



Carolina Albuquerque BeingCare: Uma Janela Virtual de Comunicação para o Bem-estar

BeingCare: A Virtual Communication Window for Wellbeing



Universidade de Aveiro

2021

Carolina Albuquerque **BeingCare: Uma Janela Virtual de Comunicação para o Bem-estar**

BeingCare: A Virtual Communication Window for Wellbeing

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor José Maria Fernandes, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e da Doutora Ana Raquel Sebastião, Investigadora do Instituto de Engenharia Eletrónica e Telemática de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

**vogais / examiners
committee**

Prof. Doutor Sergi Bermúdez i Badia
Professor Associado da Universidade da Madeira

Prof. Doutor José Maria Amaral Fernandes
Professor Auxiliar da Universidade de Aveiro

agradecimentos / acknowledges

Com o desfecho do meu percurso académico não poderia deixar de agradecer àqueles que o marcaram positivamente.

O primeiro agradecimento é destinado à minha família, especialmente ao meu pai, à minha mãe e à Eduarda, não só pelo apoio durante os meus estudos, mas pelo investimento financeiro prestado ao longo destes últimos anos na minha formação.

Agradeço ao Hugo por todo o carinho, dedicação e amparo que me transmitiu nos melhores e nos piores dias.

Gostaria de expressar a minha gratidão por ter conhecido durante estes últimos anos académicos algumas das pessoas mais importantes que levo comigo para a vida. A elas, um agradecimento especial por toda a paciência e disponibilidade quer nos tempos de trabalho árduo quer nos momentos de descontração.

Por fim, o mais sincero agradecimento pelo voto de confiança e acompanhamento excepcional prestados pelo Professor José Maria e pela Doutora Raquel Sebastião durante o desenvolvimento deste trabalho. A orientação dedicada e a envolvimento deles tornou-se num factor chave para o sucesso deste trabalho.

palavras chave

Desenvolvimento multiplataforma, Flutter/Dart, aplicação móvel, suporte comunicação humana, classificação de áudio

resumo

O crescimento tecnológico permitiu aos utilizadores tirarem partido dos seus recursos em diversas áreas como a saúde, educação, política, entre outras. Contudo, a interação humana continua a apresentar alguns desafios, não só em cenários de isolamento e prestação de cuidados básicos, mas também em restrições temporárias humanas (incapacidade de falar ou mover), mesmo recorrendo à tecnologia.

O sistema BeingCare – uma prova de conceito baseada em Flutter – é uma janela virtual que ajuda a superar restrições de comunicação levantadas em cenários relacionados com o bem-estar, onde a interação direta entre humanos envolvidos pode ser prejudicada.

O sistema BeingCare foi desenvolvido em Dart (*backend*) e Flutter (*frontend*) tendo sido implantado com sucesso em iOS e Android com pequenas adaptações específicas da plataforma. A exploração da abordagem de *edge computing* para a classificação de áudio foi bem sucedida com o uso do Tensorflow Lite, que concede processamento local sem a necessidade de serviços externos e qualquer troca de dados de áudio – reforçando a privacidade e a troca de informações pessoais de acordo com as regras e mentalidade do RGPD. Embora inicialmente não planeado, o *background* do BeingCare foi inteiramente desenvolvido na linguagem Dart, sugerindo a sua adequação a alcances para além da *web* e do *mobile* em Flutter.

keywords

Cross-platform development, Flutter/Dart, mobile application, human communication support, audio classification

abstract

The powerful growth of technology granted users to take advantage of their resources in several areas such as health, education, politics, and others. However, human interaction continues poses several challenges, not only in isolation or care support scenarios, but also with human temporary constraints (inability to speak or move), even with the resource of technology.

The BeingCare system - a proof-of-concept system based on Flutter – is a virtual window helping surpass communications constraints raised in wellbeing-related scenarios where direct interaction between the humans involved may be hindered.

BeingCare was developed in Dart (backend) and Flutter (mobile and web front ends) and successfully deployed in iOS and Android with minor platform specific adaptations. Flutter enabled an easy integration of audio acquisition, processing, and classification. Edge computing approach for the audio classification was successfully addressed by using Tensorflow Lite, which grants local processing without the need of external services and any audio data exchange - enforcing the privacy and personal information data exchange in line with RGPD rules and mindset. Although not initially planned, the BeingCare back office was based entirely on the Dart language suggesting its adequacy to scopes beyond web/mobile using Flutter.

List of Contents

List of Contents.....	i
List of Figures.....	v
List of Tables.....	vii
Glossary.....	ix
Chapter 1 Introduction.....	1
1 Motivation.....	1
2 Objectives.....	2
3 Contributions.....	3
4 Document Structure.....	3
Chapter 2 State of Art.....	5
1 Human Interaction and Communication.....	5
1.1 Monitoring to Support Communication.....	6
1.2 Audio Detection.....	6
1.3 Technology Advances in Monitoring.....	8
1.4 Technology Advances in Distance Communication.....	9
2 The Impact and Improvements of Mobile Devices.....	10
2.1 Mobile Development.....	10
2.2 Native Development.....	11
2.3 Cross-Platform Development.....	12
3 Machine Learning on Mobile.....	15
3.1 ML Mobile Frameworks Runtimes.....	17
3.2 Solutions Using Machine Learning Frameworks on Mobile.....	19
4 Application Backend.....	20
4.1 Java Enterprise Application Frameworks.....	22
4.2 The Dart from Behind the Flutter.....	23
Chapter 3 System Specification.....	27

1	Description	27
2	The Information	30
2.1	Events and Data Exchanged	31
Chapter 4	Implementation	35
1	System Architecture	35
2	BeingCare Mobile Component.....	37
2.1	Audio Processing and Classification Unit	39
2.2	Speech to Text Unit.....	42
3	BeingCare Service Component	44
3.1	Services and Controllers.....	46
3.2	Authentication and Authorization.....	49
3.3	Messaging: Consume and Publish.....	51
3.4	API Documentation	52
4	BeingCare Web Application Component.....	54
4.1	BloC Pattern in Navigation	55
4.2	Consuming Information	57
5	Messaging Runtime	57
5.1	MQTT Protocol	57
5.2	Mosquitto Message Broker.....	58
6	Deployment.....	60
7	Reflections and Considerations on Development	61
7.1	Dart as Applicational Backend Programming Language.....	62
7.2	Flutter in Web	63
7.3	Tensorflow and Physical Devices.....	64
7.4	Operating Systems	64
Chapter 5	Results Analysis.....	67
1	Mobile Application Demonstration	67
2	Web Application Demonstration	69
3	Mobile Application Size Analysis.....	70
3.1	Releases	71

4	Audio Classification Analysis	73
4.1	Model Overview	73
4.2	The Distance's Impact	74
4.3	Input Parameters	79
Chapter 6	Final Remarks.....	81
1	Conclusions	81
2	Future Work	83
	References.....	85
Appendix A	Flutter/Dart Dependencies	91
Appendix B	Database Schema.....	93
Appendix C	Docker Specifications.....	95

List of Figures

Figure 1 - Mobile framework purposed in a healthcare scenario [9].....	7
Figure 2 - Voice pathology detection method purposed in [10]	8
Figure 3 - Mobile sensors used to monitor health issues [4]	9
Figure 4 - Most usable cross-platform mobile frameworks by software developers in 2020, according to [18]	13
Figure 5 - Cross-platform mobile frameworks used by software developers in 2019 and 2020 according to [19].	13
Figure 6 - Mobile apps revenues since 2014 and projections until 2023 according to iResearch survey [25]	16
Figure 7 - Percentage of Flutter users positively satisfied with Dart during the first quarter of 2021 (Adapted from [36])	24
Figure 8 - BeingCare logical architecture	28
Figure 9 – Technical architecture diagram.....	36
Figure 10 - Sequence diagram of the flow of the information exchanged between components	37
Figure 11 – Tensorflow Lite model generated using Google Teachable Machine	40
Figure 12 – The state management problem [49]	43
Figure 13 – Authentication process returning a JWT for users’ authorization when requests API.....	50
Figure 14 – BloC pattern communication [49].....	55
Figure 15 – MQTT publish-subscribe architecture [53]	58
Figure 16 – Deployment diagram	60
Figure 17 – Main screen on Android devices.....	68
Figure 18 – History of sent messages on Android devices.....	68
Figure 19 – Speech to text screen on Android devices.....	68
Figure 20 - Main screen on iOS devices	68
Figure 21 - History of sent messages on iOS devices.....	68
Figure 22 - Speech to text screen on iOS devices	68
Figure 23 – Web application homepage	69
Figure 24 – People being cared list and individual overview of events that occurred.....	70

Figure 25 – History board of all information organized by type 70

Figure 26 - Visual comparison of the native applications' size between operating systems 73

Figure 27 – Evolution of model accuracy (left) and model loss (right) depending on the epochs elapsed 74

Figure 28 – Confusion matrix obtained from 40 inferences with distances of 0.5 meters (top) and 1 meter (bottom) tested with random samples used to build and train the classification model 76

List of Tables

Table 1 - Categorization of Machine Learning algorithms	17
Table 2 - List of Machine Learning frameworks to integrate models in mobile applications	19
Table 3 - Most used backend frameworks by programming language	22
Table 4 - Comparison between backend frameworks used for enterprise applications development	23
Table 5 - Comparison between backend frameworks based on Dart language	25
Table 6 – Personas of the system	29
Table 7- Description of defined stories specifying the system	30
Table 8 - Types of messages shared through the system and respective events associated	31
Table 9 – Packages used as support to mobile application development and features	39
Table 10 – Description of each package imported	45
Table 11 – Description of services implemented.....	43
Table 12 – Annotation’s overview used on each controller.....	47
Table 13 – API documentation summary.....	53
Table 14 – Brief description of packages imported to support the web development	55
Table 15 – Description of used message topics.....	59
Table 16 – Ports and protocols available for each message topic	60
Table 17 – Comparison between Aqueduct and Jaguar frameworks in terms of features provided and issues encountered	63
Table 18 – Comparison APK sizes between releases generated for all the targets ABIs.....	71
Table 19 – Android and iOS native applications’ size comparison	72
Table 20 – Classification report of performance metrics calculated per class using the random samples testing approach.....	78
Table 21 - Audio recording temporal window (in seconds) to be inferred on Android devices depending on sample rate and buffer size passed as input parameters.....	80
Table 22 - Audio recording temporal window (in seconds) to be inferred on iOS devices depending on sample rate and buffer size passed as input parameters	80

Glossary

ABI	Application Binary Interface
API	Application Programming Interface
APK	Android Package
CORS	Cross-Origin Resources Sharing
COVID	Corona Virus Disease
EGG	Electroglottographic
GMM	Gaussian Mixture Model
GTM	Google Teachable Machine
HTTP	Hypertext Transfer Protocol
iOS	iPhone Operating System
IoT	Internet of Things
IPA	iOS App Store Package
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
KMM	Kotlin Multiplatform Mobile
ML	Machine Learning
MQTT	Messaging Queueing Telemetry Transport
MVC	Model-View-Controller
ORM	Object-Relational Mapping
PPE	Personal Protective Equipment
REST	Representational State Transfer
SQL	Structured Query Language
UI	User Interface
XML	eXtensible Markup Language

Chapter 1 Introduction

This chapter presents the motivation and objectives that lead to the main contribution of the present dissertation, the BeingCare, a proof-of-concept system based on Flutter for helping surpass communications constraints raised in wellbeing-related scenarios where direct interaction between the humans involved may be hindered.

1 Motivation

Technological advances have granted users to take advantage of their resources in several areas such as health, education, politics, and others. However, human interaction poses several challenges when direct interaction is not possible, even with the resource of technology. This happens not only in isolation scenarios but also with human temporary constraints, such as the inability to speak or move.

Some examples can be found in health-related contexts. For instance, in hospital services, nursing homes or in a care support scenario, people are bedridden (or not) and sometimes with respiratory limitations or with aphasia, having clear communication constraints regardless of the absence of any reasoning disability. In these cases, where exist communication problems or it is not possible to direct interpersonal contact, tasks like asking or calling for someone may be cumbersome as well as the report of their pain status or other problems related with wellbeing, increasing the difficulty of understanding people.

The recent world pandemic, COVID-19, also provides another example where people being cared have limited ability to speak, either due to auxiliar respiratory devices or due isolation

scenarios. Moreover, communication between people being cared and caregivers becomes more limited, complex, and harder due to the use of uncomfortable personal protective equipment (PPE).

Accordingly, the need to create a system capable of overcoming communications restrictions between people arises in order to facilitate the interaction and the clarity of the ideas transmitted. The impact of the constraints mentioned could be reduced by improving communication between people with the support of monitoring in order to get clearer information about the condition of people being cared for caregivers. Further, the use of mobile devices coupled with this may turn the interaction easier and accessible anywhere. Therefore, it is useful to identify events using audio classification while monitoring and some events e.g. non-critical demanded by people being cared. It is also relevant to identify some events such as cough, sneeze, demands for help, or other types of requests. On the other hand, the caregiver's teams access to the events should be easy and namely as real-time notifications, especially to alarms and requests.

2 Objectives

The main objective of this dissertation is to explore the use of the Flutter, a Dart cross-platform application development solution for developing BeingCare, a proof-of-concept system based on Flutter for helping surpass communications constraints raised in wellbeing-related scenarios where direct interaction between the humans involved may be hindered (i.e., people may be bedridden or isolated) while addressing privacy concerns related with the people being cared state.

Is aimed to achieve through the BeingCare system the support implicit and explicit communication and should provide:

- Detection of some critical events related with the current state of people being cared, addressing verbal communication using audio detection techniques with the support of monitoring
- Non-critical information sharing on demand by people being cared
- Report of critical triggered events to caregivers as well as providing other relevant information related to the current state of people being cared

BeingCare will allow addressing Flutter abilities, not only to explore data collection and audio processing in mobile, but also eventual integration limitations with enterprise solutions e.g., messaging, REST endpoints, and others.

3 Contributions

The main contribution of this dissertation was the BeingCare system developed in Dart (backend) and Flutter (both mobile and web frontends). The BeingCare mobile system was developed in Flutter and successfully deployed in iOS and Android, with minor platform specific adaptations. Flutter enable:

- easy integration of audio acquisition, processing, and classification
- addressing an edge computing approach for the audio classification using Tensorflow Lite

The use of Tensorflow Lite grants local processing without the need of external processing and any audio data exchange - enforcing the privacy and personal information data exchange in line with RGPD rules and mindset.

Although not initially planned, the BeingCare back office was based entirely on the Dart language, suggesting its adequacy to other scopes beyond the web/mobile using Flutter.

4 Document Structure

Chapter 1 presents a brief introduction and the motivation of the problem as well as the objectives defined and contributions.

Chapter 2, the State of Art, addresses the existing general solutions to culminate the problems raised and explains the current technological trends used in the context.

Chapter 3 describes the specification of the system and its minimum requirements to develop a proof-of-concept.

Chapter 4 illustrates a technical architecture and the implementation details of its components, along with some technical decisions. Some technological considerations about the work developed are also addressed.

Chapter 5 presents an overview of the implemented user interfaces and a detailed discussion of the results obtained.

Chapter 6 analyses the objectives and the appropriate approaches to achieve them. It also discusses the results of this work and presents interesting suggestions for future work.

Chapter 2 State of Art

In the present chapter are introduced approaches of human interaction and the impact of mobile technology interaction during communication. A review of cross-platform frameworks for mobile development is presented as well as trended backend frameworks to support applicational back office and useful Machine Learning (ML) frameworks for mobile applications are discussed.

1 Human Interaction and Communication

Communication consists in transmitting information from one person or group to another and can be done through verbal or non-verbal interaction. While verbal communication consists of the use of words (such as speech, texts, emails, etc...) in order to transmit a message, non-verbal communication uses gestures, and body language for this purpose [1].

Verbal communication includes spoken and written communication. Spoken communication consists of transmitting messages using speech while written communication uses writing texts.

Body language includes not only facial expressions but also physical gestures and movements. Some specialists in this area argue that nonverbal communication supports verbal communication helping express ideas and feelings [2]. Nonverbal communication, usually, is explored and used by deaf or speech-impaired communities.

In social distance or isolation scenarios, communication between people becomes harder due to physical distance. In such case, the use of some gadgets and new technological trends may become an interesting solution to exceed these challenges in both ways of communication [3].

1.1 Monitoring to Support Communication

Monitoring can be a solution to bypass barriers in communication between persons. It allows observing changes in individuals namely in vital signs, actions, and movements that can be used to adapt and improve the communication process, for example adapt the volume of speaker to environment noise. It can also provide a better characterization and person's classification related with patterns that could add an insight on their own mental state. For instance, in health area, a frequent or continuous monitoring allows early diagnosis of pathologies and detection of risk situations [4].

Monitoring individuals can be based on data collection by invasive or non-invasive technics.

Invasive monitoring solutions are, most of the times, the reference monitoring gold standard in several areas such as healthcare and sports area. But their downside is that they imply special conditions of monitoring and can represent a distress constraint on the individual being measured. A good example is measure blood pressure using the insertion of a catheter into a suitable artery to display in a monitor the corresponding pressure wave [5], implying operating rooms level collection conditions or intensive care units.

Monitoring people through non-invasive techniques consists of analyzing information that is not retrieved directly from body incisions. For example, measure blood pressure invasively could be replaced by the use of a less invasive use of an upper arm cuff [6].

Currently, with the exponential growth of social networks the potential of continuous "social" monitoring people has brought the potential of ongoing analysis persons' behaviors namely on the patterns of consumed contents or time spent on it [7].

1.2 Audio Detection

The voice plays an essential role in verbal communication between people. This means of communication often allows the transmission of clearer and more explanatory ideas compared to textual communication. Technological advances have allowed new investigations in the area of sound detection applied in several

areas. For instance, audio detection and their classification have not only enhanced video surveillance systems [8] as well as the early detection of pathologies [9][10].

In the healthcare area, a study purposes voice pathology detection using Machine Learning (ML) techniques, improving the accuracy of a healthcare system, and combining it with mobile technology and sensors accessible via Bluetooth [9]. These mobile sensors are used to capture sound signals from people being cared and are sent to a local machine to be, subsequently, stored in a cloud. The cloud contains servers responsible for Deep Learning processing and its result is sent to caregivers with some samples to validate the decision. Figure 1 describes the purposed system architecture.

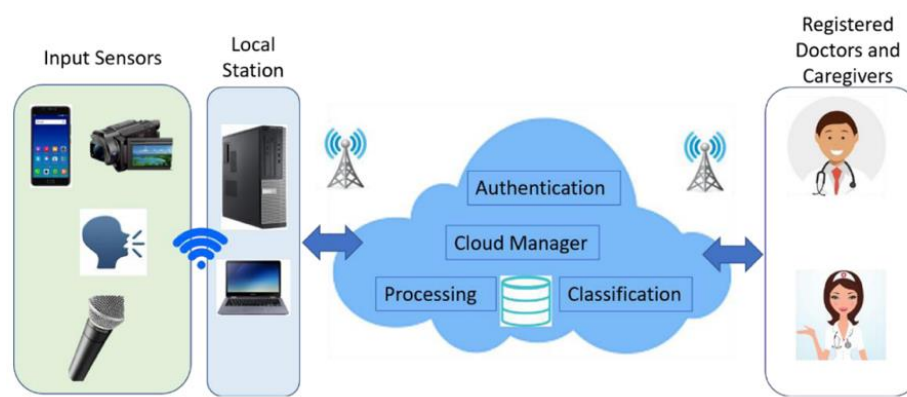


Figure 1 - Mobile framework purposed in a healthcare scenario [9]

The voice pathology detection system uses as input audio samples with one second duration. If this duration is longer than one second, the sample is cut. After that, samples are divided into overlapping frames and a Fourier transform is applied to each one in order to convert the signal into a frequency-domain signal. All these frequency-domain signals are concatenated to obtain a spectrogram filtered by twenty bandpass filters that will be the input of two Convolutional Neural Network (CNN) models investigated.

