

WeDoCare - An IoT system to help vulnerable social groups



UNIVERSIDADE D
COIMBRA



Rúben Filipe Gonçalves Saldanha

Supervisor: Prof. Jorge Sá Silva

Prof. André Rodrigues

Prof. Fernando Boavida

Department of Informatics Engineering

University of Coimbra

This dissertation is submitted for the degree of

Master in Informatics Engineering

Coimbra

June 2019

Esta cópia da tese é fornecida na condição de que quem a consulta reconhece que os direitos de autor são pertença do autor da tese e que nenhuma citação ou informação obtida a partir dela pode ser publicada sem a referência apropriada.

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognize that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without proper acknowledgement.

Dedication

I would like to dedicate this thesis to my loving parents, sister and my lovely girlfriend, for all the strength and support that they have given me throughout this hard academic course.

I would also like to thank my supervisors, Professor Jorge Sá Silva, Professor André Rodrigues, and Professor Fernando Boavida, for the guidance they gave me. A special thank you to Professor Jorge Sá Silva, for always being available to help me and for helping me be better in certain aspects.

I'd also like to thank my colleagues from the Socialite-LCT group. A thank you to Marcelo Fernandes for the technical support he gave me, and also to rest of the team, Duarte Raposo, Jorge Rivadeneira, Martha Todi, Ngombo Armando, Oswaldo Polo and Soraya Sinche.

Finally, I'd like to thank all my friends for their support and motivation through the years.

Resumo

Infelizmente, a violência contra as mulheres é um assunto que ouvimos regularmente nas notícias. No ano passado, a Índia foi considerada o país mais perigoso para as mulheres devido ao alto risco de violência sexual, escravidão sexual e casamentos forçados. Embora possamos identificar um aumento no número de movimentos que fortalecem as mulheres, como o movimento "me too" - iniciado em 2006 para ajudar as vítimas de violência sexual - a maioria das mulheres que passaram por violência doméstica ou sexual ainda não estão a reportar as suas situações à força policial.

Há um grande número de sistemas de alarme pessoal no mercado para combater esta situação, a maioria deles baseados em botões de pânico. No entanto, nenhum deles obteve ampla aceitação. No contexto desta tese, nós desenvolvemos uma aplicação inovadora que é capaz de reconhecer uma situação perigosa e acionar um alerta através de quatro métodos diferentes: (i) reconhecimento de voz, (ii) reconhecimento de gestos, (iii) reconhecimento através do uso de um nó IoT e (iv) pressionar manualmente um botão no ecrã principal da aplicação. Além disso, a aplicação também contém vídeos e dicas informativos, para que as vítimas saibam como agir em situações perigosas.

Palavras-Chave: Android, Java, IoT, HitL, Violência Doméstica

Abstract

Unfortunately, violence against women is a topic we regularly hear on the news. Last year, India was considered the most dangerous country for women due to the high risk of sexual violence, sexual slavery and forced marriages. Although we can identify a rise in the number of movements that empower women, such as the "me too" movement - started in 2006 to help victims of sexual violence – the majority of women who went through domestic or sexual violence are still not reporting their situations to law enforcement.

There is a large number of personal alarm systems in the market to combat this situation, most of them based on panic buttons. However, none of them has gotten widespread acceptance. In the context of this thesis, we developed an innovative application that is capable of recognizing a dangerous situation and trigger an alert through four different methods: (i) speech recognition, (ii) gesture recognition, (iii) recognition through the use of an IoT node and (iv) manual pressing of a button in the application's main screen. Furthermore, the application also has informative videos and tips, so the victims know how to act in dangerous situations.

Keywords: Android, Java, IoT, HitL, Domestic Violence

Table of contents

List of figures	xiii
List of tables	xv
Nomenclature	xix
1 Introduction	1
1.1 Context	1
1.2 Motivation	2
1.3 The team and partners	3
1.4 Work Plan	4
1.5 Internship risks	8
1.6 Contributions	9
1.7 Document structure	9
2 Technologies and Protocols	11
2.1 The smartphone in the XXI century	11
2.2 Internet of Things	12
2.3 Human in the Loop	14
2.4 Technologies and software used	15
2.4.1 IoT nodes	15
2.4.2 Android	17
3 WeDoCare	19
3.1 The previous WeDoCare	19
3.1.1 Functionalities	19
3.1.2 Architecture	20
3.1.3 Similar applications	22
3.2 WeDoCare 3.0	25

Table of contents

3.2.1	Objectives	25
3.2.2	Methodology	26
3.2.3	Software tools used	28
3.2.4	WeDoCare 3.0 requirements	29
3.2.5	Architecture	39
4	Development	43
4.1	Partner meetings	43
4.1.1	State Secretary for Citizenship and Equality	43
4.1.2	PSP	44
4.1.3	Ergue-te and Saúde em Português	45
4.2	Rest API Server	45
4.2.1	Organization	45
4.2.2	Technologies used/Third party libraries	47
4.2.3	Storage	48
4.2.4	Implementation	49
4.3	Chat Server	52
4.3.1	Organization	52
4.3.2	Technologies used/Third party libraries	53
4.3.3	Implementation	53
4.4	Android application	55
4.4.1	Organization	56
4.4.2	Technologies used/Third party libraries	57
4.4.3	Implementation	59
4.4.4	User Interface	65
4.5	WeDoCare Website	68
5	Tests	71
5.1	Unit tests	71
5.1.1	Rest Api Server	72
5.1.2	Chat Server	76
5.1.3	Android application	78
5.2	Performance tests	80
5.3	Battery tests	83
5.4	Usability tests	87
5.5	Real life scenarios	93
5.5.1	Ergue-te	94

5.5.2	Saúde em Português	96
6	T4SC	99
6.1	What is it?	99
6.2	New features	101
6.3	Results	102
7	Conclusion	107
7.1	Final results and Future work	107
7.2	Personal consideration	109
	References	111
	Appendix A WeDoCare Paper	119
	Appendix B WeDoCare installation guide	129
	Appendix C WeDoCare document sent to the Police	135

List of figures

1.1	Gantt diagram of the first semester	4
1.2	Predicted Gantt diagram for the second semester	6
1.3	Actual Gantt diagram for the second semester	6
2.1	IoT	13
2.2	HiTL	14
2.3	Nordic Thingy:52 [29] and its development kit [27], respectively	16
2.4	iTag	16
2.5	Android vs iPhone Market Share	17
3.1	WeDoCare main screen, address form and location tab	20
3.2	Architecture of the previous WeDoCare	21
3.3	Thesis Kanban board on Trello	28
3.4	Initial architecture of the system	39
3.5	Final architecture of the system	40
4.1	Organization of the rest API server	46
4.2	ER diagram of the database	48
4.3	Organization of the chat server	52
4.4	Organization of the Android application	56
4.5	Organization of the Android application - UI	57
4.6	WeDoCare main screen, menu drawer and location tab	66
4.7	List of group chats menu, create group chat menu, and join group chat menu	66
4.8	Chat screen, tips menu, and videos menu	67
4.9	iTag menu, help menu, and login/register menu	67
4.10	WeDoCare website - Home page	68
4.11	WeDoCare website - Features	69
4.12	WeDoCare website - Screenshots	69
4.13	WeDoCare website - Team	70

List of figures

4.14	WeDoCare website home page - Footer	70
5.1	Tests organization on the Rest API server	72
5.2	UserController test	74
5.3	UserService test	76
5.4	Chat server test	77
5.5	Login test on the left, register test on the right	79
5.6	Performance test with JMeter	81
5.7	Latency in milliseconds	81
5.8	Throughput in requests per second	83
5.9	Battery Consumption - Skype vs WeDoCare	86
5.10	Battery Consumption - Detailed Graph	86
5.11	Main screen	89
5.12	Left menu and help screen	90
5.13	Listing of the group chats	91
5.14	Results of question 1	91
5.15	Results of question 2	92
5.16	Results of question 3	92
5.17	Results of question 4	92
5.18	Results of question 5	93
6.1	T4SC home page	102
6.2	Modal to select whether you are a company, researcher or citizen	102
6.3	Researcher profile	103
6.4	Company/Organization profile - adding a reward	103
6.5	Citizen profile - listing rewards	104
6.6	Company/Organization profile - adding a challenge	104
6.7	Citizen profile - listing challenges	104
6.8	Citizen profile - volunteering to a challenge	105
6.9	Company/Organization profile - Terminating a challenge	105
6.10	Citizen profile - Number of points is now updated	106

List of tables

3.1	WeDoCare 2.0 vs Similar Applications	24
5.1	Performance test results	82
5.2	Battery stats	85
5.3	Expected results for the usability test	88
5.4	Usability test results	88
5.5	Ergue-te test results	94

Listings

4.1	UserController class	49
4.2	UserService class	50
4.3	UserRepository class	51
4.4	ChatServer class	53
4.5	Connection class	54
4.6	RestApi interface	59
4.7	Android Rest communication example	60
4.8	Android Chat communication (1)	61
4.9	Android Chat communication (2)	62
4.10	Android Chat communication (3)	63
4.11	BluetoothService class	63
5.1	UserControllerTest class	73
5.2	UserServiceTest class	75
5.3	MemoryTests class	76
5.4	LoginTest case	78

Nomenclature

Acronyms / Abbreviations

API Application Programming Interface

BLE Bluetooth Low Energy

CISUC Centre for Informatics and Systems of the University of Coimbra

CSS Cascading Style Sheets

ER Entity Relationship

GPS Global Positioning System

HiTL Human in the Loop

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

IDE Integrated Development Environment

IoT Internet of Things

JSON JavaScript Object Notation

MAC Media Access Control

MVC Model View Controller

MVP Model View Presenter

NFC Near-field Communication

ORM Object Relational Mapping

Nomenclature

REST Representational State Transfer

SMS Short Message Service

T4SC Tech 4 Social Change

TCP Transmission Control Protocol

UI User Interface

WHO World Health Organization

Chapter 1

Introduction

The present chapter serves as an introduction to the work that was performed; explaining its context and motivation, mentioning who our key partners were, presenting the work plan followed, the risks inherent to the internship, and presenting the document's structure.

1.1 Context

Unfortunately, violence against women is a topic we regularly hear on the news. Last year, India was considered the most dangerous country for women due to the high risk of sexual violence, sexual slavery and forced marriages [1]. Furthermore, a study from the WHO considered domestic and sexual violence as a worldwide problem, and romantic partners expose women to a greater risk of violence than any other people [2].

In fact, 43% of college women in the U.S. have reported experiencing violent and abusive dating behaviors, while one in every six college women has been sexually abused in a dating relationship [3]. These numbers are very concerning, however, not as concerning as knowing that 25% of young people think that it is normal for a man to pressure a woman into having a sexual relationship; with around 17% of young people thinking that "women should know their place" [4]. There is a pattern in the problem wherein gender stereotypes have a significant negative impact on young people's expectations and behaviors when it comes to intimate relationships [4]. This supports the statistics shown before in which a considerable amount of violence is found within intimate relationships.

Although we can identify a rise in the number of movements that empower women, such as the "me too" movement - started in 2006 to help victims of sexual violence [5] - the majority of women who went through domestic or sexual violence are still not reporting their

Introduction

situations to law enforcement. This can be due to the fact that they are afraid of repercussions or the fact that they just think it is normal for a man to do such things [3] [4]. There are also occasions where an intimate partner stalks the victim and threatens to humiliate them, more often than not, with sharing sexual or violent content online, creating a paralyzing sense of fear and helplessness in the victim that stops them from seeking help [6]. In other cases, it is simply because of the shame in assuming themselves as victims.

1.2 Motivation

There is a large number of personal alarm systems in the market, most of them based on panic buttons. Nevertheless, none of them has gotten widespread acceptance. Perhaps technology has not yet progressed enough towards the safety of sexual and domestic abuse victims, at least for the ones that live in low-wealth conditions.

In the past years, the Socialite-LCT team has developed a couple of mobile applications, both named WeDoCare, as a solution to the previous mentioned situation. The first version of WeDoCare was more oriented to a refugee scenario. The target audience was refugees because they did not know the city or country they fled to. Thus, this version had the possibility to mark some places in the city as a danger zone or as a helpful locale, like a hospital or police station. This information would be accessible to all users of the application. More importantly, the application would detect violent attacks through a screaming recognition algorithm, and would then send an automated alert with the attack's location to other users nearby with the technology BLE Beacons. At the same time, WeDoCare would also post the same information on the user's Facebook page in order to alert the victim's friends about the situation. It is relevant to explain that BLE is a wireless low-power solution technology, a feature firstly introduced with the Bluetooth Core Specification 4.0, which allows short-range communication between devices and exchange of small amounts of data between them [7].

The second version of WeDoCare, which is the starting point for the work of this thesis, was more focused on sexual violence and human trafficking victims. This second version was improved to detect violent attacks through speech and gesture recognition. There is also a panic button on the main screen, which is another way to give an alert. When a violent attack is detected, there is a seventeen-second window where the victim can cancel the alert in the event that it was a false alarm. If after that seventeen-second window the victim did not cancel the alert, the application gets the location where the attack occurred and sends a

message to the police with that information. No information of any kind is stored in a remote database due to the privacy concerns raised from the associations we partnered with. A more detailed explanation on how the application works is on chapter 3.

Now, a new version of WeDoCare, which encapsulates the functionalities of the previous version and some new features, was developed. The main goal is to expand the target audience. The second version of WeDoCare was more oriented to sexual violence and human trafficking victims. The current version maintains this target audience, but also expands to women who are dating. As mentioned in the previous section, a large number of women suffer from violence in their relationships, hence, the reason for us to develop a solution for this problem. However, first, we had to correct the bugs that were present in the second version of WeDoCare, so we could have a stable version before implementing new functionalities. These new functionalities resulted from the meetings that we had with our partners throughout the school year. The also UI suffered some slight changes, due to the usability tests, as presented in section 5.4.

1.3 The team and partners

The research group involved in the current master thesis internship is the Communications and Telematics (CT) group. The CISUC-CT group has a vast experience in the communications area and is currently active in research projects about low latency communications in the cloud, IoT clouds and virtualization, human-in-the-loop systems, mechanisms for resilient communication in IoT, mobile phone programming, network security, and several more projects. The master thesis intern responsible for this work is Rúben Saldanha and his supervisors are Professor Jorge Sá Silva, Professor André Rodrigues and Professor Fernando Boavida.

Our key partners are Ergue-te [8], Saúde em Português [9], ISCAC [10] and the Public Security Police (PSP). Ergue-te is a social intervention association with the main goal of fighting for gender equality and opportunity, and for the promotion of women. Furthermore, this association helps prostitutes to leave their work and find a new project for their lives. Due to the job these women perform, they are very susceptible to a high level of violence, not only from their clients but from their bosses and intimate partners as well. For this reason, they were and still are part of the target audience for WeDoCare. Saúde em Português is a Non-Governmental Organization that aims to promote social and community integration

Introduction

and human rights. We worked with a specific department, the one that provides assistance and protection to victims of trafficking. ISCAC, also known as Coimbra Business School, belongs to the Polytechnic Institute of Coimbra. It offers courses on accounting, marketing, management, informatics, among others. The main goal of this partnership is to aid in the raising of money and awareness for this application, so that in the future this application can be even better but still remain free of cost.

It is also important to mentioned that we are trying to form another partnership, this time with the State Secretary for Citizenship and Equality. This would allow us to have a strong support for the target audience that we want to expand to. As it was previously mentioned, we intend to expand our target audience to dating relationships, since more often than not it is the romantic partner who practices the violent attacks we are trying to prevent.

1.4 Work Plan

The internship started in September 2018. The first weeks were an adaptation period as a project of this dimension was a novelty. Since the beginning of the semester that a possibility of a new partnership, this time with State Secretary for Citizenship and Equality, was viable. With that in mind we tried to define new features for the application so that we could expand our target audience. Below is a more detailed explanation of each row in the Gantt diagram represented in figure 1.1.

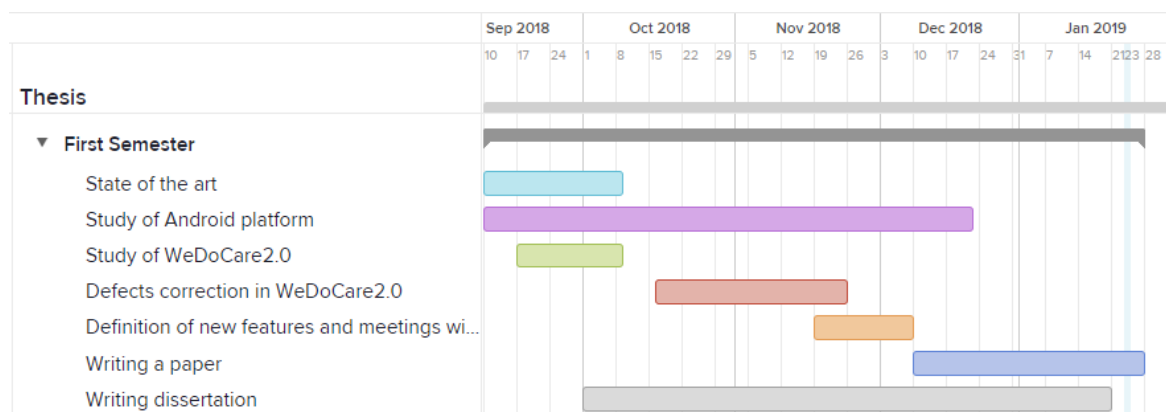


Fig. 1.1 Gantt diagram of the first semester

- **State of the art:** Throughout these weeks we studied applications similar to WeDoCare and the key concepts for this study, such as IoT and HiTL.
- **Study of the Android platform:** This task was scheduled for the whole semester since the student responsible for this work was taking a course on Mobile Computing, which was of great help to understand the Android SDK, common code patterns, common libraries and Material design guidelines.
- **Study of WeDoCare 2.0:** In order to improve and add new features to WeDoCare 2.0, the existing prototype needed to be studied. Alongside with the familiarization with the code, a study of the technologies used and code structure was also performed.
- **Defects correction in WeDoCare 2.0:** WeDoCare 2.0 had some defects that were not expected and needed to be fixed.
- **Definition of new features and meetings with our partners:** We met with Ergue-te, Saúde em Português more than once to obtain feedback on the current version of WeDoCare and to possibly define new features that could improve the application.
- **Writing a paper:** Some of the last weeks of the first semester were spent on writing a paper about WeDoCare. However, this paper suffered some changes along the second semester, namely in June, as we can see in Figure 1.3. This paper was submitted to the CLEI 2019 conference [11].
- **Writing dissertation:** As the name says, during these weeks this dissertation was being written.

For the second semester the predicted planning is represented in Figure 1.2.

The main goal during the second semester was to implement the features that are mentioned in section 3.2.4. Since we were going to have more meetings with our partners, some requirements could change, and there was a possibility of adding new requirements. However that would not be a problem, as the methodology being used is agile, as it will be stated in section 3.2.2. The Figure 1.3 illustrates a Gantt diagram of what actually was done in the second semester.

As shown in Figure 1.3 we had meetings with all of our partners in the last two weeks of February, with the goal of defining the final requirements and establishing the best way to test

Introduction

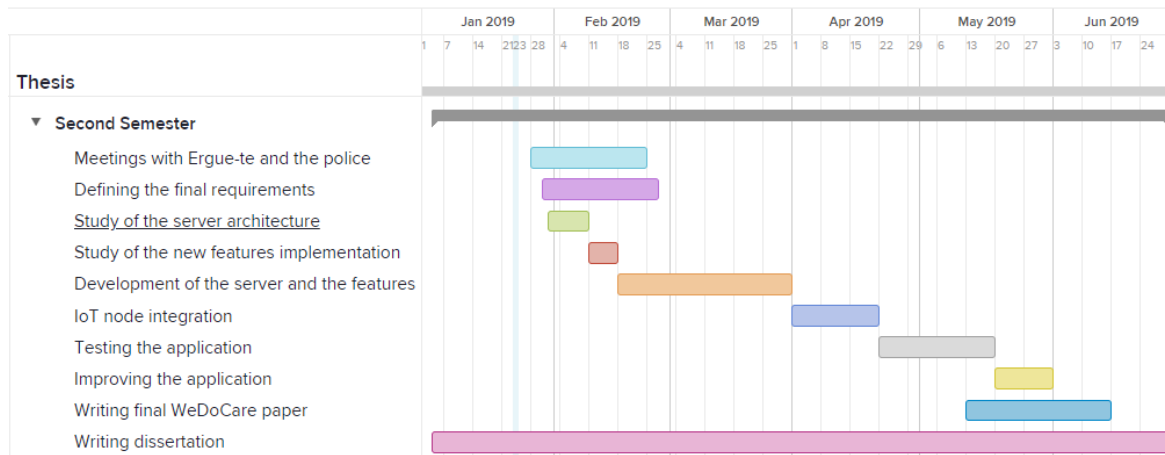


Fig. 1.2 Predicted Gantt diagram for the second semester

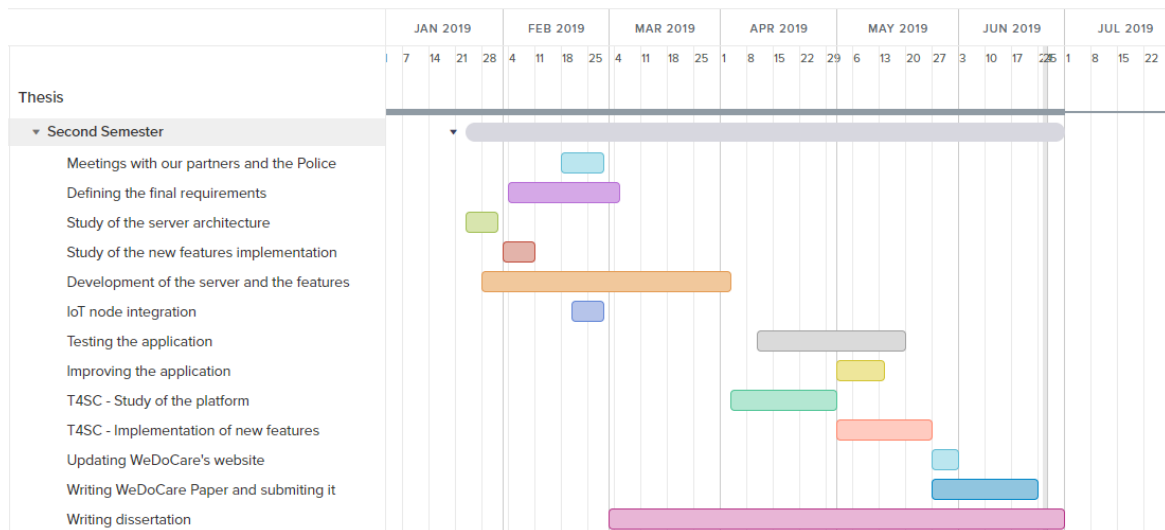


Fig. 1.3 Gantt diagram of the work produced during the second semester

WeDoCare with the help of the PSP. We even went to Lisbon to meet with the State Secretary for Citizenship and Equality. Those meetings are better described in section 4.1. So far the predicted and actual work produced during the second semester is similar. However, the study of the server architecture and the study of the implementation of new features started and was concluded quicker than expected. This is due to the fact that the report delivered in January was finished earlier than expected as well. To avoid being a week without doing anything, the studies previously mentioned began earlier to continue to be as productive as possible.

If we further compare Figure 1.2 and Figure 1.3, we observe that the development of the server and the new features also started ahead of schedule, starting in January rather than February. Nonetheless, the end of this phase was still at the beginning of April as expected. With this, we conclude that the development phase took longer than predicted. This is partly due to the fact that some new requirements were defined in the meetings we had with our partners, and also because the implementation of the video streaming was time-consuming. The IoT node integration also started sooner than expected since it was simpler than we thought, hence why it was completed in a shorter period of time.

With the development phase concluded, we contacted the State Secretary for Citizenship and Equality to let her know that the development phase had ended. We also conducted several tests to the application which are better described in chapter 5. During this time, some improvements were made due to the feedback from the tests. This is also described in chapter 5.

Since we knew that a response from the State Secretary for Citizenship and Equality could take a considerable amount of time, and to maintain a high productivity level, we decided to invest on the T4SC platform [12], since it would be useful to WeDoCare. This platform is explained in more detail in chapter 6. The code of the platform was studied for almost a month. After that, we implemented some new features which will also be mentioned in chapter 6. We also updated WeDoCare's website, so we can have another way of promoting WeDoCare, and possibly make new partnerships. The semester ended with the writing of a new paper for WeDoCare and of course, the writing of the dissertation, which was during most of the semester.

In conclusion we can see that what changed most from the predicted Gantt diagram to the diagram with the actual work that was produced was the study of the architecture of the

Introduction

server, the study of the implementation of the new features and the IoT node integration. Also, improving the application was done alongside with the tests and not after, since it would not make much sense, because then we would not be able to test those new improvements.

