gathering to invoice generation.

Furthermore, as there exist many languages to develop a cross-platform application, this thesis will discuss the advantages and floes of Flutter extensively. The choice of a multi-platform language is guided by money constraints. Small businesses cannot allow themselves to hire multiple developers for an extensive period of time. Using this technology will enable the deployment of the application across Android and IOS devices for a lower cost.

The Flutter philosophy is widget oriented. This project will also determine how this particular approach affects the development process of a full-stack application. It will answer questions regarding the accessibility for new or experienced programmers, compare it with other similar languages and provide an in-depth analysis.

1.4 Delimitation

The Flutter SDK offers the possibility to deploy the application, using Dart as base code:

- In Java for Android devices,
- In Swift for IOS devices and
- In JavaScript for desktops.

For this project and due to time constraints, we are only going to deploy our application on Android and IOS devices. The desktop part of the application will be conducted outside of this project. It means the implementation part will not cover the automatic generation of the invoice, neither extensive analytics of the sponsors gathering process.

The functionalities developed and analysed for the work will be the following:

- Administration part
    - Creation and management of an event
    - List of all the donations (search functionality)
- User part
    - Joining process to an event
    - Creation of a donation

2 Why Flutter?

2.1 Methodological approach

Throughout the development of the thesis, interviews will be conducted to gather more information regarding the user experience. Following the qualitative approach, those meetings with new and confirmed Dart developers will allow us to gain in-depth insight into specifics aspects of the user experience.

Considering the number of languages available to develop a mobile application and the time constraint of this project, the quantitative methods such as surveys didn't suit. Identifying patterns or generalising would have been difficult as those languages are in continuous evolution.

2.2 Cross-platform development

Mobile phones combine a wide variety of different functions (Heitkötter, Hanschke, & A. Majchrzak, 2012). However, rapid changes to the mobile market in terms of operating systems, size and resolution of the screen make it challenging to develop efficiently.

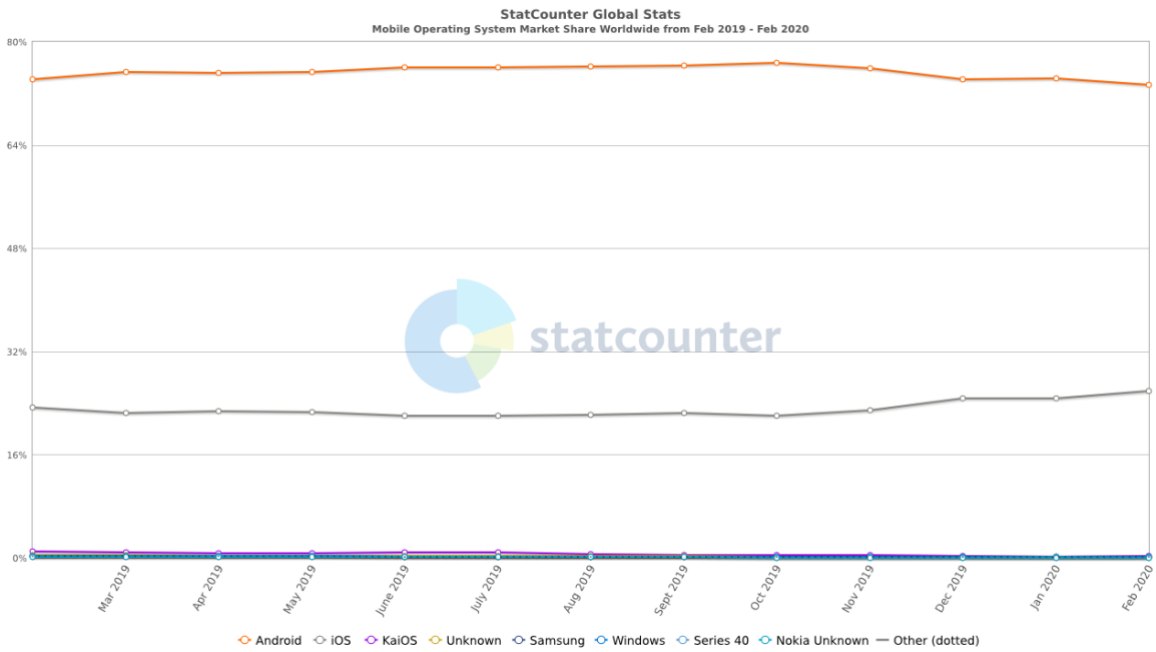


Figure 1 - Mobile Operating System Market Share Worldwide (Statcounter, 2020)

According to Statcounter, Google's Android represents 73,3% of the market share worldwide. Apple's IOS arrives second with 25.89%. Merged, they represent 99,2% of the market. Those statistics prove that from a company perspective, clients are present on those platforms.

The use of the word “platform” is key to the understanding of the mobile market. A platform is

“a (computing) platform is a technical framework on which one or more application may be run and where data are kept.” (Sturm, et al., 2017)

2.2.1 Apple Store versus Google Play

Google and Apple are known to have different philosophies, different approaches to the creation process. It takes effect by having at first two different “Application stores” where users can find the application they need. To have their application shown of those stores, developers have to comply with the listing of obligations fixed by the two companies.

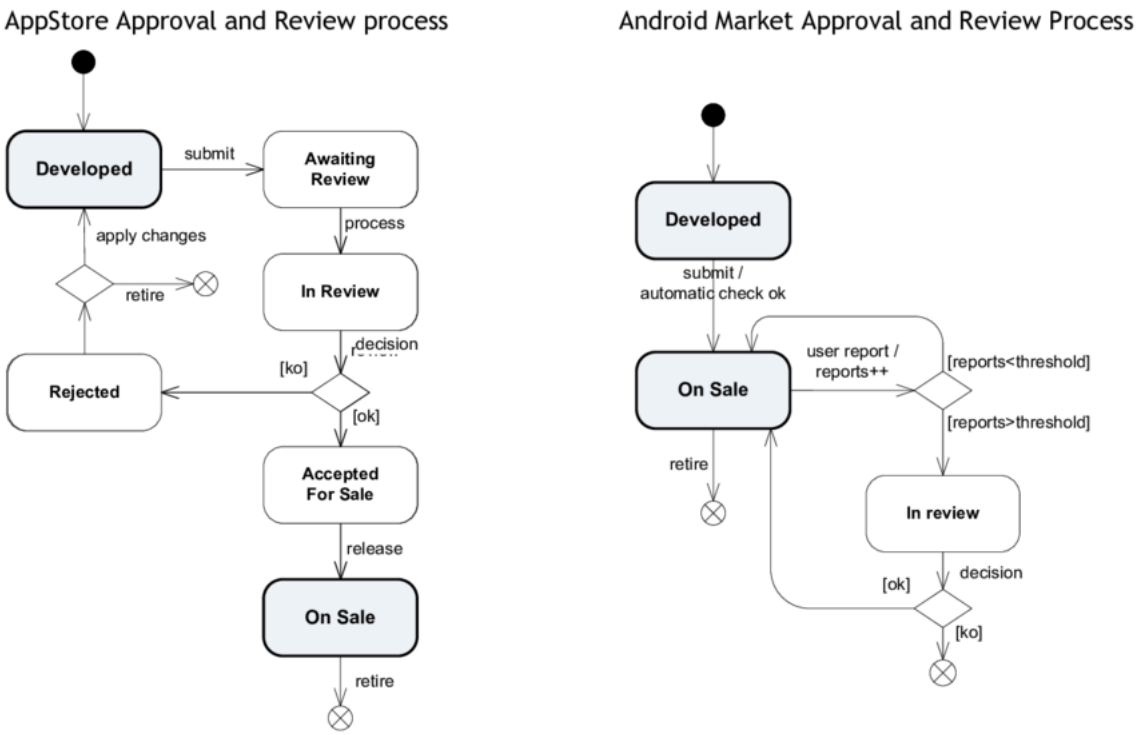


Figure 2 - AppStore and Google Play Approval Processes (Cuadrado & Duenas, 2012)

Analysing this figure, we see on the left-hand side the approval process of the AppStore. One point to be noted is that Apple manually inspects every submission, therefore, having more control over it. Since September 2010, the list of criteria is pubic, meaning for developers to develop their application proactively.

On the right-hand side, Google Play approval may seem more open. Each application that is submitted is automatically checked. Where Apple manually review each application, Google checks theirs using crowdsourcing.

“Developers are liable for the submitted applications, and users detect and report unacceptable applications.” (Cuadrado & Duenas, 2012)

Nevertheless, those differences in the approval process are evolving to adapt and react to competitors. Apple has, for example, allowed more programming languages which applications could be developed with (Evans, 2020). More than two platforms, there are also two languages: Swift for Apple and Java for Google.

2.2.2 Swift versus Java

First of all, it is noticeable to see that Swift shares to functionalities with Objective-C. We could consider this second language as the ancestor of Swift. Started in 2010 by Chris Lattner, Doug Gregor and John McCall, the swift language idea is taken from many languages such as python, ruby or C language.

Developers seem to describe it as easy to understand and adopt (Rebouças, et al., 2016). Most questions or problems concern libraries and frameworks, instead of the language itself. Besides, there have been remarks made about the compiler which is stated to be “the worst we could ever imagine”. It means that developers had real difficulties in understanding the errors shown by the compiler.

As a pure OOP (Object-oriented programming) language, Java contains the following concepts: classes, objects, inheritance, encapsulation and abstraction. Therefore, it is known as a powerful and robust language. Being a platform-independent programming language, Java can run software and hardware system.

Even nowadays, Java is still the most popular programming language overall (TIOBE Index for February 2020, 2020). It can be explained by the fact that Java had been created already in 1996. It had time to settle and evolve following the needs of the developers.