A disadvantage of the framework previously referred is the entire audio processing and classification which is carried out in a cloud, exposing the people being cared voice. A local computing and classification approach (edge computing) can overcome this disadvantage by guaranteeing people being cared privacy, as only a final decision will be shared out of the mobile device, as well as the samples to support it instead of the entire audio captured.

Another method purposed is voice pathology detection applied to the smart monitoring of healthcare in smart cities [10]. In this method, the voice signal captured by smartphones or recording devices and an electroglottographic (EGG) signal are transferred to a cloud to be processed and classified. Firstly, the noise ratio of both signals is decreased in order to improve pathology detection. Both signals are explored to extract from the respective spectrogram some features, and, after, it is used Gaussian mixture models (GMMs) as modeling process, as seen in Figure 2.

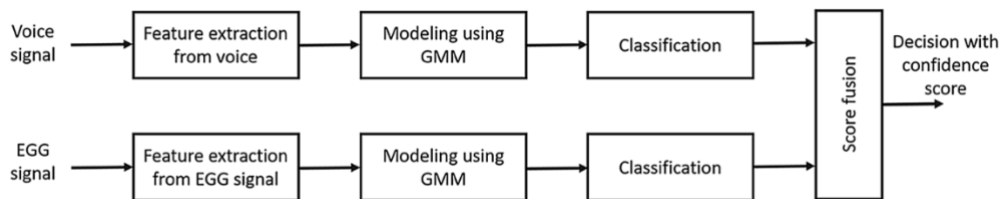


Figure 2 - Voice pathology detection method purposed in [10]

Similar to the system previously proposed, this one uses a cloud for audio processing and classification. As mentioned earlier, this fact becomes a disadvantage related to user’s privacy due to the samples of their voice are exposed outside smartphones when used for this purpose.

1.3 Technology Advances in Monitoring

With the technological advances over decades, there were developed external sensors capable of measuring vital signals [11] and recognizing a different kind of activities. Moreover, there were also explored existing sensors of mobile devices in order to track and monitor people non-invasively.

External wearable sensors are wearable objects designed to monitor several health-related issues and to provide, wirelessly, relevant data to a control device for wellbeing purposes. Smartwatches, wristbands, and body jackets are interesting examples of the most usable wearable sensors.

The evolution of mobile phones for smartphones allowed the inclusion of several sensors incorporated in a device. Although some mobile phones already had cameras and microphone, smartphones have brought new sensors such as the accelerometer, gyroscope, and GPS. These sensors are less expensive compared to wearable sensors but less accurate.

Smartphone sensors have become useful to monitor people and detect health issues. Regarding the monitoring of health issues related with heart, eyes, skin, and physical activity, several developments using smartphones’ sensors were explored before [4]. Figure 3 resumes which mobile sensors are typically used to address each issue mentioned.

Monitored Health Issues	Typically Used Smartphone Sensors
Cardiovascular activity e.g., heart rate (HR) and HR variability (HRV)	Image sensor (camera), microphone
Eye health	Image sensor (camera)
Respiratory and lung health	Image sensor (camera), microphone
Skin health	Image sensor camera)
Daily activity and fall	Motion sensors (accelerometer, gyroscope, proximity sensor), Global positioning system (GPS)
Sleep	Motion sensors (accelerometer, gyroscope)
Ear health	Microphone
Cognitive function and mental health	Motion sensors (accelerometer, gyroscope), camera, light sensor, GPS

Figure 3 - Mobile sensors used to monitor health issues [4]

1.4 Technology Advances in Distance Communication

As mentioned before, the use of technology devices may contribute to improving the communication between people. In the past, communicating at a distance was possible through smoke signals or Morse Code after the evolution of radio waves. Later, with the emergence of mobile devices and the development of new connectable gadgets, distance communication has improved connecting people simpler and faster.

Smartphones or computers allow to make video-calls, send messages, or even share voice audios in chats. Distance non-verbal communication is also possible due to the gadget’s interaction using gestures to produce actions. Several handsfree systems allow interacting with systems or other devices without touching them, using gestures or voice commands [12] for a purpose. Amazon Alexa, Google Assistant, and Siri can remotely control not only personal smartphones but also other home devices using voice commands, such as send messages, call someone, or turn on/off TV, lights, and air conditioning.

In fact, the mentioned solutions can improve distance communication between people. Moreover, people with physical problems (e.g.: bedridden people) may not be able to interact directly with these gadgets. Therefore, voice commands to communicate with the other side, or even ask for help, can be crucial to improve communication in this specific scenario.

2 The Impact and Improvements of Mobile Devices

Since the invention of the mobile phone, these devices have transformed dramatically over the last few years. Mobile phones were created to allow distance communication between people through calls, and, later, short-message-service (SMS). At the beginning of the current century, mobile phones started to take advantage of accessing information over wireless networks and started to integrate cameras. Since these improvements, a mobile revolution has emerged, allowing not only the advancement of sensors and technologies already integrated, but also granted the integration of new sensors. Nowadays, these devices have become fundamental to people's life becoming almost an essential asset.

Addressing mobile technology is an important objective of this work as part of the solution due to its ease of handling and its integrated components and sensors. The development of a mobile application as a starting point for communication is an asset to the system and should be compatible with different mobile operating systems.

2.1 Mobile Development

Over the years, the evolution of mobile devices and the easy access to them have contributed to a burst in the development of mobile applications.

In the beginning, applications were only developed in accordance with the native approach. With the evolution of the capacities and functionalities of the devices, as well as the appearance of multiple operating systems, there felt a need to change the development process in order to correspond to the perspectives of the developers, focusing on temporal effectiveness. Thus, the development of cross-platform applications arose [13].

In addition, modern mobile users already expect a user experience more simple, intelligent, and functional, and several mobile applications have started to integrate Machine Learning (ML) mechanisms to recognize, classify or even suggest content to the users. Peter Flach defined ML as "the systematic study of algorithms and systems that improve their knowledge or performance with experience" [14].

Indeed, learning requires data and information, already collected or not, and related to the target of the problem. The problems associated with ML usually are classification, regression, or even scoring [15], and its intent is to find the best connection between the base of knowledge and the target required. ML is frequently applied by mobile developers for data mining, tracking, monitoring, and search. It is possible to integrate ML models into mobile applications easily using frameworks for this purpose or developing algorithms.

During the last decade, computing paradigms have been a significant evolution affecting mobile application development. Despite the cloud computing concept has revolutionized the development of internet-based services, there are limitations that it cannot address, such as bandwidth, latency, and connectivity requirements. With the revolution of Internet-of-Things (IoT), new perspectives to lead with these challenges emerged, arising the edge computing approach [16].

There are two main approaches to mobile application development: one focuses on native development for a single operating system, and the other focuses on cross-platform development. The latter seems to be the most suitable for the proposed system since it may be able to cover not only different mobile operating systems but also allow other types of platforms, namely web.

2.2 Native Development

Native applications are written and designed for a particular operating system. In native development, for each operating system is needed a version of a native application. This becomes development more stable, with a seamless performance once the application is developed for the specific devices operating system. Also, native apps are more consistent with user interface (UI) components of the devices, since are built with technologies used by the device's manufacturer.

However, when the target consists of multiple platforms, it will be necessary to develop two or more separated apps and each operating system has its own technologies and software development kit (SDK). In addition, it is impossible to share code between separated applications, affecting the development time and test process because there are needed two or more separated codebases to build and test [13].

The Android operating system is open-source, and its development is supported by Google. As this operating system is open-source, it is used by several companies on their devices, such as Samsung, LG,

and HTC. Native Android applications are written in Java or Kotlin, recently considered the official programming language of Android.

The iOS operating system is supported by Apple and is used on iPhones and iPads devices. All iOS native applications are written in Swift or Objective-C.

2.3 Cross-Platform Development

Built mobile applications that are compatible with several operating systems is ensured by cross-platform applications development approach. Although native applications are developed through their native operating environment, cross-platform applications development frameworks use a single tool to ensure multi-operating systems compatibility.

As mentioned before, in native development, each operating system needs a version of a native application, and developing for devices with different operating systems may cause a higher budget because it requires a larger team to develop the different versions.

Therefore, cross-platform frameworks are powerful in producing versions both for iOS and Android systems using a single-engine (easy portability), presenting low-cost development [17]. On the other hand, cross-platform applications are bigger than native applications due to the fact of operating systems do not able to run the official code language of these applications. So, it is necessary to include libraries to support this code language increasing the size of applications.

During the last years, cross-platform development frameworks gain popularity inside the mobile development world. The recent JetBrains survey [18] that gathers feedback from about 20 thousand developers shows the last trends around tools, technologies, and programming languages, revealing that 45% of developers develop applications both for Android and iOS systems. Figure 4 indicates which cross-platform mobile frameworks are most used by the developers surveyed in 2020 and Figure 5 provides a comparison with the previous year.

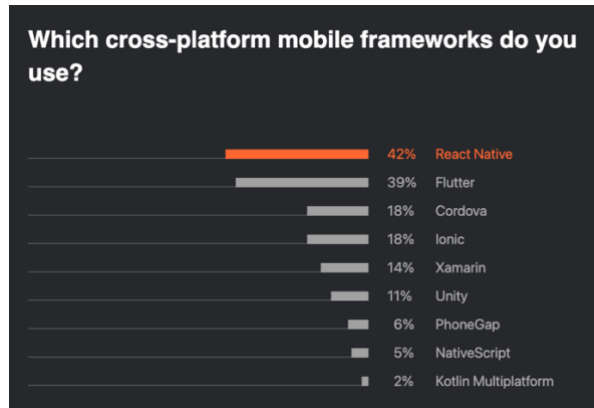


Figure 4 - Most usable cross-platform mobile frameworks by software developers in 2020, according to [18]

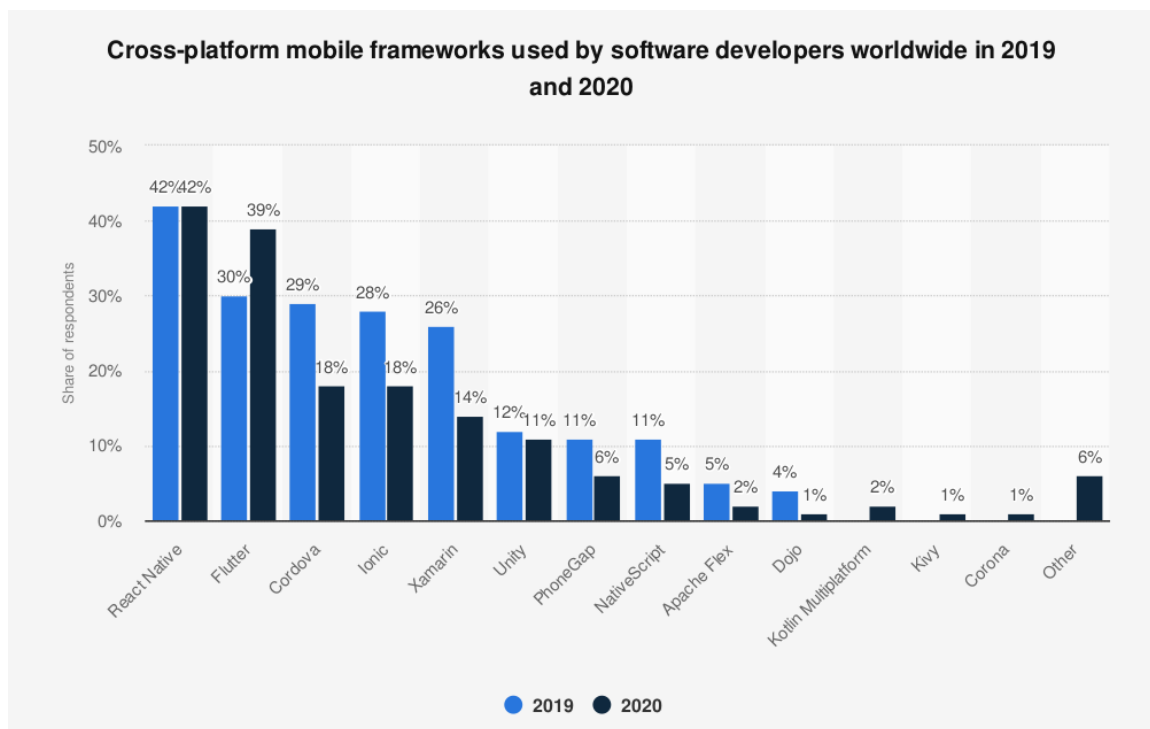


Figure 5 - Cross-platform mobile frameworks used by software developers in 2019 and 2020 according to [19]

2.3.1 React Native

React Native, created by Facebook in 2015, is an open-source cross-platform development framework based on JavaScript language, which is a wildly popular language. Due to its portability, fast development,

and more efficient sharing code between operating systems (specially iOS and Android), React Native has become the most popular framework to build cross-platform applications [20].

This framework has as main functionality the Fast Refresh, allowing that an application in development could still be running on the device even when code updates are made. This feature revolutionized the optimization of application development and testing time.

However, React Native applications are not completely native and despite having a very similar behavior and performance to the native one, these are not the same. Also, as mentioned above and as other cross-platform applications, React Native applications are bigger than native applications.

Besides being an open-source project and not being totally stable, large companies such as Uber, Instagram, Airbnb, and Tesla have already developed their applications based on React Native, supporting the promising future of this framework.

2.3.2 Kotlin Multiplatform Mobile (KMM)

Kotlin Multiplatform Mobile (KMM) is a development kit developed by JetBrains for cross-platform mobile development based on Kotlin language. Kotlin is an open-source, static and the official Android programming language. Oriented-object programming, interoperability, and safety are some advantages of this language [21].

Similar to the other cross-platform solutions, KMM allows not only the use of a single codebase for both Android and iOS applications, but also uses the multiplatform capabilities of Kotlin. KMM is still very recent in the mobile development world, however, due to its features and the base language, it is predicted that it may have an eminent growth in the next years [22].

Besides KMM documentation mentioning the possibility of developing a full-stack web application using Kotlin, some projects mentioned on Kotlin hands-on course program use React to support the web component of Kotlin applications [23]. Hence, web support in KMM cannot be confirmed.

2.3.3 Flutter

Flutter, an open-source mobile development framework provided by Google, concedes developing cross-platform applications for mobile, web, and desktop (in development stage) using only a single source code and natively. The use of a unique codebase makes the development, debug, and update processes

easier since there is only a unique place to make changes and debugging [24]. Consequently, the time needed to build an app decreases significantly compared to native development.

Based on Dart Language, an oriented-object programming and fast language, Flutter architecture relies on reactive programming. Reactive programming deals with asynchronous data streams and the specific propagation of change. This programming paradigm increases the performance once handles huge volumes of data in a quick and stable way, improves user experience due to keeping the app more responsive, and simplifies updates.

Similar to Fast Refresh of React Native, Flutter also provides Hot Reload functionality offering a more dynamic and faster development. As mentioned above, this feature is important for mobile developers and helps them to fix bugs and experiment new ideas quickly.

Flutter also has its disadvantages. As it is a new framework in growth, compared to React Native, it is less established and some libraries still not rich as native development. The application's size is affected especially when applications are written in Flutter. Flutter applications are bigger than native applications. Besides this, and as mentioned previously, Flutter's interest has been increased significantly.

Flutter also supports web development through different scenarios such as building Progressive Web Applications (PWA), single-page applications, and from existing mobile applications. Flutter web supports development compatible with modern browsers like Chrome, Safari, Edge, and Firefox. Flutter web works identical to Flutter in mobile however to generate a build release, this framework uses dart2js compiler to convert Dart code into HTML and CSS, generating a single JavaScript file and populates a specific directory with web files including assets.

According to the JetBrains survey [18] presented in Figure 5, Flutter's popularity has increased 9% while other frameworks, such as Ionic and Cordova, decreased their popularity. Companies like Ali Baba, BMW, and even Google themselves are already using Flutter in their production apps right now [24].

3 Machine Learning on Mobile

The growth of mobile technologies boosted the growth of the mobile applications market. According to a survey performed by iResearch, in 2023 mobile application revenues are expected to be almost double the revenue obtained in 2019, as seen in Figure 6.

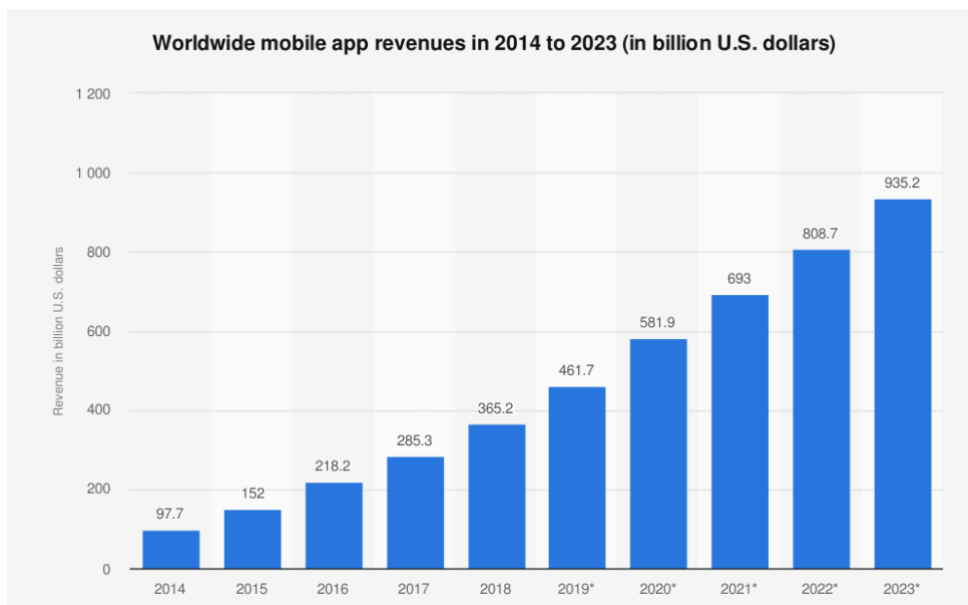


Figure 6 - Mobile apps revenues since 2014 and projections until 2023 according to iResearch survey [25]

These impressive numbers demonstrate that the mobile applications market tends to need increasingly creative and innovative applications. Some innovative features that have been incorporated into mobile applications are related to ML. The integration of intelligent virtual assistants, the connections of smartphones and IoT devices, and the development of voice user interfaces are some classic features included in modern mobile applications [26].

There are four different approaches to defining a ML algorithm for iOS and Android applications explained in Table 1.

Supervised Learning	This approach uses labeled or marked data as input and expected results. The learning algorithm, trained with the input data, intends to output results matching with the expected results. These are commonly used in healthcare applications, using historical data to predict future events.
Unsupervised Learning	As data is not labeled, in this approach algorithms explore the input data to describe or reveal hidden patterns. Commonly used in e-commerce applications and recommended systems.

Semi-supervised Learning	Match between supervised and unsupervised learning using labeled and non-labeled data, usually, a greater amount of unlabeled data when compared to labeled data. Commonly used in applications that process text, audio, or image.
Reinforcement Learning	Through interaction with the environment, this learning method finds an optimal strategy in accordance with actions and parameters, errors or rewards, defined. Commonly used in robots that need to learn all obstacles around.