1.5 Internship risks

There were some risks inherent to the internship, the main ones being related to the State Secretary for Citizenship and Equality, and to the PSP. It is well known that forming a partnership with members of the government is never easy, mainly because it takes a long time. This time it was not different. Since the beginning of the school year - September 2018 - that we had been in contact with the State Secretary for Citizenship and Equality, but only in February we managed to have a meeting with her for the first time, as explained in section 4.1. When the development phase of the application was done, we contacted her again, but to no avail. To this day we are still waiting for a reply. Because we knew that this might happen, we had a backup plan to keep the high levels of productivity. Our backup plan was the platform T4SC and the WeDoCare website. This would allow us to promote WeDoCare, and possibly make new partnerships. The T4SC platform will be described in more detail in chapter 6, while the website is described in section 4.5. With this backup plan, we managed to still be productive during April and May.

Another risk was the fact that the PSP Commander was not the same one as when WeDoCare was first presented to the PSP. For that reason, we had no choice but to present WeDoCare again, this time to the new PSP Commander, during our meeting in late February. Like it is mentioned in section 4.1, the meeting did not go as expected, and for the PSP to be involved with WeDoCare we would need to go through a long process. Due to this, Ergue-te did not want to proceed with more tests without the PSP involved. To mitigate this risk we are trying to promote WeDoCare through the platform T4SC, and through the WeDoCare website, which was updated and it is mentioned in section 4.5. We are also trying to promote WeDoCare through crowdfunding to get partnerships with more organizations. The crowdfunding is possible due to our partnership with ISCAC. Nevertheless, WeDoCare is still being tested in a real life scenario with Saúde em Português, as they agreed to test the application without the PSP being involved.

1.6 Contributions

Some of the work developed in this thesis contributed greatly to the MobiWise Project [13]. CISUC is one of the partners of the mentioned project [14], and one product that needed to be developed was an Android application that could stream videos, had a chat, amongst other features. With that said, WeDoCare has video repository feature and a chat feature. The code for these features was utilized in the Android application that CISUC wanted to develop. The back-end servers of WeDoCare also served as a starting point for the back-end servers of the mentioned Android application. This means that the some of the work performed to develop WeDoCare, was also used to develop an Android application for the previously mentioned project.

1.7 Document structure

This document is divided into seven chapters, with each chapter being subdivided into several sections.

This first chapter was an introduction to the work being performed, to the team and partners involved. It was also presented the work plan and risks inherent to this internship. The second chapter presents the technologies and protocols related to the work that was developed, like the evolution of the smartphone, which is now much more than just a device to make phone calls. There are also sections explaining what is the concept of Internet of Things and the concept of Human in the Loop. Finally, the last section will describe the technologies used. The third chapter presents the previous and current version of WeDoCare. We will see what are the requirements, as well as its architecture. It will also present the methodology applied to the development of this product.

The fourth chapter is where we look at the development phase of WeDoCare, where it is explained how the entire system works, whilst on chapter five it is presented the tests that were performed to the WeDoCare system. On chapter six we will talk about the platform T4SC and how it is related to WeDoCare. Finally on chapter seven, we present the conclusion to this internship.

There are also three appendices. The first one contains the WeDoCare paper that was written. The second one presents several instructions that should be followed to obtain a

Introduction

successful installation of WeDoCare. The third appendix shows the document we needed to send to the PSP, to initiate the process where the PSP would be involved with WeDoCare.

Chapter 2

Technologies and Protocols

This chapter explains the evolution of the smartphone and some concepts inherent to the work performed in this thesis. With that in mind, a brief introduction to IoT and HiTL will be given.

2.1 The smartphone in the XXI century

A smartphone is arguably the most popular gadget of the 21st century. Initially, mobile phones were developed solely for the army where the need to stay in touch with each other was a big necessity. They were huge in size, and the voice quality was bad and additional features were nonexistent [15].

Today cell phones are smaller and have all the features that one could think of having in an electronic gadget, hence, the name smartphone. With the fast-growing technology, it has now managed to catch up with the best digital cameras and video capturing devices [15].

Up until 1990, mobile phones only had one purpose - to call other people. However, during the 90s mobile phones started to evolve and features like text messaging, email messaging, and games started to become more common. By 2002 features like video calling, GPS navigation, predictive text, cameras, MP3 players, and Bluetooth technology were available in the majority of mobile phones. By 2006 3G was in place and mobile phones were capable of connecting to a Wi-Fi network and do full web browsing. Starting 2007 was when the concept of a mobile phone started to change with the swiping and scrolling replacing the traditional button input method. Nowadays, smartphones have voice control, multiple cameras, several sensors, facial recognition capability, and more importantly, their processing power is much higher [16].

It is difficult for us nowadays to imagine ourselves without smartphones. Whether it is to keep updated with news, events in our friends lives through social media, or for leisure time like traveling, these are all things we depend on our smartphones for. Going to an unknown place and being able to still navigate your way through the use of your smartphone's GPS is a blessing [17].

For our team, the current smartphone is very helpful. Due to its multiple sensors, voice recognition, and GPS navigation or localization abilities, we can more successfully try to develop a solution for the problems mentioned in the previous chapter. The current version of WeDoCare makes use of the sensors to detect an abrupt gesture, uses the voice recognition system to detect a keyword, and through the GPS capabilities we are able to collect the victims' current location. All of this information is sent to the police whenever a dangerous situation is detected.

2.2 Internet of Things

Firstly proposed by Kevin Ashton in 1999, Internet of Things is a system of interrelated computing devices, mechanical, and digital machines, that can transfer data over a network without requiring human-to-human or human-to-computer interaction [18], meaning that the objects gather information by themselves. Kevin Ashton defended that if we had computers that knew everything that is to know about things - without any human help - then we would be able to reduce waste, loss and cost [19].

The objects mentioned above include laptop, computers, smartphones, sensors, actuators, cars, televisions and many others. Any object that is capable of supporting wireless communication and smart connectivity can be a *thing*. This is one of the factors that explains the massive growth in the number of IoT devices. Other factors are emerging applications and business models, standardization, and falling device costs [20]. A growing number of IoT devices also means that a bigger number of devices will connect to the Internet, increasing its traffic.

As was mentioned above, device costs are falling, meaning that creating an IoT device is becoming cheaper. This means that even the simplest object could become an IoT device. This way there will be no need for humans to insert data manually since the devices can

2.2 Internet of Things

sense that information by themselves, connect with each other wirelessly and are able to connect to the Internet. For this reason, some say that the number of IoT connected devices will reach 18 million by 2022 [20]. Others say that number can be much higher as illustrated in figure 2.1.

Internet of Things - number of connected devices worldwide 2015-2025
Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)

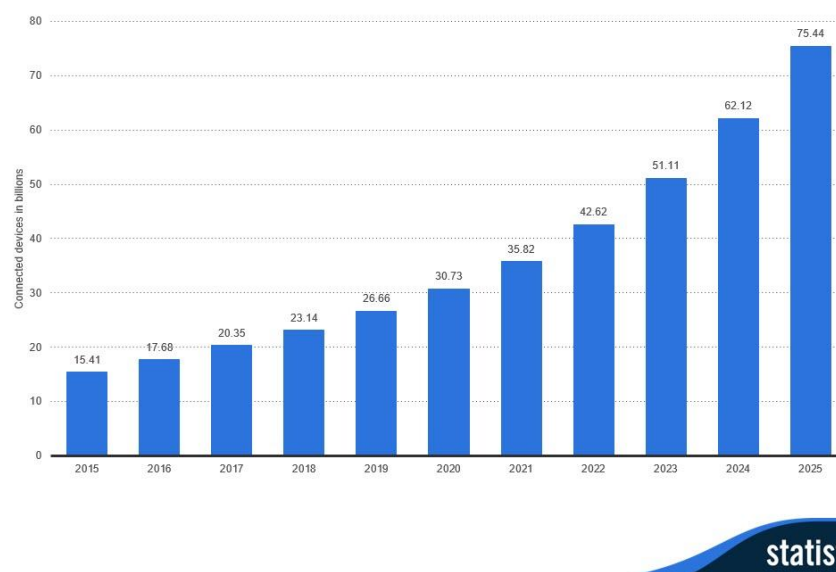


Fig. 2.1 Number of IoT devices until 2025 [21]

As seen in the figure above, the numbers of both sources vary a considerable amount, but it stills gives us an idea of how much IoT devices are going to grow.

As stated in the previous section, the smartphone is a very different piece of hardware than it was a couple of decades ago and even the cheapest smartphones have sensors. Since people always bring their smartphone with them, we can make full use of it, and in our particular case, with WeDoCare we make use of some of those sensors. The Android platform supports three different types of sensors from which you can collect data: motion sensors, environmental sensors, and position sensors [22]. Not included in these groups are a camera, fingerprint sensor, microphone, and touch screen [23]. Millions of Android applications use these sensors, which can be connected to other pieces of hardware (a smartwatch or a wristband, for instance).

It is also worth noting that we incorporated an IoT node in our application with the end goal of

increasing the reliability of the detection of a dangerous situation. This node communicates with the smartphone via BLE and can trigger an alert. This scenario is better explained in chapter 4.

2.3 Human in the Loop

HiTL is defined as a model that requires human interaction, meaning it takes human actions and intents into consideration. Stankovic mentions that HiTL applications can be classified into three categories [24]:

1. Applications where humans directly control the system;
2. Applications where the system passively monitors humans and takes appropriate actions;
3. A hybrid approach of 1) and 2).

In any of the categories mentioned above, the human influences the final output of the system. These systems have been increasing in popularity over the last few years and are implemented in automobiles, health care, and energy management applications. The figure 2.2 shows the basics of HiTL.

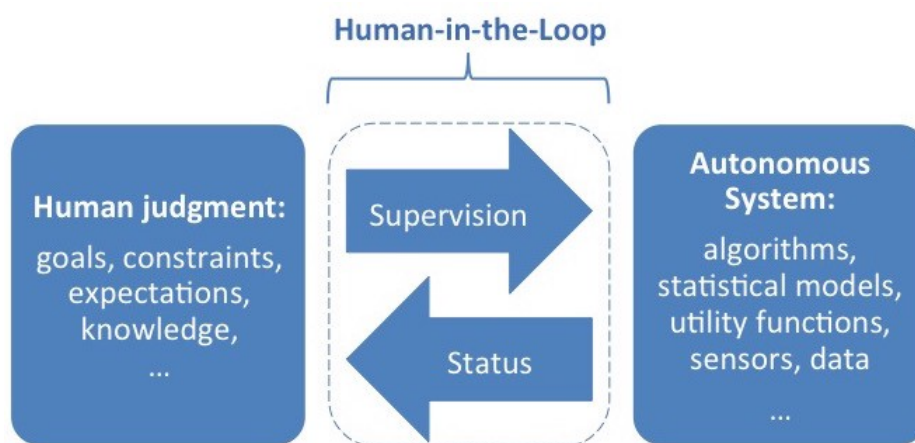


Fig. 2.2 HiTL system [25]

The WeDoCare application inserts itself in the first category, since it is up to the user if he/she wants to enable the GPS and the speech recognition module or not. Of course that we recommend our users to enable those features, since it will be of great help in detecting

violent situations.

2.4 Technologies and software used

To integrate an IoT node, as previously mentioned, we first did some research to evaluate which technology would be most appropriate for this project. This section will detail the solutions found, and it will explain the software in which this project was developed, which is Android.

2.4.1 IoT nodes

In this section we detail the solutions found, and we mention which one was chosen and the reasons that led us to choose that solution.

Nordic Thingy:52

The Nordic Thingy:52 is an easy-to-use prototyping platform, that aims to help in building prototypes and demos, without the need to build hardware or write firmware. It connects via BLE to smartphones, tablets, laptops and similar devices [26]. It comes with a development kit - nRF52 DK [27] - and it is accompanied by an application [28]. This device comes with sensors for temperature, humidity, air pressure, air quality, and color and light intensity. The Figure 2.3 shows the Nordic Thingy and its development kit.

The Nordic Thingy:52 has some documentation to those who want to use the development kit and build a demo or prototype, and it also makes available the source code of the Android application [30]. However, it can be complex at times to write a simple program. We tried using the Nordic Thingy:52. We downloaded the application from the Google Play Store [28] and some demos from the Nordic website [31]. We used Segger Embedded Studio [32] as an IDE because it was recommended by Nordic [33] and it was free. With an IDE, we could see the code of the demos and try to develop a demo of our own, that would serve the purposes of this project. Our goal was to develop a simple demo where a person would click the button on the Nordic Thingy:52, triggering an alarm in the Android application. After a while, we managed to accomplish that goal, yet the code was quite complex. Moreover, the Nordic Thingy malfunctioned after some time, and we did not find out the reason why. Therefore, we were not convinced by this technology and decided to research other options.

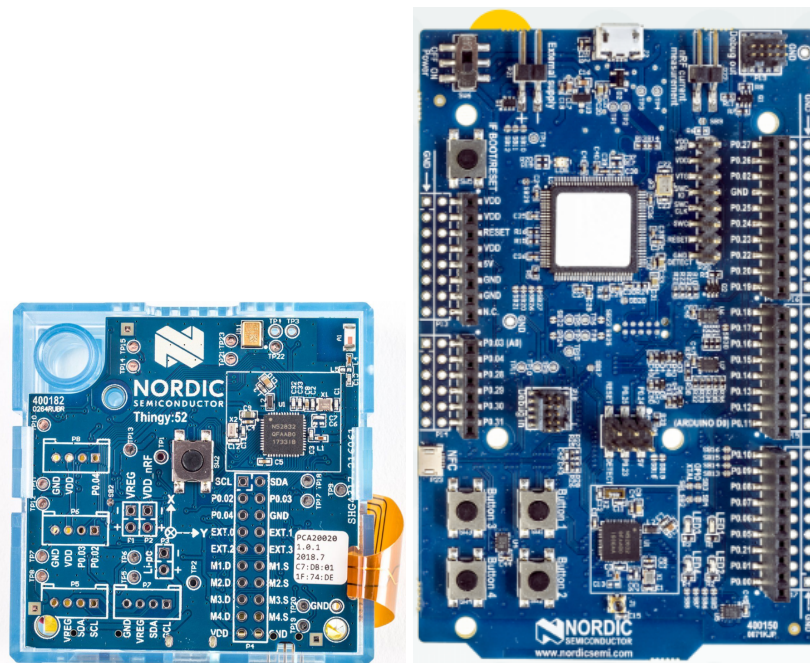


Fig. 2.3 Nordic Thingy:52 [29] and its development kit [27], respectively

iTag

An iTag is a BLE product based on the Bluetooth 4.0 version. It is mostly used as an anti-lost device [34] and there are several applications in the Google Play Store that make use of an iTag. We can see an iTag in the Figure below.



Fig. 2.4 iTag [34]

Compared to the Nordic Thingy:52, the iTag is far simpler to use, it is cheaper - around 3 euros - and smaller. We do not need to code anything in the device, just on the Android application. The red circles in Figure 2.4 show the button that one needs to press to activate the iTag. Our goal was to have a device with a button that, when pressed, would trigger an

alarm on WeDoCare. With the iTag, the process is very simple.

Since the iTag filled the necessary IoT node integration requirement, explained in section 3.2.4, and because it was cheaper, smaller and simpler to use, we decided to use this device to integrate with WeDoCare.

2.4.2 Android

The previous versions of WeDoCare were developed in Android, and so was WeDoCare 3.0. Android is a Linux-based mobile operating system developed by Google [35]. The latest version announced was Android Q, which will soon be released. Android was firstly developed for smartphones, but in the last few years it has expanded into other areas and devices. Now we see tablets, smartwatches, and even TV's running Android. Android is also the most used mobile operating system having more than triple the market share of iPhones [36]. The Figure below shows the market share of the Android and the iPhone from January 2018 to January 2019.

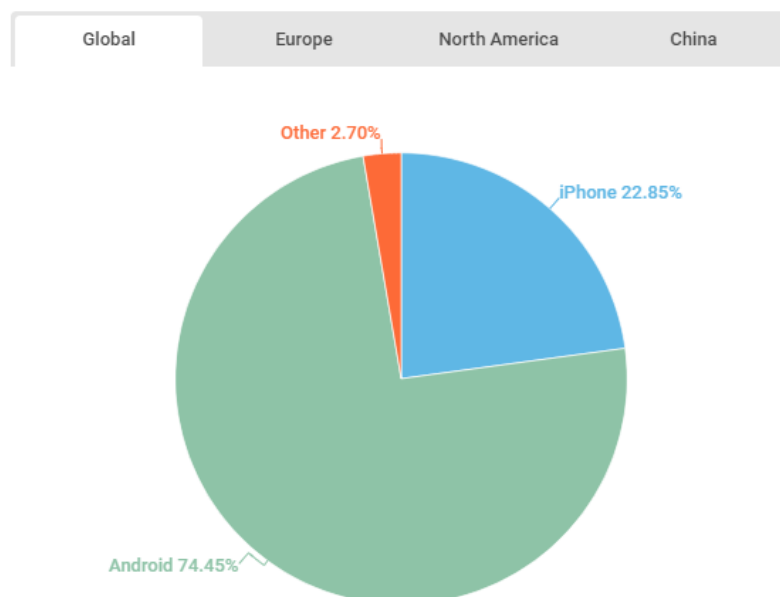


Fig. 2.5 Android vs iPhone Market Share [36]

Android is more common and cheaper, and it's the operating system that most people from our target audience will have on their smartphones. We had confirmation that all of the victims that Ergue-te and Saúde em Português are helping, use a smartphone with Android as the operating system. So, for this reason, and because the previous versions of WeDoCare were also developed in Android, WeDoCare 3.0 was also developed for Android smartphones.

Technologies and Protocols

Something to keep in mind regarding the Android operating system, is that at every new version release, some classes or functionalities become deprecated. This means that they will not work in future versions. With this in mind we had to correct several bugs on WeDoCare 2.0, to make the application stable for the most recent versions - Android O, Android P, and the most recent version, Android Q.

Chapter 3

WeDoCare

3.1 The previous WeDoCare

The previous version of WeDoCare - WeDoCare 2.0 - was a native Android application with the main goal of detecting a dangerous situation and alerting the police about it. This chapter presents the functionalities of this WeDoCare version, as well as its architecture. Lastly, it is done a comparison between WeDoCare and the existing similar applications.

3.1.1 Functionalities

The main functionality of WeDoCare is to detect a dangerous situation, meaning a situation where the person is being attacked, and alert the police via SMS. It is advised that the speech recognition is active in order for the application to be able to recognize a keyword and subsequently trigger an alarm. The GPS should also be activated so that the person's current location can be collected. This is specially important when an alarm is activated, since the application will be able to collect the attack's location. If for some reason the person does not want to have the GPS signal activated, there is an option to put an address that he/she is frequently in, as we see in figure 3.1. That address will replace the coordinates of the current location in the message to be sent to the police if an alert is triggered.

The detection of a dangerous situation can be performed in three different ways. The first one is by just pressing the button on the main screen, as shown in figure 3.1. When the button is pressed, a notification will appear with a common sentence so the perpetrator does not become suspicious of something. This notification has a button that can be pressed in the case the event was a false alarm. There is a seventeen second window where one can cancel this notification. If after that time window the notification was not cancelled, a message will

be sent to the police with the victim's current location.

Another detection method is through the recognition of the keywords "socorro" (which is the portuguese word for requesting help) or "help". The person using the application can turn on speech recognition, allowing him/her to scream for help (using the keywords mentioned) when he/she feels in danger. The application will recognize the keyword and the same notification will appear. The rest of the process is the same as before.

In addition, the application can also detect when there is a very abrupt gesture, like a chop gesture, and when that happens the same notification will pop up.

Lastly, there is also a tab where a person can see his/her current location, as illustrated in figure 3.1, and a tab which informs the person if the messages have been sent or not.

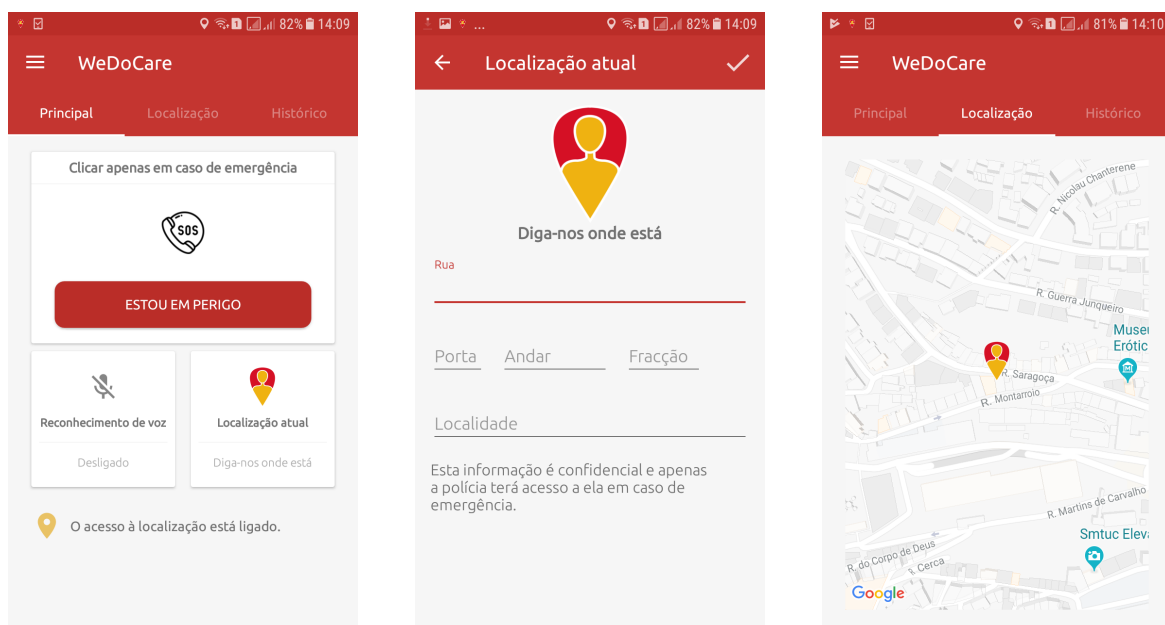


Fig. 3.1 WeDoCare main screen, address form and location tab

3.1.2 Architecture

The WeDoCare 2.0 system is limited to only two elements - the victim's smartphone, which runs WeDoCare, and the police's smartphone. The communication between these devices is performed via SMS. The Figure 3.2 shows the architecture of the previous WeDoCare.

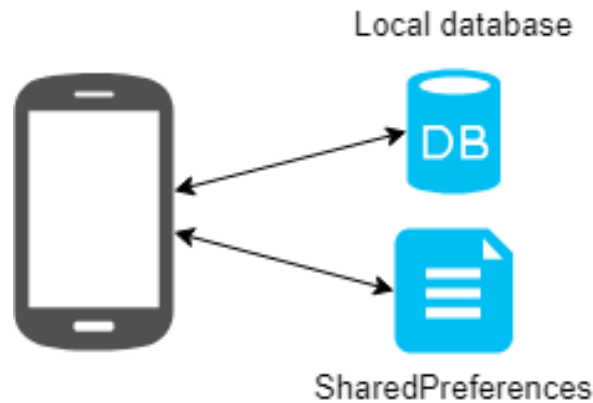


Fig. 3.2 Architecture of the previous WeDoCare

This application will not store information of any kind in a remote database or will share it any under circumstances. This is due to the privacy concerns raised from our partners. However, WeDoCare does store data persistently in a local database or using the SharedPreferences interface.

Since this application was designed only for personal use and every necessary information is processed during runtime and then forgotten, and also due to the privacy concerns of our partners, we felt that there was no need to implement authentication methods with the email or real name of the person using the application. Furthermore, this application was meant to only be installed on devices that belong to people authorized by our partners. For this reason, WeDoCare was not available on the Google Play Store for download. It is of the utmost importance that the application is not accessed by people outside the associations that we partnered with, which could happen if the mobile phone is sold or stolen. The simplest way to make sure that alerts from unauthorized people are not sent to the police is to verify that the text message comes from the same SIM card ID as the one collected during the installation.

The code follows the MVP architecture [37]. The MVP pattern establishes the separation of business logic from the interface code which results in more readable, extendable and maintainable code. In more detail, the Model provides the data we want to see in the View, meaning it is responsible for using API's, caching data and managing databases. The Presenter is the middle-man between Model and View, being able to query the Model and update the View. The View holds the code for the user interface and presents data, in a way decided by the Presenter, to the user. The View can only communicate directly with the

Presenter or other View methods. This separation of code makes it easier to create decoupled classes and to perform unit testing on the application.

3.1.3 Similar applications

In this section we describe applications similar to WeDoCare. Some projects below were the semi finalists of the Anu & Naveen Jain Women's Safety XPRIZE [38], a competition that challenged teams to develop a project to help women in India feel safer. Other projects allow the possibility of integrating additional hardware. This can improve the reliability of the solution presented, however it implies extra costs for the user since the additional hardware is not free of charge. Therefore, it is not a solution that is affordable for everyone.