2.2.3 Benefits

Even if developing cross-platform software may seem at first as a lot of work, there are many reasons which prove the opposite.

At first, it brings more portability. Having the ability to deploy an application of both IOS and Android devices will expand your market. Furthermore, some clients have heterogeneous devices. The faculty to support your customer on all platform and tools give you a considerable advantage over competitors.

Then, as the code will not be spread across multiple languages or multiple developers’ team, it will lead to better software. The quality improvements are made by reducing the language fence. Developers will focus more on the functionalities rather than technical issues due to cross-language.

Furthermore, nowadays, we could consider cross-platform development as necessary, giving the competitive environment. (Hook, 2005)

2.2.4 Drawbacks

As stated previously, new technologies are evolving rapidly. It means that languages need to follow too.

In consequence, they are in a perpetual testing phase. New experimental functionalities are continuously added. Those new capabilities should not be implemented directly into a working application. Support for those features could be inefficient and episodic. It is recommended to wait until other peers have well tested the functionality.

Even if at first sight, cross-platform may be the perfect solution, the reality remains. Developers will again deploy the application on different environments. It occurs that a feature available on some platform may not be on the other one, at least not accessible in the same way. In this particular case, all the advantages brought by cross-platform development falls apart as we must then code in each language again. (Hook, 2005)

2.3 In-depth Flutter SDK

To understand Flutter, we need to explain what a Software Development Kit (SDK) is. A way to grasp the meaning of an SDK is to compare it to an API (Application Programming Interface). Having the ability to distinguish what each of them provides is a core element to develop modern applications. (Sandoval, 2020)

An API could be described as an interface for software to interact with each other. They exist with many types or sizes. Nowadays, most actions done on a computer utilises an API. For example, when moving files from a folder to another, the data from the first folder is stored into the RAM of the computer using an API. This same API will also make the opposite transaction. We understand that APIs bring consistency to an environment. They link everything together. They bring to life concepts like consistent coding or replicable functions. (Bloch, 2006)

As its name states, an SDK provides a list of tools, documentation, processes and help developers to develop applications on specific languages or environment. Where an API is a block when constructing a house, an SDK is a workshop. Very often, the confusion between both of these concepts is that a software development kit contains an API in most of the case. (Sandoval, 2020)

2.3.1 Architecture choices

Flutter is built around three core principles: fast development, flexible UI and native performance. (Flutter-dev, Homepage, 2020)

Fast development

Fast development is built inside Flutter using hot reload. The developer can run a Dart VM (Virtual Machine) and update it by injecting new source code directly into it while running. It allows for a quick view of the changes made to the code without the need to restart the VM at each move, saving a lot of time. Hot-reloading comes with four advantages. First, as the screen you are working will not need to refresh, your console logging stays untouched. Therefore, it makes debugging a lot easier. Second, during a hot reload, the state of your application is preserved. In practice, this means you will not need to navigate to your specific component to find what has changed. Your screen will not restart and saves you time. Third, hot-reloading is faster than a rebuild or a refresh. A screen refresh can take from 2 to 5 seconds for a small app. We need to have in mind this would add up throughout the day. In comparison, a hot reload takes about half a second. To conclude, hot-reloading affects coding practice. It encourages developers to choose good coding patterns. For example, it will show you that the state must be kept in parent components instead of child components. (Mills, 2020)

Using an IDE (Integrated Development Environment) like Visual Studio Code that supports Flutter's IDE tools, you can do a hot reload using the toolbar provided. After you started your application in debugging mode, you will be able to it as follow (when clicking inside the yellow scare) :

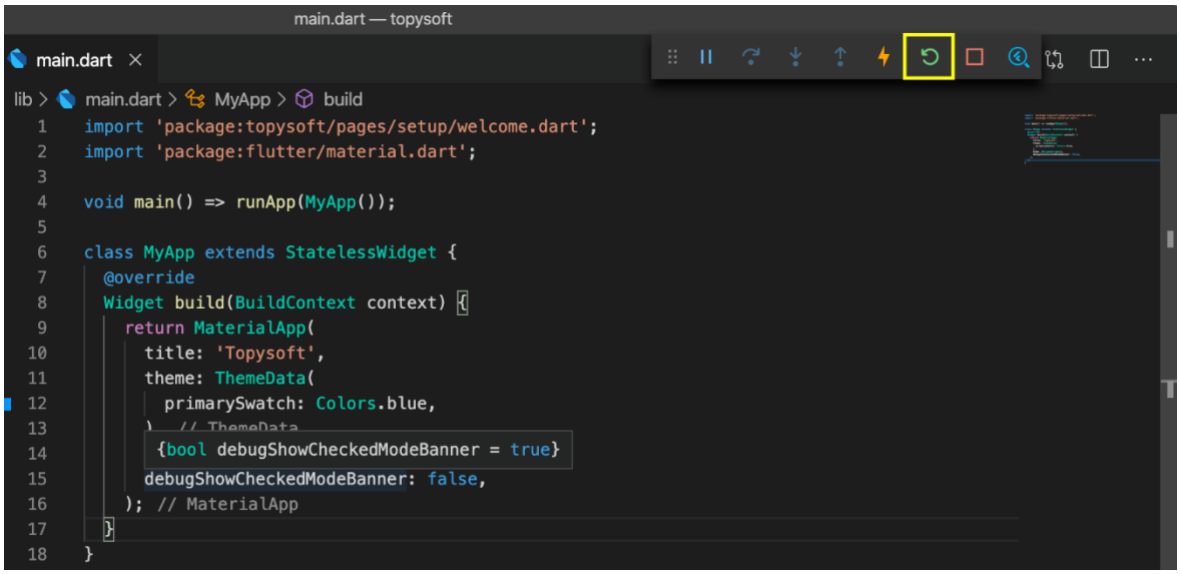


Figure 3 - Hot reload on Visual Studio Code

Figure 4 - Result after hot reload

Figure 4 shows the advantages of hot-reloading. Inside the current project application, there are 982 libraries loaded to make it work. When doing a normal restart of the application, it took 1’114ms. However, when using the hot reload functionality, we can see that the IDE only reloaded 1 of the 982 libraries. It has made this choice because the changes made to the source code were only affecting one library. The hot reload took only 196ms. It improved by 82%. To put it on a bigger scale, a project that would initially take one year (twelve months) could be reduced to nearly two months. (Flutter-dev, Hot reload, 2020)
However, there are some limitations to hot-loading. When using Dart, global variables and static fields are treated as a state. As explained earlier, they will not be reinitialized during hot reload.

Figure 5 - Changes not saved after hot reload

In the example above, we declare a table containing for items which are calls to a method “test” with one parameter. If we change the fourth element of this table while running our VM, this change won’t take effect with a hot reload. The first time you run a Flutter application, this static table is read and stored. However, when doing a hot reload, all global variables and static fields are considered as a state. By consequence, they will not be reinitialised. (Flutter-dev, Hot reload, 2020)

Flexible UI

Flutter’s architecture is based on the widget concept. Everything is a widget. This modern framework took inspiration from React. A widget has internal parameters that can be configured. All parameters put together will create a state of that widget. When a developer makes changes to the code, it will trigger the framework to evaluate what changes have been made compared to the previous state of each widget within the application. This calculation allows Flutter to rebuilt and transition the UI quickly from one state to the next. (Flutter-dev, Widgets, 2020)
To give an example of some basic code using Flutter, we will discuss it using the “Hello world” minimal application.

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

void main() {
 runApp(
  Center(
   child: Text(
    'Hello, world!',
    textDirection: TextDirection.ltr,
   ),
  ),
 );
}
```

Figure 6 - Hello world using Flutter

Situated at the core of a Flutter application, the function “runApp” makes the widget associated with as the root of the widget tree. You will use this to build every Flutter application. We can see in the example that the application is made using widgets as layers. We have a “Center” widget which contains “Text” widget as its child. Each child or widget will have heritage from its parents. The framework is built to force the root widget to take all the space possible on the screen. Therefore, the text “Hello World” will be centred.

When creating an application, you need to choose your widget based on if it will have a state or not. They are two main types of widgets, “StatelessWidget” and “StatefulWidget”. Figure 3 shows the implementation of a stateless widget. The primary function of a widget is the “build()” function. It defines what the widget does and how.