Table 1 - Categorization of Machine Learning algorithms (adapted from [26])

ML mechanisms allow companies to predict user behavior, helping to plan and implement more effective marketing campaigns, personalized content, improve clients' engagement and increase revenues. For these reasons, ML is integrated into mobile applications to create personalized content for each user and helps in performing predictive analytics [27]. Data mining, tracking, data analysis, and monitoring problems are some examples that benefit from the integration of ML techniques into mobile applications.

Google recently launched the Teachable Machine (GTM) online platform that allows a fast and an easy creation of trained classification models of three main types: image, audio or poses detection models. In this platform it is possible to gather and group examples into classes or categories labeled, and train the model created in order to use online through a sharable link or offline exporting the model in several Tensorflow formats, addressed in the next section.

3.1 ML Mobile Frameworks Runtimes

The integration of ML procedures in mobile applications can be performed by frameworks capable of integrating models. Table 2 summarizes a review about frameworks capable of integrating ML mechanisms in mobile applications.



Tensorflow allows an easy model building, robust ML production, and powerful experimentation for research. Tensorflow Lite is a lightweight version of Tensorflow developed specifically for mobile and embedded devices. It allows local data process, avoiding data sharing with external servers. The size of mobile applications using Tensorflow Lite is heavy since the models are also stored locally. Tensorflow provides a repository of trained models shared with the community through Tensorflow Hub project.

Android
iOS
Linux



ML Kit is a powerful and an easy library that brings ML experience to mobile devices. It allows Android and iOS applications to be personalized based on users' interactions and, at the same time, it is optimized to run in mobile devices. All data processing is in local devices increasing the speed. In addition, allows processing in real-time such as video and image processing. ML Kit provides several projects grouped by vision and natural language categories.

Android
iOS



Firebase ML stores models into servers that are distributed to the users. All information processing is online, and the fact of models being stored outside the devices reduces the size of applications and allows model updates easily without new applications releases.

Android
iOS



Framework to develop models for mobile applications and manage them in a production environment. This framework supports integration with cloud providers such as AWS, Azure, and IBM, or environment in order to organize data and train models [28]. This framework

iOS


	supports both Core ML and Tensorflow Lite models and allows models updating remotely.	
	End-to-end solution for the implementation of Artificial Intelligent (AI) features into mobile applications. It also provides an extra real-time dashboard to monitor the performance of models executing in devices [29]. This framework provides a set of pre-trained models and a development platform to generate, collect, label datasets, and train models ready to use in mobile with writing code. Moreover, it supports Tensorflow, Tensorflow Lite, and Core ML models [28].	Android iOS
	Numerical framework is used commonly for data science purposes and focuses on optimization and delivery of models. Also, it provides performance monitoring tools measuring on-device speed and power used during computations [28] [30].	Android

Table 2 - List of Machine Learning frameworks to integrate models in mobile applications

3.2 Solutions Using Machine Learning Frameworks on Mobile

Recently, the investigation around healthcare monitoring systems has increased due to their contribution in detecting pathologies and chronic diseases in a timely manner. A study presented in [31] addresses cough detection using different mobile devices to collect audio input for classification models. However, audio inputs were processed outside mobile devices, i.e., these devices only were used for data collection. In this study, the approaches to classify data were implemented using Tensorflow framework to train the neural networks.

In [32], approaches related to the detection of human health-related actions (e.g. fall, stomachache, nausea, sneeze) using an Android camera for video processing and using Tensorflow Object Detection API were addressed. This work contributes to monitoring children, elderly, and people with special needs

when their caregivers are absent. There were explored two different approaches, one using Tensorflow Object Detection in a mobile device and the other using the Tensorflow Object Detection Notebook technique. As result, the researchers concluded that using the first approach were precisely detected several types of human-related actions with high accuracy.

4 Application Backend

The application backend, also known as server-side, is essential in the application's development process because it combines the execution runtime providing some sort of resources management and the programming language or framework used by programmers. For example, it handles all the database interactions and data mapping using an abstraction provided by the runtime and provides content to applications through controllers.

A good conduct consists of implementing services, which can be available through webservice, Representational State Transfer Application (REST) application programming interfaces (APIs), or messaging processes, in order to create an abstraction in the direct access to the stored data. Webservices allow easy communication between client applications and application backend.

A typical REST API is a set of definitions and protocols to create and integrate application's software that is available through HTTP endpoints and following the basic architectural REST principals. A common example is API for allowing the access to data sources through abstractions, such as Object-Relational Mapping (ORM) [33]. An API consumer/client requires content through calls and API responds to the call by sending the requested content. The implementation of APIs allows multiple applications and external clients to access the same content at the same time. This is impossible if the server-side is implemented together with a client application, i.e., the client application besides presenting content to the user, is responsible for all calculations made by the backend.

With the technological growth and the need to create new software and applications, numerous backend frameworks for different programming languages have emerged. Some of the currently most used backend frameworks by language [34] are presented in Table 3.








Framework	Language	License	Characteristics	Used by
	Java	Open source	Powerful and handy framework for develop modern java-based enterprise applications, providing an easy configuration model and a comprehensive programming.	Trivago Udemy
	JavaScript	Open source under MIT License	Minimal and flexible framework used by node.js to build and maintain APIs, and provides plugins and templates that reduce the development time.	IBM Accenture
	Python	Open source	Django framework enables building quick and stable high-level backend applications and APIs. Provides authentication and security mechanisms to protect applications and is convenient for the development of feature-rich database-driven web applications.	Instagram Youtube
	PHP	Open source under MIT License	Laravel is a lightweight framework that allows the development of robust microservices and high-performance APIs. Supports caching and authentication of multiple users using tokens.	Pfizer 9GAG
	Ruby	Open source under MIT License	Ruby on Rails framework follows the model-view-controller (MVC) pattern and use some web principle such as XML or JSON for information sharing and offers database default structures, web pages and web services.	Spotify GitHub

Table 3 - Most used backend frameworks by programming language (adapted from [35])

In short, all the frameworks grants are very similar providing a quick development and maintenance of APIs, whether can be protected with security mechanisms and allowing connections to different databases, and mapping data to objects quickly and effectively.

4.1 Java Enterprise Application Frameworks

Of all the application development frameworks mentioned above, only Spring Boot is the most suitable to develop enterprise-level applications. Usually, a microservices approach is followed for enterprise application development, and technologies like Spring Boot, Quarkus and Jakarta EE present a significantly relevant performance compared to the other backend frameworks mentioned above. Table 4 compares these three Java-based frameworks.

Framework	Language	License	Characteristics
	Java	Open source	Framework to develop modern java-based enterprise applications and web services, providing an easy configuration model, starters and a comprehensive programming [35]. This framework provides faster I/O operations and reduces the effort of the developer taking advantage of configuration's files. However, it contains a huge quantity of dependencies [36]. Supports Tomcat, Undertow and Jetty as embedded servers.
	Java	Open source	Java framework that enables the delivery of small artifacts, fast boot and lower first request time. Is a Microprofile compliant efficient for serverless, Kubernetes

			environments and cloud, and provides dependency injection and REST API development. Compared to Spring Boot, Quarkus requires less memory [36].
	Java	Open source	Jakarta EE is a set of configurations that allows Java developers working on java enterprise applications. Jakarta EE applications relies on microservices or application servers running on reference runtimes such as WildFly, GlassFish or Payara Server [37].

Table 4 - Comparison between backend frameworks used for enterprise applications development

To sum up, it is clear that Java programming language is suitable for enterprise applications development. Despite Express JS, Django, Laravel and Ruby on Rails being the most used and trended technologies to backend development, developers should adapt its use to the intended architecture.

4.2 The Dart from Behind the Flutter

The notable positive satisfaction with Flutter boosted a contentment with Dart language during the first quarter of this year according to Flutter team survey results [38]. Between 2019 and March of 2021, the enjoyment increased 9% and some developers started to uncover the potential of Dart applied to other contexts beyond mobile, web and desktop development, namely after the release of Flutter 2 as seen in Figure 1.

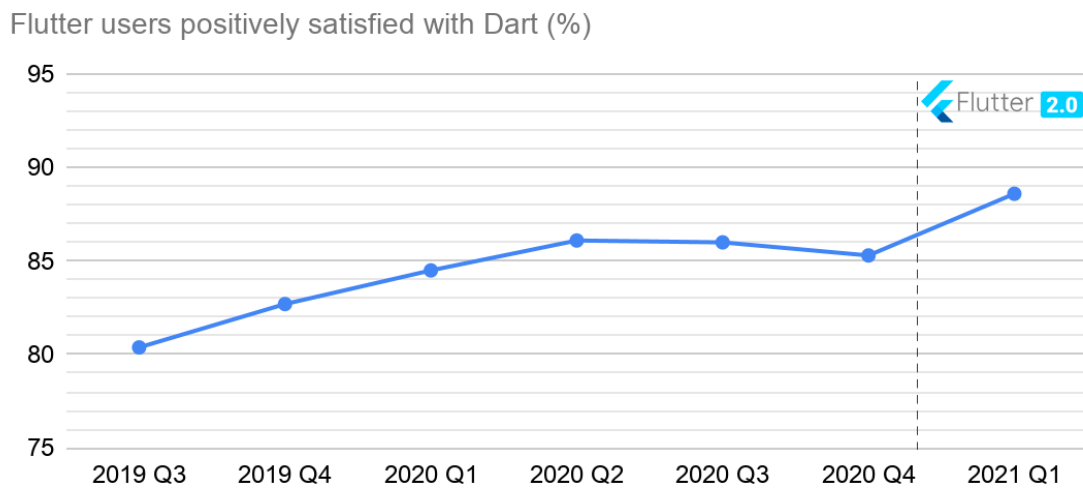


Figure 7 - Percentage of Flutter users positively satisfied with Dart during the first quarter of 2021 (adapted from [38])

Despite the most known frameworks previously mentioned, with the growth of ascertained Dart language, backend frameworks to build REST APIs and services have also emerged. The most well-known and used framework was Aqueduct that allows building RESTful APIs based on Dart and multi-threading HTTP server including ORM providing a stable and fast mapping between databases tables and data objects [39]. In late March 2021, this framework was discontinued, resulting in an increased demand for Dart-based identical frameworks. Jaguar [40] and Angel [41] are two examples of frameworks like Aqueduct, however, the latter has not changed since 2018 and is out of date with the new versions of the Dart language that have been announced. Although Aqueduct was discontinued, a comparison was made between the different Dart-based backend frameworks. Table 5 presents a review of the different Dart-based backend frameworks.



Routing	Yes	Yes	Yes
Multi-threading	Yes	Yes	No

ORM	Yes	Yes	Yes
Serialization	Yes	Yes	No
Authentication Support	Yes	Yes	Yes
GraphQL Support	No	No	Yes
Last update	August 2020 (Discontinued on late March 2021)	May 2021	May 2020 (Discontinued on late April 2021)

Table 5 - Comparison between backend frameworks based on Dart language

All previously listed frameworks are not provided by Google, the creators of Dart language. Therefore, it is predictable that their support may fall short of the developers' expectations despite showing promise in the features provided. Furthermore, a brief survey over the frameworks' documentation clarify that it is scarce and unclear in some cases, namely for Jaguar and Angel.

Chapter 3 System Specification

In this chapter, the BeingCare system specification is presented including its personas, stories, and main requirements. A logical architecture is presented as well as some examples of information flows in the BeingCare.

1 Description

In a care support scenario, often the people being cared are bedridden (or not), isolated people, or people with aphasia problems. These people clearly have difficulties in communicating with their caregivers to report for example their pain status, or problems related to their wellbeing. Indirect monitoring becomes a perfect ally to help caregivers get clearer information about their people being cared condition. Therefore, it is useful to identify events using audio classification while monitoring and some non-critical events are on demand by people being cared. It is relevant to identify some events associated with the people being cared, such as cough, sneeze, demands for help, or other types of requests. On the other hand, the caregiver's teams should be able to access the events generated by their people being cared, receiving real-time notifications related to alarms and requests.

In this context is aimed to achieve through the BeingCare system:

- Support implicit communication i.e., detect and report relevant events in the context for example coughing, gagging, paining, among others;

- Support explicit communication i.e., the person wants to communicate providing solutions according to different levels of autonomy, such as explicit speech, tempting communication using sounds like tapping

The improvement of direct communication between people or identify events with context valuable may grant overcome the barriers previously mentioned. Naturally, the system must be based on the exchange of information through messaging that may be a result of a relevant event in the context.

Figure 8 describes the system's architecture in a logical context presenting the personas involved as well as its role.

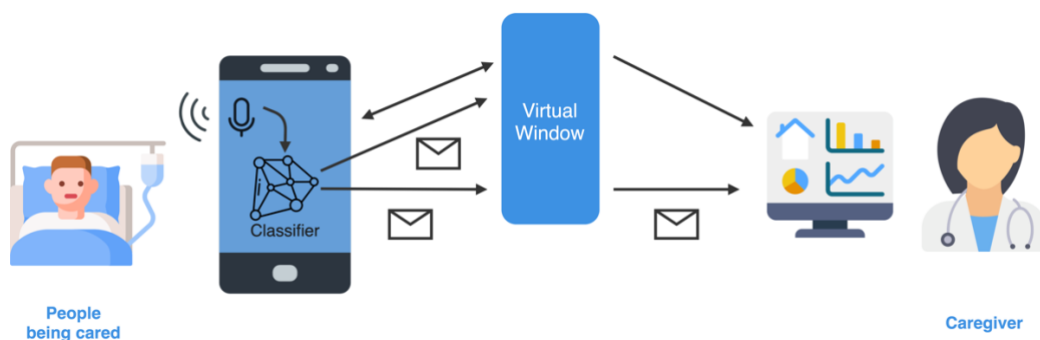


Figure 8 - BeingCare logical architecture

The system should have two user interfaces intended for each persona: the people being cared and caregiver. Each people being cared has associated a mobile device running the BeingCare mobile application. This application is responsible for recording audio and classifying it and for sending the classification's results to system services to be stored. The purpose of audio classification is to identify some relevant events according to sounds provided by people being cared. In case of the classifier detects an event considered critical, a message is sent to a message broker in order to send it directly to the web application. Caregivers through the BeingCare web application receive reports of the critical events and receive other data related to people being cared and their history.

Table 6 summarizes the role of each persona.

People being cared	All monitored people considered isolated, bedridden, or not, or people with aphasia problems. People being cared can interact with the system directly, sending messages to caregivers or indirectly through monitoring.
Caregivers	All caregivers interested in the people being cared state. Caregivers have access to all events, information, and critical situations related to their people being cared.

Table 6 – Personas of the system

Descriptive stories translating functional and non-functional requirements about the behavior of the system are detailed in Table 7.

Story Name	Description	Persona
Visualize events	As a caregiver, I want to access all events related to people being cared such as the frequency of cough, sneeze, or snore.	
Visualize requests	As a caregiver, I want to access all requests (asks for basic care assistance, water, food) made by people being cared so that I can attendee it.	Caregiver
Receive notifications	As a caregiver, I want to receive real-time notifications whenever a people being cared is in a critical situation so that I act quickly to help.	
Send requests	As a people being cared, I want to inform caregivers if I am hungry, need water, or some basic care assistance so that caregivers can help me.	People being cared
Visualize history	As a people being cared, I want to see all occurred events related to me and send them to caregivers	

Send speech to text	As a people being cared, I want to send personalized messages to caregivers without typing text
---------------------	---

Table 7- Description of defined stories specifying the system

It is expected the system allows communication between people being cared and caregivers, ensuring a point-to-point communication, able to alert the caregivers when their people being cared are in risky situations, such as pain or gagging/choking. However, as people being cared sometimes only require non-critical assistance such as basic assistance from their caregivers or even food, the system should include quick shortcuts for this type of requests, facilitating the interaction of people being cared with it.

Due to flexibility and accessibility in scenarios of monitoring, indeed the mobile technology is an interesting solution for people being cared. Addressing Flutter cross-platform is the main objective of this dissertation that grant portability between mobile operating systems ensuring the system availability on different mobile devices.

2 The Information

The information exchanged in BeingCare can be grouped into four different types: alarms, events, requests, and messages. Table 8 details each type of message as well as which events are associated.

Types	Description	Events associated
Alarm	Messages that translate risk situations that happened to a people being cared	Tap, pain, or gagging (automatic)
Event	Messages containing people being cared-related events	Cough, sneeze, or snore (automatic)

Request	Messages considered requests by people being cared, such as ask for food, water, or basic care assistance	(on demand)
Message	Voluntarily sent messages that are not considered of the other types mentioned	(on demand)

Table 8 - Types of messages shared through the system and respective events associated

2.1 Events and Data Exchanged

The information exchange in the system is encoded in JSON and timestamped. Below are brief examples of data exchange message depending on the type of event.

Example of Alarm Messages

These messages result of pain symptoms, calls for help, or even choking from the person and are produced automatically and sent as the following example:

```
{
  "timestamp": "2021-05-08T14:18:13.109",
  "sender": "22e66656dc7fd1df",
  "type": "ALARM",
  "description": "PAIN"
}
```

The details of the message also contain the description of the event that caused the alarm, which can be Pain, Tap, or Gagging.

Example of Event Messages

Events related to the people being cared health, such as coughing, sneezing, and snoring detected automatically, originate messages similar to the following.

```
{
  "timestamp": "2021-05-10T11:21:06.521",
  "sender": "22e66656dc7fd1df",
  "type": "EVENT",
  "description": "COUGH"
}
```

Like the other types of messages, the description details the type of event which caused the alarm, such as Cough, Snore, or Sneeze.

Example of Request Messages

These messages are on-demand by the people being cared, and he can send requests according to the options presented to him in the system, for example, request water, food, or basic care assistance.

```
{
  "timestamp": "2021-05-11T16:19:59.977",
  "sender": "22e66656dc7fd1df",
  "type": "REQUEST",
  "description": "WATER"
}
```

Example of Message Messages

Similar to request messages, this type is on-demand by the people being cared however he can have more freedom to customize what it wants to transmit. These messages are produced after the people being cared mentions what he wants to transmit through speech to text in order to facilitate the interaction with the system.

```
{
  "timestamp": "2021-05-11T16:19:59.977",
```

```
"sender": "22e66656dc7fd1df",  
"type": "MESSAGE",  
"description": "I would like to call to my family"  
}
```


Chapter 4 Implementation

This chapter details the implementation choices made during BeingCare implementation with special emphasis on the opting to a full Dart/Flutter solution either for the back office and frontend UI i.e., mobile and web.

1 System Architecture

In BeingCare system, there are four essential components depicted in Figure 9:

- Mobile component
- Services component
- Persistence component
- Web component

The main flow of BeingCare is centered around audio processing so that any audio relevant interaction is processed and transmitted to the caregiver. In this flow, audio captured by mobile application is classified and the result is shared with the system. Audio classification depends on the Tensorflow Lite [42] framework used to integrate an audio classification model into the mobile component which is addressed with more detail in the next section. The mobile component communicates with the services component through REST requests but also uses Message Queuing Telemetry Transport (MQTT) [43] messages to send information regarding generated alarms. Message passing relies in MQTT support on a Mosquitto [44] message broker. The web component consumes data provided from services and data that are transported by the message broker through Websockets.