Wearsafe

Wearsafe [39] is a personal or enterprise solution. It is a free mobile application with a button that when pressed, sends an emergency alert to a group chat previously defined by the user that will coordinate the emergency response. An alert can also be activated by a wearable device which can be both a smartwatch or a Wearsafe Tag. This can be extremely useful in situations when the smartphone is not within the user's reach. Nonetheless, the pairing of such devices with the mobile phone implies a subscription cost. The application also features audio streaming, location updates and the possibility for friends and family to call emergency services right from the application.

Nimb

Nimb [40] is a personal safety application that can be optionally paired with the Nimb Ring, which costs \$249 and, similarly to the previous application, is useful for when the user is unable to reach its phone. The system allows users to send alerts to friends, family, nearby users or emergency responders. However, the option to ask emergency responders for help can only be accessed through a subscription plan that costs at least \$155.88.

Guardian Circle

The Guardian Circle [41] is a free mobile application that works almost as a social network for personal safety. Users can ask their guardians - can be whoever they want, from family to neighbors - for help inside the application. Guardians can then communicate and coordinate an emergency response. When in a dangerous situation, users have to first select the alert

type (emergency, urgent, request or only a test) and can optionally write about what is going on. The down side is the lack of a quick 'help' button.

Watch over me

Watch Over Me [42] is a freemium application, meaning its basic features are free of charge, while more advanced features are paid for. The application sends emergency alerts to a safety network through push notifications for free. Nevertheless, since this feature does not work in areas with low connectivity, users can, for \$4.99 per year, unlock the 'Total safety feature' that sends alerts via SMS, with the company covering for the cost of those text messages. When the user feels insecure, he/she can set a timer inside the application so that it can watch over him/her for a specified amount of time and additionally share details. If the user does not let the system know that he/she is safe until the timer finishes, his/her friends will receive an alert.

bSafe

bSafe[43] is a mobile application that works without the need of additional hardware. The application allows the user to create a safety network of friends with whom he/she can share his/her location and send emergency alerts by tapping an emergency button. There are also additional features which include a timer that works the same way as the timer in Watch Over Me described in the previous subsection.

SAFER by Leaf

Leaf [44] is a wristband with a GPS incorporated that costs \$37 and pairs with the smartphone. With this, the user can share its live location with guardians through a mobile application. The wearable has a panic button that enables the user to send emergency alerts and share his/her live location with guardians. Another particular feature is that it helps users find safe places like police stations and hospitals. This solution won Anu & Naveen Jain Women's Safety XPRIZE.

WeDoCare vs. Similar applications

After analyzing the previous applications we can verify that location sharing is a common and important feature, and one that is also implemented in WeDoCare 2.0. The fact that WeDoCare 2.0 is an anonymous system, since there are no authentications methods, means that

WeDoCare

the location of the users will never be shared with other than the police. This 'sharing location with the authorities' feature is only implemented in three of the applications described above and it is only available when users adhere to a subscription plan, meaning one has to pay for it. WeDoCare 2.0 is free application. Users do not have to pay for extra functionalities and they do not need to buy additional hardware to pair with the application. Furthermore, with WeDoCare 2.0 the user has the possibility of activating an alarm without any wearable device or when it is not possible to reach the mobile phone. As already stated, WeDoCare can detect an attack through the recognition of a keyword and through the recognition of an abrupt gesture, in this case the chop gesture.

WeDoCare 2.0 shares some features with other applications but it clearly stands out for the privacy it offers. It is also important to mention that WeDoCare 2.0 is the only system capable of detecting violent attacks with a complete 'hands-free' way. We consider that WeDoCare is in advantage over other solutions when it comes to low-cost and reliable safety applications, since it is also free of charge.

WeDoCare 2.0 can be more easily compared with the other mobile applications in table 3.1. In the last two columns, AH stands for Additional Hardware, while IAD stands for Intelligent Attack Detection.

Table 3.1 WeDoCare 2.0 vs Similar Applications

Application	Anonymity	Location Sharing	Cost	AH	IAD
WeDoCare 2.0	Yes	Yes	Free	No	Yes
WearSafe	No	No	Freemium	Yes	No
Nimb	No	Yes	Freemium	Yes	No
Guardian Circle	No	Yes	Free	No	No
Watch Over Me	No	Yes	Freemium	No	No
bSafe	No	Yes	Free	No	No
SAFER by Leaf	No	Yes	Paid	Yes	No

It is worth noting that the current version that was developed - WeDoCare 3.0 - also comprises all of WeDoCare's 2.0 features, as well as some new ones.

3.2 WeDoCare 3.0

In this section a reflection is made about the objectives of this work. It is also explained which methodology was used in the approach to this project and the planning followed in the first semester. Lastly, we take a look at WeDoCare's requirements and architecture.

3.2.1 Objectives

The first version of WeDoCare was developed during the academic year of 2015/2016. This application was designed to help refugees. These are groups of people who have fled their home country in hopes of finding a better life. Since they do not know the country they have fled to, more often than not, they find themselves taking shelter in places that lack security. It should be mentioned that the majority of violent attacks are witnessed in less developed countries where more people face poverty conditions. This was the main reason for the conceptualization and design of WeDoCare 1.0 as a possible IoT and HiTL solution. Fortunately, this reality is uncommon in Portugal, namely in Coimbra, however this meant that it would be complicated to validate WeDoCare 1.0.

Regarding WeDoCare 2.0, the focus shifted to other social minorities, namely prostitutes and victims of human trafficking. These are women who suffer a high level of violence, not only from their clients but from their bosses and romantic partners as well. Partnerships with some local associations were made and a new version of WeDoCare was conceptualized and implemented during the academic year of 2017/2018. With these local partnerships WeDoCare had a greater chance of being validated since the target audience would be able to test the application. WeDoCare 2.0 was able to detect violent attacks through 3 different ways: i) through a manual trigger, by pressing the alarm button in the main screen, ii) through the recognition of the keyword "*socorro*" or *help*, iii) through the recognition of an abrupt gesture - chop gesture. It is worth noting that neither WeDoCare 1.0 or WeDoCare 2.0 were ever tested in a real life scenario.

Currently we are still trying to improve and to provide a better solution for these social minorities. Therefore a more stable and reliable version of WeDoCare was developed. With WeDoCare 3.0 we wanted to improve some features and add others, so that we can also expand our target audience. Alongside with prostitutes and victims of human trafficking, we want WeDoCare 3.0 to reach dating relationships, because, as it was mentioned in chapter 1, sometimes it is the romantic partner who is the violent perpetrator. At this moment we are trying to establish another partnership, this time with the State Secretary for Citizenship and

Equality.

One of the new features we were able to implement is another mechanism to detect a dangerous situation. This was done through the use of an IoT node as mentioned in section 2.4.1. That node is an iTag, which is a small object, similar to a keychain, with a button that, when pressed, sends a message via Bluetooth to the smartphone, triggering an alert. One of our goals was to have a stable prototype in May so it could be tested during *Queima das Fitas*, as we consider it provides the right circumstances for testing an application like WeDoCare.

Other features are a text chat and a REST API server. The text chat would allow people registered in the application to talk to each other. This can be useful since the people using this application are a part of a social minority, and therefore have (unfortunately) some things in common. This way they can stay in contact with each other and possibly help each other. The difference in WeDoCare is that it does not store personal information. The username chosen by the person using the application can be any string of words, meaning it does not even have to be the person's name or email. With this we can guarantee to the user that his/her privacy is intact. Regarding the server, it will contain informational videos and tips on how the user can protect him/herself better in danger situations or on the precautions that he/she should have when alone in places unknown to him/her. These features were proposed by the State Secretary for Citizenship and Equality during our first contacts in the beginning of the school year.

The main goal for this internship was to test WeDoCare 3.0 in a real life scenario, as its previous versions were never tested in such conditions. And, as it will be explained in the following chapters, we accomplished our goal, because WeDoCare 3.0 is stable, it's being tested with people who are actual victims of violence, and so far, there were no bugs being reported.

3.2.2 Methodology

During the first and second semesters, weekly meetings, usually on Thursdays, were scheduled with Professor Jorge Sá Silva and Professor André Rodrigues with the purpose of defining new tasks for the following week, according to the progress made in the previous one. The tasks of the previous week were analyzed by the supervisors, who then evaluated and informed about their strengths and weaknesses, thus deciding the steps to be performed

during the next week accordingly. Alongside with these individual meetings, weekly meetings with the whole Socialite-LCT group were scheduled, also on Thursdays. During these meetings, everyone would explain what they were working on and the progress they had made since the previous week so that all the team members could constructively share their opinion and suggest possible improvements to everyone's work. However, it should be clear that the intern responsible for this internship was the only person to work on WeDoCare 3.0.

Throughout the first semester we also had the chance to meet with some of our partners. We had two meetings with Ergue-te. The aim of the first meeting was to evaluate WeDoCare's current situation and to draw attention to some bugs on the, at the time, current version of WeDoCare. The second meeting had the purpose of teaching the Ergue-te team how could they install WeDoCare in the smartphones of the women they are helping. Since these women want to keep their privacy intact, we can not see them, so we can not teach them directly, hence why we had to show the Ergue-te team the steps to follow to a successful WeDoCare installation. During the second semester we had more meetings, this time with the PSP, alongside Ergue-te. We also met with Saúde em Português and with the State Secretary for Citizenship and Equality. These meetings are more detailed in section 4.1.

There is an office at the Department of Informatics Engineering where our team can work. During the first semester, since there were still classes, most of the work developed was done remotely and asynchronously. In order to make it work, we made use of some productivity tools and platforms that will be mentioned in the next section. However, throughout the second semester the work was performed in the mentioned office.

Since weekly meetings were scheduled with Professor Jorge Sá Silva and Professor André Rodrigues and some meetings were scheduled with Ergue-te, PSP and the State Secretary for Citizenship and Equality, a Scrum methodology was adopted. Scrum is an agile process with an incremental life cycle [47] that follows a set of fixed length iterations in which the product is developed. Each of these iterations is called a sprint. Usually, each sprint can have a duration of two to four weeks. These types of agile methodologies are based on incremental procedures for continuous delivery of software and have become the golden standard in software development. An agile methodology copes well with changes that the client - in our case, our partners - might want to make, so it is the best methodology to approach this project. To implement agile software development we used the Kanban [48] framework which is better explained in the next section.

3.2.3 Software tools used

As mentioned in the previous section, since there were classes during the first semester most of the work was done remotely and asynchronously, hence why we decided to use productivity tools and platforms such as Slack [45] for communication and Gitlab [46]. It was also stated that an agile methodology was chosen - more specifically, Scrum - because it copes better with changes in the requirements. For this reason we used the Kanban framework for an easier agile development.

The Kanban framework allows you to visualize the workflow in a Kanban board, which is itself divided in cards for the allocation of tasks according to their status in the project. This board enables transparency throughout the entire development since everyone who looks at it is able to see the state of the project at all times in a highly visual manner. This framework is very lightweight and flexible because it does not require someone in the team to perform pre-defined agile roles, thus making Kanban a great choice for small teams. All the tasks were defined and assigned according to their priorities by Professor Jorge Sá Silva and Professor André Rodrigues, who could also change their priority if need be. Kanban promotes the minimization of the work in progress. Trello [49] was used as a platform for project management, to create our Kanban board and to keep track of our progress. The board is represented on figure 3.3.

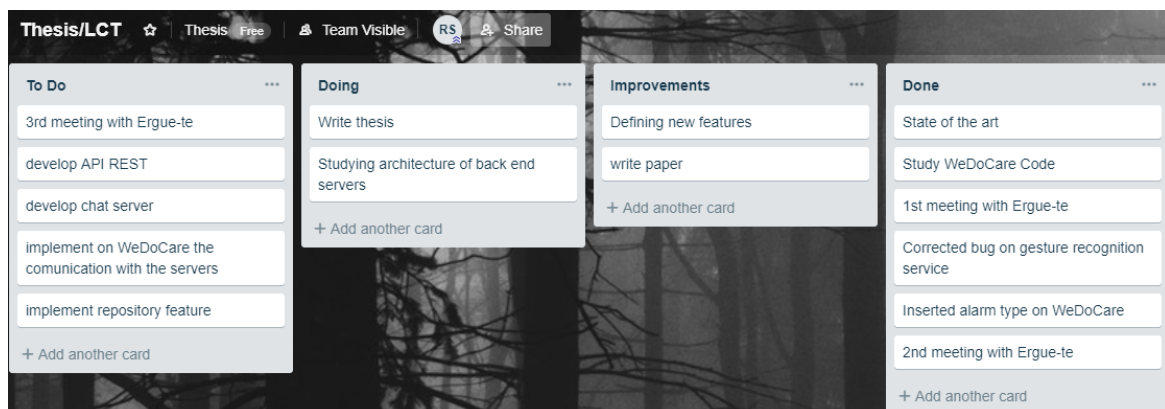


Fig. 3.3 Thesis Kanban board on Trello

As it is illustrated in the figure, there are four cards in the Kaban board, from which in the first three the tasks were organized in order of highest to lowest priority, from top to bottom. When a task was finished, it was moved to the "Improvements" card, and the next task on the

top of the "To Do" card was moved to the "Doing" card. Below is a more detailed description of each card.

- **To Do:** This card holds the tasks given by Professor Jorge Sá Silva and Professor André Rodrigues on the Thursdays weekly meetings. This card could occasionally include tasks from the preceding week in the event that those tasks were not completed before.
- **Doing:** As the name says, if any task was in this card, it means that it was being performed.
- **Improvements:** This card contains the tasks that have been finished, but might benefit from a code review, whether it is for a clean up or for a possible bug correction.
- **Done:** After the task was finished and reviewed, it was considered done and added to this card. The completed task were always reviewed with Professor Jorge Sá Silva and Professor André Rodrigues.

To implement the new features in WeDoCare, Android Studio IDE [50] was used, since it was already used to study WeDoCare 2.0 and to correct some defects. Currently, using Android Studio, an android application can be coded using the Java [51] or Kotlin [52] programming languages. Both previous versions of WeDoCare were implemented in Java, and so was WeDoCare 3.0. The back-end rest server and the chat server were also coded in Java. These servers were implemented through the use of the IntelliJ IDE [54].

We made use of several more tools such as the TeamGantt platform [55] to draw the Gantt diagrams related to the first and second semesters. To draw the architecture of the system the *draw.io* platform [56] was used. The ER diagram that will be discussed in section 4.2.3 was designed with the tool Visual Paradigm [57]. To write this thesis and the paper about WeDoCare we used the Overleaf platform [58], which is an online LaTeX editor.

3.2.4 WeDoCare 3.0 requirements

A requirement is a necessary condition to satisfy an objective [59]. Requirements are subdivided in functional and non-functional requirements. In the next subsections are listed the functional and non functional requirements for WeDoCare. Furthermore, we divide the functional requirements in two parts. The first part refers to the requirements of the mobile application, while the second part mentions the requirements of the back-end servers.

Functional requirements

A functional requirement describes the functional behaviour of a system. Each one of the requirements will be describe as follows:

- **Priority**
 1. **Must:** Mandatory requirements, whose absence causes the application to malfunction;
 2. **Should:** Requirements that should be implemented, but whose absence will not stop the application from working;
 3. **Could:** These requirements are not mandatory and have the lowest priority.
- **Description:** overall description of the requirement;
- **Sequence of stimulus and response:** description of the actions needed to accomplish the requirement and the results expected.

Next are defined the functional requirements of the **mobile application**. It is important to mention that the requirements 1 through 4, and 12 through 15, were already part of the previous WeDoCare version. Nonetheless, these requirements should still be listed since these functionalities were improved when the bugs were corrected. It should also be noted that requirement number 4 is specially important for Saúde em Português, as they have people that are from different countries.

1. Send an emergency alert manually

- **Priority:** Must.
- **Description:** The user must be able to trigger an emergency alert manually through the click of a button.
- **Sequence of stimulus and response:** The user clicks the emergency button that is present on the main screen, which is very easily accessible and visible. The application must send an automatic message to the police with the user's location (coordinates, address and link to open the pinned location on Google Maps application) and the type of alarm triggered.

2. Send an emergency alert automatically

- **Priority:** Must.
- **Description:** The emergency alert must be activated automatically when it detects the words 'socorro' and 'help', or when it detects a chop gesture.
- **Sequence of stimulus and response:** If the speech recognition and gesture detection service are activated, the system must detect, either, the keywords 'socorro' or 'help', or the chop gesture, meaning that the person is in danger. Subsequently, the application must send an automatic message to the police with the user's location (coordinates, address and link to open the pinned location on Google Maps application).

3. Cancel an emergency alert

- **Priority:** Must.
- **Description:** The user must be able to cancel an emergency alert in case of a false alarm.
- **Sequence of stimulus and response:** When the emergency alert is activated, the application displays a discrete notification to the user, that, apparently, is not related to the application. There is a seventeen second window where the user has to acknowledge this notification. If the notification is acknowledged before that time period, the application must not send the text message to the police and the alarm is cancelled.

4. Support for other languages

- **Priority:** Must.
- **Description:** The system must present the application in English (default), Portuguese, Nepali, Russian or Romanian.
- **Sequence of stimulus and response:** When WeDoCare is launched it defines its language according to the language defined in the user's mobile phone.

5. Register into WeDoCare

- **Priority:** Must.

- **Description:** The user should be able to register into the application.
- **Sequence of stimulus and response:** If the user does not have an account yet, he or she should be able to register an account with a username and a password. The username is unique for every person.

6. Log in into WeDoCare

- **Priority:** Must.
- **Description:** The user should be able to log in into the application.
- **Sequence of stimulus and response:** If the user has an account, then he or she should be able to log in into the application through the use of a username and a password.

7. Communicate with other users through a text chat

- **Priority:** Must.
- **Description:** The user should be able to communicate with other users that have registered an account for WeDoCare via a text chat.
- **Sequence of stimulus and response:** If in a chat group, the user should be able to send and receive messages from that group allowing him or her to communicate with other registered users.

8. Create a group chat

- **Priority:** Must.
- **Description:** The user should be able to create a group chat.
- **Sequence of stimulus and response:** The user should be able to see all the group chats that exist and he himself or she herself should be able to create a group chat with a name for the group and a password.

9. Watch a video stored in a server

- **Priority:** Must.

- **Description:** The user should be able to watch an informational video that is stored in a server.
- **Sequence of stimulus and response:** The user should be able to see all the informational videos that are stored in the server and he/she should be able to select a video of his/her choice. Once selected, the video should be downloaded and played for the user to watch it.

10. Read a tip from the tips list

- **Priority:** Must.
- **Description:** The user should be able to a tip from the tips list.
- **Sequence of stimulus and response:** The user should be able to see all the informational tips that are stored in the server and he/she should be able to select a tip of his/her choice to read.

11. Integrate an IoT node to improve reliability

- **Priority:** Must.
- **Description:** An IoT node should be integrated with the application to improve the reliability of the system.
- **Sequence of stimulus and response:** The IoT node has a button. When the button is pressed a message should be sent via Bluetooth to the smartphone. This message has to be recognized by the application, even if it is in the background, and should trigger an alert. If the alert is not cancelled, the application must send an automatic message to the police with the user's location (coordinates, address and link to open the pinned location on Google Maps application) and the type of alarm triggered.

12. Store the users current location in the system

- **Priority:** Should.
- **Description:** The user should be able to provide his or her current location whenever he or she desires.

- **Sequence of stimulus and response:** The user should be allowed to fill in a form with details about a location's address (street, number, floor, fraction). This information should be editable anytime and stored in a local database.

13. Activate/deactivate the keyword detection algorithm

- **Priority:** Should.
- **Description:** The user should be able to activate or deactivate the speech recognition service so that it is only running when he or she feels that a violent situation might happen and, therefore, reduce power consumption.
- **Sequence of stimulus and response:** When the user wishes to activate the speech recognition service, he or she should be able to turn it on by clicking a button that is visible in the main screen. From that moment on the application will start recording audio and searching for the word 'socorro' or 'help' in the background.

14. Inform the user about functionalities that might not work

- **Priority:** Should.
- **Description:** The user should be properly informed if something is interfering with the regular functioning of the application.
- **Sequence of stimulus and response:** The application should present warning messages to the user explaining that a feature might not work correctly.

15. Consult the history of sent messages

- **Priority:** Could.
- **Description:** The user could be able to consult a list with information regarding the messages sent to the police.
- **Sequence of stimulus and response:** The user could consult the list with information about the time a message was sent to the police and the location sent. This list could also contain information about the messages that, due to some malfunction, were not sent.

Next we take a look at the functional requirements for the **back-end servers**.

1. Registration of a user

- **Priority:** Must.
- **Description:** The REST API server must be able to register a user and save his/her username and password.
- **Sequence of stimulus and response:** When a user tries to register from the mobile application, the server should be able to verify if the username, that was selected by the user, is not already used. If it is used then a negative response is sent to the mobile application, which in return warns the user about it. Otherwise, the user will be registered successfully.

2. Log in of a user

- **Priority:** Must.
- **Description:** The REST API server must let a user log in when he or she uses the correct credentials.
- **Sequence of stimulus and response:** When a user tries to log in, the username and password will be sent to the server to verify if they match. If so, then the server will send a positive response to the mobile application and the user will be logged in. Otherwise, a negative response will be sent and the user will get a warning that the credentials he has inserted do not match.

3. Creation of a chat room

- **Priority:** Must.
- **Description:** The REST API server must let a logged user create a chat room with a name and a password.
- **Sequence of stimulus and response:** When a user wants to create a chat room, he or she defines a name and a password. The information should be stored in the server so when another user wants to join the chat room, he or she has to insert the correct password for the room, which will be verified by the server.

4. Sending chat messages to all participants in the same group

- **Priority:** Must.
- **Description:** The chat server must send the chat messages to all participants in the same room.
- **Sequence of stimulus and response:** When a user wants to send a message, that messages is first sent to the server, which in return must send it to all the remaining participants in the chat room.

5. Store informational videos

- **Priority:** Must.
- **Description:** The REST API server must store informational videos that can later be viewed by users.
- **Sequence of stimulus and response:** The server must store informational videos. When a user selects a video in the mobile application the server must provide a download link to the user in order for the video to be downloaded.

6. Store informational tips

- **Priority:** Must.
- **Description:** The REST API server must store informational tips that can later be read by users.
- **Sequence of stimulus and response:** The server must store informational tips. When a user clicks to see the tips, the server must provide them.

7. Store necessary information

- **Priority:** Must.
- **Description:** Both servers must store every information regarding registrations and chat rooms.
- **Sequence of stimulus and response:** The REST API server must store the user-name and password of every user, as well as every chat room created, and of course its name and password. On the other hand, the chat server must store

every message that is sent in each of the chat groups.

8. Store statistics information

- **Priority:** Must.
- **Description:** The Rest API server must store statistic information regarding the tips, videos, alarms triggered, and application usage.
- **Sequence of stimulus and response:** Whenever a user reads a tip, watches a video, triggers an alarm or logs in or out of the application, the Rest API server must store that information for statistics purposes.

It is important to mention that there was a change in the requirements. Due to the meeting we had with the State Secretary for Citizenship and Equality, the mobile application requirement identified by number ten, and the back-end server requirement identified by number 8 were added. It was discussed during the meeting that it would be interesting to have a list of informative tips for the users to read, hence why the requirement number 10 of the mobile application was added. It was also mentioned that it would be good to verify how many people read the tips and watch videos. And if they watch videos, we should verify the percentage of the video they watched. They also asked to retrieve information on the alarms that were triggered, so they could see which alarm was triggered more often. Finally, they would like to know how often people use the application, hence why we collect information about the app usage. For these reasons we needed to establish another requirement for the back-end servers - in this case, it is just the Rest API server - which is requirement number 8.

Non-functional requirements

While functional requirements describe what the system needs to do, a non-functional requirement describes what the system needs to be. The following non-functional requirements are important for designing the system's architecture.

- **Privacy:**
Our partners expressed how privacy is very important for the victims they are helping. With that in mind the only information the REST API server stores about the user is its username and password. The username can be any string of words that the user desires.

Regarding the location information of the user, that is stored locally in the mobile phone in a database or through the use of the SharedPreferences interface. Since this information is more sensitive, that is the reason why it is stored locally, so only the user is able to access it.

- **Battery usage:**

The application must be designed in such a way that it does not drain all the energy from the mobile phone and does not fail in a danger situation.

Must: Last at least one day.

- **Reliability:**

The application should always give the right result on the recognition of the keyword and should always detect the chop gesture, this way sending a 'true positive' alarm to the police. Regarding the REST API server, it should always correctly verify if a user has already registered, logged in or if the credentials for the chat room match.

Desired: 100 % of the times

Must: 97 % of the times

- **Usability:**

The application should be easy to use since a large portion of our target audience may not be accustomed to some of the features implemented. The application should also explain to the user why certain features might not work if the user does not allow certain permissions.