```
class NewDonation extends StatefulWidget {
  final String idEvent, idUser;

  NewDonation(this.idEvent, this.idUser);

  @override
  _NewDonationState createState() => _NewDonationState(idEvent, idUser);
}

class _NewDonationState extends State<NewDonation> {
  String idEvent, idUser;
  _NewDonationState(this.idEvent, this.idUser);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Add a new donation'),
      ), // AppBar
      body: StepperBody(idEvent, idUser),
    ); // Scaffold
  }
}
```

Figure 7 - Stateful Widget

Figure 7 shows the implementation of a stateful widget. As previously stated, in this type of widget, we have to implement a state. The primary function will be “createState”. It requires to return class of the type “State”. When creating “_NewDonationState” class, we extend it from “State” class. By consequence, it will be forced to have a build function.

Flutter comes with some basic widgets (Flutter-dev, Widgets, 2020):

- Text: It displays a style text
- Row, Column: It creates a flexible layout in both directions. These widgets are based on the flexbox layout model.
- Stack: This widget allows developers to position every widget relatively to the stack itself.
- Container: It creates a visual element that can be decorated by adding background, borders or shadows.

Native performance

Flutter’s framework uses the Dart language. As Dart can be chosen to build simple scripts or complex applications, it is very flexible.

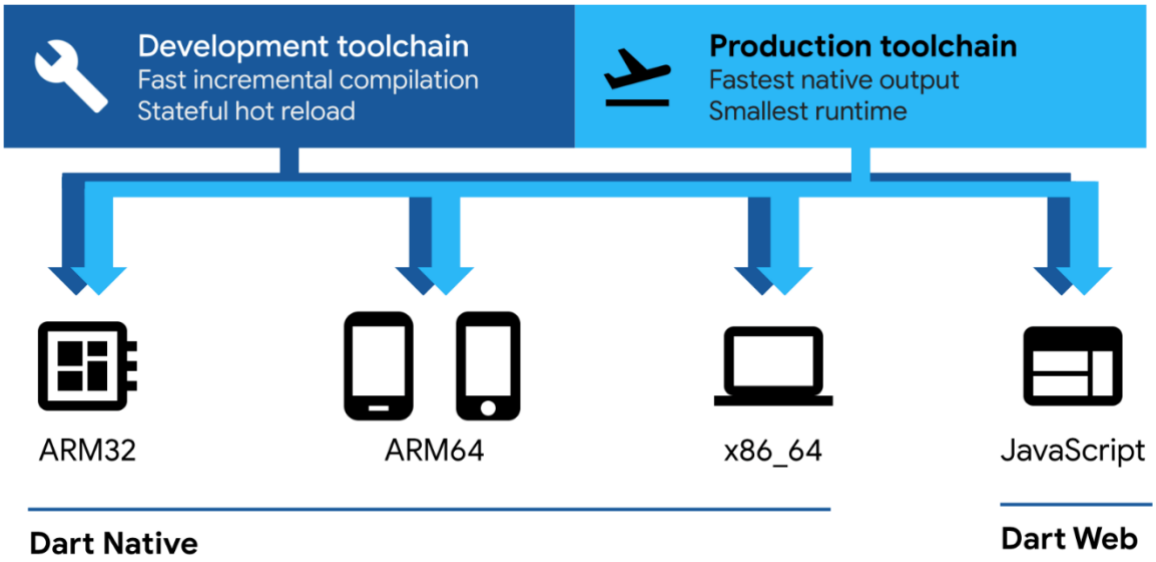


Figure 8 - Flexible compiler (Flutter-dev, Platforms, 2020)

Any code that uses Dart can be compiled on a mobile device, a desktop, a server and more. It is possible because Dart includes technologies like VM with JIT (just-in-time) and AOT (ahead-of-time) to produce machine code. JIT enables fast developer cycle into each iteration. It supports pure interpretation (needed on iOS devices) and runtime optimisation. AOT manages memory with fast object allocation and enhanced garbage collector. (Flutter-dev, Platforms, 2020)

Having the ability to compile on both iOS and Android from the same code is one capability. Flutter also has the functionality to display native elements from both environments. For example, the following code will display native items for iOS or Android.

"return CircularProgressIndicator();"

Figure 9 - Progress indicator Android

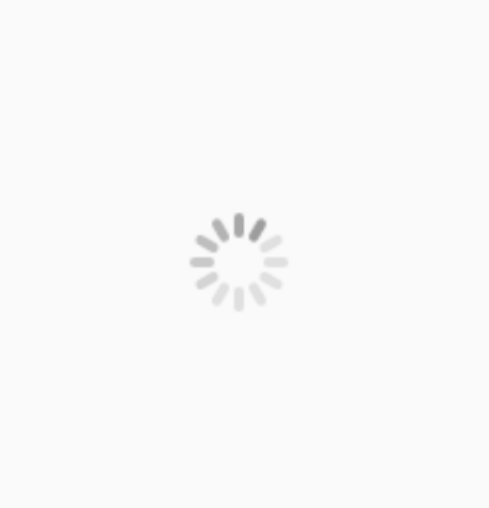


Figure 10 - Progress indicator iOS

2.3.2 Future capabilities

Planned for the final application but not chosen for the requirements of this project, the Dart framework has enabled web support on a beta channel. Therefore, using Flutter, the application could render screens using web technologies such as HTML, CSS or JavaScript.

When deploying Flutter application on the web, what you do is putting code on top of the standard browser APIs. The code you wrote using Flutter is compiled into a single Javascript file. Therefore, it can be deployed on any web server. (Flutter-dev, Web support, 2020)

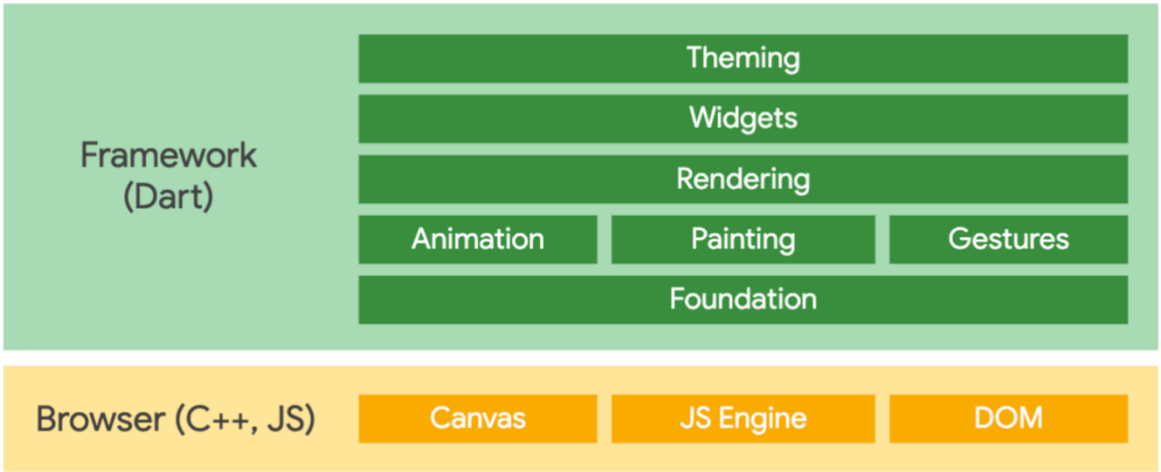


Figure 11 - Web support on Flutter

To conclude, the web version of a Flutter application could be implemented in various cases, such as a PWA (Progressive Web Application), embedded web content within a mobile application.

2.4 Other alternatives

Flutter is not the only alternative when considering cross-platform development. This chapter will expose the differences or standard features between the tools available on the market.

2.4.1 React native

React native was developed by Facebook, compared with Flutter, who is affiliated with Google. There are some similarities when looking at the philosophy behind those technologies. One of them is the modularity idea. It means that developers need to think about the creation process of their applications with smaller parts put together. React Native constructs a page with components while Flutter uses widgets. Modularity brings reusability. Once a component or a widget is created, it can be reused throughout the application but also for other projects.
Choosing one element to make the comparison, Flutter has some advantages over React native as it has native access performance.

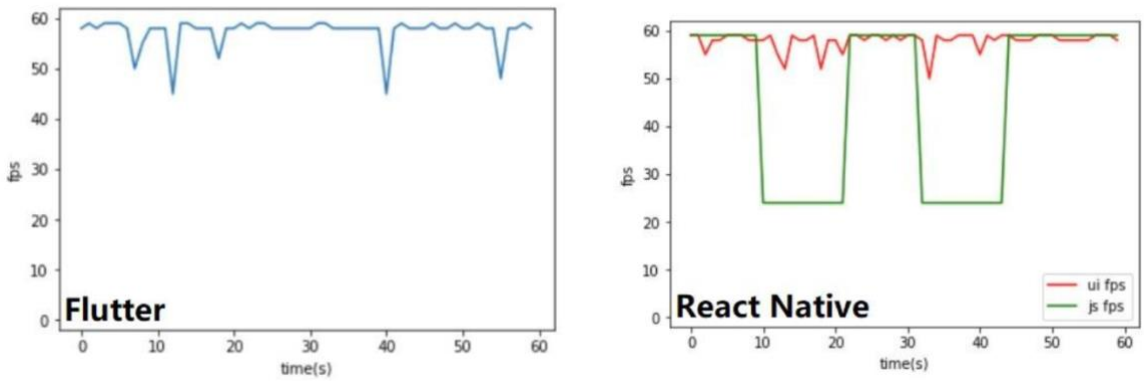


Figure 12 - FPS tracking for scrolling (Wenhao, 2018)

To conclude, we can say that both are doing great when analyzing performances.
However, it appears that Flutter was designed from the start to handle mobile architecture. It gives him more viability when doing rapid scrolling, for example.

3 Implementation of the application

The implementation of the mobile application will be held as follows.

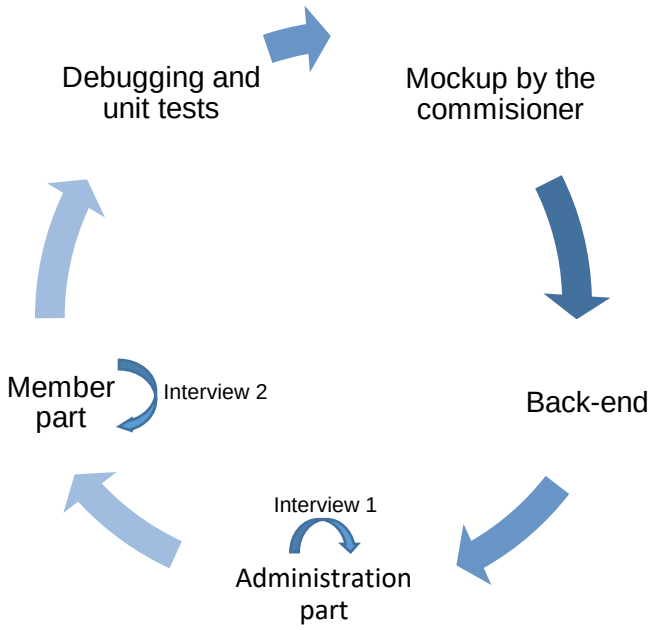


Figure 13 - Application complete implementation process diagram

The project will be conducted, trying to follow an agile philosophy. As described in the figure above, the administration and member part will be analysed, criticised during two interviews. It will also be sent back to the commissioner for update purposes but also comments or corrections.

Having feedback coming from insiders (commissioner) and outsider (flutter developers), the quality of the project will increase. The technical decisions will also have more legitimacy after being reviewed by peers.

3.1 Mockup from the commissioner

To start the project, the commissioner provided the writer with a visual prototype of the application. It gives a good representation of the functionalities to be implemented. Furthermore, it also provides the writer with a first look at the way the application is naturally thought to be used.

For example, we can see with the screen captures below that the commissioner has created several steps for the subscription process. This information will lead our development process into researches to find out if those steps affect or not on the user.

Finally, this mockup qualifies the writer to forecast the data-model needed for the mobile application.

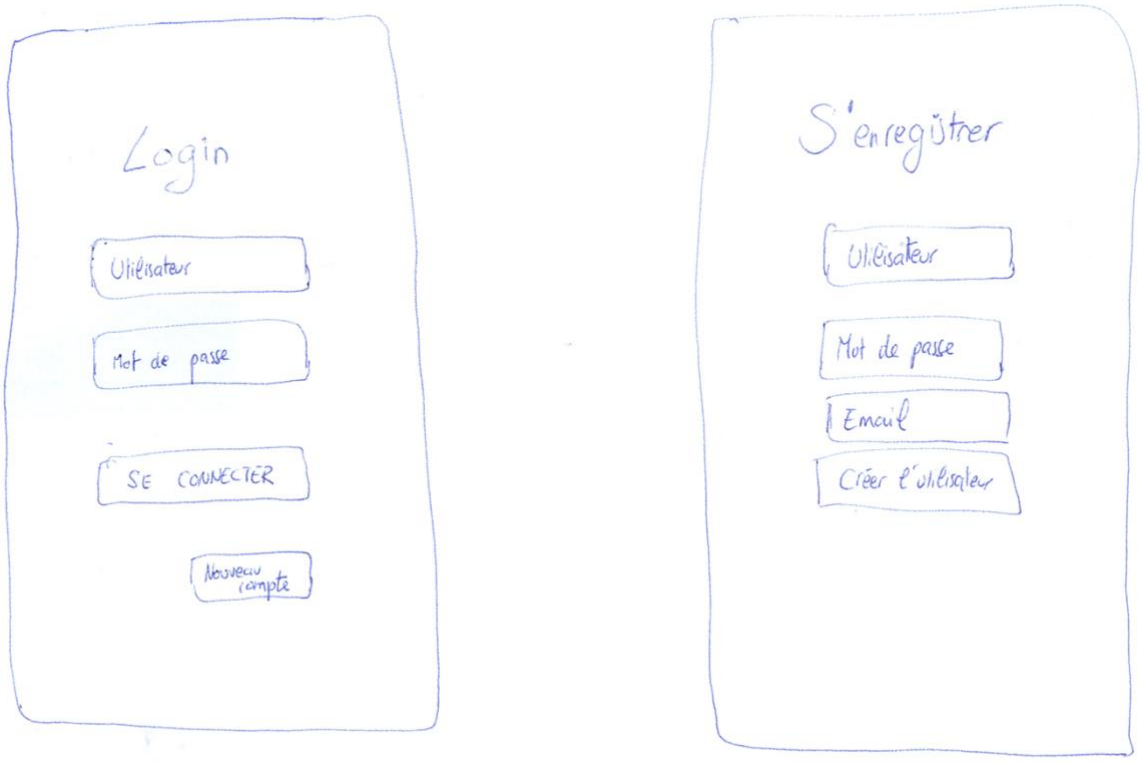


Figure 14 - Screenshot from Mockup n°1

The complete mockup is added as Appendix n°1. From this, we can confirm that the commissioner wants a straightforward UI (User Interface). However, it gives more freedom regarding the design choices. The commissioner provides no instructions for colours or fonts.

3.2 User stories

Analysing the mockup given by the commissioner and using the knowledge we have about the brass band community; we can create the following user stories.

"As an administrator, I can create an event so that an event is created."

"As an administrator, I can manage an event so that the information related to this event are up to date."