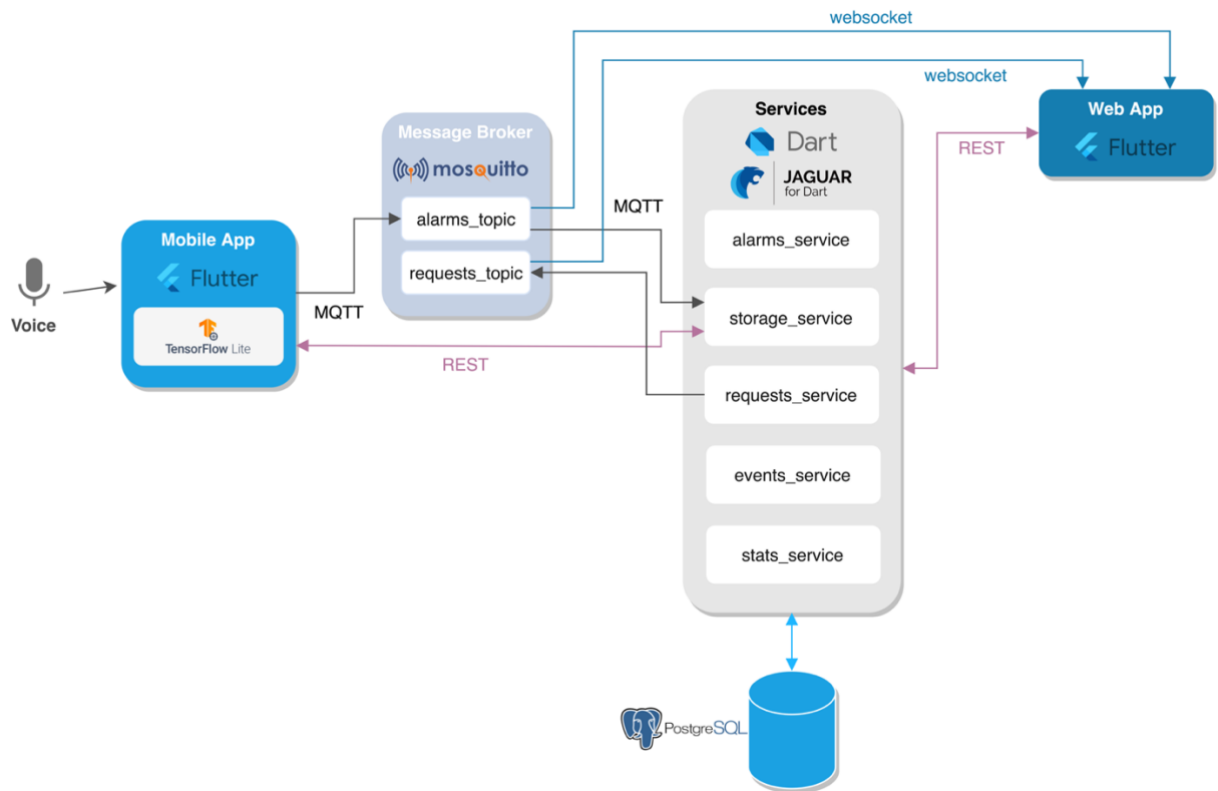


Figure 9 – Technical architecture diagram

Figure 10 details how operations between components are carried out and what information is sent and when. Each component will be discussed in more detail in the next sections as well as some technological decisions made to implementing the components. While the red flow represents an automatic information flow, the yellow and green represents, respectively, information sent on demand and information that is visualized.

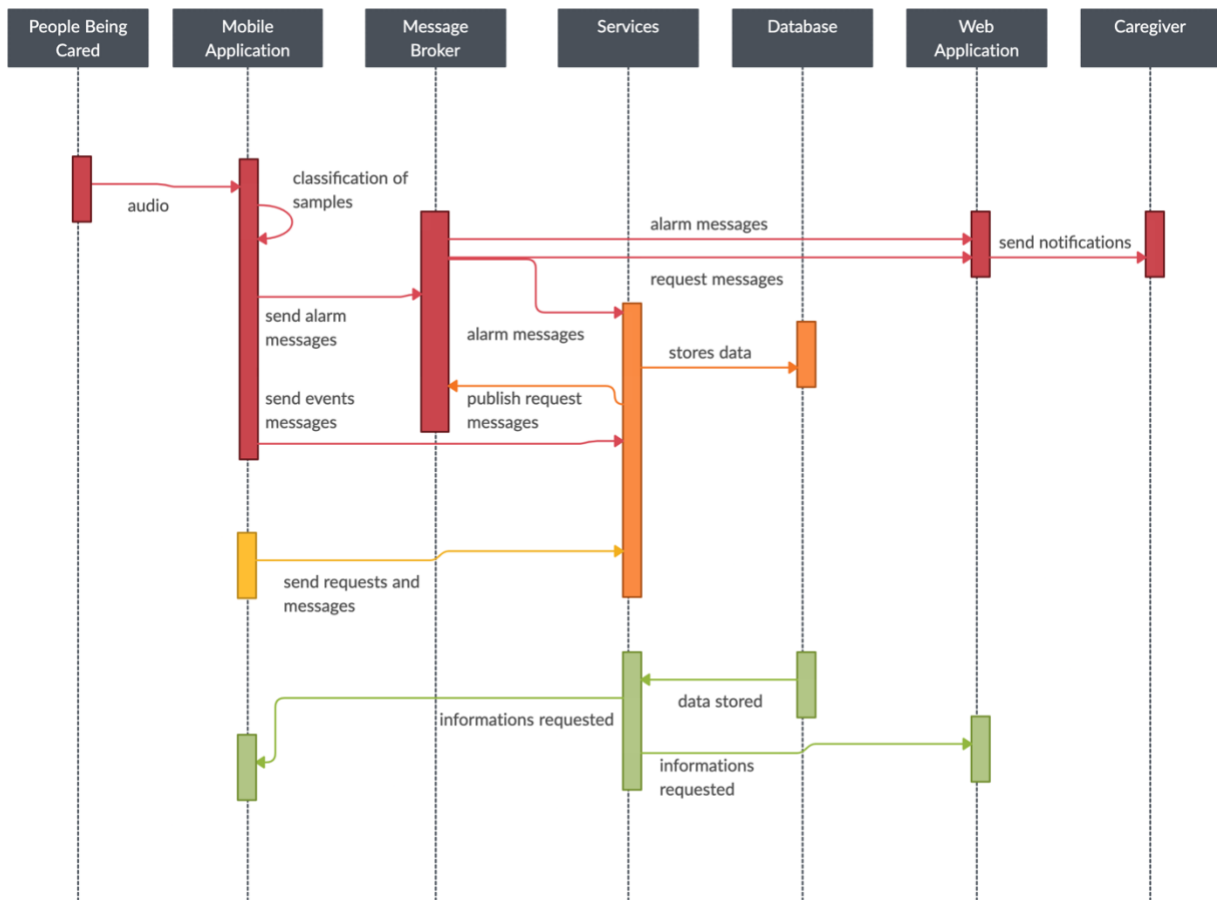


Figure 10 - Sequence diagram of the flow of the information exchanged between components

2 BeingCare Mobile Component

The mobile component is implemented in cross-platform framework Flutter [24] and uses the Jaguar [40] as a client to support REST requests. This component is responsible for audio acquisition, classification, and event propagation. It relies on an audio processing unit described in other section containing two sub-units:

- audio processing and classification unit responsible for detecting audio events according to a classification model;
- speech to text unit to transform speech into text in order to sharing more descriptive information to the system based on speech.

While speech to text is a voluntary audio detection, audio classification occurs automatically in the background when the application is running.

The mobile component propagates via REST or through the share of MQTT messages if an alarm is to be triggered. Mobile application also requests information related to the messages sent by its user via REST to the services component. As the collection of sensitive data requires greater attention regarding sharing and storage issues, all acquired audio data is collected locally and not transmitted outside the people being cared device.

The mobile application was developed using Flutter 2.0 version and there were used packages to support some features and external frameworks. The Appendix A contains the dependencies that are specified into `pubspec.yaml` file of the mobile application to support the development and some features implemented. Below in Table 9 is a brief explanation of each one package used.

cupertino_icons	Adds the Cupertino Icons font to application to support iOS style icons
http	Package used to consume HTTP resources through high-level functions and classes for this purpose
jaguar_resty	Provides building fluent functional REST clients supporting interceptors and authenticators.
speech_to_text	Package that contains classes use speech recognition capabilities in Flutter
device_info	Provides detailed information about the device such as model, name, make and others
tflite_audio	Audio classification Tensorflow Lite package
mqtt_client	Adds a server and browser based MQTT client for Dart supporting normal, secure sockets and web sockets
intl	Package used to deal with internationalized date and number formatting and parsing

isolate_handler	Package to support concurrent execution of code through isolates
provider	Wrapper around InheritedWidget to make them easier to use and more reusable
curved_navigation_bar	Add curved and animated style to navigation bars

Table 9 – Packages used as support to mobile application development and features

Verbal data and intentions of users are collected, respectively, using the smartphone’s microphone and touch inputs. The microphone is used not only to record audio samples to classify and detect events but also to transform speech into text messages that will sent to the system service.

2.1 Audio Processing and Classification Unit

The implementation of this unit helped to classify automatically relevant events related to people being cared using the smartphone microphones as input. Thereunto, the Tensorflow Lite framework was used to integrate ML classification models in mobile applications.

2.1.1 Tensorflow Lite

TensorFlow Lite [42], a lightweight version of Tensorflow [45] framework that allows the integration of ML models into mobile solutions, was recently released. It allows running Tensorflow models locally on device, and all audio classification models in a Tensorflow format are supported on Tensorflow Lite. As Tensorflow Lite framework works offline, it grants a non-dependency on the network to send, for example, data to be classified and provides greater efficiency because data never leaves the mobile device, and the entire process is local. So, the use of this framework guarantees the user’s privacy since it handles sensitive data on the device itself without data being exposed.

The integration of Tensorflow Lite models into a Flutter mobile application was performed by a Flutter package [46].

2.1.2 Tensorflow Lite Audio Classification Models

The classification model tested and used were created with the help of Google Teachable Machine (GTM) [47] framework since it enables the generation of Tensorflow Lite models. An audio classification model was created with several classes, each composed of about 14 audio samples recorded not only from different aged people but also from online sources providing free license samples. The option for using GTM with Tensorflow Lite was a trade off in the proof-of-concept scenario, while enabling an easy and straight forward model creation and usage, it is black box design hindered any clear view on both the model (based on neural networks) [48] and the learning process, relying on GTM automated process.

Figure 11 shows the model created properties using the Netron application [49] to visualize the entire model workflow. The model input is a tensor. All computations in Tensorflow involves tensors [50], an immutable and multi-dimensional array of a uniform type of values with a known shape (dimensionality of the array).

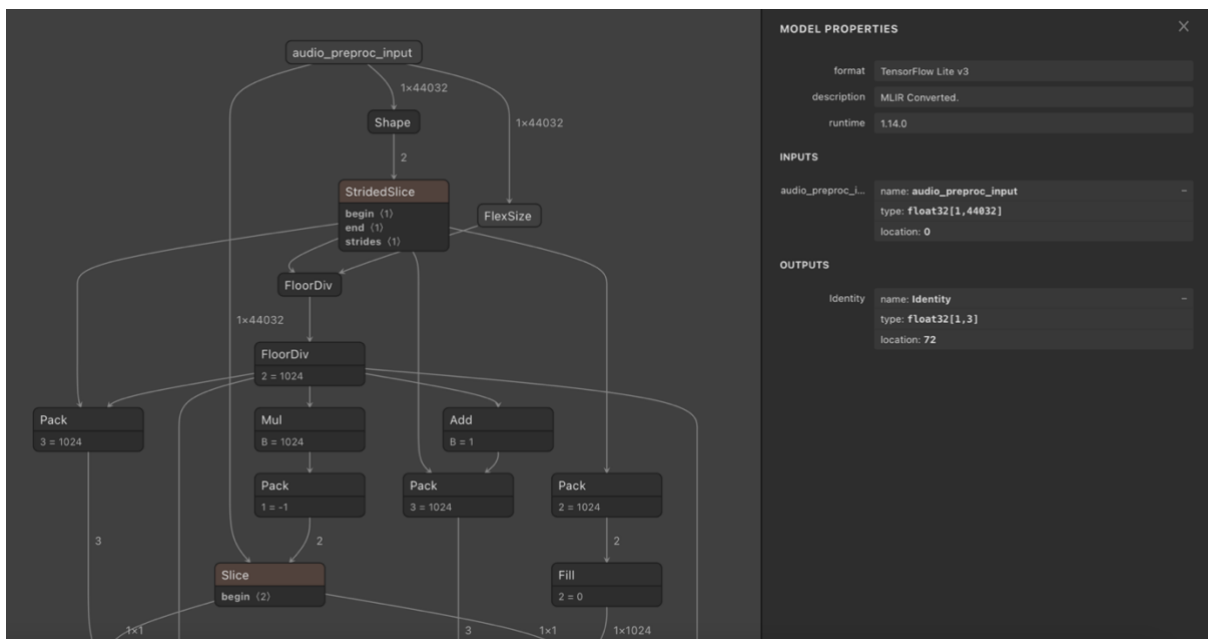


Figure 11 – Tensorflow Lite model generated using Google Teachable Machine

The audio is recorded and inferred using the `startAudioRecognition` method provided by package where are passed some important arguments:

- **Sample rate:** number of samples per second;

- **Input type:** specifies the type of input: *rawAudio* for GTM models or *decodedWav*;
- **Recording length:** determines the size of tensor input and must be equal to the tensor input of classification model;
- **Buffer size:** size of the buffer that iterates over recording length;
- **Detection threshold:** threshold to ignore any predictions below the inference precision of the event detected.

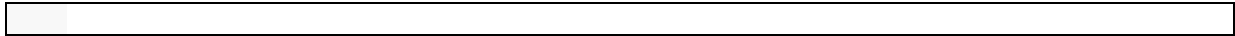
2.1.3 Audio Classification on the Fly

The audio recognition process is performed parallel to the other features of the mobile application using an Isolate [51]. Dart Isolate is analogous to a Thread but with some differences namely in the way of working. An Isolate is an independent worker that does not share memory; however, it is possible to communicate with other Isolates by passing messages over communication's channels. In Flutter, the application runs into a main Isolate. When the application started, it is created other Isolate responsible for running audio recognition parallelly and whenever the process recognizes an event, a message is sent containing the classification obtained to the communication channel so that the main Isolate knows the results. Code 1 shows the implementation of the audio recognition process running in a separated Isolate.

```

1  static void startRecording(Map<String, dynamic> context) {
2      String recognition = "";
3      int sampleRate = 44100;
4      double detectionThreshold = 0.6;
5      final messenger = HandledIsolate.initialize(context);
6      _isRecording = true;
7      messenger.listen((msg) async {
8          result = TfliteAudio.startAudioRecognition(
9              numOfInferences: 1,
10             inputType: 'rawAudio',
11             sampleRate: sampleRate,
12             recordingLength: 44032,
13             bufferSize: 2000,
14             detectionThreshold: detectionThreshold,
15         );
16
17         result.listen((event) {
18             log(event.toString());
19             recognition = event["recognitionResult"];
20         }).onDone(() {
21             _isRecording = false;
22             recognition = recognition.split(" ")[1];
23             TfliteAudio.stopAudioRecognition();
24             messenger.send(recognition);
25         });
26     });
27 }

```



Code 1 – Audio recognition process

The classification model is preloaded and when the Isolate communication channel initializes and receives order to start the audio processing, Tensorflow Lite starts the audio recognition according to the input parameters defined in the respective method (lines 8-15). A stream provides receiving a sequence of data [52]. The result of the audio recognition process is populated into a data stream that is listened to and retrieved the inference result sending it through the communication channel to the initial Isolate (lines 17-25).

2.2 Speech to Text Unit

Speech to text uses the *speech_to_text* package mentioned above. This package transforms the audio provided by the microphone returning the correspondent text and supports multiple languages speech recognition, namely native Portuguese.

Both audio detection and speech to text require the use of the microphone to record audio. As the audio recognition process runs in loop on an Isolate, whenever the speech to text is initialized, it kills the Isolate in order to interrupt the audio recognition, giving priority to the speech to text process. After the speech to text is completed, the audio recognition process is restarted.

Provider is a package usually used to manage rebuilding UI based on state changes. The used of Provider helped with state management problem between audio recognition process and speech to text regarding the share of microphone resources. Figure 12 demonstrates the state management problem that could be solved using the Provider.

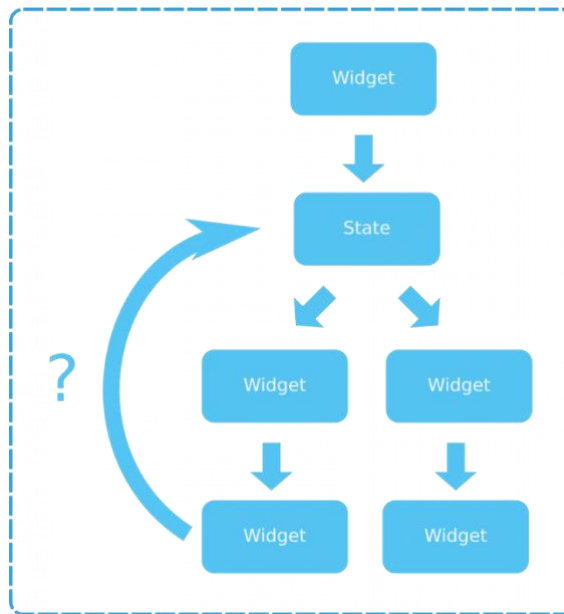


Figure 12 – The state management problem [53]

Provider manages the audio in order to start or stop the audio processing and classification unit whenever speech to text is used. More precisely, it was used a `ChangeNotifier` pattern from Flutter responsible for notifying the listeners about state updates, as seen in line 8 of Code 2. An `Audio` model that extends from `ChangeNotifier` class was implemented and contains all the state data that are being used by the different units (speech to text and audio processing and classification), allowing notify structure listeners whenever some data is changed. `ChangeNotifier` is identical to the Controller's role in the Model-View-Controller pattern.

```

1  class Audio extends ChangeNotifier {
2    String _status = "INIT";
3
4    String get getStatus => _status;
5
6    void setStatus(String s) {
7      _status = s;
8      notifyListeners();
9    }
10 }

```

Code 2 – Audio model implementation to notify listeners when data changed

It was used the `ChangeNotifierProvider` type that provides the instance of the audio model to access state data in the widget. Whenever speech to text tries to start a recording, firstly, the audio status data is changed to "STOP" to notify the audio recognition listener so that it can interrupt it and release the microphone, as presented in Code 3. After speech to text is completed, audio status data is again changed.

```
1 Provider.of<Audio>(context, listen: false).setStatus("STOP");
```

Code 3 – Changing audio status using Provider

Every time the audio recognition listener is notified, according to the data updated, the audio recognition process is interrupted or restarted. Through Code 4, whenever the status is changed, is received the new status after the listener being notified.

```
1 var status = Provider.of<Audio>(context).getStatus;
```

Code 4 – Receiving new audio status after listener being notified

The sharing of messages with the others system components differs according to their type. For example, an alarm message is automatically published to a specific topic in the message. On the other hand, events and request messages are sent via HTTP requests directly to the storage service.

3 BeingCare Service Component

The BeingCare service component was implemented in Dart using the full stack Dart server framework Jaguar [40], in order to develop a system's proof-of-concept based on Dart language. Jaguar framework includes ORM, serialization, MVC template, authentication, and security with JSON Web Tokens (JWT) [54]. As advantages, this framework enables writing fast and elegant RESTful APIs based on Dart language.

Its syntax is analogous to the syntax of the Spring Boot framework where it is possible to build groups of routes associated with classes inheriting from the Controller pattern.