- **Maintainability:**

The application should be coded in a way that is easy to test, to correct defects and to implement new features.

- **Availability:**

The system must have permission to get the users location as well as to send text messages, meaning the phone in which the application is installed must have a SIM card on it. Regarding the servers, they must be available and functional so a user can register/log in and maybe create a chat room and send messages.

Desired: 100 % of the times

Must: 98 % of the times

- **Performance:**

The application must not be slow in changing screens and detecting a dangerous situation. Furthermore, the servers must not prolong their response to the mobile application when a request has been made. The user should feel a smooth experience when using the application.

3.2.5 Architecture

WeDoCare 3.0 presents some changes in its architecture when comparing to the previous version. Figure 3.4 illustrates WeDoCare's 3.0 initial architecture.

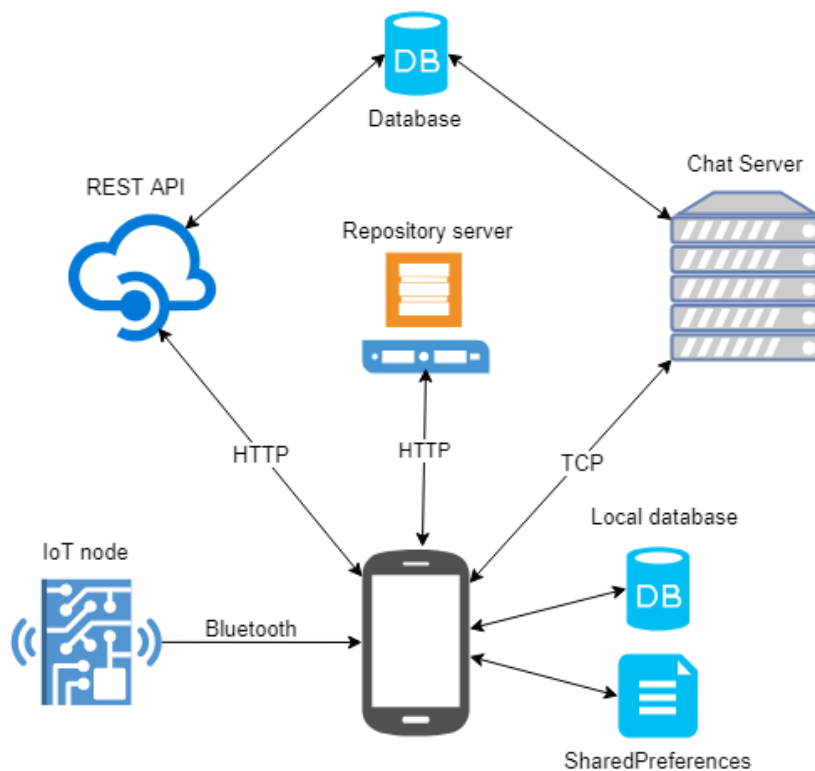


Fig. 3.4 Initial architecture of the system

There were some changes in the architecture of the system in the beginning of the second semester. We realized that there was no need to have a separate repository server for the video streaming, hence why the videos are now streamed from the REST API server. Furthermore,

the chat server does not save data in the database when the application is distributed to the victims that Ergue-te is helping, since it was decided not to save messages due to their privacy concerns. Nevertheless, the chat server does store the messages when the application is distributed by our partnership with the State Secretary for Citizenship and Equality. This is because a regular chat server stores the messages that each user send to each other, and that is what the State Secretary for Citizenship and Equality wanted. The Figure 3.5 shows WeDoCare's final architecture.

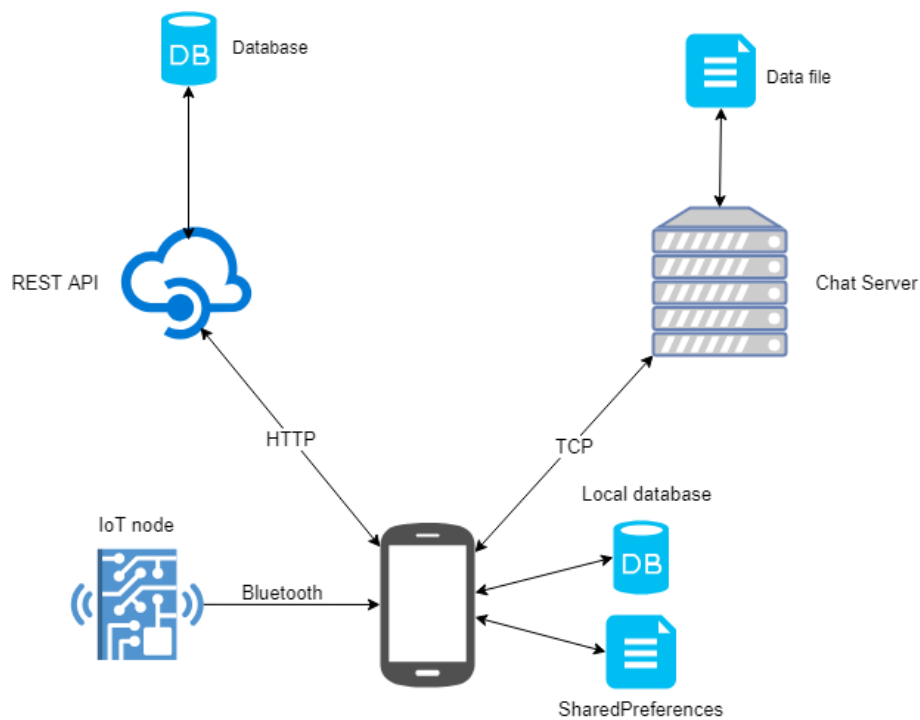


Fig. 3.5 Final architecture of the system

In the previous version, the WeDoCare system was limited to only two elements - the victim's smartphone, which ran WeDoCare, and the police's smartphone. The communication between these devices was performed via text message. Moreover, the application did not store information of any kind in a remote database. Instead, WeDoCare would store data persistently in a local database or using the SharedPreferences interface.

As shown in figure 3.5, WeDoCare 3.0 retains the concept of storing some information in a local database of the mobile device, and the concept of using the SharedPreferences interface. SharedPreferences writes information into an XML file that is located in the application folder in the mobile device. A SharedPreferences object contains all the preferences

in the application. The content stored in the local database is the information regarding the messages that were (or were not, due to some malfunction) sent to the police.

However, the architecture of WeDoCare 3.0 has more components. A REST API server, a chat server and a remote database were added. The information saved in the database will be detailed in section 4.2.3. A user while using the application will be able to connect to the REST server through HTTP requests. The operations that will need to connect to the REST server are log in, registration, displaying of the group chats available and creation of groups chats. This server will also allow for users to watch informational videos and read informational tips. If the user wishes to watch videos on how he/she can protect him/herself better, then the user can do that by watching those videos stored in the server. The application will make an HTTP request for the desired video, which will then be played for the user to watch. The chat server will be used only when users are in a group chat and sending messages. A TCP connection will be used in order for the user to connect to this server. As it is well known, the TCP protocol assures delivery of the data that was sent. If a packet is lost, then it will be sent again, and again until it reaches its destiny. TCP also has a congestion control mechanism to prevent losing too many packets when the network is congested. With this strategy, no information will be lost while communicating with the server, which is what is desired. A TCP connection will be established whenever a user enters a group chat.

When the application launches the user can log in or register. For any of these two options, the application will contact the REST API server to verify the credentials the user inserted. This server will store information in the remote database about the user credentials and the group chats available. Regarding the chat server, it will only be used when a user has joined a group chat and wants to send a message. When a user joins a group chat, then a TCP connection will be established with the chat server. Every time a user sends a message, it is first sent to the chat server, and then sent to the remaining participants in the group chat. This server does not store information about the messages sent in each group chat.

Finally, the remaining component in the architecture is the IoT node. As seen in section 2.4.1, this node is an iTag. It is similar in shape to a keychain and possesses a button. This will be very useful in circumstances where the smartphone is out of reach of the user. When this happens and the user feels in danger, he/she can press the button in the iTag. This will send a message via Bluetooth to the smartphone and should be recognized by the application, even if it is running in the background, generating an alert. If the alert is not cancelled, then a message with the user's current location and the type of alarm triggered

WeDoCare

will be sent to the police.

The mobile application's code follows the MVP architecture [37].

Chapter 4

Development

This chapter will reflect the whole process and work performed during the second semester to develop WeDoCare 3.0, including meetings with our partners.

4.1 Partner meetings

Throughout the second semester we had some meetings with our partners. The objective and outcome of each meeting is reported below.

4.1.1 State Secretary for Citizenship and Equality

On the 18th of February 2019 we went to Lisbon to meet with the State Secretary for Citizenship and Equality. We ended up meeting with her chief of staff and a colleague of hers. The purpose of this meeting was to explain the WeDoCare project and show a small demo of the application.

Following the presentation of WeDoCare and the showing of a small demo, they seemed quite interested in the application, and there was a brief exchange of ideas. We talked about how to test WeDoCare during Queima das Fitas and even the possibility of using WeDoCare in events with more people, like the music festivals that are held in June/July. Moreover, the State Secretary for Citizenship and Equality had mentioned in a previous meeting with our supervisors that a chat feature and a video repository feature would be very interesting, and that was reinforced during this meeting. They also mentioned that it would be good if the application had a set of tips that a user could read to stay more informed on how to act or avoid certain situations. They also wanted to gather statistics about the application usage, and how many people would read the tips and watch the videos. This is the reason why we

Development

needed to add more requirements to the application.

At the end of the meeting, we stated that we would contact them as soon as the chat, repository videos, and tips features were implemented. Therefore, as soon as those features were implemented, we contacted them, however still to this day we are waiting for a reply.

4.1.2 PSP

On the 28th of February 2019 we met with the PSP Commander Rui Filipe Coelho de Moura and his two deputy commissioners, alongside with one person responsible for the Ergue-te organization. The meeting took place at the Coimbra District Command. The purpose of this meeting was to establish the best way on how to conduct the first experience using WeDoCare, involving the PSP and the women aided by Ergue-te.

This meeting did not go as well as we would have hoped. We were told that this experience with WeDoCare would not be a simple process, as it is a real life scenario. Since our idea was to send a message to the police whenever a dangerous situation is detected, this would imply a new communication method with the police, differing from the existing ones. Considering that the police would be contacted in case of a dangerous situation, resources would need to be allocated in order to check the situation, and it would not be good if for some reason the alarm was a false one. We were told that to move to a real life scenario the WeDoCare project would need to be re-evaluated by higher ranked police officers and possibly by government officials. It is also important to mention that since the last time we had met with the PSP, there was a change in the PSP Commander. So in this recent meeting we had to explain the project again to the new PSP Commander. This is the main reason why WeDoCare needs to be re-evaluated.

For the re-evaluation process, we would need to elaborate a report explaining how WeDoCare works and how the police would be involved. The report would then be evaluated by those higher ranked police officers and government officials. This means the process needed for WeDoCare to be available for a real life scenario experience involving the police can, and most probably will, be very time consuming. To have an idea, it took us almost three weeks just to be able to schedule a meeting with the PSP. The next days after the meeting we reflected on what was said, and we decided to write the previously mentioned report, explaining how WeDoCare works. The report can be viewed in Appendix C.

4.1.3 Ergue-te and Saúde em Português

Following the meeting with the PSP, and after a week of reflection, we met with Ergue-te and Saúde em Português, with the aim of finding a suitable option to test WeDoCare without the intervention of the Police. We suggested using a method where the alert messages are not sent to the Police, but to a person responsible for those who use the application. If that person receives an alert message, then he/she decides if he/she should contact the Police or not.

Ergue-te chose not to use that approach, as they did not want to assume the responsibility of deciding if they should contact the Police or not. Therefore, it was decided that we would wait for the process to be approved by the Police, and only then we would test WeDoCare with Ergue-te. However, Saúde em Português decided to go with the aforementioned approach. With that said, WeDoCare is being tested in a real life scenario, through the people of the Saúde em Português organization, who are actual victims of violence. So far no bugs or malfunctions have been reported. This allows us to accomplish our main goal, as mentioned in section 3.2.1.

4.2 Rest API Server

As mentioned in section 3.2.5 where we discussed WeDoCare's architecture, the Rest API server handles requests like, login or register, requests for getting the list of tips, or even the informative videos. This section will explore this server's organization, technologies used and its implementation.

4.2.1 Organization

The Figure 4.1 shows the organization of the Rest API server. This server follows a slight variation of the MVC structure - Model-View-Controller. The MVC architectural pattern has existed for a long time in software engineering. MVC separates an application into three components - Model, View and Controller. The Model represents shape of the data and business logic, meaning it retrieves and saves data in a database. The View is a user interface and displays data to the user, while also allowing the user to modify the data. The Controller handles the user requests [60]. In this case, the server has a slight variation, since there is really no View implemented, as the user utilizes the mobile application. Nevertheless, when the user, for instance, wants to login, he has to insert the data on the mobile application. That data is sent via POST request to this server. The Controller then, handles that request and

Development

passes the data inserted by the user to the Model. The model saves the data, or verifies if the data is correct, by making queries to the database.

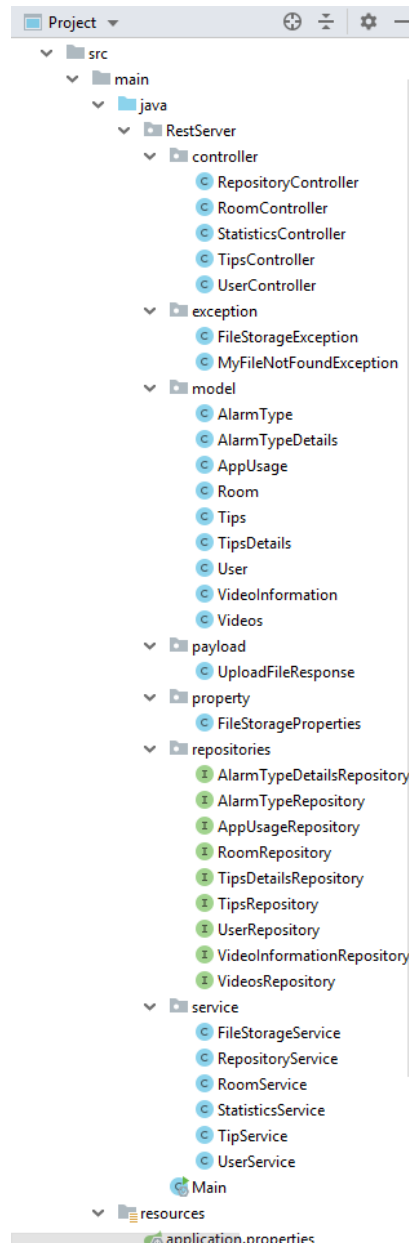


Fig. 4.1 Organization of the rest API server

In Figure 4.1 we can observe this MVC structure. There is a package named "controller", which holds the controllers used in the server. There is also a package named "model", where the entities of the database are located. For example, the User class is marked as an entity, meaning the database will create a table with the name "user". The package "repositories" has the interfaces that hold queries to the database, while the package "service" has the services

invoked by the controllers. In the subsection 4.2.4 we will see an example of all the process that goes through the controller and the model.

It is worth mentioning that the MVC pattern is very useful because it makes the code easier to understand, and it is very good for unit testing, as we can test each part individually, like we will see in chapter 5.

4.2.2 Technologies used/Third party libraries

The technologies and/or third party libraries used to develop this server are listed below:

- **Spring Boot:**

Spring Boot [61] provides a good platform for Java and Kotlin developers to develop a stand-alone and production-grade application with minimal configuration. Therefore, it is easy to use and understand, which leads to an increase in productivity and a decrease in development time. Spring Boot was used to develop this server.

- **Spring Data JPA:**

Spring Data JPA [62] simplifies the implementation of JPA based repositories. Implementing a data access layer of an application can be very time consuming, even for the simplest queries. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the amount of boilerplate code that we need to implement.

- **MySQL Connector/J:**

MySQL Connector/J [63] is the official JDBC driver for MySQL. It is used so the application can connect to the database and perform the desired operations.

- **JUnit:**

JUnit [64] is an open source unit testing framework for Java. It is very useful for developers to write and run repeatable tests, allowing them to discover bugs early and keep the code more reliable.

- **AssertJ:**

AssertJ [65] provides a rich set of assertions, and you can even write your own custom assertions, improving test code readability. It is also designed to be very easy to use within your favorite IDE.

- **Mockito:**

Mockito [66] is a mocking framework that lets you write tests with a clean and simple

Development

API. It is used mainly to mock dependencies. For example, if you want to write a test for a service that does some operations and needs to connect to the database, you do not want to really connect to the database. You just want to mock that connection to keep the test simple and fast.

4.2.3 Storage

The majority of the information is stored in a database. Below is Figure 4.2 that illustrates a ER diagram of the database implemented.

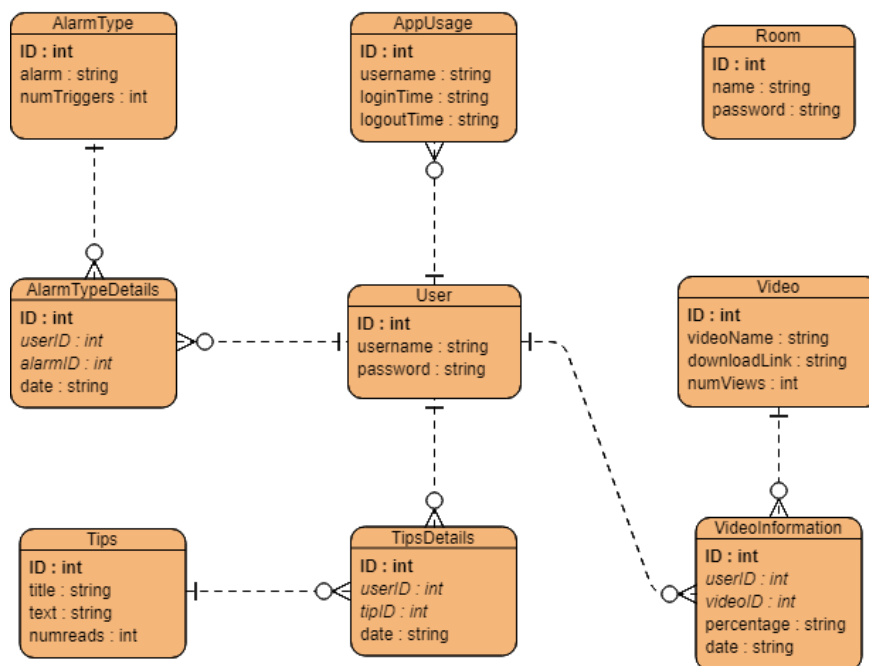


Fig. 4.2 ER diagram of the database

By observing figure 4.2 we observe that in each entity there is a variable written in bold style, meaning it is a primary key. In some entities there are also variables written in italic style, meaning they are foreign keys. The *User* and *Room* entities, as the names suggest, store information about the users and group chats, respectively. The *AppUsage* entity stores information about the application's usage, like when the user logs in and logs out. The *AlarmType* entity stores information about the alarms, more precisely how many times each alarm has been triggered, whilst the entity *alarmTypeDetails* saves information about each time an alarm was triggered. The server also saves information about how many times each tip has been read, through the entity *Tips*. And each time a tip is read, the entity *TipsDetails* stores additional information about the tip and the user that read the tip.

Regarding the videos, the information stored is the video name, the download link, and the number of views it has. The actual video is stored in the file system. This is because videos nowadays have a huge size and storing big files in a database is not always the best choice. A simple query to get the list of videos will also load all the content of the videos and that may affect the performance of the server. On the other hand, saving the videos in the file system is preferable for several reasons [67]:

- Better performance - since there is no need to load all the data of the videos.
- Simpler to save.
- Easier to migrate the data since one can just copy and paste the videos to the desired destination folder.

Finally, the entity *VideoInformation* stores information each time a video is viewed.

4.2.4 Implementation

In this subsection it will be shown a small example of how the Rest API server works, and how the controllers, services and repositories interact with each other. Let us take a operation like a user logging into the mobile application. The user will insert his/hers credentials on the smartphone and then the Android application will do a POST request with those credentials. This POST request will be handled by the *UserController* class, as we can observe in the Listing 4.1.

```
1 @RestController
2 @RequestMapping("/api")
3 public class UserController {
4     private final UserService userService;
5
6     public UserController(UserService userService) {
7         this.userService = userService;
8     }
9
10    @PostMapping("/login")
11    public void loginUser(@RequestBody User user, HttpServletResponse
12    response) {
13        if (!userService.loginUser(user)) {
14            response.setStatus(HttpServletResponse.SC_NOT_ACCEPTABLE);
15        }
16    }
17
18    // 406
```

Development

```
14     }
15 }
16
17 @PostMapping("/register")
18 public void registerUser(@RequestBody User user, HttpServletResponse
response) {
19     if (userService.registerUser(user)) {
20         response.setStatus(HttpServletResponse.SC_CREATED); // 201
21     } else {
22         response.setStatus(HttpServletResponse.SC_CONFLICT); // 409
23     }
24 }
25 }
```

Listing 4.1 UserController class

Since it is a login operation, the endpoint will be `/api/login`, meaning it is the `loginUser` method that is called. The credentials are sent in a JSON file and then mapped into a `User` object as we can see in line 11 of the listing above. This method just has an *if* condition to check if the user inserted the correct credentials. Nonetheless, the *if* condition calls a method in the `UserService` class, which we can see in Listing 4.2.

```
1 @Service
2 public class UserService {
3     private final UserRepository userRepository;
4
5     public UserService(UserRepository userRepository) {
6         this.userRepository = userRepository;
7     }
8
9     public boolean loginUser(User user){
10         String password = DigestUtils.sha256Hex(user.getPassword());
11
12         if(user.getUsername().equals("") || user.getPassword().equals("")){
13             return false;
14         }
15         return userRepository.verifyUserLogin(user.getUsername(),
password) != null;
16     }
17
18     public boolean registerUser(User user){
```

```
19     String password = DigestUtils.sha256Hex(user.getPassword());
20
21     if (user.getUsername().equals("") || user.getPassword().equals(""))
22     ){
23         return false;
24     }
25
26     if (userRepository.verifyUserRegister(user.getUsername()) ==
27     null) {
28         userRepository.save(new User(user.getUsername(), password));
29         return true;
30     } else {
31         return false;
32     }
33 }
```

Listing 4.2 UserService class

The method *loginUser* in this class first verifies if the username and password received are valid. If they are not, then it returns *false* and there is no need to connect to the database. On the other hand, if the credentials are valid, then a method in the *UserRepository* class is called, to communicate with the database, as seen in Listing 4.3.

```
1 public interface UserRepository extends CrudRepository<User, Long> {
2     @Query(value = "select id from users where username = ?1 and
3     password = ?2", nativeQuery = true)
4     Long verifyUserLogin(String username, String password);
5
6     @Query(value = "select username from users where username = ?1",
7     nativeQuery = true)
8     String verifyUserRegister(String username);
9 }
```

Listing 4.3 UserRepository class

The method called is *verifyUserLogin* which returns the user id in the database if it found a match with the credentials received. If for some reason a match is not found, then the method returns a null value. It is worth noting that in line 1 of the Listing above, the interface extends the *CrudRepository*. This is helpful because it already provides basic methods like, saving a user, updating a user or deleting a user, hence we do not need to code those methods.

This is the interaction between the controllers, services and repositories. It is identical for the remaining operations that the Rest API server needs to handle. First, the controller handles the request and passes the data to the service. The service performs some verification and, if needed, it passes the data to the repositories so it can be stored.

4.3 Chat Server

As mentioned in section 3.2.5, the chat server will store the messages when the application is distributed by our partnership with the State Secretary for Citizenship and Equality, but not when it is distributed by the people Ergue-te is helping, due to their privacy concerns. In this section it will be described the version where the messages are stored. Both versions of the chat server are almost identical, the only difference is the saving messages feature.

4.3.1 Organization

The Figure 4.3.1 illustrates how the server is organized.

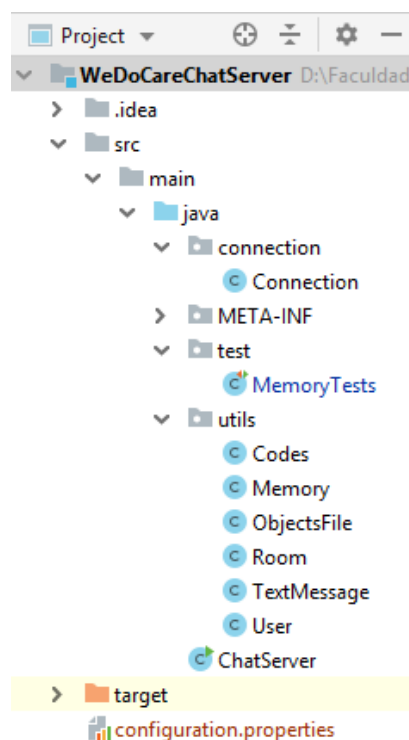


Fig. 4.3 Organization of the chat server

We can observe the package "utils" contains classes for general use, whilst the package "connection" holds the *Connection* class. This class is a thread and it is responsible for receiving and sending messages. The class *ChatServer* is the main class, where the server is initialized. In the section of the implementation we take a more detailed look at these previously mentioned classes.