"As an administrator, I can delete an event so that the list of events I can see is up to date."

"As a member, I can register myself to an event so that I can start searching for new donations."

"As a member, I can search a donator inside the application so that I can confirm if it has already made a donation."

3.3 Typical modern architecture

A typical architecture used to develop a mobile application is to separate it into two categories: Front-end and Back-end. The first is regrouping all the elements needed for the interface. All the pages, buttons, images the user of the application sees can be considered as front-end. A developer in charge of this part of the project will create the graphical user interface. At the minimum, the following skills are required (Harnil, 2019):

- HTML (HyperText Markup Language)
- CSS (Cascading Style Sheets)
- Javascript

As this list is minimalistic and non-exhaustive, we should also consider other frameworks. The writer will not make a list of them as the number, preference and capabilities of those framework change quickly. However, for this specific project, the writer will develop the application using Dart. Using Flutter as a framework, the project will be built using pre-made elements called as widgets. Another interesting fact about Flutter is that programmers can use libraries made from different providers such as Google or Apple. An application made using Flutter could have the design attributes of an Android or iOS device.

As a front-end would be an empty shell without a back-end. This work is not visible from a user perspective. Three pillars form a back-end solution: the hosting server, the database and the application itself. The server could be described as a hard disk. All the files needed for the application to run smoothly are stored on it. The back-end developer builds the database. Its goal is to keep in tables all necessary material for the application to deal with the everyday workload. Finally, for the database to be updated or changed, the developer will use dynamic languages to link the application and the database. At the minimum, a back-end programmer should have the following skills (Harnil, 2019):

- PHP (Hypertext Preprocessor)
- Python
- SQL

3.4 Back-end

This project will develop a full-stack application. To accomplish this objective, the writer will also create the back-end to support the application in both environments.

3.4.1 Which technology for the back-end?

In the early stages of the development process, multiples technologies were considered. After extensive research, Firebase has been chosen. Several points have driven this choice. First, with Firebase, the Real-time Database stores data as JSON (JavaScript Object Notation). It is also coordinated in real-time to each linked client. (Lahudkar, Sawale, Deshmane, & Bharambe, 2018)

Storing data using JSON is key to this project; it allows for easy transformation of the data. Another point of interest is type management; this application does not have exotic data types. Finally, the Flutter SDK has efficient serialisation and deserialisation native methods, making it easy to work with. (Peng, Cao, & Xu, 2011)

3.4.2 UML Model of the Database

Even if Firebase uses NoSQL (Not Only SQL), we will at first model our database using a relation type of database. Through the UML (Unified Modelling Language) diagram below, we will discuss the choice made to model it.

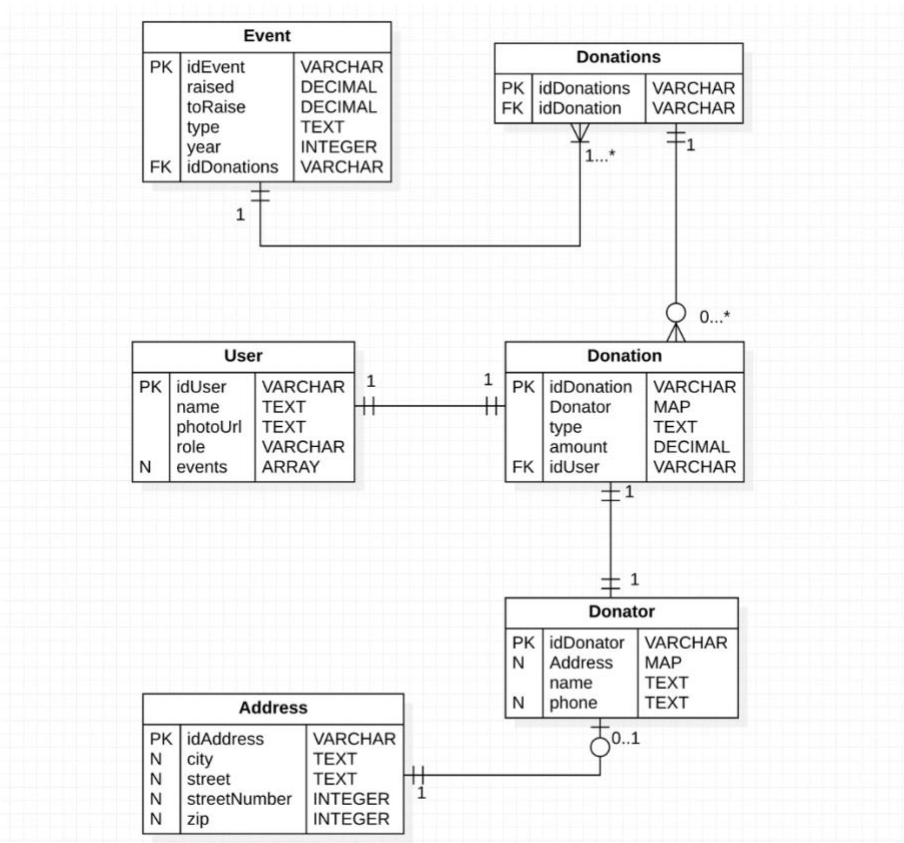


Figure 15 - UML of the Database

To understand this model, we will go through each table, allowing in-depth analysis.

Event table

Both “raised” and “toRaise” fields are set as a decimal. It allows more precision to the administration part when analysing the gathering process of sponsors during an event. The “type” field is defined as text to include spaces character. Finally, the “year” field will only accept integer values. By doing that, we restrain the error margin on the user’s input.

Donation table

The field “Donator” is declared as a map. It is a data type used by Firebase. It allows quick mapping of an object inside another one. This table is linked to the event table using an intermediate table “Donations”. Each event has a list of donations. The final implementation is this concept will be done differently using Firebase’s real-time database.

User table

This table gathers all the information related to the user of the application. It contains an array to list all the events in which the user is a member of. It has been done to iterate quickly through this array.

Donator table

As for the donation table, this table has a map field. In this situation, the embedded object has only an esthetic purpose. All the data gathered about the donator is not put inside only one table. In the future of the project, we could consider controlling the quality of the addresses. To define an address as good enough, it would be tested to see if it is unique.

3.4.3 NoSQL Database description

We previously explained why Firebase was our choice for the back-end. Driven by the potential functionalities offered by this technology, we had to transform our data model to make it work inside a NoSQL configuration.

The first topic to discuss is the data type restriction. Firebase has a limited choice regarding data type:

- string
- number
- boolean
- map
- array
- null
- timestamp
- geopoint
- reference

For this project, the need for exotic data type was not a problem. For example, the address field has been stated as a string.

Inside the Cloud Firestore, we created two main collections: events and users. Each of these collections contains a list of documents. Then, for the events collection, each document can contain multiple mandatory fields such as “type” or “year”. It also includes a collection of donations. This structure allows us to navigate easily inside the donations made for a specific event.

The user collection is less complicated. It contains all the fields related to the user itself. Then, to link it to the events collection, we created an array of events. This way, we simulate a foreign key. The only downside of using the method is the work that will need to be done regarding the creation, update and deletion of events. Each array in each user document will need to be updated. Nevertheless, exploiting all the capabilities of Firebase will provide us will an excellent workaround to solve this problem. We will discuss it during the creation of the screens related to the member.

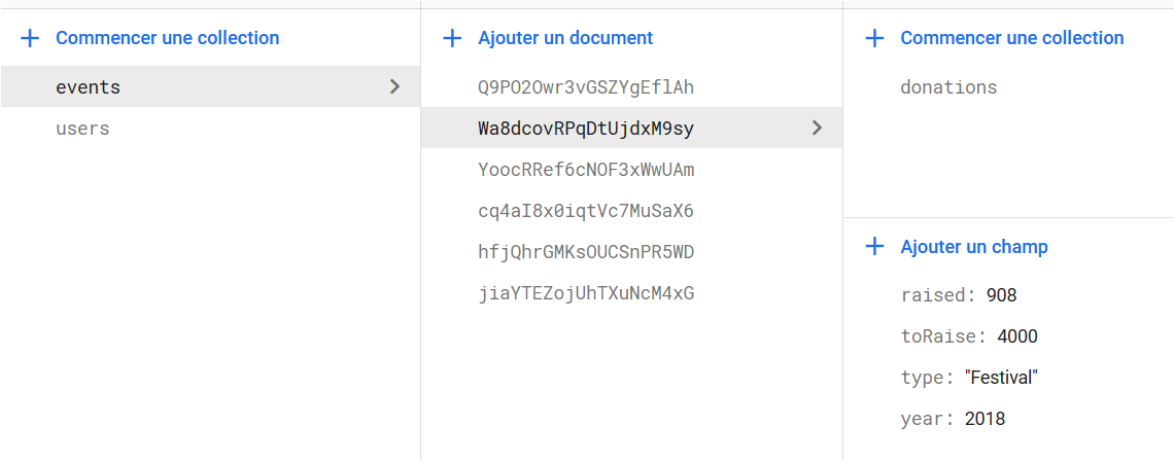


Figure 16 - Events collection example

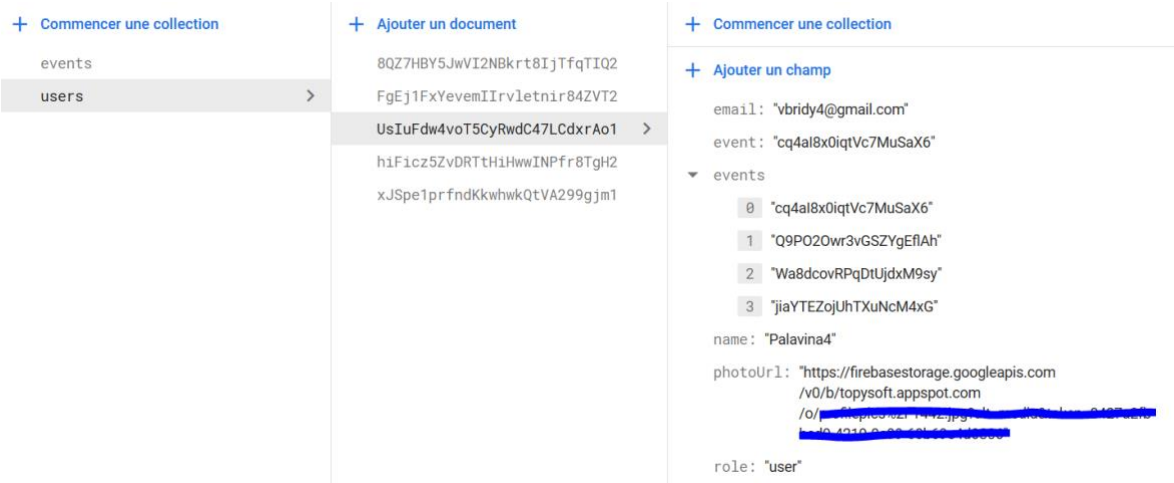


Figure 17 - Users collection example

3.5 Administration part