Similar to the Flutter application, this component is a Dart application containing several packages to support the development of the entire service. Jaguar framework is imported as a dependency as well as other Jaguar packages to support the server. The Appendix A contains the dependencies imported in `pubspec.yaml` file and the Table 10 provides a brief description of each package used:

jaguar	Provides a production ready HTTP server for building REST APIs in Dart
jaguar_reflect	Provides reflection of Jaguar Controllers into Jaguar routes
jaguar_session_jwt	Provides JWT session managers to build server sessions on JWT standard
jaguar_auth	Provides username-password based authentication interceptors and helper methods for Jaguar
jaguar_common	Jaguar common components to both server and client
jaguar_cors	Provides Cors interceptor and CorsOptions configuration to add CORS support to Jaguar server applications
http	Provides classes to consume HTTP resources
postgres	Driver for connecting and querying PostgreSQL databases
mqtt_client	Adds a server and browser based MQTT client for Dart supporting normal, secure sockets and web sockets

Table 10 – Description of each package imported

3.1 Services and Controllers

The BeingCare service offers 5 logical services, described in **Error! Reference source not found.** and developed based on their contextual logic and provide REST endpoints implemented using Jaguar Controllers.

Storage Service	Service responsible for store messages and retrieve information related to people being cared
Alarms Service	Service responsible for retrieve information related to alarms stored in the repository
Events Service	Service responsible for retrieve information related to events stored in the repository
Requests Service	Service responsible for retrieve information related to requests stored and send new requests received to a message topic
Statistics Service	Service responsible for retrieve statistical information related to the overall data stored

Table 11 - Description of services implemented

The information is retrieved through querying on the database. All messages exchanged through the system are stored in a database to ensure that information is not lost if a service fails or restart. For this purpose, PostgreSQL [55] was used as database due to its advantages, such as low maintenance and simplicity of querying operations. This database supports not only SQL (Structured Query Language) but also JSON (JavaScript Object Notation) for relational and non-relational queries.

The database is composed by a single table:

- **Message:** Responsible for store all information related to messages exchanged

The Appendix C contains the database schema that creates tables with specific fields and rules in order to prepare the database to receive information during the services run.

In the end, all information can be accessed by REST API endpoints implemented using REST Controllers.

Controller actions must be created in order to expose resources to final clients through the REST API. There were created six different controllers annotated with @GenController annotation. All routes' handlers were implemented as methods and annotated using HTTP method annotations (such as @Get, @Post, @Put, @Delete). Table 12 describes all annotations used grouped by the respective controller.

Annotations	Annotation Description	Controller	Method
@PostJson	Responsible for storing records provided by the JSON body of the request	Storage Controller	insertMessages
@GetJson	Responsible for returning a JSON object with all records associated with the respective service	Storage Controller Alarms Controller	
@HttpMethod	HttpMethod annotation is used to allow Options methods due to Cross-Origin Resource Sharing (CORS) [56]	Events Controller Requests Controller Statistics Controller	getRecords
@HttpMethod	Responsible for user's authentication through a login allowing Get and Options methods	Auth Controller	login
@Post	Responsible for clear the user from session		logout

Table 12 – Annotation's overview used on each controller

Below in Code 5 is depicted the Storage Controller implementation as an example of implementing controllers using the Jaguar Dart framework and the other controllers are analogous.

```
1  const corsOptions = CorsOptions(  
2    allowAllOrigins: true, allowAllHeaders: true, allowAllMethods: true);  
3  
4  @GenController(path: '/storage')  
5  class StorageController extends Controller {  
6  
7    @PostJson()  
8    Future<Map> insertMessages(Context ctx) async {  
9      await Authorizer.authorize<User>(ctx);  
10     var body = await ctx.bodyAsJsonMap();  
11     await StorageService().saveRecords(body);  
12     return body;  
13   }  
14  
15   @GetJson(path: '/info/:beingCaredId')  
16   Future<List<dynamic>> getRecordsByCareer(Context ctx) async {  
17     if (ctx.req.method != 'GET') return null;  
18     await Authorizer.authorize<User>(ctx);  
19     var param = ctx.query.getInt('hour');  
20     return StorageService()  
21       .getAllRecordsById(ctx.pathParams['beingCaredId']  
22         .toString(), param);  
23   }  
24  
25   @HttpMethod(path: '/messages', methods: ['GET', 'OPTIONS'])  
26   @GetJson(path: '/messages')  
27   Future<List<dynamic>> getMessageRecords(Context ctx) async {  
28     if (ctx.req.method != 'GET') return null;  
29     await Authorizer.authorize<User>(ctx);  
30     return StorageService().getMessageRecords();  
31   }  
32  
33   @HttpMethod(path: '/peopleBeingCared', methods: ['GET', 'OPTIONS'])  
34   @GetJson(path: 'peopleBeingCared')  
35   Future<List<dynamic>> getPeopleBeingCared(Context ctx) async {  
36     if (ctx.req.method != 'GET') return null;  
37     await Authorizer.authorize<User>(ctx);  
38     return StorageService().getAllPeople();  
39   }  
40  
41   @override  
42   void before(Context ctx) {  
43     cors(ctx, corsOptions);  
44   }  
}
```

Code 5 – Storage Controller implementation

Some endpoints defined in Storage Controller are accessed by web services. CORS mechanism uses additional HTTP headers to allow a server to indicate any other origins. The configurations related with CORS options are specified in lines 1-2 and 40-43 in Code 5, and the endpoints annotated with `@HttpMethod` supports Options requests (example in line 24, Code 5). The endpoint path is defined inside each annotation using the `path` property.

One of the benefits of Dart centers on its asynchronous operations that enables the program to be autonomous while it waits for another operation to finish [57]. Asynchronous operations are related to:

- Fetch data from the network;
- Writing or reading data from databases or files;

Dart uses the `Future` class and the `async` and `await` keywords providing a declarative way to define and handle with asynchronous operations. These operations are denoted with `async` keyword before the function body (see line 8, Code 5) and the `Future` represents the result of the operations containing two states: completed or uncompleted before having a value. The `await` keyword allows getting the result of the asynchronous operations in the completed state. The session user authorization and the storage of inserted records methods (lines 9 and 11, respectively in Code 5) are asynchronous operations that require waiting before proceeding with the rest of the operations. In other words, if the user is not authorized for example, he cannot retrieve the information available after the `await` declaration.

All storage endpoints require authorization. Initially, the user authenticates through the endpoint created for that purpose in order to obtain a Bearer token through the session that must be used to access the remaining protected endpoints. The authentication and authorization process will be covered in the section 3.2

3.2 Authentication and Authorization

Besides all information's stored and sent to the BeingCare Service being indirect information's related to users, additional security issues were considered to protect data accessed. For this purpose, JWT method was implemented with the help of `jaguar_session_jwt` package for an authorization scenario where the user is required to login before accessing other API routes. Figure 13 describes the authentication and authorization processes workflow.

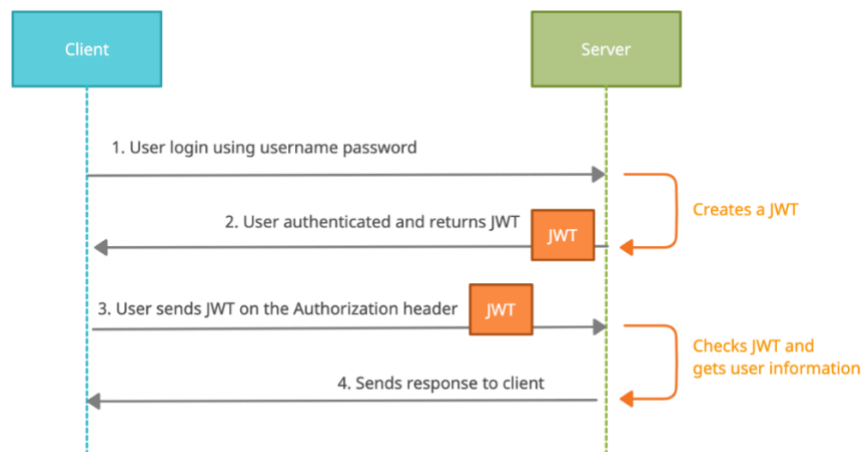


Figure 13 – Authentication process returning a JWT for users’ authorization when requests API

Of the three possible authentication methods offered by *jaguar_auth* (Basic Auth, Form Auth, and JSON Auth), authentication is based on Form Auth where an authenticator expects an `application/x-www-form-urlencoded` encoded body with username and password fields.

According to documentation, authorization implementation requires the *jaguar_auth* package that revolves around three basic principles according to documentation:

- **User Model:** user model uniquely identified
- **User Fetcher:** fetch the user model by its unique identification
- **Authorizer:** verification of a proper user identity in the request

The user model implements a `PasswordUser` interface provided by *jaguar_common* for a user model with a password to operates with Authorizer.

A fixed list of users authorized for login has been defined and all controller’s routes only return results if the user is authenticated and, consequently, authorized.

3.3 Messaging: Consume and Publish

Messages published in alarms topic are sent directly from mobile application to web application. However, for historic effects it is necessary that the storage service saves all information's exchanged. For this purpose, it was implemented a MQTT client capable of connecting to the message broker and subscribe a certain topic in order to listen the data stream receiving real-time messages. Whenever a message of alarm type is received (lines 2-10, Code 6), the storage service stores it into the database (line 9, Code 6).

```

1  _client.subscribe(topic, MqttQos.atLeastOnce);
2  _client.updates.listen((List<MqttReceivedMessage<MqttMessage>> c) {
3      final recMess = c[0].payload as MqttPublishMessage;
4      final pt =
5  MqttPublishPayload.bytesToStringAsString(recMess.payload.message);
6      print('[MOSQUITTO] Received message on topic <${c[0].topic}>:
7  $pt\n');
8          var json = jsonDecode(pt);
9          StorageService().saveRecords(json);
10 });

```

Code 6 – Subscription and listening alarms to store information sent via MQTT

When the storage service receives a POST request (lines 7-13, Code 5) with messages of request type, these messages in addition to being stored in the repository, are also published in the requests' topic so that the web application can consume this information through a Websocket (Code 7).

```

1  void publish(String message, String topic) {
2      final builder = MqttClientPayloadBuilder();
3      builder.addString(message);
4      _client.publishMessage(topic, MqttQos.exactlyOnce,
5  builder.payload);
6      print('[MOSQUITTO] Published message on topic <$topic>:
7  $message\n');
8  }

```

Code 7 – Publishing messages into a specific topic

3.4 API Documentation

The API was documented using Swagger and it is available¹. Table 13 summarizes the created endpoints and a brief description of each one.

	Request	Endpoint	Authorization	Description
Login	POST	/login	No	Endpoint for user authentication
	POST	/logout		Endpoint for logout session
Storage	POST	/storage	Yes	Endpoint for post records
	GET	/storage		Endpoint for returning all records
	GET	/storage/info/{beingCaredId}		Endpoint for returning all records of a people being cared in the last hours
	GET	/storage/peopleBeingCared		Endpoint for returning all people being cared
	GET	/storage/messages		Endpoint for returning only messages records
Alarms	GET	/alarms	Yes	Endpoint for returning all alarms
	GET	/alarms/last		Endpoint for returning the alarms occurred in the last x hours
	GET	/alarms/last/{beingCaredId}		Endpoint for returning the alarms occurred in the last x hours given a people being cared ID

¹ <https://app.swaggerhub.com/apis-docs/cmambuquerque/BeingCare/1.0.0>

Requests	GET	/requests		Endpoint for returning all requests
	GET	/requests/last	Yes	Endpoint for returning the requests occurred in the last x hours
	GET	/requests/last/{beingCaredId}		Endpoint for returning the requests occurred in the last x hours given a people being cared ID
Events	GET	/events		Endpoint for returning all events
	GET	/events/last	Yes	Endpoint for returning the events occurred in the last x hours
	GET	/events/last/{beingCaredId}		Endpoint for returning the events occurred in the last x hours given a people being cared ID
Statistics	GET	/stats/all		Endpoint for returning statistics related to overall records
	GET	/stats/alarms		Endpoint for returning statistics related to alarms records
	GET	/stats/events	Yes	Endpoint for returning statistics related to events records
	GET	/stats/requests		Endpoint for returning statistics related to requests records

Table 13 – API documentation summary

4 BeingCare Web Application Component

The development of web application appealed to the address of Flutter in order to try to prove a proof-of-concept of the entire system based on Dart language. Flutter web allows the compile of Dart code into HTML, CSS, and JavaScript code and works like Flutter in mobile. Thus, Flutter packages are integrated into applications in the same way as mobile by adding dependencies to the `pubspec.yaml` file described in the Appendix A and the Table 14 provides a brief description of each package imported to support the development:

<code>http</code>	Provides classes to consume HTTP resources
<code>jaguar_resty</code>	Provides building fluent functional REST clients supporting interceptors and authenticators.
<code>flutter_bloc</code>	Provides pre-implemented Widgets to easily implement BloC (Business Logic Component) pattern
<code>provider</code>	Wrapper around InheritedWidget to make them easier to use and more reusable
<code>mqtt_client</code>	Adds a server and browser based MQTT client for Dart supporting normal, secure sockets and web sockets
<code>line_icons</code>	Icon library based on Awesome Line Icons
<code>intl</code>	Package used to deal with internationalized date and number formatting and parsing
<code>pie_chart</code>	Provides a beautiful Pie Chart Widget with animations
<code>syncfusion_flutter_charts</code>	Provides data visualization by creating beautiful, animated, and high-performed charts

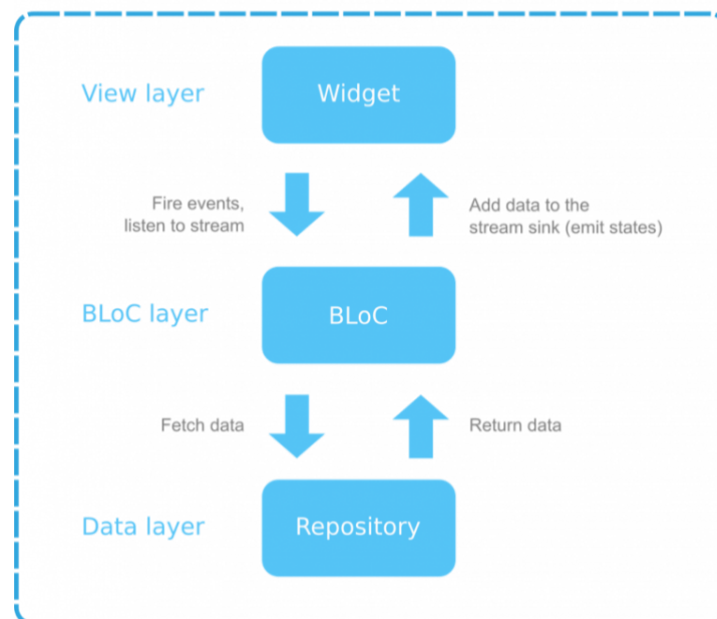
flash

Provides customizable and easy-to-use alerts to display messages

Table 14 – Brief description of packages imported to support the web development

4.1 BloC Pattern in Navigation

BLoC (Business Logic Component) pattern helped with state management to provide the navigation between pages through the sidebar clickable buttons. This pattern is similar to other suitable mobile patterns such as Model-View-Presentation (MVP) and Model-View-View-Model (MVVM) [53]. Indeed, these patterns represent a very similar form of interaction between the model side and a view, only varying the flow of communication. BloC allowed an efficient management state across the web application without a tight coupling between the view and the logic side. Figure 14 shows the details of the communication flow in the BloC pattern.

*Figure 14 – BloC pattern communication [53]*

For navigation events management associated with a navigation button, BlocProvider was used to update the navigation state. BloC is based on streams used to provide and consume a continuous data flow. The implementation focused on Events, States and the transformation of Events into States through the

mapEventToState method provided by the BloC package. Code 8 addresses the BloC pattern implementation applied in navigation.

```
1 class NavigationBloc extends Bloc<NavigationEvents, NavigationStates> {
2   @override
3   NavigationStates get initialState => MyHomePage();
4
5   @override
6   Stream<NavigationStates> mapEventToState(NavigationEvents event)
7   async*{
8     switch (event) {
9       case NavigationEvents.HomePageClickedEvent:
10        yield MyHomePage();
11        break;
12       case NavigationEvents.HistoryPageClickedEvents:
13        yield HistoryPage();
14        break;
15       case NavigationEvents.DevicesPageClickedEvents:
16        yield DevicesPage();
17        break;
18     }
19   }
20 }
```

Code 8 – Navigation BloC implementation

Similar to async keyword used in asynchronous operations, the async* is related to streams. The yield keyword works similarly to a return keyword; however, it adds a single value to the output stream. Depending on the event to be mapped, the associated Widget page is added to the output stream of the surrounding async* method through the yield. Each Widget page implements the NavigationState and each navigation button has an associated event that is added to the BlocProvider stream to notify navigation state has changed. BlocProvider is analogous to the Provider used in the mobile application as can be seen in Code 9.

```
1 BlocProvider.of<NavigationBloc>(context)
   .add(NavigationEvents.HomePageClickedEvent);
```

Code 9 – BlocProvider adding a new navigation event

4.2 Consuming Information

Web application consumes information from two different sources: REST API through HTTP requests and Message Broker consuming two message topics.

A generic service class responsible for client authentication and for the remaining requests after being authenticated was implemented. This web client consumes information from the different services provided by the REST API, and each service consumed on the client side is an extension of the implemented generic service.

The information provided by the message broker arrives distributed by topics. Identical to the BeingCare service component, an MQTT client was implemented allowing the connection between the web application and the broker in order to consume messages that are published in both topics. Whenever a message appears in the topic, this is populated in the data stream so that Flutter, later, builds Widgets based on the data stream's content.

5 Messaging Runtime

In the context of the problem, as the system focuses on an event-driven approach and the exchange of information between components, the use of a message broker is advantageous to act as a communication mediator facilitating the communication process. A message broker is a technology that enables applications, services, and systems to exchange information through messages translated by a formal messaging protocol.

5.1 MQTT Protocol

MQTT is a publish-subscribe protocol for message transport which architectures relies on Figure 15. This protocol is extremely lightweight and efficient and allows bi-directional remote communication between devices with a minimal network bandwidth [43].

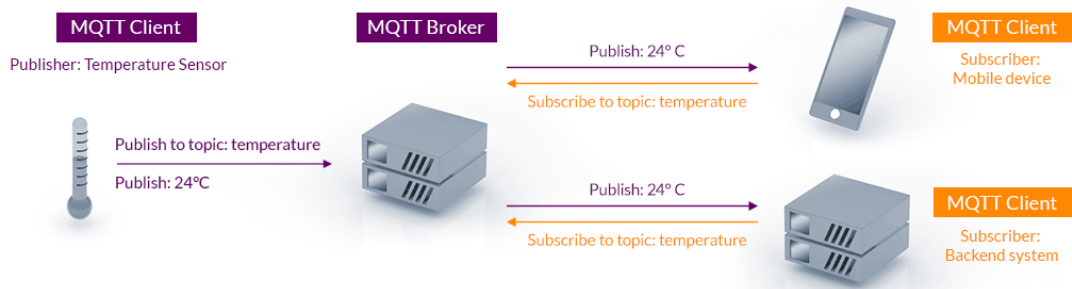


Figure 15 – MQTT publish-subscribe architecture [43]

A MQTT client can publish messages on a specific topic, which could be subscribed by other clients receiving the messages published on it. This protocol significantly reduces the network bandwidth consumption and increases scalability due to the characteristics of the publish/subscribe model.

To achieve the purposed goals, this protocol was used for messages exchanged between the components described in the architecture, whenever possible.