4.3.2 Technologies used/Third party libraries

This is a simple chat server that makes use of sockets to establish TCP communications. For this reason, it was used only one third party library, and it was for testing, not for implementation. The third party library used was TestNG [68]. TestNG is a testing framework inspired from JUnit and NUnit but it introduces some new functionalities that make it easier to use. This is specially true in regular Java projects like this server. It was much easier to use this testing framework on the chat server.

4.3.3 Implementation

As previously mentioned, the class *ChatServer* initializes the server. In the following listing we can see that process.

```
1 private void initServer() {
2     try {
3         int clientID = 0;
4         //this is the class that stores and loads the saved rooms and
        users
5         Memory memory = new Memory();
6
7         //creates a socket for the server
8         ServerSocket serverSocket = new ServerSocket(port);
9
10        // waiting for clients
11        while (!serverSocket.isClosed()) {
12            //accept() is a blocking call meaning it won't advance
        further until a client connects
13            Socket clientSocket = serverSocket.accept();
14            //this synchronized method is to avoid more than one thread
        accessing something at the same time
15            synchronized (allClients) {
16                allClients.put(clientID, clientSocket);
17            }
18        }
19    }
20 }
```

Development

```
18
19         //connection between client and server established
20         new Connection( allClients , clientID ,memory );
21         clientID ++;
22     }
23 } catch (IOException e) {
24     System.err.println("Error launching server");
25     e.printStackTrace();
26 }
27 }
```

Listing 4.4 ChatServer class

In line 8 of the above Listing, we see that the server initializes in the port specified by the variable *port*. It then advances to a *while* where it is always expecting users to connect. When a user connects - line 13 - then its ID is saved, as well as the socket that he/she is using. In line 30 we see a new thread being created to deal with the user that just joined. The Listing 4.5 shows how a thread receives and sends messages.

```
1 public void run() {
2     String roomName = "";
3     try {
4         roomName = in.readUTF();
5         synchronized (memory) {
6             memory.addUserToRoom(roomName, threadNumber);
7         }
8
9         while( true) {
10            // it's a chat
11            roomName = in.readUTF();
12            String message = in.readUTF();
13
14            ArrayList<Integer> clientsToSendMessage;
15            TextMessage msg;
16
17            String [] messageAux = message.split(": ");
18            msg = new TextMessage(messageAux[0], messageAux[1]);
19
20            synchronized (memory) {
21                clientsToSendMessage = memory.getUsersInRooms().get(
22                roomName);
23                memory.addMessage(roomName, msg);
```

```
23     }
24
25     for(Integer clientID : clientsToSendMessage){
26         synchronized (allClients){
27             this.out = new DataOutputStream(allClients.get(
clientID).getOutputStream());
28             this.out.writeUTF(message);
29         }
30     }
31 }
32 } catch (EOFException e){
33     synchronized (memory){
34         memory.removeUserFromRoom(roomName, threadNumber);
35     }
36     synchronized (allClients){
37         allClients.remove(threadNumber);
38     }
39 } catch (IOException e){
40 }
41 }
```

Listing 4.5 Connection class

When a user connects to a group chat, the thread adds that user to that group chat - line 6. Then while the user is connected, if someone sends a message to the group, this thread will receive that message - line 12 - and will determine to which users to send the message - line 21. Since the information is stored in an Hashmap, the search time is very fast. In fact, it is in constant time. In line 22 the message is saved in a data file, and in line 25 there is a *for* loop that will send the message to each user that is in that group chat.

4.4 Android application

As illustrated in section 3.2.5 the Android application now communicates with a Rest API server, a chat server and with an IoT node - iTag. With that in mind, the next subsections will explain how the code is organized, what technologies and/or third party libraries were used, and it will be shown some examples of how the application was implemented.

4.4.1 Organization

In Figure 4.4 we can observe how the Android application is organized.

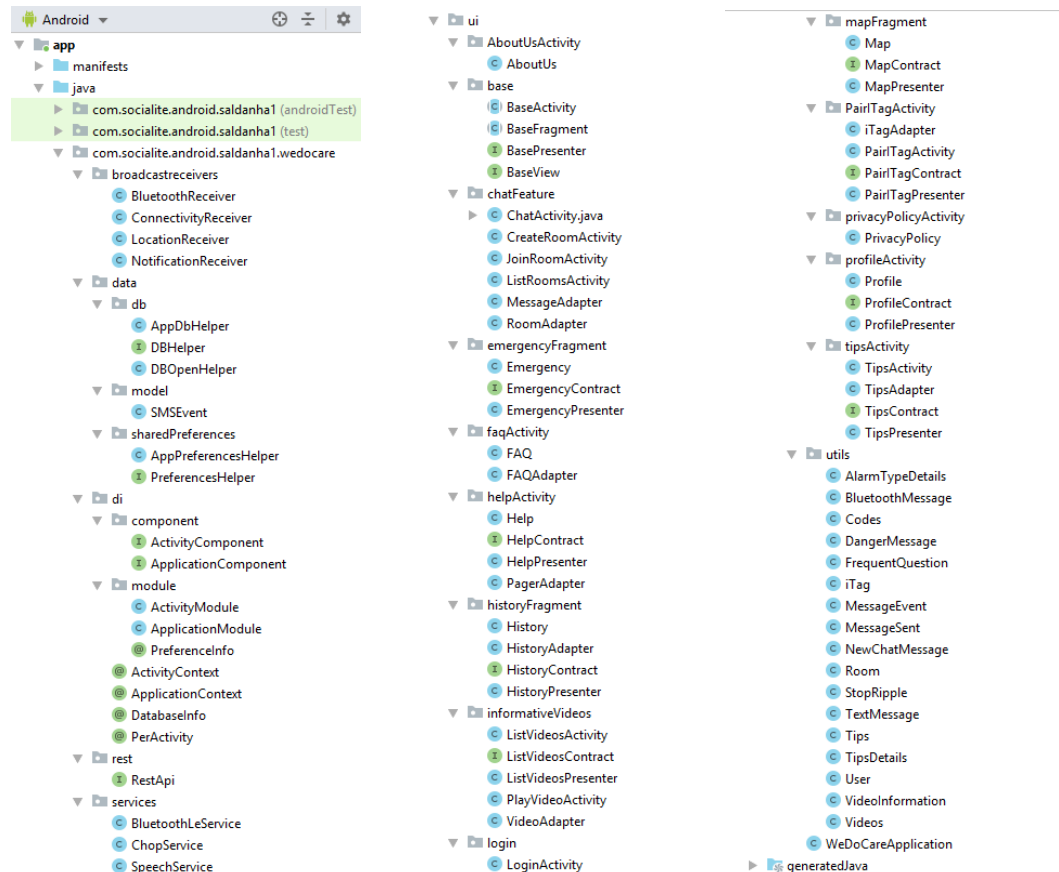


Fig. 4.4 Organization of the Android Application

As we can see, the project is quite extensive. Let us start with the package "broadcastreceivers". This package contains classes that will be listening for changes in the Bluetooth service, connectivity service, location service, and when a notification is triggered. The package "data" holds classes that will save data in the local database, and in the SharedPreferences file. The package "di" contains classes for dependency injection. The concept of dependency injection is better explained in the next subsection, where it is mentioned the third party libraries used.

The "rest" package has an interface with the methods used to communicate with the Rest API server, whilst in the package "service" there are classes for the Bluetooth service, Chop service - gesture recognition -, and the Speech service, for the speech recognition. The "ui"

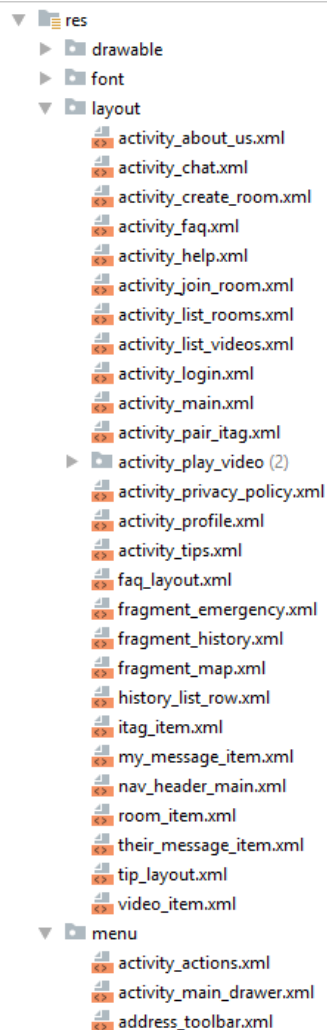


Fig. 4.5 Organization of the Android Application - UI

package, which is the most extensive one, contains all the packages and classes responsible for the code logic and UI interaction. This means that it is in this package where all the features are implemented. In the package "utils" it is where we see the most general classes, used to help with the code logic.

Finally, in Figure 4.5 we see the files responsible for the UI part of the application.

4.4.2 Technologies used/Third party libraries

The technologies and/or third party libraries used to develop the Android application are listed below:

- **greenDao:**

greenDAO [69] is an Android ORM used to create SQLite databases by mapping java objects to database tables. Methods like storing, updating and deleting are already provided by the framework, making it easier for the developers and it is less time consuming. Other advantages of this framework include encryption of data, low memory consumption and small library size.

- **Dagger:**

Dagger [70] is a framework for dependency injection, which is a technique in software development whereby an object depends on other object. Dagger generates code for proper dependency injection, relieving the developer from writing that code himself/herself. It makes use of a set of annotations to build a graph of objects linked by their dependencies.

- **Butter Knife:**

Butter Knife [71] is a library for binding views and methods that makes use of annotations to prevent developers from having to write repeated code.

- **Event Bus:**

Event Bus [72] is a annotation based library that enables communication between the several classes used in the application, employing the publish/subscriber pattern. It has a high and fast performance, delivering results in the main thread and background threads.

- **Retrofit:**

Retrofit [73] is a type-safe HTTP client for Android and Java, allowing the developer to easily make HTTP request to a Rest server.

- **LibVLC:**

LibVLC [74] is a very powerful, yet simple, library designed by the VideoLan Organization. Its features include playing songs, videos, and even capture live streaming, while making it simple for the developer to implement said features.

- **Mockito:**

This framework was already explained in section 4.2.2.

- **PermissionsDispatcher**

PermissionsDispatcher [75] is an annotation based API to handle Android runtime permissions with minimum lines of code, making it less time consuming for the developer to deal with the permission results.

- **Google Play Services:**

With Google Play Services [76], the application can use Google features, such as Maps, Location, Speech Recognition API, and receive automatic Google updates and bug fixes if it is deployed on the Google Play Store.

- **Material Dialogs:**

Material Dialogs [77] is an API to create dialog messages according to the Material Design Principles. With this API it is easier and less time consuming to show Dialogs to users when an error occurs or when critical information needs acknowledgment or input.

- **Ripple Background:**

Ripple Background [78] is customizable animation which can be integrated with other elements, such as an `ImageView`. WeDoCare starts the Ripple Background animation when the user clicks the panic button.

- **Sensey:**

Sensey [79] is a library which makes playing with sensor events and detecting gestures very simple, by eliminating a lot of boilerplate code, thus reducing development time.

4.4.3 Implementation

This subsection presents two small examples of how the Android application communicates with the Rest API server and the chat server. It also explains how the application detects that a person has pressed the button in the iTag, and consequently, triggers an alarm.

Below is Listing 4.6, where there is an interface which holds some of the methods used to communicate with the Rest API server.

```
1 public interface RestApi {
2     @POST("api/login")
3     Call<Void> loginUser(@Body User user);
4
5     @POST("api/register")
6         // register
7     Call<Void> registerUser(@Body User user);
8
9     @POST("api/updateViews/{fileName}")
10    Call<Void> updateViews(@Path("fileName") String fileName);
11    ...

```

12 }

Listing 4.6 RestApi interface

This interface contains more methods but the logic is similar for all, so it is unnecessary to show everything. Every method has an annotation which specifies the HTTP request, and the endpoint to which that request should be sent. With the annotation "Body" the Retrofit API knows that it needs to map the User object to a JSON object. This JSON object will be sent to the Rest API server, which was already explained how it works. The Retrofit API also allows the developer to pass arguments in the url as it is seen in the method *updateViews* of the above listing. The Listing 4.7 illustrates how one of these methods can be used to send data to the Rest API server.

```
1 @OnClick(R.id.button_login)
2 public void login() {
3     username = editUsername.getText().toString();
4     password = editPassword.getText().toString();
5     User user = new User(username, password);
6     Call<Void> call = restApi.loginUser(user);
7     call.enqueue(new Callback<Void>() {
8         @Override
9         public void onResponse(Call<Void> call, Response<Void> response)
10        {
11            if(!response.isSuccessful()){
12                //something failed
13                wrongCredentials.setText(getString(R.string.
14                wrong_credentials));
15                return;
16            }
17            if(!wrongCredentials.getText().toString().equals("")){
18                wrongCredentials.setText("");
19            }
20            preferencesHelper.setUsername(user.getUsername());
21            weDoCareApplication.setUser(user.getUsername());
22            //login was successful so we start the next activity
23            Intent start = new Intent(LoginActivity.this, Help.class);
24            startActivity(start);
25        }
26        @Override
27        public void onFailure(Call<Void> call, Throwable t) { Log.e(TAG,
28        t.getMessage()); }
29    });
```


27 }

Listing 4.7 Android Rest communication example

As shown in Listing 4.7, it is simple to do a rest call to the server. In line 5 it is created a *User* object with the credentials inserted by the user, and then in line 6 that object is passed to the method *loginUser*. The Retrofit API issues a callback, waiting for the response of the Rest server. If the credentials do not match, then a error text will appear - line 12. However, if the credentials are correct then the next activity is started, which in this case will be the main screen where there is the panic button. The process is the same for all the other rest requests to the server.

Now, we take a look at how the Android application communicates with the chat server. The Listing below shows a user connecting to the chat server.

```

1 private class ConnectToServerTask extends AsyncTask<Void, Void, Boolean> {
2     @Override
3     protected Boolean doInBackground(Void... voids) {
4         //now we try to connect to the server
5         Log.i(TAG, "Connecting to server");
6         try {
7             socket = new Socket(CHAT_HOST, CHAT_PORT);
8             Log.i(TAG, "Connected to the server");
9             DataOutputStream out = new DataOutputStream(socket.
10 getOutputStream());
11             out.writeUTF(roomName);
12             return true;
13         } catch (IOException e) {
14             Log.e(TAG, "Error connecting to the server");
15             e.printStackTrace();
16             return false;
17         }
18     }
19     @Override
20     protected void onPostExecute(Boolean connected) {
21         if(connected){
22             readMessages = new ConnectionClient(socket);
23             readMessages.start();
24         }
25     }
26     else {

```

Development

```
25         Toast.makeText(getApplicationContext(), "Unable to connect to the
server", Toast.LENGTH_SHORT).show();
26     }
27 }
28 }
```

Listing 4.8 Android Chat communication (1)

The listing above shows that in line 7 the user is trying to connect to the chat server using a socket and specifying the IP address and port where the server is running. It should be mentioned that the piece of code illustrated in the above Listing is running in a background thread, in this case an *AsyncTask*. It needs to be run in a background thread because Android does not allow developers to do network operations in the main thread, as it could take a long time and make the user experience very poor. Nevertheless, when the user is connected, then the method *onPostExecute* is called and creates a new thread for this user. This thread will be listening for messages that others users might be sending. The following two listings show the existing flow when a user receives a message.

```
1 class ConnectionClient extends Thread {
2     private final String TAG = this.getClass().getSimpleName();
3     private DataInputStream in;
4     private Socket s;
5
6     public ConnectionClient(Socket serverSocket){
7         try {
8             s = serverSocket;
9             in = new DataInputStream(s.getInputStream());
10        } catch (IOException e){
11            Log.e(TAG, "Error getting InputStream");
12            e.printStackTrace();
13        }
14    }
15    public void run(){
16        try{
17            while (true){
18                String data = in.readUTF();
19                EventBus.getDefault().post(new NewChatMessage(data));
20            }
21        } catch (IOException e){
22            Log.e(TAG, "Error while reading from the server: "+e);
23        }
24    }
}
```

25 }

Listing 4.9 Android Chat communication (2)

```

1 @Subscribe
2 public void OnEvent(final NewChatMessage event){
3     runOnUiThread(() -> {
4         String [] content = event.getMessage().split(": ");
5         TextMessage textMessage = new TextMessage(content[0], content
6         [1]);
7         textMessagesList.add(textMessage);
8         adapter.notifyDataSetChanged();
9     });

```

Listing 4.10 Android Chat communication (3)

In Listing 4.9 we can observe the code for a simple thread that is waiting to receive messages. When any user within that group chat sends a message, this thread will receive it - line 18 - and then, through the use of the Event Bus library, it will send the message to the class responsible for the chat group. This can be seen in line 19. In Listing 4.10 we have the method that will receive the message sent through the Event Bus library. As it was mentioned in section 4.4.2, this library follows a publish-subscribe pattern, hence why the method has the annotation *Subscribe*. This method receives the message and displays it in the chat, although it needs to run on the main thread. This is because Android does not allow for background threads to interact with the UI.

Lastly, now we explain how the application detects that a person has pressed the button in the iTag. As mentioned before in this report, the iTag communicates via BLE. This means that we needed to use the BLE feature in the Android operating system. We created a class named *BluetoothService*. This class will continuously search for BLE devices, such as an iTag. Listing 4.11 shows the most important methods of this class.

```

1 public class BluetoothLeService extends Service {
2     ...
3     private void scanLeDevice(final boolean enable) {
4         try{
5             if (!mBluetoothAdapter.isEnabled()){
6                 stopSelf();
7             }
8             if (enable) {
9                 // Stops scanning after a pre-defined scan period.

```

Development

```
10         handler.postDelayed(() -> {
11             mScanning = false;
12             if (mBluetoothAdapter.getBluetoothLeScanner() != null
13         ){
14                 mBluetoothAdapter.getBluetoothLeScanner().
15         stopScan(mScanCallback);
16             }
17             }, SCAN_PERIOD);
18             mScanning = true;
19             mBluetoothAdapter.getBluetoothLeScanner().startScan(
20         mScanCallback);
21         } else {
22             mScanning = false;
23             mBluetoothAdapter.getBluetoothLeScanner().stopScan(
24         mScanCallback);
25         }
26     } catch (Exception e){
27         e.printStackTrace();
28         Log.e(TAG, "scanLeDevice: Bluetooth turned off ..." + e);
29     }
30 }
31
32 private ScanCallback mScanCallback = new ScanCallback() {
33     @Override
34     public void onScanResult(int callbackType, ScanResult result) {
35         super.onScanResult(callbackType, result);
36         if (result.getDevice().getName() != null && result.getDevice
37         ().getName().trim().equalsIgnoreCase("iTAG") && !detected) {
38             detected = true;
39             mBluetoothAdapter.getBluetoothLeScanner().stopScan(
40         mScanCallback);
41             search = false;
42             if (searchDevices != null){
43                 searchDevices.interrupt();
44             }
45             searchDevices = null;
46             EventBus.getDefault().post(new DangerMessage("Alerta
47         iTAG", result.getDevice().getAddress()));
48         }
49     }
50 };
51 ...
```

45 }

Listing 4.11 BluetoothService class

First, for this detection mechanism to work properly, the user must pair his/her iTag with the application. A screenshot of that menu can be seen in Figure 4.9. Once the iTag is paired, its the MAC address is saved in the application. Now, when this Bluetooth service executes, first it checks if the Bluetooth is enabled on the smartphone - line 5. If it's enabled, then the scanning starts - line 17. This service scans for 15 seconds, which is the value of the variable *SCAN_PERIOD* - line 15. After the 15 seconds, it stops for 2 seconds. When the 2 seconds are over, it does the same cycle, never stopping until the application is closed by the user, or until the user disables the Bluetooth.

Whenever a user clicks the button in the iTag, the iTag signalizes it via BLE, and the application detects it and executes the callback defined in line 28. If the signal comes from an iTag, then it uses the already mentioned Event Bus library to send a message to the main class - line 40. The main class will verify if the MAC address of the iTag matches the MAC address previously saved. If so, then it triggers an alarm. We can also observe that the service stops when it recognizes the pressing of an iTag - line 37. This is to avoid triggering too many alerts in the case that a person clicks on the iTag button several times. If this happens, the alert is only generated once, and only notification will appear. That way, only one message is sent, instead a multiple messages.

4.4.4 User Interface

Below are some screen shots of the application, already improved with the feedback of the people who did the usability test, which will be explained in more detail in section 5.4.

We decided to use the design of the previous WeDoCare, but with some slight improvements. Therefore, for the new menu/screens that need to be implemented, we used the color red as seen in the screenshots above.

Development

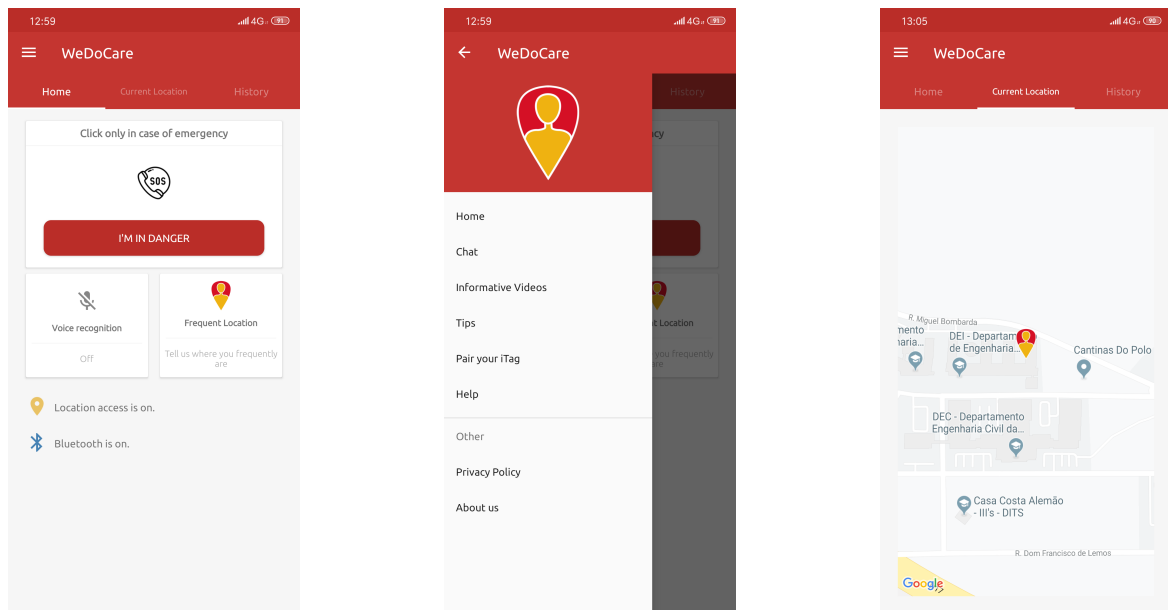


Fig. 4.6 WeDoCare main screen, menu drawer and location tab

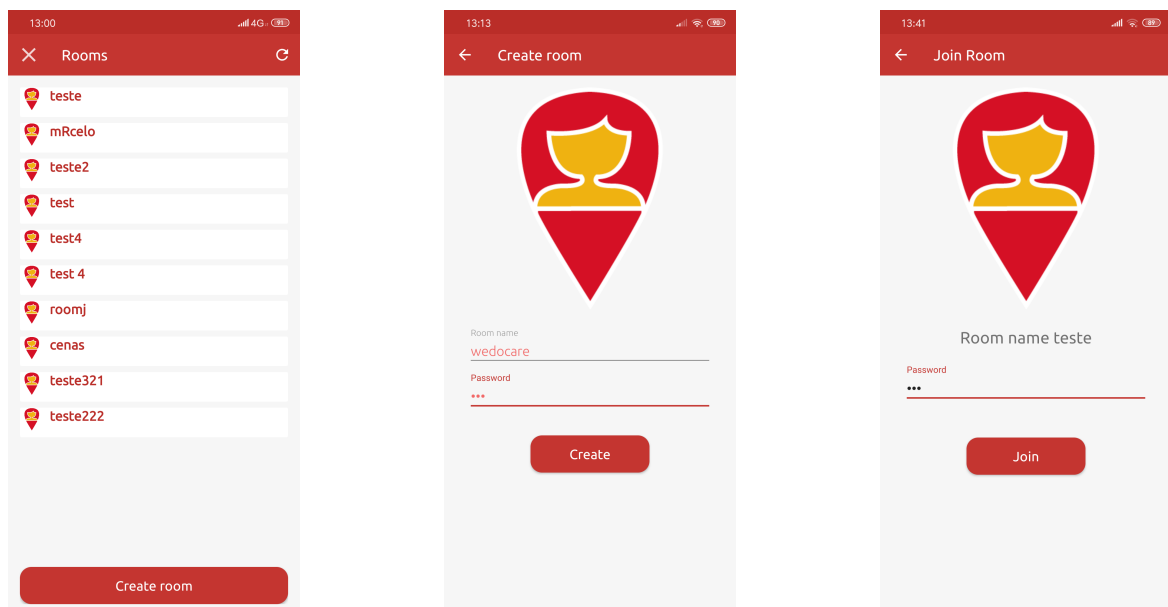


Fig. 4.7 List of group chats menu, create group chat menu, and join group chat menu