The secretary of the brass band will mostly use the administration part of the application.

3.5.1 Goals

Without the application, the gathering process of donation is very time-consuming. One of the first goals of the application is to eliminate the writing issue. As the data is digitally stored from the beginning, we don't have to copy all the hand-written into an excel file.

The research capabilities will also be enhanced. Nowadays, if the secretary wants to find if a donator has already donated previously, he or she must either know it by heart or needs to search it manually through each excel file made throughout the years. There are also no links made between the different kind of events organised by a brass band.

From the outside, it seems that all the data gathered during those events are not adequately put for contribution. They are only stored for accounting purposes. The other goal of the project is to give this data some value, either by providing live status to the secretary or allowing extensive searching capabilities.

3.5.2 Admin interface

On the first screen, the administrator is given the list of all the events he or she is related to. The advantages of Firebase are directly put into action here. This list is automatically updated at any moment. We could remain sceptical regarding this feature and ask if this will not drain all the network usage (endless repetitive queries to check the data in the database).

However, Firebase is well integrated into Flutter. Packages are available to handle all the interaction with Firebase.

The writer has encountered some difficulties when trying to implement the Firebase connection. The main issue was that the actual implementation is different on IOS or Android devices. You are required to add the package inside your work environment. It will only provide the necessary libraries to work with Dart but not the one for Swift or Java. To make it all work together, some modification needed to be done manually inside each proprietary language. The work was especially tricky inside the Android environment. To put it simply, Google is pushing to merge all existing libraries to Android X. This new

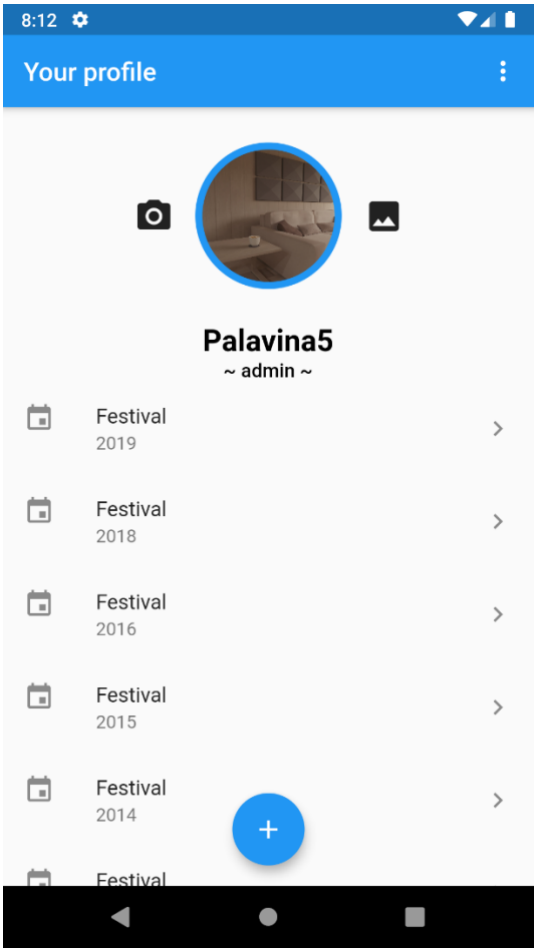


Figure 18 - Admin interface

platform has the capabilities to be deployed on multiple types of devices, from wearables to computers. To make this migration process more manageable, Google provides an option inside Android Studio to automatically migrate a library to Android X. This is not always successful. Having libraries from Android X and others working in the same application can create conflicts. It was the case during this project.

The solution to this issue has been to work with 100% Android X compatible libraries. One downside is that nowadays, the choice of libraries is not as developed.

From the IOS point of view, the only concern was to modify the rights given by the application. One of the functionalities implemented in this first screen is picture management. To access the camera or the library, you must provide the permission manually using a certificate generated by Xcode.

From the start, we face the main challenge of cross-platform development. The Dart language is easy to get familiar with as it has many resemblances with Java in its widget (or class) approach. However, it gets annoying to modify each environment depending on the quality of the library implementation we use. We can consider that official packages found on “pub.dev” are more reliable than others.

3.5.3 New event

To create a new event, the administrator must enter three information: the type of the event, the year it will occur and the objective fixed.

The implementation is done using a form widget. Build-in capabilities are useful to prohibit administrator actions. In fact, all the fields are mandatory, and the “create” button will not work until all the fields are completed. The “type” field has no other limitations than not to be empty. The “year” and “toRaise” fields have the same type of constraints. They are numeric fields. However, the “year” field will only accept integer numbers. It raises to problematic of keyboards. Without even considering other platforms, keyboards are usually a significant source of issues when developing a mobile application on Android.

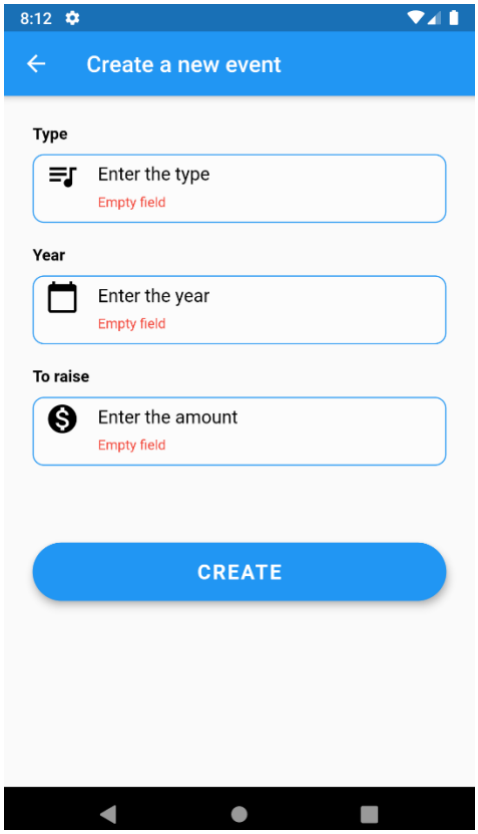


Figure 19 - New event

The main problem with the Android world is how fragmented it is. Each constructor is given the possibility to implement a different type of keyboards. Most of the time, the packages

accessible with Dart will use the basic keyboard provided by Google itself. It won't work for constructors such as Samsung or Huawei.

The best solution for the problem would be to develop a custom keyboard for this application. This way, we could have precisely the functionalities needed for the application while having consistency throughout all the platforms. Given the time constrain for this project, the writer will not develop this custom keyboard.

3.5.4 Adding a member

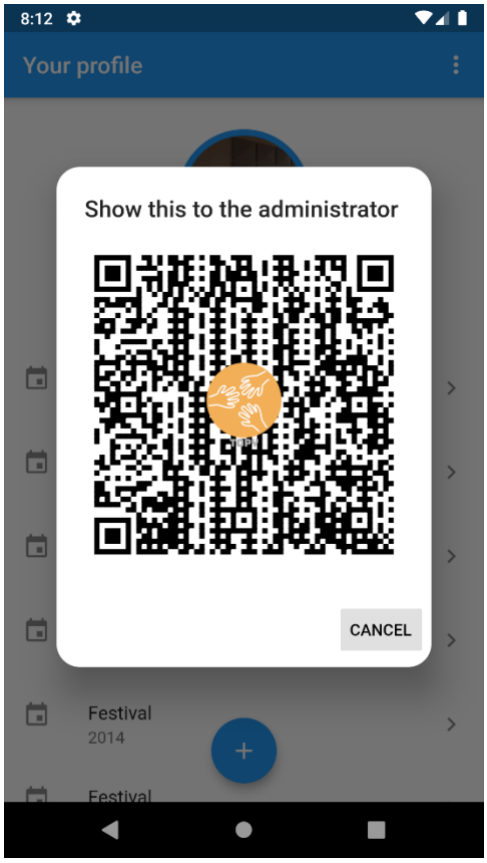


Figure 20 - Adding a new member

There exist multiple ways of adding someone to a list. In the real-time database, an event has no record of all the members working to gather the sponsors.

Instead, each member that has the application can apply for an event. In short, each user document will contain a list of each event this user is a member of. Then, to apply for an event, we need the administrator side of the application to know the name of the new member.

As the name is not enough to identify a member, we will take the id (identification number) of the member.

To transmit this information from the mobile phone of the new members to one of the administrators, the writer first thought it could simply be displayed on the screen A (new member). Then, the administrator simply must enter the number in a field on his phone (screen B). This method has some downsides. The number is shown publicly to others. To counter this issue, the number could be shown on the screen A after some encryption. The new problem is now that if we want enough security, the encrypted number will become very long and complicated. It will require time and focus from the administrator to copy it entirely without mistake. The writer is acquainted with the fact that most of the enrolments in this environment are done at the end of a repetition. It means that this action is done in a short period of time.

To make to process as quick as possible, the writer opted for the QR (quick response) code. As stated in the name, it allows for immediate transmission of data with various adjustment capabilities. The implementation of the functionality has been done in two parts: the scanning and the generation.