5.2 Mosquitto Message Broker

As MQTT broker was used the open-source Eclipse Mosquitto (version 2.0.9). The basic configurations are defined into a configuration file editable in order to specify other configurations needed in the context. Besides allowing MQTT protocol, this broker also allows the use of WebSocket protocol and the use of several listeners at the same time. The default configuration only allows MQTT protocol available on port 1883 and, since version 2.0 was released, anonymous clients connected to the broker are not authorized. For these reasons, its settings have been changed in the configuration file in order to allow, simultaneously, MQTT and WebSocket protocol in different ports. For proof-of-concept purposes, certain security aspects were discarded and therefore the connection of anonymous users was allowed.

Below in Code 10 are the configurations defined for Mosquitto broker through its configuration file:


```

1 listener 1883
2 listener 9001
3 protocol websockets
4 allow_anonymous true
    
```

Code 10 – Configurations defined in Mosquitto configuration file

Message broker clients can subscribe two topics related to the context. Table 15 describes the topics used in the system, as well as its logic context and which clients subscribe or publish on it.

MQTT Topic	Description	Operation	Client
alarms	Topic used to exchange messages of alarm type, generated from an event detected in mobile component.	Publish	Mobile
		Subscribe	Web Application
			Storage Service
requests	Topic used to send requests-related messages to web application	Publish	Storage Service
		Subscribe	Web Application

Table 15 – Description of used message topics

As defined in the Mosquitto configuration file, two different ports are used with different protocols. Each topic is available for consumption through Websocket protocol on port 9001 while the alarms topic is consumed on port 1883 by storage service through a MQTT client in order to store all messages published on it. A resume of ports and protocols available for each topic are detailed in Table 16.

Topic	Port	Protocol
alarms	1883	MQTT
requests	9001	Websocket

Table 16 – Ports and protocols available for each message topic

6 Deployment

For data privacy reasons, the system was deployed locally using docker containers. Figure 16 shows the deployment diagram where the components of the message broker, database and services are deployed in different docker containers.

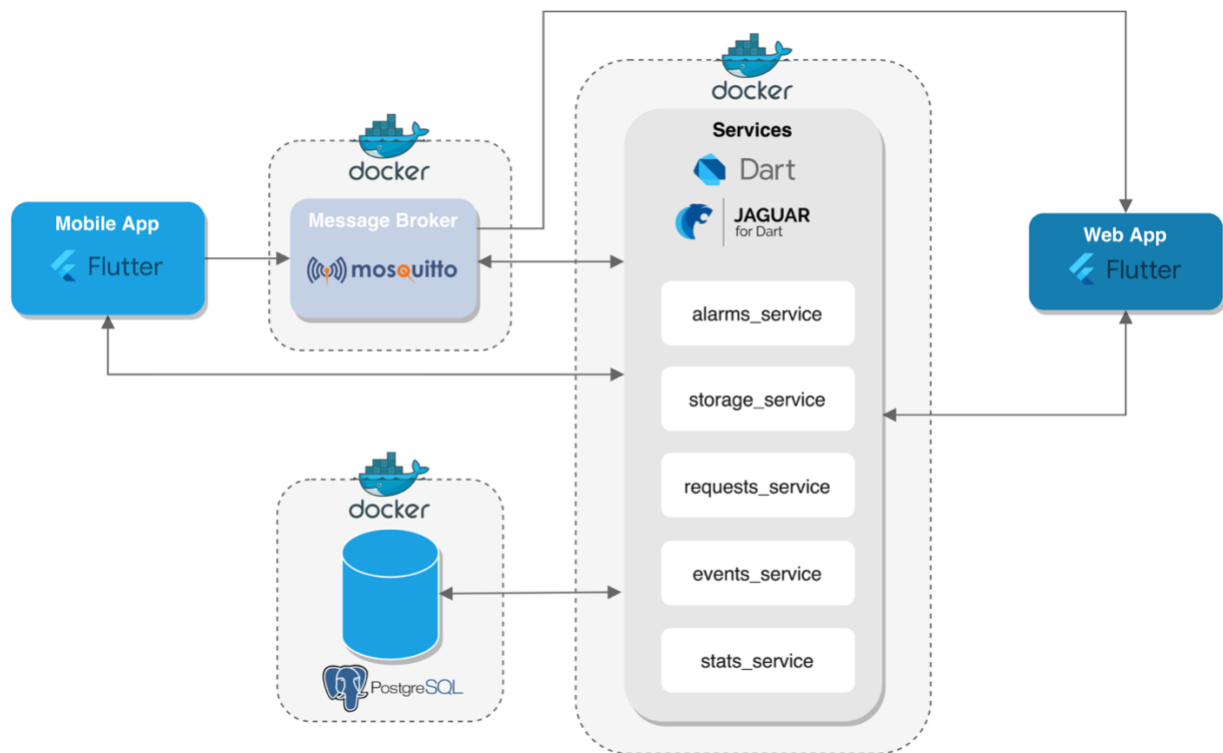


Figure 16 – Deployment diagram

Docker compose defines and runs multi-container Docker applications running in containers. In the `docker-compose.yml` file the services were defined, and for each one, it was indicated the respective image used or a pre-compilation of a Dockerfile, ports exposed, and volumes mounted.

The details specified for each service are described below:

- `postgres` – this container uses an already pre-defined PostgreSQL image available on Docker Hub, a hosted repository service provided by Docker containing containers images. The environment variables stated are related to the database access credentials as well as its name. As PostgreSQL uses the port 5432 as its official port, it was decided to keep it and expose it in order to allow mapping all requests arriving at the port to the container's port. Having created a schema with the database structure, the schema is specified on the volumes tag to prepare the database;
- `mosquitto` – like `postgres` container, this container uses also a pre-defined image available on Docker Hub. In this case, as Mosquitto message broker uses two different ports in accordance with the protocol as seen before, both two ports need to be exposed (1883 for MQTT and 9001 for Websockets). The environment variable stated specifies the host and the configuration file is specified in the volumes tag;
- `beingcare-services` – this container uses a custom image built using the Dockerfile in the `beingcare_services/.` Directory and depends on `postgres` and `mosquitto` containers already addressed.

The Appendix C contains the Docker-compose specifications and the Dockerfile with the custom image that compiles and runs the BeingCare services Dart application.

7 Reflections and Considerations on Development

During the implementation of this work, several constraints have arisen over time largely due to the novelty that Dart presents as a new programming language. Following are considerations and opinions about the work process.

7.1 Dart as Applicational Backend Programming Language

The Dart language syntax, similar to Java, is more concise and straightforward, and an example of this are prints and main functions. It resembles Python in its simplicity however the fact that it is object-oriented makes it more robust. The great advantage of Dart is the simplicity of synchronization and the range of patterns provided are easy to apply for cleaner state management.

However, compared to other application backend frameworks namely the popular Java-based Spring Boot, the Dart ecosystem is still very inconsistent. Despite the several existing frameworks like Jaguar, Angel, Aqueduct among others, these lack still a clear community support for now. This was clearly reflected in progress of BeingCare development. Initially, Aqueduct framework was adopted however, as soon the released of Flutter 2, Aqueduct did not keep up with language changes and, in the end discontinued at the end of March. Besides this downside, Aqueduct provided some valuable features that were already being using namely the ORM and the automatically API documentation.

The second option was the Jaguar framework. Jaguar framework has positive aspects namely in the development of REST APIs and the routes mapping is more intuitive and effective thanks to the annotations available when compared with Aqueduct. Yet, its ORM support is not functional in contrast with ORM provided by Aqueduct. Aqueduct version worked perfectly, automatically creating the necessary migrations to build the database schema.

With Jaguar, the ORM was dropped, and it were ended up in a classical SQL handmade based approach in model creation. To mapping to objects, queries were also custom made. Despite some interesting integrations with Jaguar from a developer perspective its documentations are quite disorganized and flawed.

Jaguar has the advantage of being very consice and intuitive for write REST APIs. The endpoint creation process works based on self-explanatory annotations allowing greater control to the developer through path configurations, path parameters, among others.

Table 17 summarizes some relevant considerations about the functionalities provided by each Dart-based applicational backend framework explored during this dissertation.



ORM	Consistent and functional only for PostgreSQL	Not functional
Documentation	Guides well detailed	Disorganized and flawed
Community Support	More considerate due to the number of users	Sparce
REST APIs Documentation	Automatically and integrating Swagger	Not supported

Table 17 – Comparison between Aqeduct and Jaguar frameworks in terms of features provided and issues encountered

Along the development, although not particular to Dart, the quick evolution of Dart lead to constant issues with package incompatibilities due to versioning, several time indirect ones not fully clear from the start. It is natural to assume this will stabilize but currently it is an obstacle in the development process.

There is still a discrepancy between packages released by Google and packages provided by Dart developers open community. If the development process is constrained mostly to Google released packages besides being well documented, they are clearly more reliable them packages provided by the community. The community’s efforts are notorious, but certain aspects need to be enhanced. Nonetheless, there are renowned IT companies that have started to provide consistent packages as well.

7.2 Flutter in Web

Flutter Web was only released for the stable kernel with the fresh Flutter 2 announcement. Only after this release, the lack of packages available supporting web compared to mobile began to be surpassed.

As the screen of a web application is significantly larger than the screen of a mobile device, sometimes the web application will have more content per window, i.e., there trends to be a greater number of

Widgets to be displayed in a single window. This factor can lead to more extensive and complex code if the developer does not reuse or organize the code well. Flutter has the advantage of granting the creation of reusable and customizable Widgets which can be applied in different contexts depending on the input data passed to them. Indeed, the messiness caused by the complexity of many Widgets can be overcome by adopting the referred conducts.

7.3 Tensorflow and Physical Devices

It is important to emphasize there were constraints related to Tensorflow Lite support on emulators (both on Android and iOS emulators). Recently, Tensorflow stopped supporting debugging and testing on emulators with x86_64 architectures due to, mainly, a CocoaPods issue. CocoaPods is a dependency manager for Objective-C, Swift, and other related languages that allows managing external libraries in a standard format. Tensorflow Lite dependencies are managed by CocoaPods on iOS devices, however, the Tensorflow community decided to withdraw support for any type of emulator. This constraint can be surpassed by using physical devices to test and debug applications that use this package or packages dependent on this. As the Tensorflow Lite package used also depends on Tensorflow, applications will crash when using virtual devices (both in Android and iOS).

7.4 Operating Systems

Flutter is officially supported for both Android and iOS. During this dissertation, two operating system-related obstacles were discovered:

- iOS mobile development by itself is quite restricted;
- the release of stable packages for native iOS apps takes longer.

Besides the development and deployment for Android being simple and effortless, the same does not happen in iOS. Apple is very strict about mobile development and the complexity with iOS is mainly due to Apple restrictions but not from Flutter itself.

With iOS, the Flutter development is coupled to the actual Apple development and deployment toolkit and the process of releasing iOS applications requires enrollment in the non-free App Developer Program. A non-re-enrollment poses limitations e.g., impossibility to generate IPA (iOS App Store Package) files and

publish apps on the App Store. For this reason, besides the BeingCare mobile application being functional on iOS devices (tested using the debug mode) its IPA file which stores an iOS app was not generated.

Another issue faced with iOS is that evolutions in Flutter may take some time to be fully covered by Flutter iOS native apps. Some packages especially provided by the Dart community, sometimes in a first release/update have bugs related to the iOS component.

Also, during the proof-of-concept development phase, there was a greater difficulty in the iOS component compared to the Android component. Settings about the developer's authenticity must be considered and when it is not well defined, the operating system does not authorize the debugging of applications. Furthermore, initially there were founded bugs related to the plugins used, namely the Tensorflow Lite, since the developer team sometimes release new versions without testing their operation on iOS. This fact is due to the difficulty of testing on iOS because these tests require the use of an iPhone or an emulator that is only supported on Apple computers.

Chapter 5 Results Analysis

The pandemic, namely in relation due to interpersonal restrictions, made unpractical the conduction of system usability tests in scenarios related to wellbeing and care support for people – the options of using volunteers to perform were excluded from the start as they are not the typical target population. Therefore, the BeingCare system evaluation focus on the UI and audio classification using local ML. Firstly, the user interfaces developed for each interaction point of the system are presented and explained. Regarding the machine learning model, issues related to the variation of the size of mobile application, are also analyzed, as well as an evaluation related to the detection of audio varying input parameters and distances between the audio source and the recognition device.

1 Mobile Application Demonstration

The mobile application has been designed considering that it could be used by elderly people or people with motor constraints. Thus, the application was built based on simplicity to be intuitive and provide an easy user experience. Figure 17, Figure 18, and Figure 19 show the three screens created, running on Android devices.

The main screen contains a kind of interactive command that, depending on the clicked button, sends the associated message to the system. Thus, an elderly person who is not used to interacting with smartphones can easily communicate with the other side of the system.

The mobile application also supports iOS devices and as seen in Figure 20, Figure 21 and Figure 22, the design is identical since the used code base is the same. However, Flutter contains a panoply of native Widgets of each operating system that can be used instead of the Widgets provided by the Material Design.

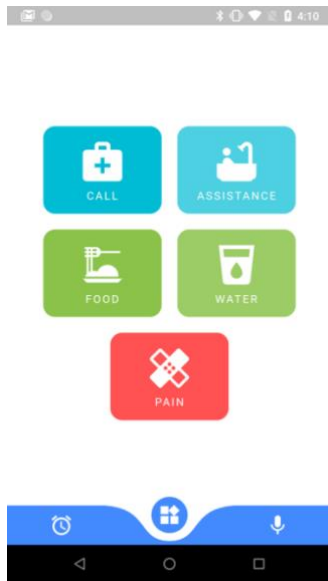


Figure 17 – Main screen on Android devices

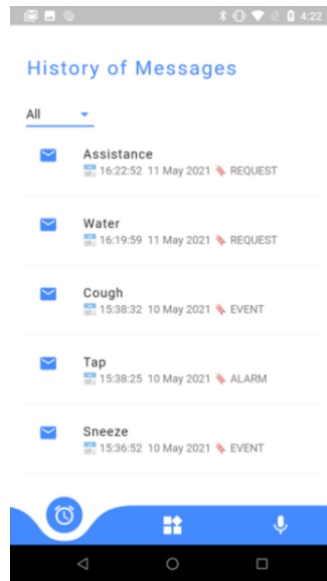


Figure 18 – History of sent messages on Android devices

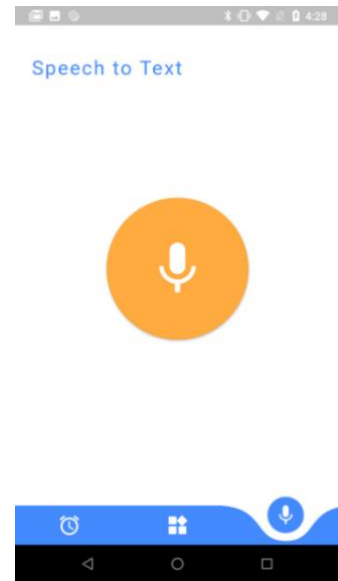


Figure 19 – Speech to text screen on Android devices



Figure 20 - Main screen on iOS devices

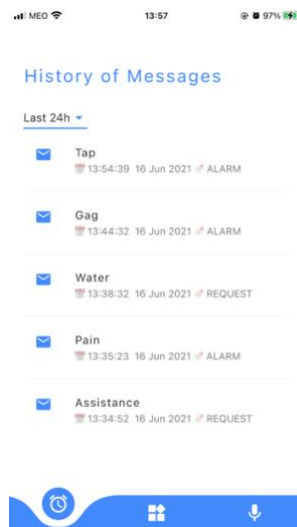


Figure 21 - History of sent messages on iOS devices

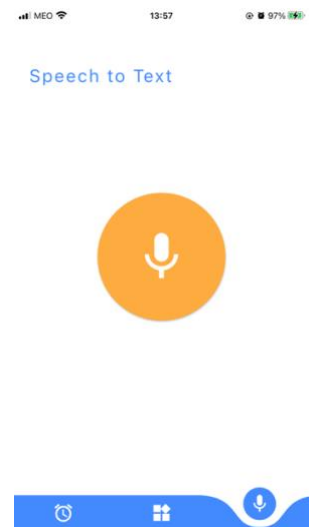


Figure 22 - Speech to text screen on iOS devices

2 Web Application Demonstration

The web application was specially designed so that caregivers can have a visual idea of the overall events happening or already happened. Therefore, at the homepage shown in Figure 23, there are presented graphs in order to summarize the percentage of each type of information and the evolution of these types over the last 24 hours. When critical people being cared-related events occur, the caregivers receive a notification with a description of the event and who rises that alert.

Caregivers can access individual information related to each people being cared with de UI represented in Figure 24 containing a chart of all records separated by type and occurred in the last 24 hours accumulated per hour.

Finally, a raw history of all records stored in the system, divided by their type, is presented sorted by recent date allows caregivers to have a gross information, as seen in Figure 25.



Figure 23 – Web application homepage

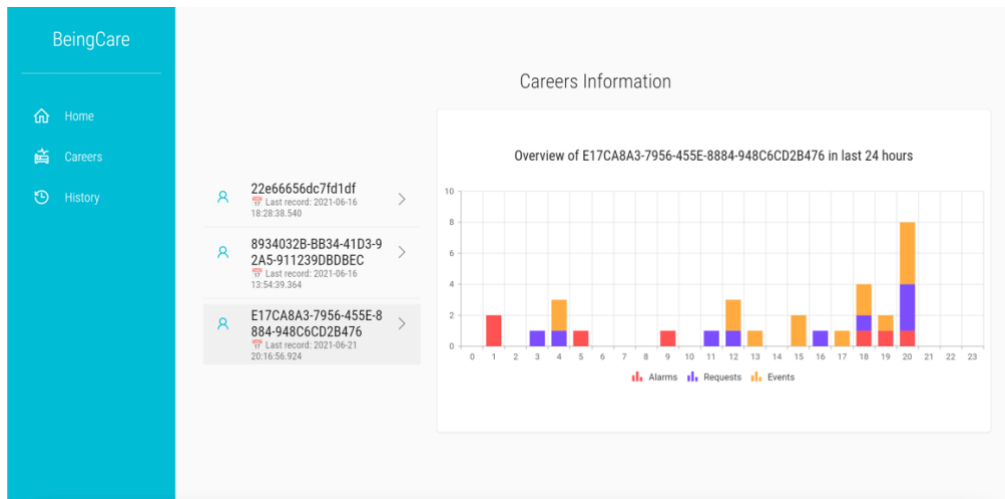


Figure 24 – People being cared list and individual overview of events that occurred

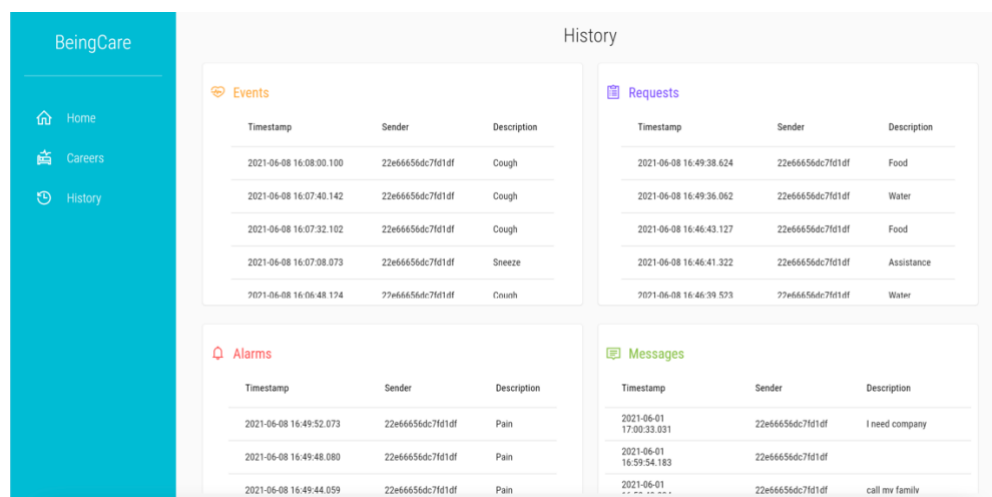


Figure 25 – History board of all information organized by type

3 Mobile Application Size Analysis

The application size is an important factor to be discussed to understand, mainly, the role of machine learning mechanisms related to the memory occupied in devices. Despite the use of a cross-platform framework in mobile development, Flutter enables the generation of application native files for each operating system. An analysis of the size of these files will be presented below.