4.4 Android application

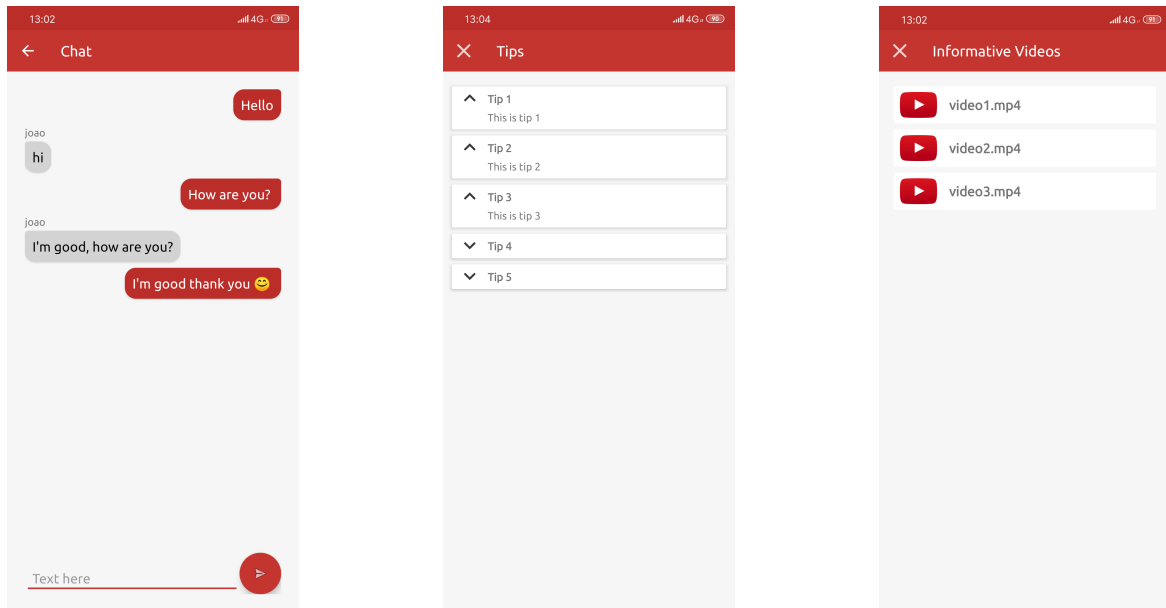


Fig. 4.8 Chat screen, tips menu, and videos menu

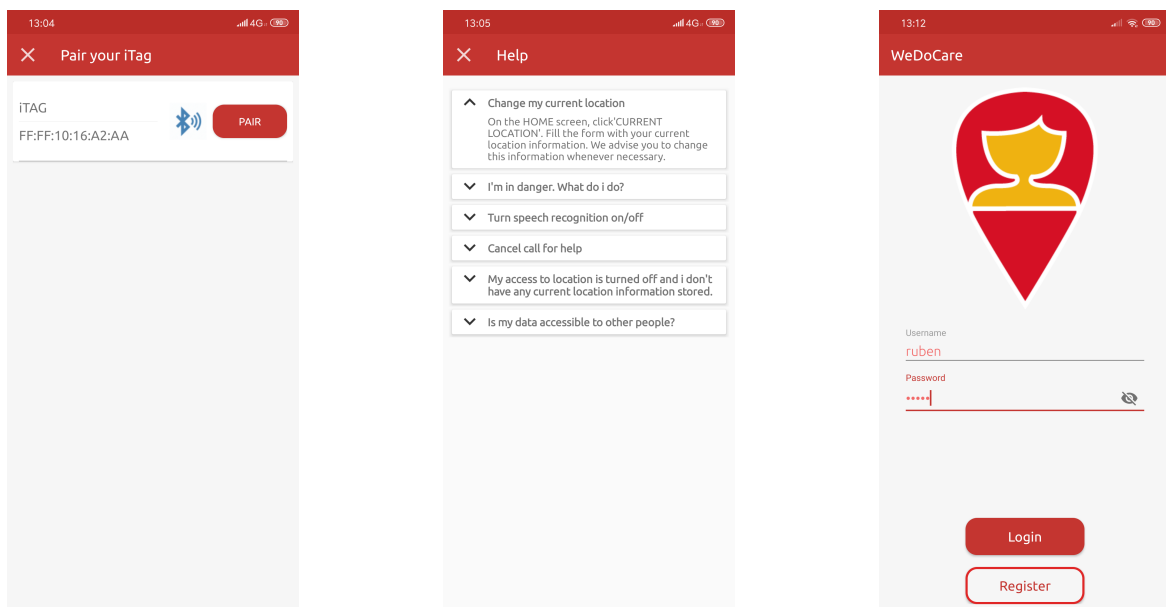


Fig. 4.9 iTag menu, help menu, and login/register menu

4.5 WeDoCare Website

WeDoCare already had an active website [85], although it was outdated since it had information about the previous version of WeDoCare. We chose to keep the webpage's main structure and layout and change only the information content. For that, we utilized some of the most common tools for web development, which are HTML, CSS and JavaScript. To make the development process easier, we used already integrated libraries. These libraries are Bootstrap [80], WOW.js [81], Animate.css [82], Font Awesome [83], and jQuery [84]. Moreover, the website is supported in both English and Portuguese languages. The website is illustrated in the figures below.

With the website updated, we can now divulge even more our application, and try to find more associations that can benefit from using WeDoCare.



Fig. 4.10 WeDoCare website - Home page

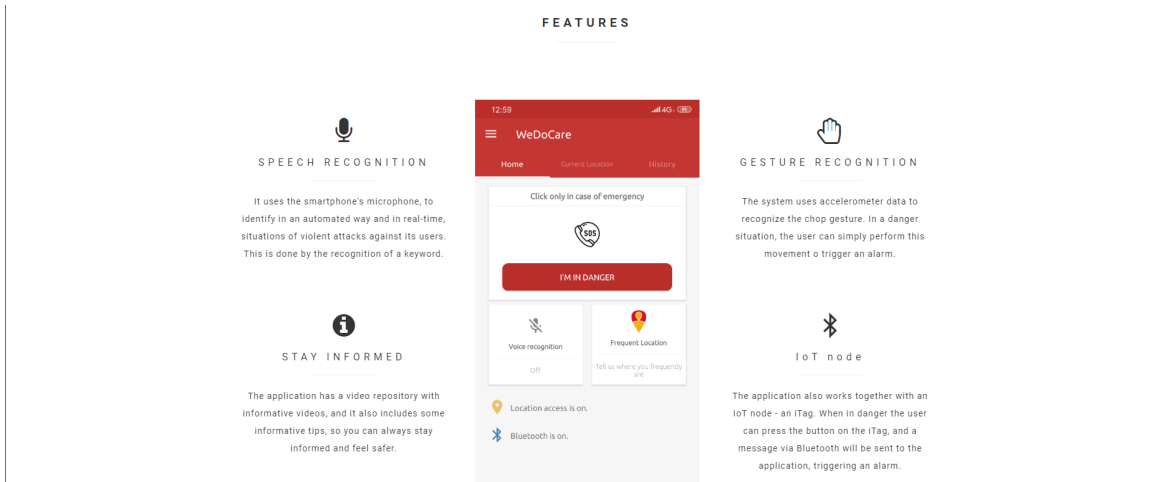


Fig. 4.11 WeDoCare website - Features

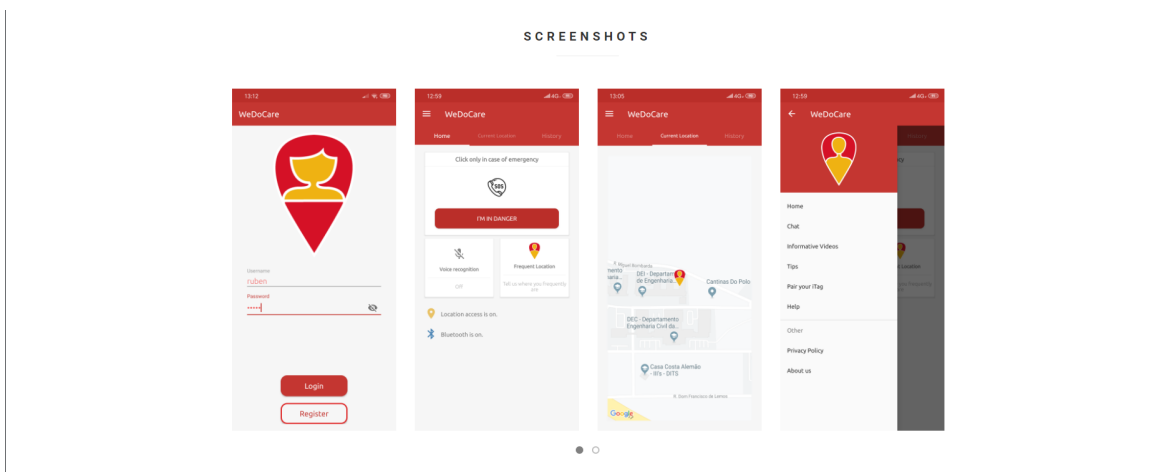


Fig. 4.12 WeDoCare website - Screenshots

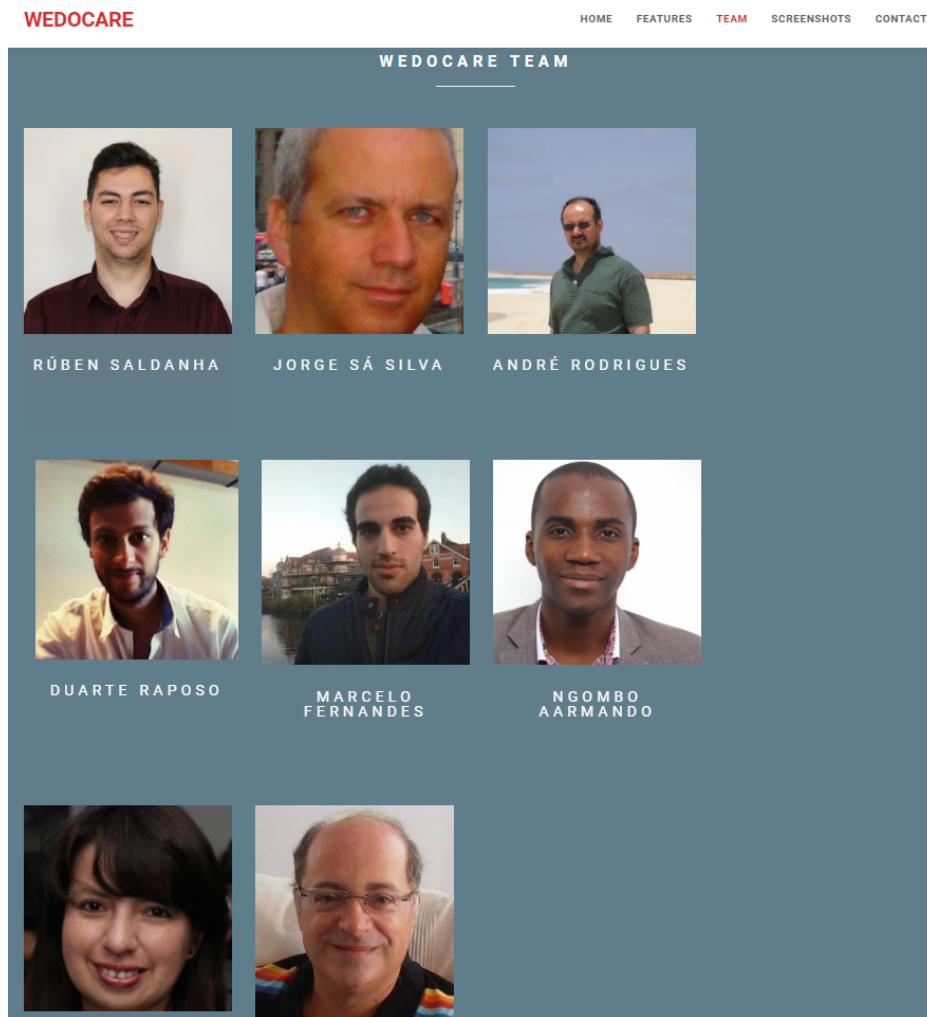


Fig. 4.13 WeDoCare website - Team



Fig. 4.14 WeDoCare website home page - Footer

Chapter 5

Tests

Software testing is defined as an activity to check whether the actual results match the expected results [86]. One of the main goals is to ensure that the software system is defect free. It should be mentioned that a defect is different from a bug. A bug is the consequence of a coding fault, while a defect is a variation or deviation from the original business requirements [87]. For instance, when a tester tests a login module and inserts the correct credentials, but somehow it is shown an error message, then the tester has just found a defect. When the defect is reported to the developer, and he recognizes it, it is called a bug.

For this project, we ran unit tests on the Rest API server, chat server, and on the Android application. We wanted to ensure that the existing methods were capable of handling, as expected, numerous different types of inputs. This is related to the reliability requirement established in section 3.2.4. For this requirement we also ran some tests with Ergue-te and Saúde em Português, as is it explained in section 5.5.1 and in section 5.5.2. We also ran performance tests, so we could validate the performance requirement, as well as battery tests, with the goal of fulfilling the battery requirement. Lastly, we performed usability tests with several people to validate the usability requirement, and to verify if WeDoCare's design was good or if it needed some improvement. In the following sections, we explore how each test was run, and what were the results attained.

5.1 Unit tests

Unit tests test the smallest testable part of any software. The purpose is to validate that each unit of the software performs as required [88]. Below are three subsections explaining how the different servers, and Android application, were tested using this technique.

5.1.1 Rest Api Server

Below, in Figure 5.1, we observe how the tests are organized in this server.

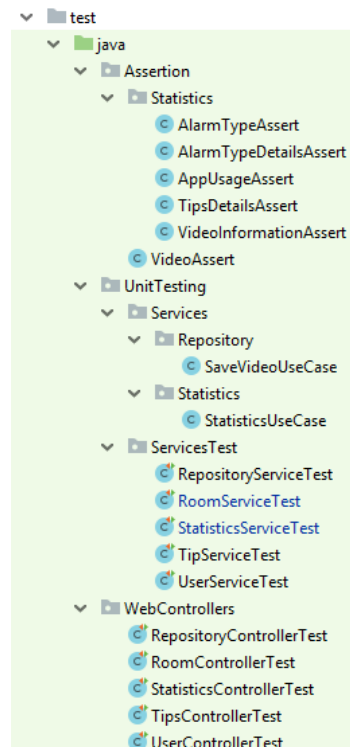


Fig. 5.1 Tests organization on the Rest API server

As it was explained in section 4.2.4 the Rest API server has web controllers, that handle the HTTP requests, and services, which perform some operations and then call repository methods to communicate with the database. It is divided like this because it follows a MVC pattern, which makes it easier to test. For this reason, and as it is shown in Figure 5.1, unit tests were performed in each web controller and in each service. The tools used for the unit testing were JUnit [64], Mockito [66] and AssertJ [65]. These tools were already explained in section 4.2.2.

Web Controllers

As previously mentioned, every controller was tested. However, since there are five controllers, and the test logic is always the same, here it will only be shown the unit tests for the *UserController*. As illustrated in Listing 4.1, the *UserController* class would return the code 200 if the user inserted the right credentials for the login, or it would return the code 406 if the credentials were wrong. Furthermore, the class would also return the code 201 if a user

registered successfully, but it would reply with the code 409 if the user failed to register. The Listing 5.1 shows all the tests to the *UserController* class.

```

1 @RunWith(SpringRunner.class)
2 @SpringBootTest(classes = Main.class)
3 @TestMethodOrder(MethodOrderer.Alphanumeric.class)
4 public class UserControllerTest {
5     private MockMvc mockMvc;
6     private ObjectMapper objectMapper;
7     @Autowired WebApplicationContext webApplicationContext;
8
9     @Before
10    public void setUp() {
11        this.mockMvc = MockMvcBuilders.webAppContextSetup(
12            webApplicationContext).build();
13        objectMapper = new ObjectMapper();
14    }
15
16    @Test
17    public void awhenValidRegistration() throws Exception {
18        User user = new User("test1", "test123");
19        mockMvc.perform(post("/api/register")
20            .contentType(MediaType.APPLICATION_JSON)
21            .content(objectMapper.writeValueAsString(user)))
22            .andExpect(status().isCreated());
23    }
24
25    @Test
26    public void bwhenInvalidRegistration() throws Exception {
27        User user = new User("test1", "test");
28        mockMvc.perform(post("/api/register")
29            .contentType(MediaType.APPLICATION_JSON)
30            .content(objectMapper.writeValueAsString(user)))
31            .andExpect(status().is(409));
32    }
33
34    @Test
35    public void cwhenValidLogin() throws Exception {
36        User user = new User("test1", "test123");
37        mockMvc.perform(post("/api/login")
38            .contentType(MediaType.APPLICATION_JSON)
39            .content(objectMapper.writeValueAsString(user)))
40            .andExpect(status().isOk());

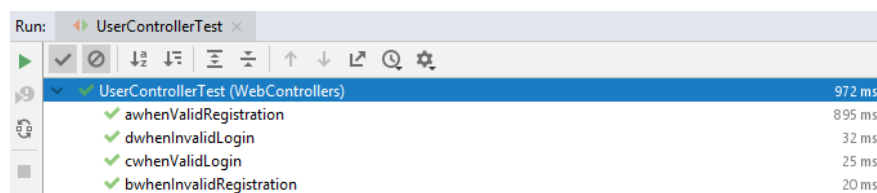
```

Tests

```
40     }
41
42     @Test
43     public void dwhenInvalidLogin() throws Exception {
44         User user = new User("test2", "test456");
45         mockMvc.perform(post("/api/login")
46             .contentType(MediaType.APPLICATION_JSON)
47             .content(objectMapper.writeValueAsString(user)))
48             .andExpect(status().is(406));
49     }
50 }
```

Listing 5.1 UserControllerTest class

Before the actual tests are performed, first there is a setup method - line 10 - that creates the context of a Spring Boot application, in order for the controller to be tested. After this setup method, there are 4 test methods. One tests a valid registration, another tests an invalid registration, another tests a valid login, and the last one tests an invalid login. Each method expects a different code as a reply. If the code is the expected one, then the test is passed, otherwise, the test is failed.



Test Method	Execution Time
UserControllerTest (WebControllers)	972 ms
✓ awhenValidRegistration	895 ms
✓ dwhenInvalidLogin	32 ms
✓ cwwhenValidLogin	25 ms
✓ bwhenInvalidRegistration	20 ms

Fig. 5.2 UserController test

The Figure 5.2 shows that all tests were passed. Although the tests performed on the other controllers will not be detailed here, since the logic is always the same, every controller was tested successfully.

Services

The same reasoning used in the last subsection will be applied here, meaning it will be shown only the tests performed to the *UserService* class. All the other services were tested as well, but the logic is the same, so there is no point in being repetitive. The Listing 5.2 illustrates all the tests performed to the *UserService* class.

```
1 @ExtendWith(MockitoExtension.class)
2 @RunWith(MockitoJUnitRunner.class)
3 public class UserServiceTest {
4     @Mock private UserRepository userRepository;
5     @InjectMocks private UserService userService;
6
7     @Test
8     public void invalidRegistration(){
9         User user = new User("", "w54vt");
10        boolean savedUser = userService.registerUser(user);
11        Assert.assertFalse(savedUser);
12    }
13
14    @Test
15    public void validRegistration(){
16        User user = new User("test2", "test123");
17        when(userRepository.save(any(User.class))).thenReturn(firstArg());
18        boolean savedUser = userService.registerUser(user);
19        Assert.assertTrue(savedUser);
20    }
21
22    @Test
23    public void validLogin(){
24        User user = new User("test1", "test123");
25        boolean savedUser = userService.loginUser(user);
26        Assert.assertTrue(savedUser);
27    }
28
29    @Test
30    public void invalidLogin(){
31        User user = new User("test1", "");
32        boolean savedUser = userService.loginUser(user);
33        Assert.assertFalse(savedUser);
34    }
35 }
```

Listing 5.2 UserServiceTest class

In line 4 we see the use of the annotation *Mock*. In this case it is mocking the dependency of the *UserRepository* class. This is useful because it does not actually create that variable, and therefore it does not connect to the database. Here we want to test the service part of the server and not the query to the database, hence why that dependency is mocked. Following

Tests

that, there are again 4 tests to be executed. Again, one tests a successful login, another a successful registration, while the other two test a failed login and a failed registration.

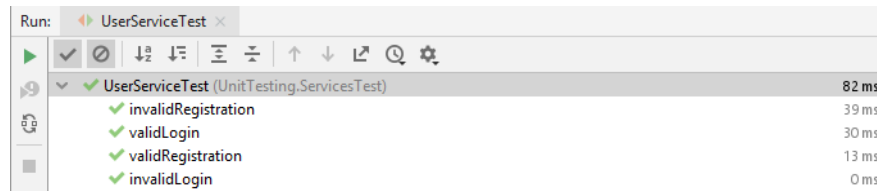


Fig. 5.3 UserService test

The Figure 5.3 shows that there are no defect in the *UserService* class. Although the tests to the other service classes will not be detailed here, all of them were tested successfully.

5.1.2 Chat Server

It was slightly harder to test the chat server using JUnit, so the tests were executed using the framework TestNG [68]. In Listing 5.3 we can see which tests were executed.

```
1 public class MemoryTests {
2     private Memory memory;
3     @BeforeMethod
4     public void setup () {
5         memory = new Memory ();
6     }
7     @Test
8     public void addUserToRoom_ReturnsTrue () {
9         boolean expected = true;
10        boolean actual = memory.addUserToRoom("test", 50);
11        Assert.assertEquals(actual, expected);
12    }
13    @Test
14    public void addUserToRoom_ReturnsFalse () {
15        boolean expected = false;
16        boolean actual = memory.addUserToRoom("test", -5);
17        Assert.assertEquals(actual, expected);
18    }
19    @Test
20    public void removeUserFromRoom_ReturnsTrue () {
21        boolean expected = true;
22        boolean actual = memory.removeUserFromRoom("test", 50);
```



```

23     Assert.assertEquals(actual, expected);
24 }
25 @Test
26 public void removeUserFromRoom_ReturnsFalse() {
27     boolean expected = false;
28     boolean actual = memory.removeUserFromRoom("test", -3);
29     Assert.assertEquals(actual, expected);
30 }
31 @Test
32 public void saveMessages() {
33     boolean expected = true;
34     boolean actual = memory.saveMessages();
35     Assert.assertEquals(actual, expected);
36 }
37 @Test
38 public void loadMessages() {
39     boolean expected = true;
40     boolean actual = memory.loadMessages();
41     Assert.assertEquals(actual, expected);
42 }
43 @AfterMethod
44 public void tearDown() {
45     memory = null;
46 }
47 }

```

Listing 5.3 MemoryTests class

As is shown in the previous Listing, there are test methods to add a user to a group chat, to remove a user from a group chat, and to test if the server saves the data and loads it properly. Again, first there is a method for setting up the tests, which creates an instance of the class *Memory*. There is also a final method that is executed after all the tests are run - line 52. That method cleans the variable *memory*, giving it a null value. Figure 5.4 shows that all tests performed were concluded successfully.