The scanning of a QR code is done through the camera of the device on which the application is run. The writer used a pre-made package available to solve the scanning part. The complicated part of this process was the fact that IOS and Android platform deal differently with permission to access the camera. It took time for the writer to make everything work together.

The generation part of the QR code was also done using a pre-made package. The package used in this project gives the possibility to define the different levels of generation; the higher the number, the more complicated the QR code is. There is no educated choice regarding the level of generation chosen for this project. The focus has been put into using QR code to quickly transmit the information from the screen A to the screen B. The encryption of the ID itself has handled the security part.

The details regarding the choice of the encryption method will not be discussed during this project as it is not the primary objective.

3.6 Member part

The member part of the application could be used by every person wanting to help a business gather sponsors for a particular event.

3.6.1 Goals

Nowadays, the actual job of finding new sponsors is done carrying a small paper book. A copy of amount pledged is given to the new sponsors. Nevertheless, the secretary has to control, input every new donation manually.

Furthermore, you cannot proactively have information regarding old sponsors, how much they gave for the previous edition of the event. The goal of the application is to provide assistance and meaningful information to the person searching for sponsors.

3.6.2 Member interface

On the first screen, the member has a view of all the sponsors that have been already gathered, also by other members. Using the functionality, it can quickly see if a person has donated or not. (Figure 21 third capture)

The primary purpose of this project is to ease the process of collecting sponsors. Personal experiences of the writer as a member show that you can gather sponsors for multiple

events at the same time. Without the application, you would need numerous physical notebooks. (Figure 21 second capture)

When a member clicks on a donation, it displays more detailed pieces of information about this particular information. (Figure 21 first capture)

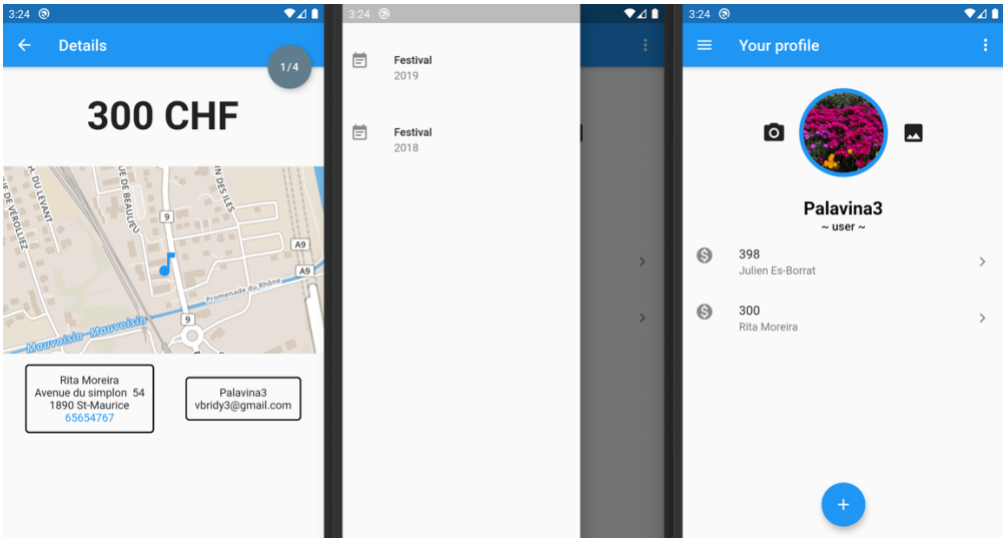


Figure 21 - Member interface

3.6.3 New donation

When a member has the opportunity to gather a new donation, he or she clicks on the plus button at the bottom of the screen. Then, the member is asked to enter the detailed information of the donation. The process is separated into three steps. Those are not final as the member can go back to one particular step if there is the need to change something. The first step is to provide the amount and type of donation. The type defines if the money is given cash or if the association will have to send an invoice. The second and third steps need to be provided with the information of the donator.

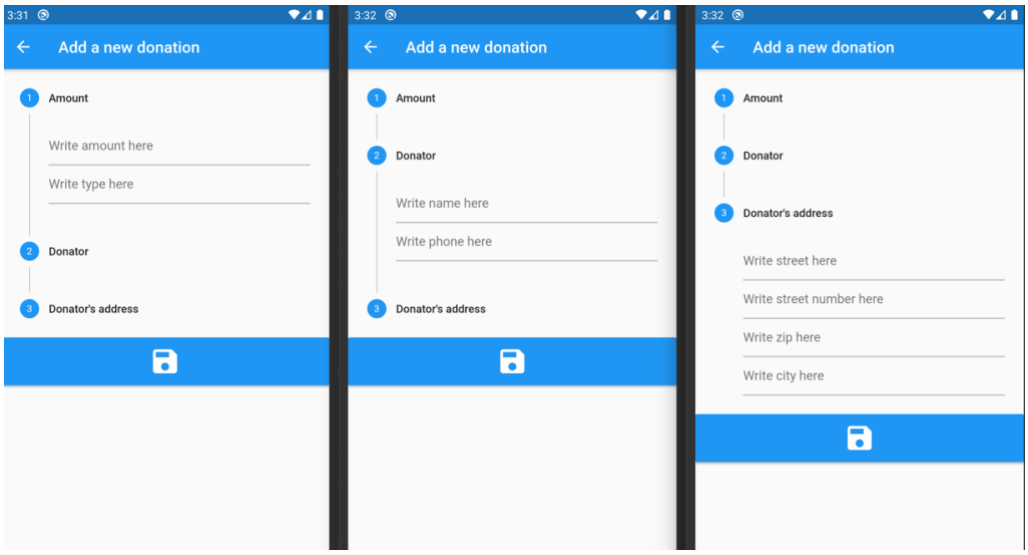


Figure 22 - New donation process

4 Testing of the application and requirements

This part of the project will discuss the testing process of this project and how it has or not met the criteria described at the beginning of this document.

4.1 Testing methodology

"Program testing can be used to show the presence of bugs, but never their absence" (E. W., O.-J., & C. A. R., 1972). Already 48 years ago, testing a software was a concern for developers. Software development tools have evolved at a rapid pace. When considering software testing, (Goodenough & Gerhart, 1977) were already aware of the central question developers should ask themselves when considering what a functional test is. The essential elements are the criteria on which the test are conducted. Based on those, you can determine if a test is adequate or not. (A. V. Hall, H. R. May, & Zhu, 1997)

However, the notion of adequacy can be challenging to define. Some sources describe it as

"what properties of a program must be exercised to constitute a 'thorough' test, i.e., one whose successful execution implies no errors on a tested program."

(Goodenough & Gerhart, 1977)

Reliability and validity of a test are the requirements for it to be considered an adequate test. The first notion requires the test always to produce consistent results. If you have different test sets that meet the same criteria, you should have the same test results. The second notion requires meaningful results. For every error possible in a program, it exists a test to reveal this error.

Following the previous requirements, the writer will implement a unit test for the main methods of the application. This low-level testing has been chosen because it enables the developer to test individual components separately. Furthermore, as Flutter is all about widgets, those tiny components can be classified, tested in groups or individually. Unit tests meet the requirements to prove that the application is doing what is supposed to do. If some errors occur, the developer can quickly find the widget responsible and make the changes needed.

The Flutter SDK also allows for more high-level testing such as widget test (also referred to in other frameworks as a component test). The writer also took some time to write widget tests. However, as this project has a limited amount of hours dedicated to it, the widget tests will not cover all the screen developed for this mobile application.

To resume the testing methodology, the writer chose to test the application by each function (with unit tests) and by each widget (with widget tests), assuring the consistency and reliability of the application.

4.2 Testing implementation

To analyse different types of testing, the writer chose to create three categories of requirements: confidence, maintenance, cost and execution speed.

4.2.1 Unit tests

Unit tests will not cost a lot of maintenance as they test small functions, methods or classes of the application. They are also the fastest way to run tests on a code to prove the functions created are correctly implemented and work as expected. However, as a developer, you need to specify each parameter you expect the function, method to handle. Unit tests bring low confidence for unexpected events or new errors. In particular condition, a mistake could not be spotted, and the developer would not be aware of it.
When using Flutter, all the developers needs to do to enable unit testing is to add the “test” package to the application’s project. This will provide the core functionalities.