3.1 Releases

As Android devices use various CPUs with different characteristics that support different instructions set, the generation of an APK that works for all platforms can cause problems regarding the size of APK. To fight this constraint, there were generated APK files per ABIs (Application Binary Interface) using the `–split-per-abi` flag provided by Flutter. Table 18 analyzes the size of each APK generated according to the inclusion or exclusion of the classification model.

APK file generated	Size with classification model	Size without classification model
app-armeabi-v7a-release.apk	29,9 MB	24,2 MB
app-arm64-v8a-release.apk	33,4 MB	27,6 MB
app-x86_64-release.apk	37,2 MB	31,4 MB

Table 18 – Comparison APK sizes between releases generated for all the targets ABIs

Using the APK `app-arm64-v8a-release.apk`, it was installed in several Android devices the BeingCare application including and excluding the classification model and the comparison is reported in Table 19. Due to the constraints related to the limitations of iOS releases discussed in 7.4, the generation of the application native file (IPA) was prevented. Nevertheless, comparisons related to the native application size with and without the classification model have been established assuming the iOS native application installed during debug mode, and the results are as well depicted in Table 19.

Device	Operating System	Size with classification model	Size without classification model
LG Nexus 5	Android	121 MB	116 MB
Xiaomi Redmi Note 5	Android	131 MB	125 MB
Samsung Galaxy A21s	Android	129 MB	123 MB

iPhone 7	iOS	318 MB	312 MB
iPhone 7 Plus	iOS	318 MB	312 MB

Table 19 – Android and iOS native applications’ size comparison

Some documentations [46] refer that GTM models may increase the application’s size. Analyzing the results presented in Table 18, comparing the size of APK generated per ABIs, it is concluded the GTM classification model adds up to a maximum of 5.8 MB (4% in worst case). This value coincides with the size of the Tensorflow Lite classification model file generated by GTM framework and which is added to the application’s assets. As GTM requires select Tensorflow operators to work, this can increase the overall application’s size and it is recommended the built of an own custom classification model to overcome this constraint [46]. Therefore, it is expected that building and training an own Tensorflow Lite model without the help of GTM will decrease the size of the APK because of the size of the model will be smaller.

The relation between the model size (5.8 MB) and the size of the application installed on Android devices is equal to the relation discussed above. The results obtained and exposed in Table 19 reinforce both native applications (Android and iOS) which include a classification model are about 5.8 MB weightier than applications without the model.

Regarding Android native applications, the behavior of the size of the installed applications varies according to the device itself. In the worst scenario, this variation is about 10 MB, comparing LG Nexus 5 and Xiaomi Redmi Note 5. On the other hand, iOS native applications, regardless of the device model, installed applications are the same size.

Figure 26 represents a visual comparison between Android and iOS native applications’ size including and excluding the classification model. For Android application size was assumed the average of sizes obtained and referred in Table 19.

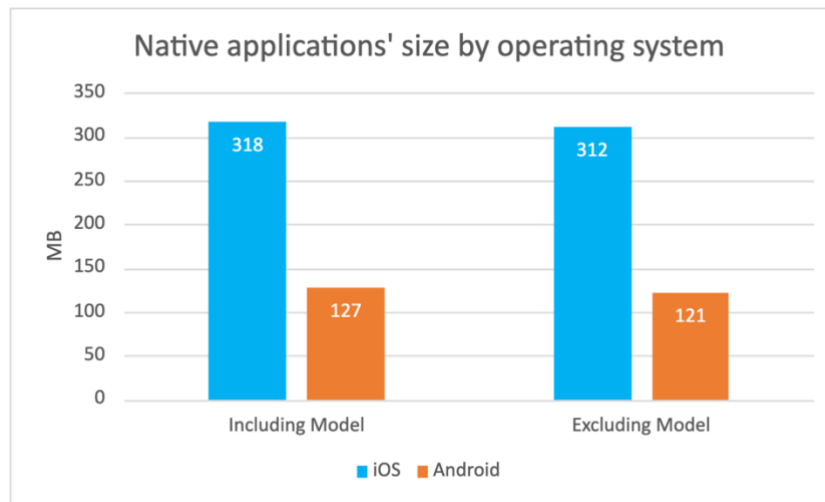


Figure 26 - Visual comparison of the native applications' size between operating systems

Overall, it's noticeable that native iOS applications are significantly heavier than Android apps, around 60% at least.

Another interesting observation, resulting from the analysis of the classification model's size, concerns the fact that its size remains constant, i.e., regardless of the number of the classes' model and its samples, the model generated by GTM will always have 5,8 MB.

4 Audio Classification Analysis

In this section, several tests are presented as a starting point for analyzing the performance of the classification model under different scenarios, as well as discussed obtained results that are considered curious and pertinent. As mentioned before, the use of a GTM model with Tensorflow Lite, while allowing easy and straight forward model creation, the GTM black box design did not allowed a model analysis.

4.1 Model Overview

GTM framework generates Tensorflow Lite models already trained. The model built using GTM framework was based on a balanced dataset where each class contains about 14 samples, splitting the dataset into two buckets:

- **Training samples:** corresponding to 85% of the samples and are used to train the model in order to learn the classification of the samples into the classes defined
- **Test samples:** corresponding to 15% of the samples that are never used to train the model but to evaluate the performance of on new samples of never-before-seen data

The model was trained with 50 epochs, which means that model works through the entire training dataset 50 times. Below, in Figure 27, are the charts that analyze the accuracy (metric used to evaluate or monitor the performance of the model during testing and training, respectively) and the loss (function to optimize the model’s parameters) of the model according to the course of the epochs.

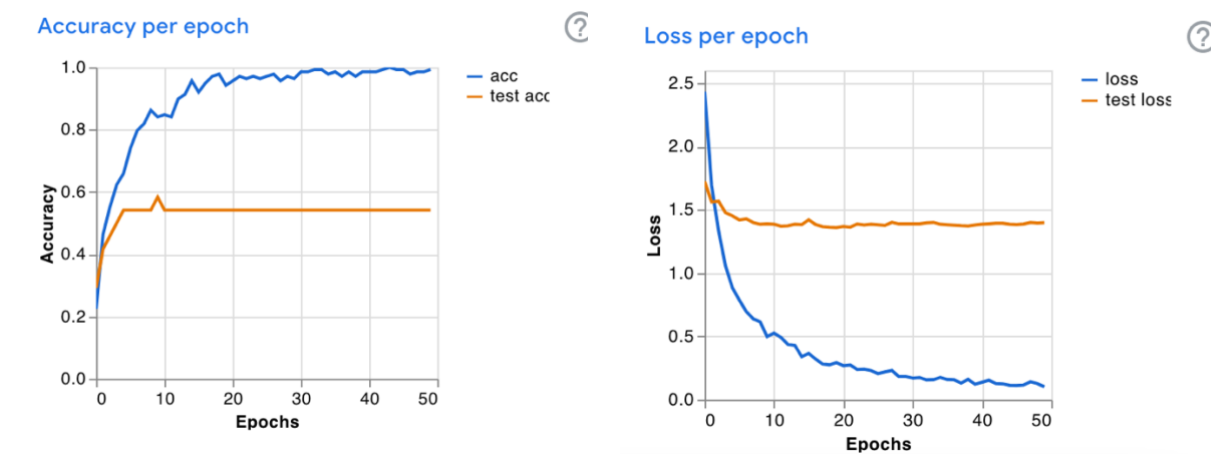


Figure 27 – Evolution of model accuracy (left) and model loss (right) depending on the epochs elapsed

An accuracy close to one means that the model is predicting correctly and, according to the previous graph, the accuracy of the model, in training data, increases along with the epochs. The predictions of the model are considered perfect if the loss is zero, otherwise, the loss is greater than zero. In this case, after 50 epochs, the trained model achieves a loss a little bit greater than zero.

4.2 The Distance’s Impact

The experimental objective is to analyze the impact of the audio distance on the performance of the classification model.

Thus, the distance impact assessment was performed based on random samples taken from the initial dataset. For each class, 3 random samples were randomly repeated into a sequence and 40 inferences, i.e., audio events, were drawn from it. For that, the input parameters of the recorder method, sample rate and buffer size, were 44100 and 22000, respectively.

As those 360 inferences were presented to the classification model at different distances between the model's device and the audio source, this procedure leads to two different test sets. Notwithstanding that the samples randomly used to create the audio sequence, were used by the learning algorithm when training the model, it is also worthwhile to mention that, by randomly drawn the audio events from the sequence and by varying the distance between the model's device and the audio source, these test sets were not entirely used in the training. Hereupon, these test sets are related to the trained concept and the resemblance of these new test sets with the set for training the classification model is not overwhelming.

Therefore, and considering that the aim is to analyze the impact of the audio distance rather than generalize on the performance of the model, these test sets fit the purpose.

As stated, these experiments were performed varying the positioning distance of the mobile device that runs the model to classify the events from the audio source that produces the events and were executed on the Android mobile device LG Nexus 5.

With the results obtained, a simple Python script was implemented to obtain the confusion matrix (a tabular summary denoting the visualization of correct and incorrect predictions returned by the classification model) and evaluate the model performance through the computation of the following metrics: accuracy, precision, recall, and F1 score.

The confusion matrices with the results from both experiments are presented in Figure 28.

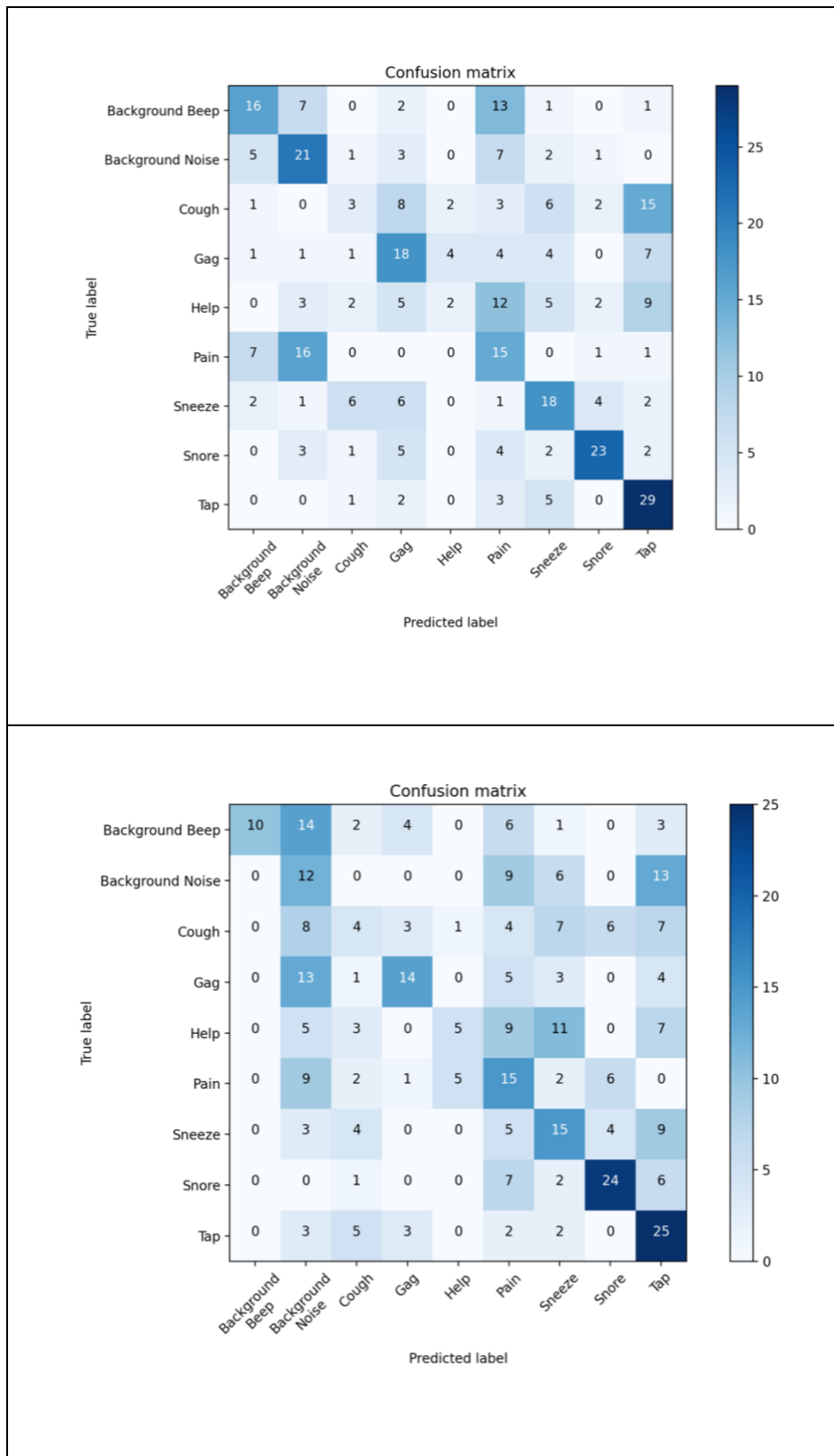


Figure 28 – Confusion matrix obtained from 40 inferences with distances of 0.5 meters (top) and 1 meter (bottom) tested with random samples used to build and train the classification model

The confusion matrices obtained allow concluding that, to some extent, the model can correctly predict the event that occurred. However, as the distance increases, the decrease in the number of hits is notorious, eventually assigning a proximal class to the event other than the true one (background beep is, majority, classified as background noise). At first glance, it is notorious that, regardless of the distance, the model presents a higher performance for short and simple sounds, as is the case of taps, as well for sonorous sounds such as snore. There are some cases where the model confuses classes that have identical sounds, namely, the gagging sound is very similar to the coughing sound. A person who gags ends up coughing involuntarily. At a shorter distance, the cough is mostly classified as gag, while at a higher distance as background noise, pointing the audio quality deteriorates. Another important conclusion is related to the Help class. This class is composed of audio samples that comprise help request words. Compared to the other classes, it is clear that the model is not able to infer well this type of class, but, surprisingly, at a greater distance, the classification of this class is better.

Comparing both confusion matrices, it is clear the distance between the mobile device and the audio source has an impact on classification results and longer distances negatively affect the correct classification. Also, the performance metrics endorse this conclusion. As the problem addresses multiclass, for a deeper analysis on the model performance, for each class was calculated the accuracy, precision, recall, and sensitivity resorting the Python's module `sklearn`:

- **Accuracy** – returns the fraction of the total samples that were correctly classified;
- **Precision** – returns the fraction of correct predictions and all the predictions of a given class;
- **Recall or sensitivity** – returns the fraction of correct predictions and all the examples that belong to that class;
- **F1 score** – weighted arithmetic mean of recall and precision.

In addition, macro average metrics were also calculated, and the results are shown in Table 20 (weighted average metrics are omitted as the results are equal to macro average metrics due to the equal support of each class).

	0.5 meters			1 meter			Support
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	
Background Beep	0.500	0.400	0.444	1.000	0.250	0.400	40
Background Noise	0.404	0.525	0.457	0.179	0.300	0.224	40
Cough	0.200	0.075	0.109	0.182	0.100	0.129	40
Gag	0.367	0.450	0.404	0.560	0.350	0.431	40
Help	0.250	0.050	0.083	0.455	0.125	0.196	40
Pain	0.242	0.375	0.294	0.242	0.375	0.294	40
Sneeze	0.419	0.450	0.434	0.306	0.375	0.337	40
Snore	0.697	0.575	0.630	0.600	0.600	0.600	40
Tap	0.439	0.725	0.547	0.338	0.625	0.439	40
Accuracy							360
Macro Average	0.391	0.403	0.378	0.429	0.344	0.339	360

Table 20 – Classification report of performance metrics calculated per class using the random samples testing approach

The classification report indicates the accuracy of each class varies dramatically. For example, the Pain class has a very depressed F1 score (0.242) compared to the Snore class (0.697), regardless of the distance. The F1 score obtained for each class conveys interesting results from classes where distance does not affect performance as in classes Background Beep, Cough, Gag, Pain, and Snore. It remains to notice the surprising improvement in the performance of the Help class with increasing distance, however, the performance is still quite low.

The overall model accuracy in the best-case scenario, which is at a smaller distance, is about 40% indicating the model misclassifies 60% of the events that occurred. Accuracy is further negatively affected

by increasing the distance, as it is decreased by 6%. These results prove that the performance of the classification model is not perfect and the need to be improved is obvious, but that distance does not pose a major burden.

To improve the overall performance of the classification model, more examples should be considered during the learning step. The model built using GTM framework was based on a balanced dataset, where each class has about 14 samples. This number is considered low since only 85% of these 14 samples are used to build the model and only 15% of samples are used for training. This means that there was no wide variety of samples used during the training process, which drastically affects the results. The poor performance of the classification model does not result from an unbalanced dataset and most possibly can be related to the lower number of samples, the similarity between the samples assigned to different class or the algorithm cannot distinguish between classes. Thus, it is expected that performance may increase if a larger number of samples per class is used, and these should be most distinct from each other.

4.3 Input Parameters

Another interesting analysis consists in varying some input parameters required by the Tensorflow package during audio recording. The recording length must be equal to the tensor input size expected in the model, i.e., 44032.

According to the package's documentation, it is recommended the use of 16000, 22050, or 44100 for sample rate, corresponding, respectively, to 16 kHz, 22 kHz and 44,1 kHz. In order to analyze the behavior of the audio recording time window, the sample rate values were varied along with the buffer size values and the results are registered below. For testing purposes, two devices with different operating systems and hardware components were used, LG Nexus 5 (Android) and iPhone 7 Plus (iOS), and the results are expressed in Table 21 and Table 22, respectively.

Sample Rate	Buffer Size			
	2000	8000	16000	22000
16000 Hz	3.059 s	3.254 s	3.298 s	3.698 s
22050 Hz	2.365 s	2.408 s	2.443 s	2.775 s
44100 Hz	1.448 s	1.350 s	1.328 s	1.546 s

Table 21 - Audio recording temporal window (in seconds) to be inferred on Android devices depending on sample rate and buffer size passed as input parameters

Sample Rate	Buffer Size			
	2000	8000	16000	22000
16000 Hz	2.583 s	2.712 s	2.739 s	2.956 s
22050 Hz	1.976 s	2.012 s	2.172 s	2.306 s
44100 Hz	1.032 s	0.803 s	0.785 s	1.123 s

Table 22 - Audio recording temporal window (in seconds) to be inferred on iOS devices depending on sample rate and buffer size passed as input parameters

The temporal window is an important factor not only for the model's behavior but also for the system application context. Scarce windows may not properly capture the event that is occurring and, consequently, cause a poorly classification performance. Comparing both results obtained for different mobile devices, it is noticeable that devices with a better-quality processor speed up the recording time.

To sum up, buffer size influences the time window of audio captured for inference. A lower value increases the time window however, it is computationally heavier due to the load exerted on the device's processor. There is an exception to the rule for a buffer size of 22000, which proves to be the ideal value so as not to raise computational forces for an ideal time window. It is expected that the variation of sample rates will influence the accuracy of the inference and higher values will improve the accuracy.