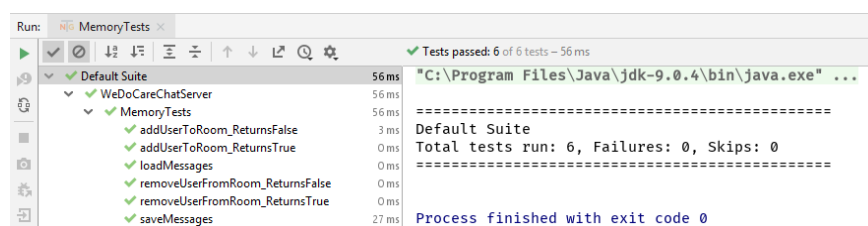


Fig. 5.4 Chat server test

5.1.3 Android application

As previously mentioned, the Android application follows the MVP pattern. Using this pattern makes it easier to understand the code and to add new features, fulfilling the maintainability requirement. The MVP architecture also turns the testing phase much simpler. All features were tested, nevertheless, since the logic is always identical, it will only be shown the test case for the login/register module. The Listing 5.4 shows the tests performed.

```
1 @RunWith( AndroidJUnit4 . class )
2 public class LoginActivityTest {
3     @Rule
4     public ActivityTestRule <LoginActivity> loginActivityRule = new
5     ActivityTestRule <>(LoginActivity . class );
6     private MockWebServer server ;
7
8     @Before
9     public void setUp () throws Exception {
10         server = new MockWebServer ();
11         server . start ();
12     }
13
14     @Test
15     public void invalidLogin () throws Exception {
16         server . enqueue ( new MockResponse () . setResponseCode ( 406 ) );
17
18         Intent intent = new Intent ();
19         loginActivityRule . launchActivity ( intent );
20
21         onView ( withId ( R . id . wrongUserPassowrd ) ) . check ( matches ( isDisplayed
22         ( ) ) );
23     }
24
25     @Test
26     public void invalidRegister () throws Exception {
27         server . enqueue ( new MockResponse () . setResponseCode ( 409 ) );
28
29         Intent intent = new Intent ();
30         loginActivityRule . launchActivity ( intent );
31
32         onView ( withId ( R . id . wrongUserPassowrd ) ) . check ( matches ( isDisplayed
33         ( ) ) );
34     }
35 }
```

```
33     @After
34     public void tearDown() throws Exception {
35         server.shutdown();
36     }
37 }
```

Listing 5.4 LoginTest case

In line 5 we can see that there is a mock instance of the server. In the test, there is no connection to the server, just a mocking of that connection. This mocked server starts in line 10. Following that, there are two methods. One will test an invalid login, while the other one will test an invalid registration. In line 20 and in line 30 we are verifying if the error message shows in the screen of the application after these failed attempts at login or register. In Figure 5.5 we can observe that the error message does show up.

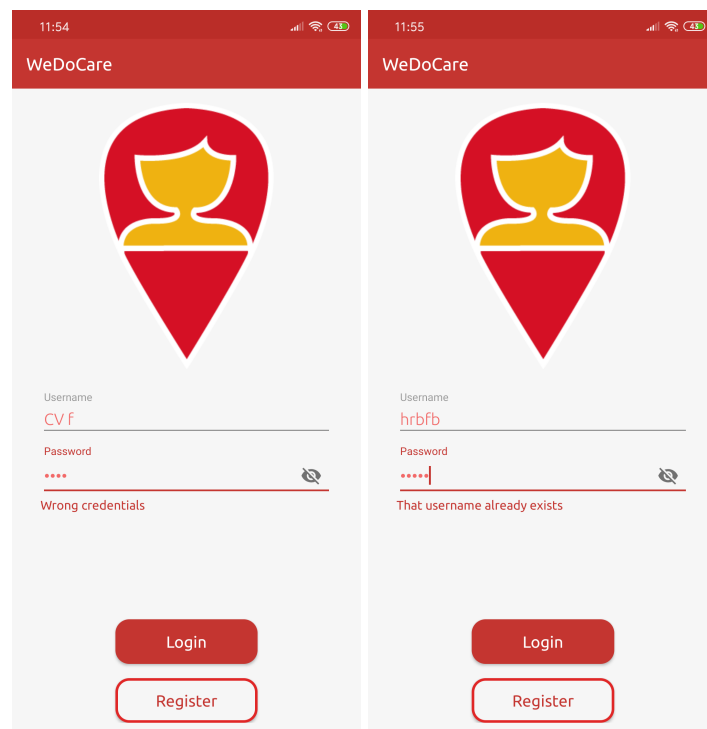


Fig. 5.5 Login test on the left, register test on the right

With all of the unit tests completed successfully, we can safely verify that all the functional requirements were implemented with success.

5.2 Performance tests

Nowadays, the response time of an application, in many situations, determines if a user continues to use the application or not. Users expect pages to load as quickly as possible and when they do not, satisfaction decreases [89].

Performance Testing is non-functional type of software testing technique to ensure software applications will perform well under their expected workload [90]. The main goal of this testing technique is not to find bugs, but to eliminate performance bottlenecks. A bottleneck is a single point or component within a system's overall function that holds back the overall performance [91].

Testing the performance of an application is not always trivial, as sometimes there are some obstacles to overcome. For instance, executing these performance tests on a machine that is in the same network as the server might create a network bottleneck. On the other hand there is a possibility of running the server and the performance tests on the same machine. However, in this last scenario the server would not utilize all available resources, meaning the performance would be lower. Nevertheless, to fulfil the performance requirement, we put the server on the *cloud*, in a virtual machine, within the *DEI* network. We used the tool *jmeter* [92] do perform these tests. This tool allows you to create test plans, which are composed of thread groups, logical controllers, listeners among other elements. The Figure 5.6 shows the tools being used with the test plan that was executed for this performance test.

As previously stated, the goal of this test was to fulfill the performance requirement, which affirms that the server should not prolong its response to a request made by the application. With that in mind, in these performance tests we increased the number of users in each iteration of the tests, to see how the server would handle multiple requests at the same time. In the first iteration we simulated 10 users making requests to the server. On the last iteration the numbers of users was 200. These requests were arbitrary in the sense that some were at the same time, and others were separated by a few seconds. All of this is possible to configure using *jmeter*.

In Figure 5.6 we can observe the test plan used. The thread group is called users, and specifies how many requests should be made to the server, meaning it simulates the number of users making requests. Below the thread group, we see 3 controllers, and each controller has an HTTP request, implying that there will be three different request. There will be a *GET* request to get the list of videos, a *POST* request to simulate a login, and a *GET* request

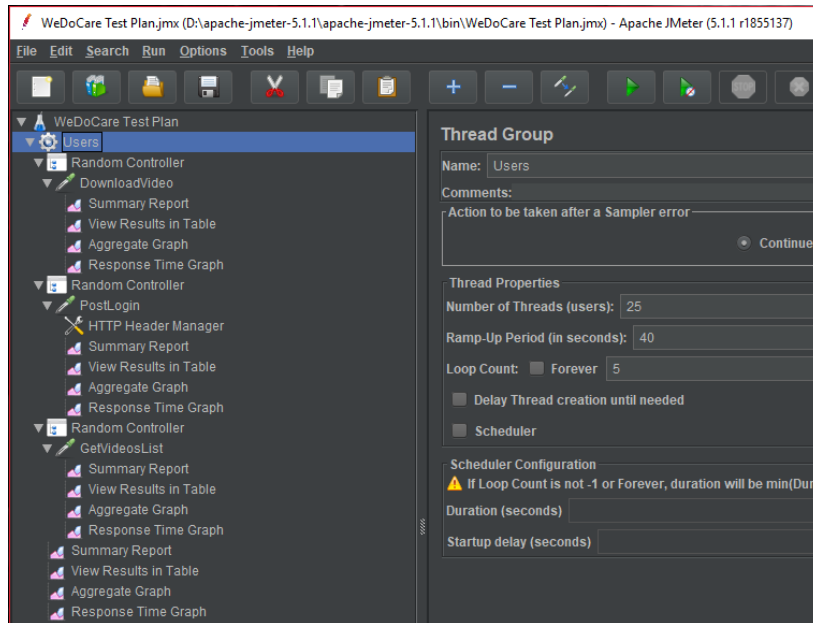


Fig. 5.6 Performance test with JMeter

to watch a video. It was expected that this last request to watch a video would take longer than the other two requests, since there was a larger amount of data being exchanged. We evaluated the latency and the throughput of these requests. Each iteration was run five times, so we will present the mean value of the results of each iteration. The table 5.1 presents the results obtained from executing the previously mentioned tests. These results were calculated automatically by *jmeter*.

To make it easier to read the results of the performance tests, and to see how the latency and throughput vary as the number of users goes up, we created four graphs which are represented in Figure 5.7, and Figure 5.8.

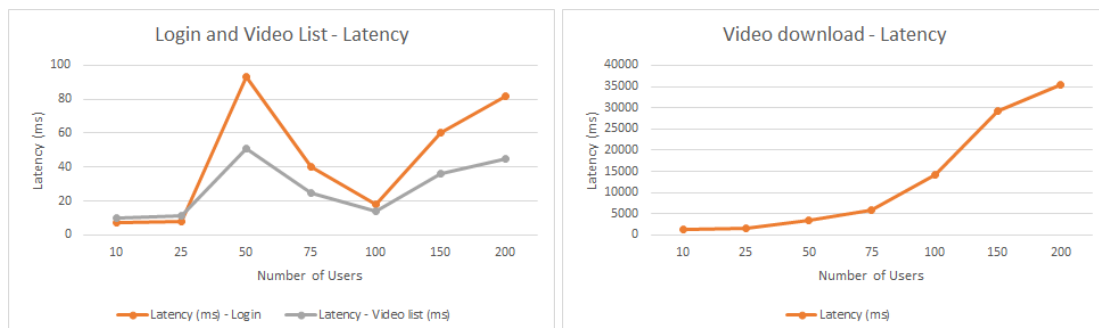


Fig. 5.7 Latency in milliseconds

Tests

Table 5.1 Performance test results

Number of Users		Latency (ms)			Throughput (req/s)
		Average	Minimum	Maximum	
10	Video Download	1397	1229	1833	0.12053
	Login	7	6	19	0.12091
	List of Videos	10	8	22	0.12091
25	Video Download	1642	1221	9160	0.20182
	Login	8	5	102	0.20223
	List of Videos	11	7	44	0.20223
50	Video Download	3405	1411	4848	0.34094
	Login	93	7	458	0.34169
	List of Videos	51	9	149	0.34169
75	Video Download	5962	865	7101	0.60646
	Login	40	4	251	0.60746
	List of Videos	25	6	112	0.60745
100	Video Download	14060	768	39896	0.15674
	Login	18	3	181	0.1568
	List of Videos	14	5	155	0.1568
150	Video Download	29250	533	170652	0.31552
	Login	60	3	1132	0.31565
	List of Videos	36	5	610	0.31565
200	Video Download	35485	1018	100147	0.51104
	Login	82	5	1103	0.43419
	List of Videos	45	6	406	0.43419

Analyzing the latency result in Figure 5.7 we verify that, as expected, the latency is much higher when a user wants to watch a video, since the amount of data exchanged is much higher. It is important to mention that from 100 users and above, the number of requests was done more sparsely. Our application will be mostly used to run in the background of the smartphones, so it will be very improbable that 100 users, or more, will be watching videos at the same time. We can verify that the latency still goes up when performing the *GET* request to watch a video. However, the latency of the other two requests drop when the requests are done more sparsely, although when the number of users is 200, they go up again. We also verify that the login request has, in general, a higher latency than the request to get the list of videos. It is understandable since you send data to the server, and then the server has to do some operations to verify if the credentials are correct or not. From these latency graphs we conclude that the server is prepared to handle regular *GET* and *POST* requests even with a high number of users. Nonetheless, the server is not fully prepared if a huge

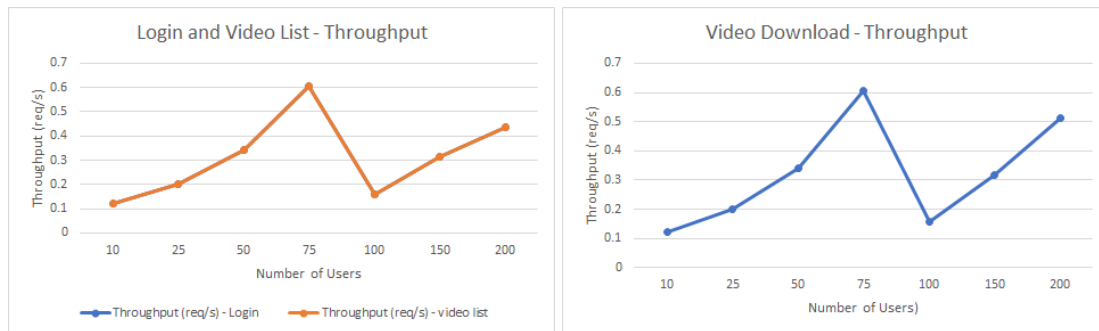


Fig. 5.8 Throughput in requests per second

number of users decides to watch videos at the same time. Be that as it may, as previously explained, our application will mostly run in the background of the smartphones, to detect dangerous situations. It is not, and it was never, intended to be a video streaming application. With that said, and analyzing these results, we still think WeDoCare performs well, and is ready to be distributed to our target audience.

Regarding the graphs illustrated in Figure 5.8, the throughput is very similar for all three requests, hence why in the left graph it is only visible the values for the video list. Since the values are so similar, the orange line overlaps the blue line. The only difference is when the number of users is 200. In that test case, we can observe that the throughput is higher for the *GET* request to watch a video. We can also see that there is a drop when the numbers of users is 100. This happened, because, as explained before, the number of requests was done more sparsely.

5.3 Battery tests

One of the most important aspects to keep in mind when developing an application is the battery life. No one wants to install an application that drains all the battery of the smartphone. If WeDoCare was one of those applications, our users would likely uninstall it, but worse than that is the fact that it could fail them in an emergency as well, leading them to believe that WeDoCare is not a viable solution. With that in mind we tried to develop WeDoCare to be reliable, but still not consuming large amounts of battery. Implementing location awareness in an application can be very battery-consuming and, in general, there is a direct proportionality ratio between the precision of the location data and the battery consumption. This means the higher the accuracy, the more battery the application will drain. There are

Tests

four levels of accuracy from which we can choose, to get the most accurate location possible. The levels are the following:

- **PRIORITY_HIGH_ACCURACY:**
Uses whatever resources available (GPS, Wi-Fi, cell towers) to compute the most accurate location possible.
- **PRIORITY_BALANCED_POWER_ACCURACY:**
Avoids using GPS to compute location, preferring Wi-Fi and cell tower information, thus saving battery.
- **PRIORITY_LOW_POWER:**
Uses cell towers to find the phone's location, reducing a lot the consumption of battery, but also reducing a lot the accuracy of the location obtained.
- **PRIORITY_NO_POWER:**
Receives location information from other location aware applications.

The location is retrieved periodically using the `PRIORITY_HIGH_ACCURACY` value. Nevertheless, constantly trying to convert physical address to coordinates is battery consuming, therefore we developed WeDoCare to listen for changes in the connectivity state and, if the conversion has not already happened, then it tries to compute the coordinates when an Internet connection is available. However, obtaining the location is not the only feature that drains battery. With the integration of the IoT node - iTag - WeDoCare has a Bluetooth service that actively listens for when a user presses the button in the iTag. This will of course consume some battery as well.

In order to validate our battery requirement we had to perform battery tests. This requirement stated that the battery should at least last one day. To accomplish this goal we tested WeDoCare during 5 days. The application was always running in the background. To evaluate the battery consumption, we used the tools Batterystats and Battery Historian [93]. Batterystats is a tool included in the Android framework that collects battery information from the smartphone, and reports of the battery usage. On the other hand, Battery Historian converts the report from Batterystats into an HTML visualization that can be viewed on a web browser.

To perform these tests we used the smartphone Xiaomi Mi 8, with a battery capacity of 3400 mAh. During the 5 days we used the phone like any other regular day. The difference was that WeDoCare was running in the background. Below is listed the test protocol designed to evaluate the battery performance:

1. Charge phone during night time to reach 100% battery in the morning;
2. In the morning, when the phone is charged, reset battery stats;
3. Open WeDoCare and turn on Bluetooth and GPS;
4. Use the smartphone normally, but with WeDoCare running in the background;
5. Register battery level after 12 hours, and upload battery report to the computer;
6. Repeat process for 5 days.

To reset the battery stats, as mentioned in step number 2, we need to connect the smartphone to the computer, open a terminal/command line window, and execute the following command: *adb shell dumpsys batterystats --reset*. In step number 5, to upload the battery report to the computer, the process is the same as in step number 2, but the command to be executed is: *adb bugreport directory_to_save_file/stats.zip* [93]. We decided to reset the battery stats in the beginning of every day, so we could analyze the battery consumption in each day. Below is Table 5.2, which illustrates the percentage of battery consumption by WeDoCare, as well as the remaining percentage of battery at the end of each day.

Table 5.2 Battery stats

Day	Battery consumption (%)	Battery left (%)
1	3	55
2	2.57	24
3	2.88	41
4	2.42	66
5	1.68	55

By observing the results, we can easily verify that WeDoCare is very battery friendly. Even with the application running all day in the background, with the GPS and Bluetooth turned on, the battery consumption levels are always below 3%. On the second day, the battery percentage remaining was quite lower when comparing to the other days, but by analyzing the data provided by Battery Historian, we see can that it was due to the usage of the Skype application. In this day, Skype was used more heavily. The Figure 5.9 shows the comparison between Skype and WeDoCare on that second day, regarding the battery consumption.

As we can see in Figure 5.9, if we look at the "Device estimated power use", we confirm that

Tests

Application	com.skype.raider	Application	com.socialite.android.saldanha1
Version Name	8.46.0.60	Version Name	1.0
Version Code	1250078353	Version Code	16
UID	10139	UID	10442
Device estimated power use	29.12%	Device estimated power use	2.57%
Foreground	5 times over 59m 22s 423ms	Foreground	3 times over 10s 325ms
CPU user time	51m 1s 210ms	Vibrator use	26 times over 2s 600ms
CPU system time	12m 7s 120ms	CPU user time	21m 23s 925ms
Device estimated power use due to CPU usage	0.00%	CPU system time	10m 39s 240ms
Total number of wakeup alarms	0	Device estimated power use due to CPU usage	0.00%
Audio	2 times for a total duration of 1h 7m 57s 558ms	Total number of wakeup alarms	187
Camera	8 times for a total duration of 1h 8m 25s 239ms		

Fig. 5.9 Battery Consumption - Skype vs WeDoCare

in the second day, the main reason why the battery level of the smartphone was so low was due to Skype, and not WeDoCare.

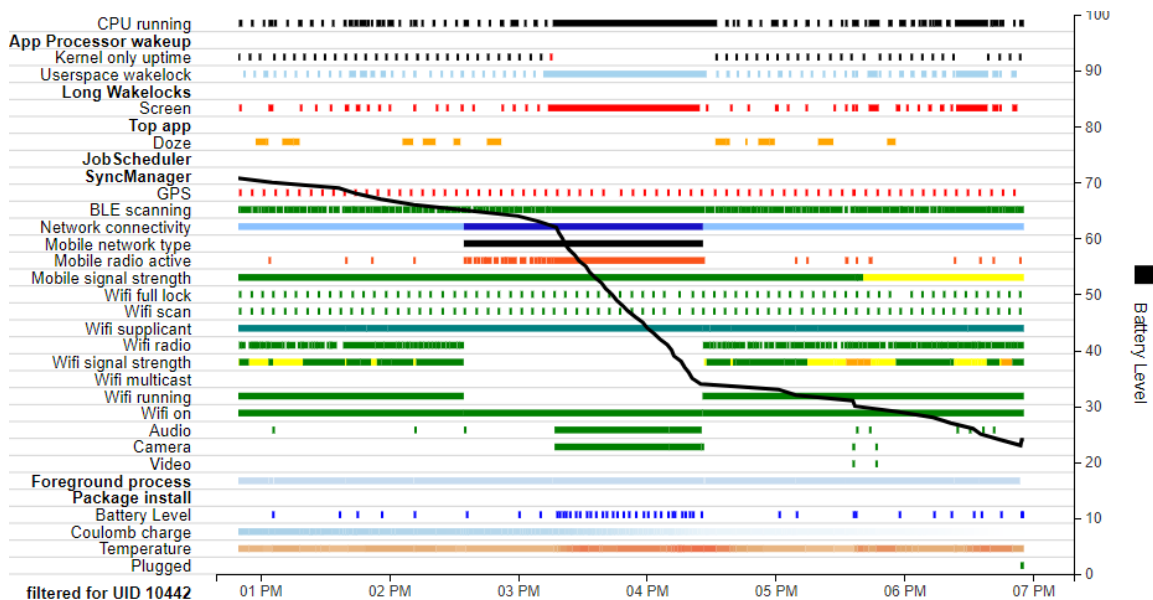


Fig. 5.10 Battery Consumption - Detailed Graph

Figure 5.10 illustrates part of a detailed graph of the battery consumption on the second day of these tests. We opted to only show a small part since the full graph is very large, and it would have been difficult to see the full contents of the graph. This graph offers a large amount of information. For instance, we can see the GPS use. WeDoCare updates the location whenever it detects changes in the connectivity. Otherwise it does not use the GPS much, hence why, despite using the most accurate option for the location, it does not drain much battery. There is also data regarding the scanning of BLE devices. It is a continuous green line in the graph. As stated earlier in this report, WeDoCare has a Bluetooth service always running, however, it only drains a small amount of battery. We can also

see information regarding the Wi-Fi use, how much time the screen of the phone was on, amongst many other informations.

Battery Historian is an extremely helpful tool to analyze the battery consumption of one or more applications. With this tool we were able to conduct our tests and collect large amounts of data. With the data collected and the results presented, we can conclude that WeDoCare is battery friendly, allowing us to fulfill our battery requirement.

5.4 Usability tests

In order to meet the usability requirement specified in section 3.2.4, usability tests were done. Usability tests are intended to find flaws in the design and usability of the application with the goal of identifying any changes that might be done to facilitate the use of the application [94]. With this in mind, we asked some users to complete a set of tasks and explain their reasoning, so we could take some notes on how they interpreted the screen and menus. For each task we counted the numbers of clicks, and the number of screens the user had to navigate through to complete those tasks. We also timed how long each user took to complete each task. The tasks to be completed are listed below:

- Task 1: Trigger an alarm through the panic button;
- Task 2: Change and save the frequent address;
- Task 3: Watch a video from the video repository;
- Task 4: Create a chat group;
- Task 5: Join a chat group;
- Task 6: Read the tips of the application;
- Task 7: Pair an iTag with the application.

The tests were performed equally for everyone. Users would start in the main screen - Figure 5.11 - then the task would be explained to them, and then we would ask them to start completing the task. As mentioned before, while the user was completing each task, the number of clicks and screens that he/she navigated through were counted. It is important to mention that the click when a user was inserting data were not counted. In Table 5.3 we can observe the expected results for this usability test, regarding the number of clicks and number of screens navigated through.

Tests

Table 5.3 Expected results for the usability test

	Taks 1	Taks 2	Taks 3	Taks 4	Taks 5	Taks 6	Taks 7
Screens	1	2	3	4	4	3	3
Clicks	1	2	3	4	4	3	3

Table 5.4 Usability test results

User		Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7
1	Screens	1	8	3	4	4	3	3
	Clicks	1	10	3	4	4	3	3
	Time (s)	1.04	85.3	7.31	13.94	36.27	2.95	8.36
2	Screens	1	7	3	4	4	3	3
	Clicks	1	9	3	4	6	3	3
	Time (s)	0.73	40.47	5.63	17.7	14.94	4.84	7.75
3	Screens	1	4	5	4	4	5	6
	Clicks	1	4	5	4	4	5	6
	Time (s)	0.8	23.16	7.25	19.46	11.1	6.92	9.92
4	Screens	1	8	3	4	5	3	3
	Clicks	3	9	3	4	5	3	3
	Time (s)	20.2	42.46	4.97	13.65	11.57	2.35	6.46
5	Screens	1	4	8	4	5	3	3
	Clicks	1	4	8	4	5	3	3
	Time (s)	2.94	22.39	29.06	22.45	13.96	3.91	11.62
6	Screens	1	2	3	4	5	3	3
	Clicks	1	2	3	4	9	3	3
	Time (s)	0.72	38.95	6.93	15.33	25.85	4.05	10.69
7	Screens	1	2	3	4	5	3	3
	Clicks	1	2	3	4	5	3	3
	Time (s)	0.92	20.63	2.62	14.31	15.29	2.26	10.19
8	Screens	1	2	5	4	4	3	3
	Clicks	1	2	5	4	4	3	3
	Time (s)	1.31	7.2	11.29	25.26	11.03	3.27	12.13
9	Screens	1	11	3	4	4	3	3
	Clicks	1	11	3	4	4	3	3
	Time (s)	0.8	35.82	2.68	15.67	9.1	2.74	11.62
10	Screens	1	4	3	4	4	3	3
	Clicks	1	4	3	4	4	3	3
	Time (s)	1.14	11.86	4.04	17.62	9.6	2.68	13.5

The Table 5.4 shows the results obtained after completing this usability test. It is visible that all the users managed to complete all required tasks.

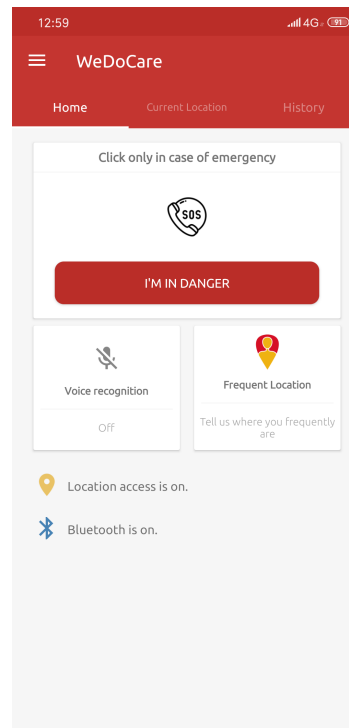


Fig. 5.11 Main screen

By analyzing Table 5.4 we can see that most users completed the task without any problems, although some took a bit more time completing certain tasks. Except for user 4, everyone completed the first task with ease. For this task, the users should click the panic button to trigger an alarm. The panic button is in the main screen - Figure 5.11. User number 4 did not understand well the task required, thus it took 20 seconds to complete the first task.