```
dev_dependencies:
  flutter_test:
    sdk: flutter
```

Figure 23 - Test dependency with Flutter

The flutter SDK has a top-level “test” function. This function has at least two parameters: the name of the test and the parameters necessary for it to work. The SDK also provides an “expect” function. It accepts two parameters: the value calculated by your application and the results the developers excepts to have in this specific case.

```
test('Counter value should be incremented', () {
  final counter = Counter();

  counter.increment();

  expect(counter.value, 1);
});
```

Figure 24 - Unit test example (Flutter-dev, An introduction to unit testing, 2020)

This « expect » function will return a Boolean (true or false) depending on the comparison between the value of the two parameters. The downside of it is that it gives no other information. The developers have no other meaningful information if the calculated value is not the same as the expected one.

4.2.2 Widget tests

Each widget test will test one single widget. The purpose is to certify that the widget looks and interacts as the developers intended. In a bigger project, a widget test should receive and respond to actions or events initiated by the user. (Flutter-dev, Testing Flutter apps, 2020)

To implement widget testing within a flutter application, the same packages necessary for the unit tests are required. Those will have the following tools:

- The “WidgetTester” will create the environment in which tests will take place
- The function “testWidgets()” is used to create a new WidgetTester.
- The “Finder” classe enables the developer to search for a specific widget inside the test environment.
- To define what the finder needs to look for, the developer will use “Matcher” constants.

As this project has a limited amount of hours defined, the writer only implemented necessary widget tests. The following example describes a widget that is called. When the environment receives confirmation of the loading of the widget, the writer can search for a specific text. A button could contain a string that will be found if the widget behaves and loads correctly.

```
  testWidgets('MyWidget has a title and message', (WidgetTester tester) async {
    await tester.pumpWidget(MyWidget(title: 'T', message: 'M'));