Chapter 6 Final Remarks

The conclusions and final considerations about the encouraging results obtained in this work are summarized during this chapter and future work is purpose in order to refine and improve the presented developed system.

1 Conclusions

The focus of this Dissertation consisted in the development of a proof-of-concept for a system capable of surpassing some communications constraints raised in some wellbeing-related scenarios and improving the direct interaction between the humans involved. It is considered that the work covered throughout this document has successfully reached all previously defined objectives, validating the concept.

The main objective was the development of a Flutter-based proof-of-concept system addressing mobile technologies to integrate part of the solution in restricted communication scenarios. This objective was successfully fulfilled due to the development of a mobile application that gathers the necessary conditions for clean and clear communication with the monitoring support. Flutter cross-platform framework allowed portability between operating systems through a fast and efficient development. Flutter follows a reactive programming approach increasing the performance by handling huge volumes of data in a quick and stable way.

Audio detection granted the analysis of verbal communication through the recognition of several audio sounds characteristic of the health and wellbeing of individuals for detecting not only critical but also non-critical events. For this, the benefits of Machine Learning allowed the creation of an audio classification model, being possible its successful integration in mobile devices through specific frameworks, in this case, the Tensorflow Lite. Tensorflow Lite framework guaranteed the processing of audio locally without

its exposure outside the mobile device, safeguarding the user's privacy according to the RDPG rules. Speech to text techniques that allowed the sharing of non-critical information were also addressed.

The possibility of developing an applicational backend using the Flutter's base language, Dart language, was investigated and the entire support backend system was implemented using the Dart-based framework Jaguar. Backend application development is feasible despite some verified limitations. However, given the evolution felt in the satisfaction of the use of Flutter accompanied by the Dart language, and with its growth, it is believed that the support for the applicational backend development will stabilize as well as the range of functionalities will increase according to the developers' needs.

During the process of implementing the work, several constraints were faced due to the novelty of the frameworks addressed. Relevant reflections and considerations from a technological point of view were also discussed and some comparisons between programming languages and frameworks were established.

Due to the current pandemic situation, it was impossible to perform system usability tests in limited communication or care support scenarios. Thus, the discussion of results focused on the size of the native applications generated for each operating system and on the classification model. The process of integrating models in mobile applications through Tensorflow Lite is quite accessible, allowing the insertion/update of new files with the model.

As result, the native applications' sizes had different sizes depending on the operating system and the weight of the classification model within the application is considered ridiculous given the overall applications' size. iOS applications installed are weightier than Android although its content is pretty much the same.

In addition, tests were conducted on the classification model in order to assess the impact of distance on the classifications obtained. According to the calculated performance metrics, it is concluded that the performance of the classification model used is not ideal, which may be due to the limited number of examples in each class, during the training, and to the fact that there is low variability in the subjects that produce the examples of audio.

Despite these limitations of the classification model, the proof-of-concept system has been successfully demonstrated, indicating that it is possible to easily integrate ML models into mobile applications capable of processing data locally using Tensorflow Lite. In addition, the concept addressed by Tensorflow Lite

proved to be an asset to the BeingCare system as it avoids exposing people being cared' data outside of mobile devices.

2 Future Work

The main objective of this dissertation consisted in the development of a proof-of-concept of the BeingCare system. The pandemic we are facing has limited the carrying out of tests and experiments in a real context of care support scenarios, avoiding contact with people sometimes considered at risk in the context of the problem. Therefore, it is encouraged that in the future the system is tested in the appropriate context in order to adapt it to the needs of more limited users.

It is clear that the system needs some improvements, in particular related to the ML component. The classification model used did not present an ideal performance, so it would be interesting to train the model with a broader dataset. Also, according to [46], using GTM models may increase the mobile application's size and it is recommended the creation of an own Tensorflow Lite model, more lightweight, to decrease the memory affected.

With the results obtained in the analysis of the distance's impact on the classification of events, it would be interesting to create several models from audio at different distances, varying their noise, to make the classification process closer to reality.

In the context of illness, the health condition can change quickly and seriously compromise the quality of the audio. It is advisable to create a poll of models in order to be able to choose the one that presented the best performance. Moreover, in these scenarios, if none of the models in the poll presented a suitable performance, it would be interesting to adapt to these cases, training a new model with new data from this context.

The implementation of features that aim to fill the system is encouraged, such as:

- Fall detection
- Breath and sleep analysis – may include detection of apnea situations
- Inclusion of more languages (speech to text also supports native Portuguese)

In addition, a study on the percentage of CPU or battery consumed during the background audio classification process is encouraged.

Finally, increasing the personal data associated with each people being cared in accordance with RGPD rules and mindset, namely age and gender, creating clusters to obtain new conclusions associated with the status of people being cared, would be an interesting approach to be addressed.

References

- [1] SkillsYouNeed, "What is Communication? Verbal, Non-Verbal & Written | SkillsYouNeed." <https://www.skillsyouneed.com/ips/what-is-communication.html> (accessed Nov. 22, 2020).
- [2] D. Phutela, "The Importance of Non-Verbal Communication," *IUP J. Soft Ski.*, vol. IX, No. 4, pp. 43–49, 2015.
- [3] H. Wanga, T. Joseph, and M. B. Chuma, "Social Distancing : Role of Smartphone During Coronavirus (COVID – 19) Pandemic Era," *Int. J. Comput. Sci. Mob. Comput.*, vol. 9, no. 5, pp. 181–188, 2020.
- [4] S. Majumder and M. J. Deen, "Smartphone sensors for health monitoring and diagnosis," *Sensors (Switzerland)*, vol. 19, no. 9, pp. 1–45, 2019, doi: 10.3390/s19092164.
- [5] MEMSCAP, "MEMSCAP | Invasive blood pressure." <http://www.memscap.com/applications-and-market-segments/medical-and-biomedical/invasive-blood-pressure> (accessed Dec. 04, 2020).
- [6] A. S. Meidert and B. Saugel, "Techniques for Non-Invasive Monitoring of Arterial Blood Pressure," *Front. Med.*, vol. 4, no. JAN, p. 231, Jan. 2018, doi: 10.3389/fmed.2017.00231.
- [7] M. J. Zhao, A. R. Driscoll, S. Sengupta, N. T. Stevens, R. D. Fricker, and W. H. Woodall, "The effect of temporal aggregation level in social network monitoring," *PLoS One*, vol. 13, no. 12, pp. 1–21, 2018, doi: 10.1371/journal.pone.0209075.
- [8] N. Almaadeed, M. Asim, S. Al-Maadeed, A. Bouridane, and A. Beghdadi, "Automatic detection and classification of audio events for road surveillance applications," *Sensors (Switzerland)*, vol. 18, no. 6, p. 1858, Jun. 2018, doi: 10.3390/s18061858.
- [9] M. Alhussein and G. Muhammad, "Voice pathology detection using deep learning on mobile

- healthcare framework,” *IEEE Access*, vol. 6, pp. 41034–41041, Jul. 2018, doi: 10.1109/ACCESS.2018.2856238.
- [10] M. S. Hossain, G. Muhammad, and A. Alamri, “Smart healthcare monitoring: a voice pathology detection paradigm for smart cities,” *Multimed. Syst.*, vol. 25, no. 5, pp. 565–575, 2019, doi: 10.1007/s00530-017-0561-x.
- [11] J. Kim, A. S. Campbell, B. E.-F. de Ávila, and J. Wang, “Wearable biosensors for healthcare monitoring,” *Nat. Biotechnol.*, vol. 37, no. 4, pp. 389–406, Apr. 2019, doi: 10.1038/s41587-019-0045-y.
- [12] M. Sreejith, S. Rakesh, S. Gupta, S. Biswas, and P. P. Das, “Real-time hands-free immersive image navigation system using Microsoft Kinect 2.0 and Leap Motion Controller,” in *2015 5th National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics, NCVPRIPG 2015*, Jun. 2016, pp. 1–4, doi: 10.1109/NCVPRIPG.2015.7489999.
- [13] D. Alymkulov, “Desktop Application Development Using Electron Framework Native vs. Cross-Platform,” South-Eastern Finland University of Applied Sciences, 2019.
- [14] P. Flach, *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*, 1st ed. Cambridge, UK: Cambridge University Press, 2012.
- [15] “Common ML Problems | Introduction to Machine Learning Problem Framing.” <https://developers.google.com/machine-learning/problem-framing/cases> (accessed Jun. 24, 2021).
- [16] M. De Donno, K. Tange, and N. Dragoni, “Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog,” *IEEE Access*, vol. 7, pp. 150936–150948, 2019, doi: 10.1109/ACCESS.2019.2947652.
- [17] M. Martinez and S. Lecomte, “Towards the Quality Improvement of Cross-Platform Mobile Applications,” *Proc. - 2017 IEEE/ACM 4th Int. Conf. Mob. Softw. Eng. Syst. MOBILESoft 2017*, pp. 184–188, 2017, doi: 10.1109/MOBILESoft.2017.30.
- [18] JetBrains, “The State of Developer Ecosystem in 2020 Infographic | JetBrains: Developer Tools for Professionals and Teams,” 2020. <https://www.jetbrains.com/lp/devecosystem-2020/> (accessed Nov. 26, 2020).

- [19] Statista, “Cross-platform mobile frameworks used by global developers 2020,” 2020. <https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/> (accessed Nov. 25, 2020).
- [20] B. Eisenman, *Learning React Native: Building Native Mobile Apps with JavaScript*, 1st ed. O’Reilly Media, 2015.
- [21] “Kotlin for Cross-Platform Mobile Development | Kotlin Multiplatform Mobile.” <https://kotlinlang.org/lp/mobile/> (accessed Apr. 07, 2021).
- [22] “Kotlin Multiplatform Mobile Goes Alpha | The Kotlin Blog.” <https://blog.jetbrains.com/kotlin/2020/08/kotlin-multiplatform-mobile-goes-alpha/> (accessed Apr. 07, 2021).
- [23] KotlinLang, “Welcome to Kotlin hands-on.” [https://play.kotlinlang.org/hands-on/Full Stack Web App with Kotlin Multiplatform/01_Introduction](https://play.kotlinlang.org/hands-on/Full%20Stack%20Web%20App%20with%20Kotlin%20Multiplatform/01_Introduction) (accessed May 05, 2021).
- [24] “Flutter - Beautiful native apps in record time.” <https://flutter.dev/> (accessed Apr. 07, 2021).
- [25] Statista, “Mobile app revenues 2014-2023.” <https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/> (accessed May 04, 2021).
- [26] “Machine Learning in Mobile Applications.” <https://theappsolutions.com/blog/development/machine-learning-in-mobile-app/> (accessed May 27, 2021).
- [27] KeyUA, “How to Use Machine Learning (ML) in Mobile Applications | KeyUA.” <https://keyua.org/blog/how-to-use-machine-learning-in-mobile-applications/> (accessed May 04, 2021).
- [28] Zain Sajjad, “Comparing Mobile Machine Learning Frameworks,” *Heartbeat*. <https://heartbeat.fritz.ai/comparing-mobile-machine-learning-frameworks-f8d498c2562a#e3f3> (accessed May 05, 2021).
- [29] “Fritz AI | Machine Learning for iOS, Android, SnapML.” <https://www.fritz.ai/> (accessed May 05, 2021).

- [30] "Numericcal." <https://www.numericcal.com/> (accessed May 05, 2021).
- [31] F. Barata, K. Kipfer, M. Weber, P. Tinschert, E. Fleisch, and T. Kowatsch, "Towards device-agnostic mobile cough detection with convolutional neural networks," Jun. 2019, doi: 10.1109/ICHI.2019.8904554.
- [32] F. Al Azzo, A. M. Taqi, M. Milanova, and F. Al-Azzo, "Human Related-Health Actions Detection using Android Camera based on TensorFlow Object Detection API," *Artic. Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 10, 2018, doi: 10.14569/IJACSA.2018.091002.
- [33] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, 2000.
- [34] "Stack Overflow Developer Survey 2020." <https://insights.stackoverflow.com/survey/2020#most-loved-dreaded-and-wanted> (accessed Jun. 21, 2021).
- [35] "Top 10 backend frameworks | Which is the best option for you? | Back4App Blog." <https://blog.back4app.com/backend-frameworks/> (accessed May 04, 2021).
- [36] Javasterling, "Spring Boot Vs Quarkus - Difference Between Two Microservices." <https://javasterling.com/spring-boot/spring-boot-vs-quarkus/> (accessed May 05, 2021).
- [37] "Jakarta EE Working Group: the Future of Cloud Native Java Applications | The Eclipse Foundation." <https://jakarta.ee/about/> (accessed Jun. 21, 2021).
- [38] J. Lee, "Which factors affected users' decisions to adopt Flutter? — Q1 2021 user survey results," *Medium*, 2021. <https://medium.com/flutter/which-factors-affected-users-decisions-to-adopt-flutter-q1-2021-user-survey-results-563e61fc68c9> (accessed May 29, 2021).
- [39] "Aqueduct | Multi-threaded Dart Server-Side Framework." <https://aqueduct.io/> (accessed May 04, 2021).
- [40] "Jaguar." <https://jaguar-dart.com/> (accessed May 04, 2021).
- [41] "Angel - Dart on the Server." <https://angel-dart.dev/> (accessed May 04, 2021).
- [42] "TensorFlow Lite | ML for Mobile and Edge Devices." <https://www.tensorflow.org/lite> (accessed Jun. 20, 2021).

- [43] "MQTT - The Standard for IoT Messaging." <https://mqtt.org/> (accessed Apr. 07, 2021).
- [44] "Eclipse Mosquitto." <https://mosquitto.org/> (accessed Jun. 20, 2021).
- [45] "TensorFlow." <https://www.tensorflow.org/> (accessed Jun. 20, 2021).
- [46] "tflite_audio | Flutter Package." https://pub.dev/packages/tflite_audio (accessed May 11, 2021).
- [47] "Teachable Machine." <https://teachablemachine.withgoogle.com/> (accessed Jun. 20, 2021).
- [48] "Teachable Machine Diving Deeper." <https://teachablemachine.withgoogle.com/faq#Diving-Deeper> (accessed Jul. 14, 2021).
- [49] "Netron." <https://netron.app/> (accessed Jun. 20, 2021).
- [50] Tensorflow, "tensorflow:: Tensor Class Reference | TensorFlow Core v2.4.1." https://www.tensorflow.org/api_docs/cc/class/tensorflow/tensor#classtensorflow_1_1_tensor (accessed Apr. 07, 2021).
- [51] "Isolate class - dart:isolate library - Dart API." <https://api.flutter.dev/flutter/dart-isolate/Isolate-class.html> (accessed Jun. 20, 2021).
- [52] "Stream class - dart:async library - Dart API." <https://api.flutter.dev/flutter/dart-async/Stream-class.html> (accessed Jun. 21, 2021).
- [53] "What is the BLoC pattern in Flutter?" <https://www.flutterclutter.dev/flutter/basics/what-is-the-bloc-pattern/2021/2084/> (accessed May 24, 2021).
- [54] "JSON Web Tokens - jwt.io." <https://jwt.io/> (accessed Jun. 20, 2021).
- [55] "PostgreSQL: The world's most advanced open source database." <https://www.postgresql.org/> (accessed Jun. 20, 2021).
- [56] "Cross-Origin Resource Sharing (CORS) - HTTP | MDN." <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> (accessed Jul. 14, 2021).
- [57] "Asynchronous programming: futures, async, await | Dart." <https://dart.dev/codelabs/async-await> (accessed Jun. 21, 2021).

Appendix A Flutter/Dart Dependencies

This appendix contains additional information regarding imported packages and respective versions

```
1 dependencies:
2   flutter:
3     sdk: flutter
4
5   cupertino_icons: ^1.0.0
6   http: ^0.12.2
7   jaguar_resty: ^2.10.15
8   speech_to_text: ^3.1.0
9   device_info: ^1.0.0
10  tflite_audio: 0.1.7+1
11  mqtt_client: 9.0.0
12  intl: ^0.17.0
13  isolate_handler: 1.0.0
14  provider: ^5.0.0
15  curved_navigation_bar: ^0.3.7
```

Code 11 - Flutter dependencies imported to support mobile development

```
1 dependencies:
2   jaguar: ^2.4.32
3   jaguar_reflect: ^2.4.6
4   jaguar_session_jwt: ^2.4.2
5   jaguar_auth: ^2.4.4
6   jaguar_common: ^2.1.4
7   jaguar_cors: ^2.4.4
8   http: ^0.11.3
9   postgres: ^2.1.1
10  mqtt_client: 8.2.0
```

Code 12 - Dart packages used to develop the BeingCare service component

```
1 dependencies:
2   flutter:
3     sdk: flutter
4
5   http: ^0.13.0
6   jaguar_resty: ^3.0.0
7   flutter_bloc: 3.1.0
8   provider: 4.3.3
9   mqtt_client: ^8.2.0
10  line_icons: ^2.0.1
11  intl: ^0.17.0
12  pie_chart: ^5.0.0
13  syncfusion_flutter_charts: ^19.1.64
14  flash: ^2.0.0
```

Code 13 - Packages imported and used by web application

Appendix B Database Schema

This appendix contains a database modeling script

```
1 CREATE TABLE public.message (  
2     id bigint NOT NULL,  
3     "timestamp" text NOT NULL,  
4     "type" text NOT NULL,  
5     sender text NOT NULL,  
6     "description" text NOT NULL  
7 );  
8  
9  
10 ALTER TABLE public.message OWNER TO beingcare_user;  
11  
12 CREATE SEQUENCE public.message_id_seq  
13     START WITH 1  
14     INCREMENT BY 1  
15     NO MINVALUE  
16     NO MAXVALUE  
17     CACHE 1;  
18  
19  
20 ALTER TABLE public.message_id_seq OWNER TO beingcare_user;  
21  
22 ALTER SEQUENCE public.message_id_seq OWNED BY public.message.id;  
23  
24 ALTER TABLE ONLY public.message ALTER COLUMN id SET DEFAULT  
25 nextval('public.message_id_seq'::regclass);  
26  
27 ALTER TABLE ONLY public.message  
28     ADD CONSTRAINT message_pkey PRIMARY KEY (id);
```

Code 14 - Database schema file

Appendix C Docker Specifications

This appendix contains the Docker files which support the deployment

```
1  version: '3'
2  services:
3    postgres:
4      image: "postgres:13"
5      container_name: "postgres_database"
6      environment:
7        - POSTGRES_PASSWORD=password
8        - POSTGRES_USER=beingcare_user
9        - POSTGRES_DB=beingcare
10     ports:
11       - 5432:5432
12     volumes:
13       - ./postgres/schema.sql:/docker-entrypoint-initdb.d/schema.sql
14     restart: unless-stopped
15
16     mosquito:
17       image: eclipse-mosquitto
18       container_name: "mosquitto"
19       ports:
20         - 1883:1883
21         - 9001:9001
22       environment:
23         - MQTT_HOST=mosquitto
24       volumes:
25         - ./mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf
26       restart: unless-stopped
27
28     beingcare-services:
29       build: beingcare_services/.
30       container_name: "beingcare_services"
31       ports:
32         - 8888:8888
33       depends_on:
34         - mosquito
35         - postgres
```

Code 15 - Docker-compose file

```
1 FROM google/dart
2
3 WORKDIR /app
4 ADD pubspec.* /app/
5 RUN pub get --no-precompile
6 ADD . /app/
7 RUN pub get --offline --no-precompile
8
9 WORKDIR /app
10
11 ENTRYPOINT ["dart", "bin/beingcare_services.dart"]
```

Code 16 – Dockerfile for BeingCare services Dart application