    // Create the Finders.
    final titleFinder = find.text('T');
    final messageFinder = find.text('M');
  });
}
```

Figure 25 - Widget test example (Flutter-dev, An introduction to widget testing, 2020)

4.3 Testing limitations

The Flutter SDK is also built to conduct high-level testing such as integration tests. Those will allow developers to test their application as a whole. Such tests can spot performance problems. The creation of such tests is done as the following. The developer will code a set an action the simulation needs to do. Each action is done in order and logged. We could describe it as a virtual user that would perform the tasks automatically. Logging all actions is primary as we will be able to spot where errors came from and what was the set of actions that led to it. It increases the effort put into testing but also develop the feedback received from the application itself. To enable the integration testing, the writer could have added

“flutter_driver” dependency. However, as this project is limited in time, it was considered out of scope. (Flutter-dev, An introduction to unit testing, 2020)

Finally, when considering state of the art testing, developers should look after continuous integration services. Those will perform unit tests, widget tests and integration tests each time the code is saved and contains changes. Automating testing brings consistency to the development process, as every step is checked and validated before being added to the application. As stated above, the writer could not implement such services as this project is limited in time. However, as the commissioner expects quality and consistency from this project, the writer will develop such services shortly.

5 Conclusion

This concluding chapter will be divided into two parts. First, it will present the technical results and reflect on them. Then, the writer will analyse the process itself and describe the next steps regarding the product development.

5.1 Technical results

During this project, the writer used the Flutter SDK to develop a mobile application. The goal was to demonstrate the utility of cross-platform development for small businesses. Flutter has a lot of well-implemented capabilities. The developer can use pre-made widgets that will do exactly what they are meant to do. Their number is relatively limited.

Developers have to turn to third-party widgets to fulfil their needs. Using "pub.dev" packages, you can find a wide choice of widgets. However, you will find that not every package work on both iOS and Android. For some of them, you need to follow some instructions to put specific values into the Android or iOS code. An example would be a keyboard widget. As keyboards are not the same on iOS or on every brand that uses Android, you will need to use a custom keyboard. The trickiest part is that a "number" keyboard on iOS will allow decimal numbers where an Android device will not. In this regard, using a cross-development tool is counter-intuitive. Working on one base code should put away maintenance or the need for developers to look after each devices implementation particularities. Programmers will lose time to look after each environment separately.

Furthermore, when looking at Android, there is an annoying part to take into account that could not be avoided. During the time of this project's implementation, Android is migrating all his libraries to Android X. However, old libraries that worked on previous Android SDK versions will not necessarily work with Android X. The opposite is also exact. New libraries developed fully with Android X will not work when used with an old SDK. During this project, the writer was stuck with the following case. He used a widget to access the camera of the device and another to upload this image to Firebase. After extensive researches, he found that those were incompatible due to different SDK version on Android. However, there were both working on iOS devices.

To conclude this first part, we can say that Flutter comes with many advantages, such as gain of time or development resources. Nevertheless, it appears to the writer that cross-platform tools such as Flutter are challenging to maintain at a high level of efficiency. It is due to the extremely rapid development speed of native SDKs. Flutter developers will find themselves stuck with incompatibilities. Being able to have both iOS and Android items is a chance to work with but also a hassle to maintain or update with consistency.

5.2 Process results

Implementing a functional prototype in twelve weeks could be considered as challenging. However, the writer found that the process itself was well organised. The goal of this thesis was to develop an electronic subscription application using Flutter. Having to plan a theoretical framework made the implementation process much smoother. The writer took time to gather sources and pieces of information regarding Flutter and gain valuable knowledge while doing it. Those researches proved that developing a mobile application using Flutter for a small organisation such as the commissioner can save time and money. Furthermore, this project gave a framework for other businesses who are willing to develop a mobile application with few resources and time.

The current implementation of the application will be tested with real users during the next event organized by the commissioner. It will act as a stress test to prove that the functionalities implemented are useful and that the architecture put in place can handle more data than the samples created for this project. If this test is passed, the writer will bring this application to more brass bands or other similar businesses. The tests will grow bigger and bigger. When the application reaches 100 members, the writer will publish it on the market. He will also deploy it on the Apple Store and Google Play Store.

To conclude, this project has started a new product aiming to simplify the handling of events for small businesses. It will continue to grow and add new functionalities as the product is used by more and more users, each of them giving feedback and contributing to the development of this electronic subscription book.

6 References

A. V. Hall, P., H. R. May, J., & Zhu, H. (1997, 12). Software Unit Test Coverage and Adequacy. (A. C. Surveys, Ed.) Milton Keynes, UK.

Bloch, J. (2006). How to design a good API and why it matters. *OOPSLA*, 506-507.

Cuadrado, F., & Duenas, J. (2012, 11 01). Mobile Application Stores: Success Factors, Existing Approaches, and Future Developments.

E. W., D., O.-J., D., & C. A. R., H. (1972). *Structured Programming* (Vol. Notes on structured programming). Academic Press.

Evans, J. (2020, 02 05). *Swift is again replacing Objective-C, report claims.* Retrieved from Computer World: https://www.computerworld.com/article/3519437/swift-is-again-replacing-objective-c-report-claims.html

Flutter-dev. (2020, 04 20). *An introduction to integration testing*. Retrieved from Flutter: https://flutter.dev/docs/cookbook/testing/integration/introduction

Flutter-dev. (2020, 04 19). *An introduction to unit testing*. Retrieved from Flutter: https://flutter.dev/docs/cookbook/testing/unit/introduction

Flutter-dev. (2020, 04 20). *An introduction to widget testing*. Retrieved from Flutter: https://flutter.dev/docs/cookbook/testing/widget/introduction

Flutter-dev. (2020, 03 23). *Homepage.* Retrieved from Flutter: https://flutter.dev

Flutter-dev. (2020, 03 23). *Hot reload*. Retrieved from Flutter: https://flutter.dev/docs/development/tools/hot-reload

Flutter-dev. (2020, 03 24). *Platforms*. Retrieved from Dart: https://dart.dev/platforms

Flutter-dev. (2020, 04 20). *Testing Flutter apps*. Retrieved from Flutter: https://flutter.dev/docs/testing#recipes

Flutter-dev. (2020, 03 27). *Web support*. Retrieved from Flutter: https://flutter.dev/web

Flutter-dev. (2020, 03 23). *Widgets*. Retrieved from Flutter: https://flutter.dev/docs/development/ui/widgets-intro

Goodenough, J., & Gerhart, S. (1977). *Current Trends in Programming Methodology.* Englewood Cliffs.

Harnil, O. (2019, 01 19). *Front-End and Back-End Development: What Are The Differences?* Retrieved 04 2020, from Hyperlink InfoSystem: https://www.hyperlinkinfosystem.com/blog/front-end-and-back-end-development-what-are-the-differences

Heitkötter, H., Hanschke, S., & A. Majchrzak, T. (2012). Evaluating Cross-Platform Development Approaches for Mobile Applications. Münster, Germany.

Hook, B. (2005). *Write Portable Code : An Introduction to Developing Software for Multiple Platforms.* No Strach Press.

Lahudkar, P., Sawale, S., Deshmane, V., & Bharambe, K. (2018, 03 01). NoSQL Database - Google's Firebase: A Review. Chikhli, MH, India.

Mills, A. (2020, 03 23). *Hot-reloading in 2018*. Retrieved from Medium: https://medium.com/@the1mills/hot-reloading-with-react-requirejs-7b2aa6cb06e1

Pardis Pourghomi, Gheorghita Ghinea. (2012, 12 10). Managing NFC payment applications through cloud computing. London, England.

Peng, D., Cao, L., & Xu, W. (2011, 12). Using JSON for Data Exchanging in Web Service Applications. Shanghai, China.

Rebouças, M., Pinto, G., Ebert, F., Torres, W., Serebrenik, A., & Castor, F. (2016, 03 01). An Empirical Study on the Usage of the Swift Programming Language. Recife, Brazil.

Sandoval, K. (2020, 03 23). *What is the Difference Between an API and an SDK?* Retrieved from Nordic APIs: https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/

Statcounter. (2020, 3 3). *Mobile Operating System Market Share Worldwide - February 2020.* Retrieved from Statcounter GlobalStats: https://gs.statcounter.com/os-market-share/mobile/worldwide

Sturm, U., Schade, S., Ceccaroni, L., Gold, M., C. M. Kyba, C., Claramunt, B., . . . Luna, S. (2017, 10 10). Defining principles for mobile apps and platforms development in citizen science. Berlin, Germany.

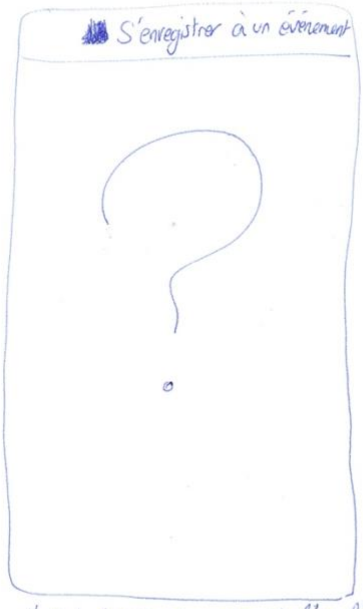
TIOBE Index for February 2020. (2020, 03 03). Retrieved from Tiobe: https://www.tiobe.com/tiobe-index/

Wenhao, W. (2018, 03 01). React Native vs Flutter, cross-platform mobile application frameworks.

Appendices

Appendix 1. Mockups from the commissioner

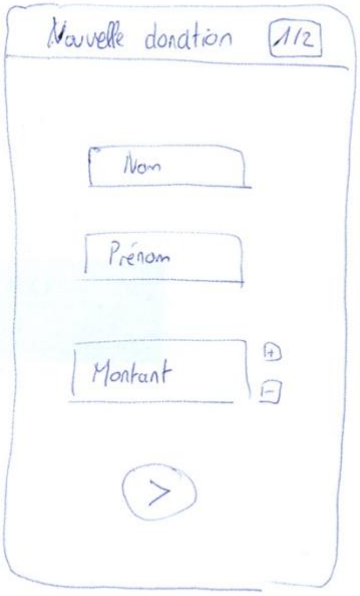
Partie Membre

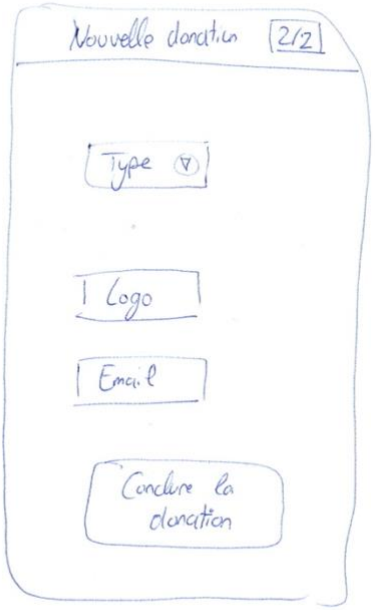


Il faut trouver un moyen de lier l'administrateur et le membre.

3

Partie Membre





Appendix 2. Interviews

As stated in the introduction of the thesis, the implementation process will be conducted using peer reviews.

The first interview will be held with Rita Moreira. She is also a student in IT (information technology) and has worked with Flutter for the same period of time as the writer. Her view on the project will be helpful as she is more focused on design aspects.

The second interview will feature Camille Oggier. He has worked for IBM for an extended period of time and is now a freelance developer. His view will mostly give contradiction as he is currently working with React, an alternative to Flutter.

Interview 1

Rita understands the need in the brass band community for this application. However, she asked the writer if it was planned to implement payment methods directly into the application.

The in-app payment method could be an exciting idea. It could increase the cash flow. By experience, most of the donations are made while being in a restaurant or a bar. People already have their credit card with them. It would also reduce the time needed to gather all the money. Since most of the member of the committee are volunteers, reducing the time required for administrative tasks is essential. Enabling such capabilities requires more information about NFC (Near Field Communication).

While based on the Bluetooth technology, NFC was built to carry sensitive information. (Pardis Pourghomi, Gheorghita Ghinea, 2012). It caused delay towards its adoption. Constructors have different policies regarding NFC. Android has a more approach. It allows for reading and writing using the NFC chip on the smartphone while Apple only grand access for reading. Taking those elements into consideration, it means that the writer would develop a different application for Android users. They would have the capabilities to use their phone a payment terminal.

The purpose of this project is to demonstrate the capabilities of Flutter to develop a mobile application at a lower cost and higher efficiency for the brass band community. Considering this, it would not make sense to promote any functionalities only for one platform. It would require extensive knowledge about Java. Furthermore, every aspect of dealing with in-app payments are far too advanced and complicated for the writer. Those elements contribute to the decision made by the writer and the commissioner not to include in-app payments inside the application, even for the foreseeable future.

Secondly, the interview has also brought to light the need for a simple login process. The application at this stage, during the first interview, only include a form to identify the user. It uses an email address and a password. Nevertheless, the writer will conduct some research

to integrate Google Sign-in within the application. This new functionality will ease the login process by eliminating the need for the user to create another specific account for this application.

Interview 2

Camille has a more critical view of the project. Apart from the project itself, he thinks that React native gives more tools to create better mobile applications.
His choice is motivated by the fact that React Native is more mature. As the technology is older than Flutter, the community had more time to build itself. As a result, coding with React Native as a beginner or a confirmed developer is improved by the community. Furthermore, the writer expressed some concerns about the lack of libraries for basic usages, such as a customised keyboard. It appears that React Native gives developers more choices regarding the amount of libraries available.
To conclude, he argues that React Native uses Javascript. This language is well known by the developers and does not require a lot of learning to get used to. In the context of this project, he finds attractive to test Flutter capabilities and push the envelope as much as possible to find value for this technology over React Native.