The goal of task number 2 was to change and save an address the user is frequently in. This is also on the main screen. However, only 3 users were able to complete this task with the number of clicks, and number of navigation screens expected. In Figure 5.11 we can see the button to change the address. Be that as it may, the version of WeDoCare that was run during these usability tests had the text "Current Location" in the button, instead of the current text which is "Frequent Location". This change was due to the feedback of the users and after seeing that only 3 users were able to complete the task how it was expected. Trying to complete this task, most of the users clicked first in the menu on the left side, and after they realize the button they were looking for was not there, most of them went to the Location tab of the main screen. One of the users even went to the help page to try to find out to change the location. Screenshots of these two screens can be seen in Figure 5.12.

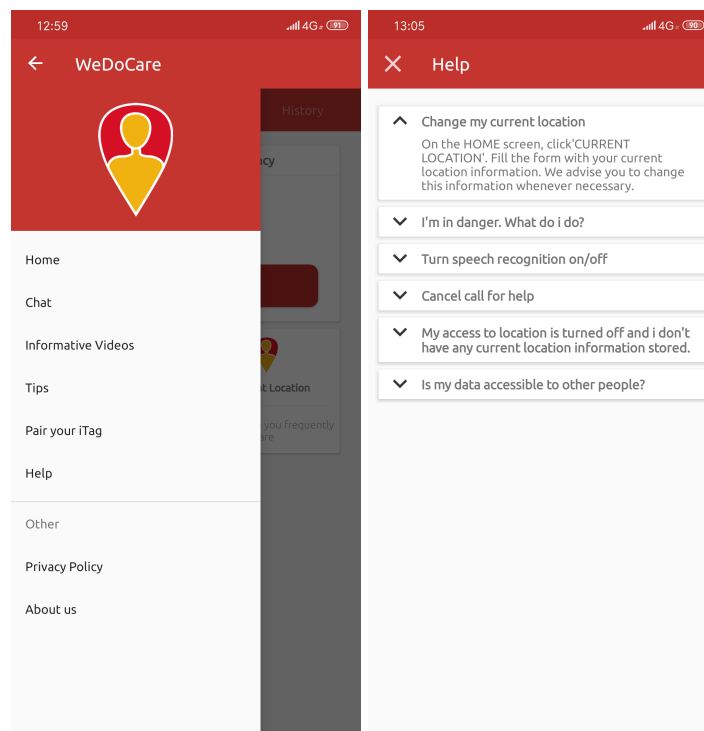


Fig. 5.12 Left menu and help screen

Task number 3 was also quite simple for most users, although user number 5 took longer than expected. This user did not see that there was a menu on the left. To watch a video one needs to click that menu on the left, and that is why this user took longer. Regarding task 4, which was to create a group chat, every single user was able to complete it with the expected number of clicks and the expected number of screens navigated.

On task number 5 the goal was to join a chat room. All users knew how to complete this task, nonetheless, for user number 2 and 6, the number of clicks was much higher than expected. In Figure 5.13 we can see the screen that lists the existing group chats, from which you can choose any to join. Before, the rectangle where the name of the chat appears was smaller and sometimes when the user clicked on the rectangle it did not assume the user was clicking on it. That is why users 2 and 6 have a higher number of clicks. The rectangles are bigger now, so it does not happen again.

Tasks 6 and 7 were completed as expected except for user number 3, who did the same mistake in both tasks. This user knew where to click to complete the tasks, but when he was on the menu on the left side, he clicked in a different place by accident, which is the reason

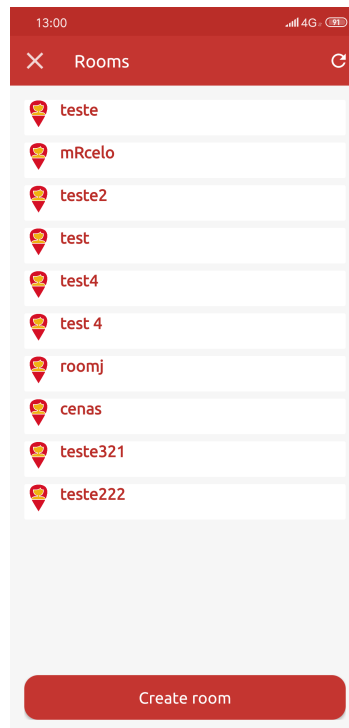


Fig. 5.13 Listing of the group chats

why he has two more clicks than expected on task 6 and three more click on task 7.

After completing these tasks, the users were asked to complete a small survey, so they could express their opinion regarding the application. The survey consists of 5 questions and in each question the user can assign a value from 1 to 5. The value 1 means that the user strongly disagrees, while the number 5 means that the user strongly agrees. The results can be seen in the Figures below.

1- I found the application simple to use

10 responses

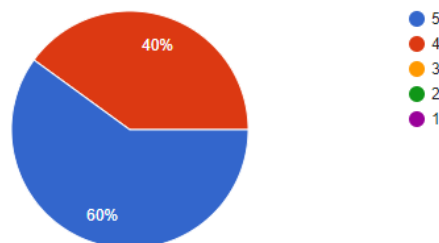


Fig. 5.14 Results of question 1

2- I can use all the features of the application easily

10 responses

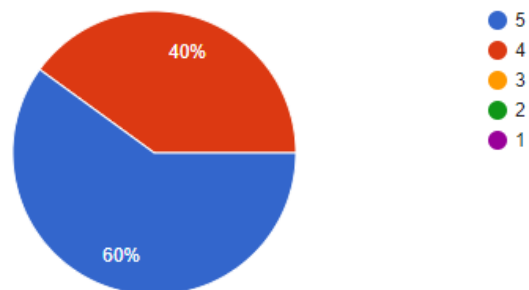


Fig. 5.15 Results of question 2

3- The navigation between the different menus is easy to understand

10 responses

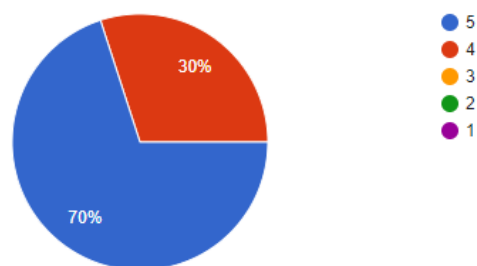


Fig. 5.16 Results of question 3

4- The content of the screens is as expected

10 responses

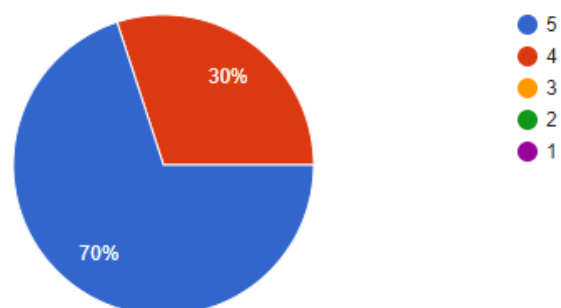


Fig. 5.17 Results of question 4

5- Would you recommend this application to people who do not feel safe?

10 responses

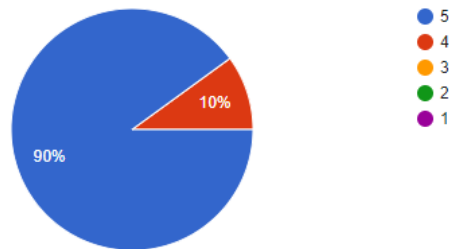


Fig. 5.18 Results of question 5

This survey allowed us to understand what the users thought of the application. The results were very positive. All the users said that the application was easy to use, and that they could use all the features easily. Furthermore, since in most of the tasks, users had to use the menu on the left, it is very easy to learn where you can go to use all features of the application. This means that if a user utilizes the application a second time, he/she will have no difficulty in performing the tasks required, and the number of clicks will be even lower, where possible. This fulfills our usability requirement, that stated that the application should be easy to use. Moreover, all users would recommend this application to people that do not feel safe.

In the survey there was also a space for people to leave their suggestions, in what they would want to see improved in the application. One of the suggestions was to make it more clear that the button to change the frequent location was on the main screen. As explained previously, this suggestion was already applied. Several users commented that they were expecting to change their frequent location in the location tab, on the main screen. One even suggest to use a pin marker and the user would put that pin on the desired location.

Given the results obtained we consider than this usability test was successful.

5.5 Real life scenarios

We did not only perform laboratory tests on WeDoCare. With the help of Ergue-te and Saúde em Português, we are able to test WeDoCare in a real scenarios. These tests are described in the next subsections.

5.5.1 Ergue-te

During the month of January, we conducted some simple tests with Ergue-te. The purpose was to identify possible bugs, and then correct them if need be. We also wanted to have a stable, bug free version, of WeDoCare before our meeting with the PSP, which took place in late February. We distributed WeDoCare for the people responsible in the Ergue-te association. The goal was for each person to trigger each alarm once. At this time WeDoCare only had 3 methods of detecting a dangerous situation: i) manual pressing of the panic button, ii) recognition of the keyword "socorro", iii) gesture detection. The tests were done during a period of 5 days. It is worth mentioning that on the first day, one of the people from Ergue-te did not do these tests, so the first day has one less sample. The Table 5.5 below shows the results from these tests with Ergue-te.

Table 5.5 Ergue-te test results

Day	User	Panic Button	Speech Recognition	Gesture Recognition
1	1	0	2	4
	2	0	5	0
	3	2	0	2
	4	-	-	-
2	1	1	0	4
	2	0	3	2
	3	3	0	0
	4	3	0	0
3	1	1	8	3
	2	1	2	4
	3	0	0	1
	4	2	0	1
4	1	1	0	1
	2	1	1	2
	3	0	0	3
	4	1	7	0
5	1	1	1	3
	2	1	1	2
	3	1	1	5
	4	1	1	3

By examining the results in Table 5.5, we can see that the first 3 days were not very good. However, it came to our attention that some people from Ergue-te misunderstood some of our instructions. The goal was to each of them trigger each alarm once, meaning this table

should be filled with nothing but 1's, in the best case scenario of course. After the second day we contacted them and explained how to perform the tests again. We can see a gradual improvement from the third day onward.

This misunderstanding explains some strange results, such as the fact that the panic button was not triggered by some users in days 1 and 2. It also reveals why some other users pressed the panic button three times in one day. Another situation that we detected was the fact that, when one of the test users tried to trigger the alarm through speech recognition, the smartphones of some users nearby would also detect that keyword and trigger an alarm as well. This, coupled with the fact that some of the tests users could not cancel the notification at first, resulted in an abnormal number of alarms triggered by the speech recognition module. This can be verified in days 1, 3, and 4.

However, WeDoCare was still buggy. One of the of the problems reported by the test users was the fact that the application was not updating the location. They gave us an example. One of the users triggered an alarm from home, but the coordinates sent in the message were from the work location. The good news is that we discovered why this happened. The problem was the Android version running on the smartphone of the user that experienced the bug. As previously mentioned, the location is updated when the application detects connectivity changes. To detect these connectivity changes, WeDoCare uses a *BroadcastReceiver*[95]. This *BroadcastReceiver* was declared in the manifest file of the application. The manifest file describes essential information about the application and it is where the permissions of the application, as well as the services, are declared. [96]. Before Android O - Android 8.0 - a *BroadcastReceiver* could also be declared in the manifest file. However, in Android O and above, the *BroadcastReceiver* needs to be declared within the code of the application, otherwise it will not work. This user affected by this bug had Android O running on his smartphone, and that is why the location was not updated. Upon discovering this bug, it was immediately corrected. Now all *BroadcastReceivers* are declared within the code of the application instead of being declared in the manifest file.

On the last 2 days the results were better. What could be improved is the gesture recognition. Sometimes it is more sensitive than others. However, that is part of the library code, so it is harder to modify, although we did manage to tune it a bit. Taking into account these results, and, although the first days were not so good, in the last days we saw an improvement. We consider it was a good experience, mainly because more bugs were re-

ported, and so, we were able to fix them. Now the current version of WeDoCare is very stable.

5.5.2 Saúde em Português

Although our partnerships with the Police and the State Secretary for Citizenship and Equality are not official yet, we still were able to test WeDoCare in a real environment, thanks to Saúde em Português. We were also counting on Ergue-te to test our application, however, since our partnership with the Police was delayed, they could not receive the SMS with the victim's location, and the only solution was to send those messages to the associations. As mentioned before, the people in the Ergue-te association did not feel comfortable with that kind of responsibility. They are still our partners, but they will only allow us to test our application with them when the partnership with the Police is effective. Fortunately, Saúde em Português agreed to collaborate with us under this condition, that the messages would be sent to a member from their association.

It would have been easier to upload WeDoCare to the Google Play Store, at least for the version requested by the State Secretary for Citizenship and Equality, in a testing version, and only make it available to certain users. However, Google recently updated their policy regarding the use of permissions to send messages, and to make calls, from an application [97]. This is due to the fact that a large amount of applications would declare these permissions, but they were not essential to the core functionalities of the applications. With this new policy, Google is trying to improve the security of their users, making developers only declare the permissions that are truly necessary for the application to function properly. Now developers should not declare those permissions unless it is critical for the application. Google wrote an article explaining what type of applications can be the exception to this policy [98], and another article showing how you should declare your permission and how to justify the use of those permissions [99]. We read both articles. In the first article it is mentioned that applications that send messages in a case of emergency can declare the SMS permission. Therefore, we followed the instructions on the second article, to justify the use of that permission. Nevertheless, Google did not consider that the permission is critical to our application, and for that reason WeDoCare is still not available on the Google Play Store. We are still trying to justify that WeDoCare needs that permission, but the process is been going on for some time.

Since WeDoCare is not on the Google Play Store, we had a meeting with our partner, where we gave the WeDoCare installation file to the member from Saúde em Português. This

member would then distribute WeDoCare with the people that this organization is helping. To make the process easier, we also provided them with an installation guide, which can be seen in Appendix B. We also explained, as best as we could, how the application works, and all its functionalities, so this member could also explain to the people being helped how WeDoCare works.

The application was distributed successfully and it is still being tested to this day. So far, and fortunately, we have no reports of people being attacked. Nonetheless, the testers did say they feel safer knowing that have they can easily ask for help if necessary. It is also important to mention that no defects or malfunctions were reported, proving that the application is fully functional and reliable.

Chapter 6

T4SC

This chapter presents the platform T4SC. As mentioned in section 1.5, we were aware that our partnership with the PSP, and the State Secretary for Citizenship and Equality, could be delayed. It was one of the risks inherent to this internship. Nonetheless, to maintain a high productivity level, we decided to invest some time in the T4SC platform. This way, we were still able to be productive during the months of April and May. The following sections explain in more detail what is this platform and how it relates to WeDoCare, what were the new features that were implemented, and the results that came from those features.

6.1 What is it?

Universities are often accused of not being involved in real world problems. They tend to focus more on the scientific value of the work produced and not on the humanitarian value. T4SC tries to change that. T4SC is a platform that aims to promote the resolution of research challenges and social challenges. It is an innovative database/repository of challenges with real impact in the world [100]. Research challenges are meant to be solved by people with knowledge of the problem in question. The work made by researchers can be stored and used in a project. If that research is picked as a solution for a problem, then the researcher gets recognition for it, by becoming referenced in that project.

There are also social challenges to which a Citizen can volunteer. These social challenges are created by companies or organizations. The Citizen will gain points for volunteering and completing the challenge. He/She can then exchange those points for rewards. These rewards are posted by companies or organizations as well.

Companies or organizations can register in this platform and create challenges. They can

create a challenge for a Researcher to solve, or a social challenge, to which a Citizen can volunteer. If this company/organization decides to create a social challenge, then it also needs to specify the number of points to be given to each Citizen when they complete the challenge. For instance, the Coimbra Municipal Council can issue a challenge with the goal of 'cleaning the streets of Coimbra', and it will give 100 points to each user that volunteers. Citizens can volunteer to that challenge, and if they complete it, they will receive the number of points specified by the Coimbra Municipal Council in the challenge - in this case it is 100 points. The Citizen can then exchange these points for rewards. The platform has the following profiles:

- **Administrator**

1. The administrator has access to all the features of the platform.

- **Company/Organization:** overall description of the requirement;

1. It can create a new challenge, either of a research nature (for a researcher to solve) or a social one (for a citizen to volunteer to). When creating a social challenge, it also has to choose the number of points to give to the citizens who volunteer;
2. It can see all the challenges it has created;
3. It can add rewards. For this, it is necessary to specify the number of points needed for the citizen to obtain this reward;
4. It can see all the rewards it has created;
5. It has the option to terminate a challenge. If the it is of a social nature, then the company/organization will have a list of citizens who volunteered. Citizens who attended the challenge must be selected, and then it should be closed, so that each citizen who volunteered and attended the challenge, can receive the number of points that was assigned to this challenge.

- **Researcher:**

1. It can list all the research challenges that exist;
2. It can add a project from the Orcid;
3. It can list all the projects it has added.

- **Citizen**

1. It can list all the social challenges that exist;

2. Can volunteer to any social challenge;
3. It can list all the rewards that exist;
4. It can always see the current number of points it has.

The profiles that were already implemented were the Administrator and Researcher profiles, meaning that we still needed to implement the Company/Organization and Citizen profiles. The next section will present how these last two profiles were implemented.

6.2 New features

As previously mentioned, the new features that were implemented were the Company/Organization and Citizen profiles. These are the profile with more functionalities, but fortunately we could reuse some code logic from the profiles that were already implemented. During the development of these features, there was the need to be a full stack developer, meaning there was work to do on the front-end and on the back-end of the platform.

The front-end was developed using HTML, CSS and AngularJS [101], which is a JavaScript-based open-source front-end web framework. The back-end was developed using Java and Spring Boot, like WeDoCare. There was a need to study the code of the platform for almost a month, specially the front-end code, since the student responsible for this internship had never worked with AngularJS before. After that month, these new two profiles were implemented, which also took approximately another month. Nevertheless, since both ends made use of the MVC pattern, it was easier to study the code and implement new features.

We implemented the functionalities shown in the last section, regarding only these two profiles - Company/Organization and Citizen. Functionalities 1, 2, and 4 of the Company/Organization profile were implemented using some code logic that was already used for the development of the Administrator and Researcher profiles. However, functionalities 3 and 5 were implemented from scratch. Regarding the Citizen profile, functionalities 1 and 3 were implemented by reusing some code logic previously developed, whilst functionalities 2 and 4 were implemented from scratch. There was also a need to create new tables in the database to store the rewards created by Companies/Organizations, to store the citizens who volunteered to challenges, amongst other data.

In the end, everything that was needed was implemented, and now the platform is be-

ing tested by one of our partners. The next section shows the results of these implementations with some screenshots of the platform.

6.3 Results

The Figures below illustrate the T4SC platform.

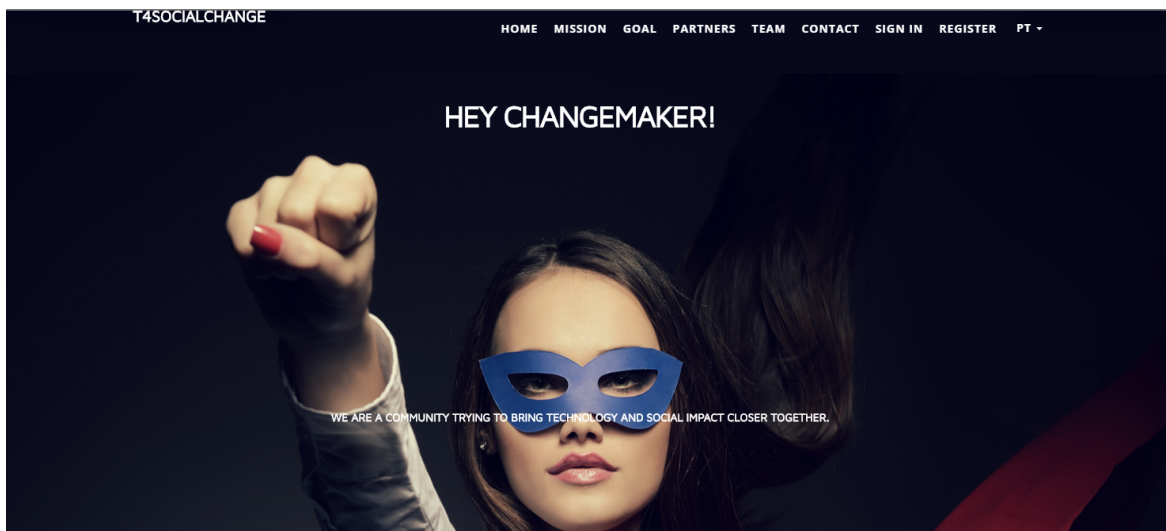


Fig. 6.1 T4SC home page

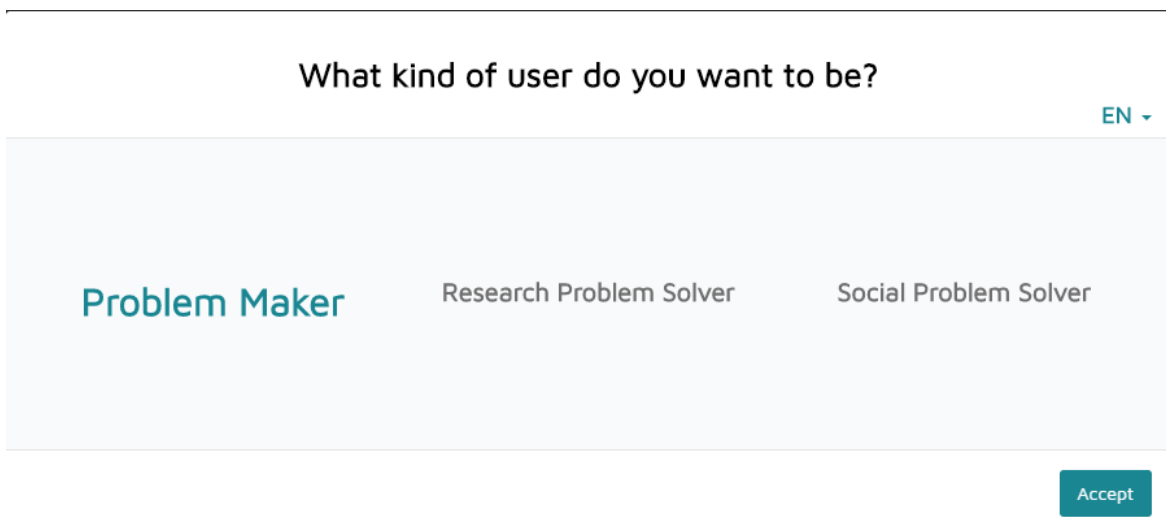


Fig. 6.2 Modal to select whether you are a company, researcher or citizen

Figure 6.2 shows a modal presented to the user, so it can select whether it is a company/organization, researcher or citizen. Figure 6.3 shows the profile for a researcher. This profile was already implemented.

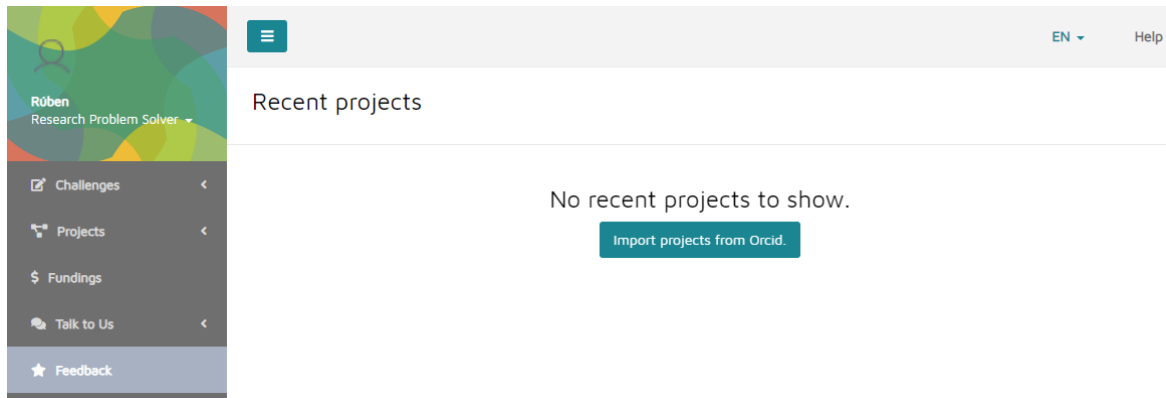


Fig. 6.3 Researcher profile

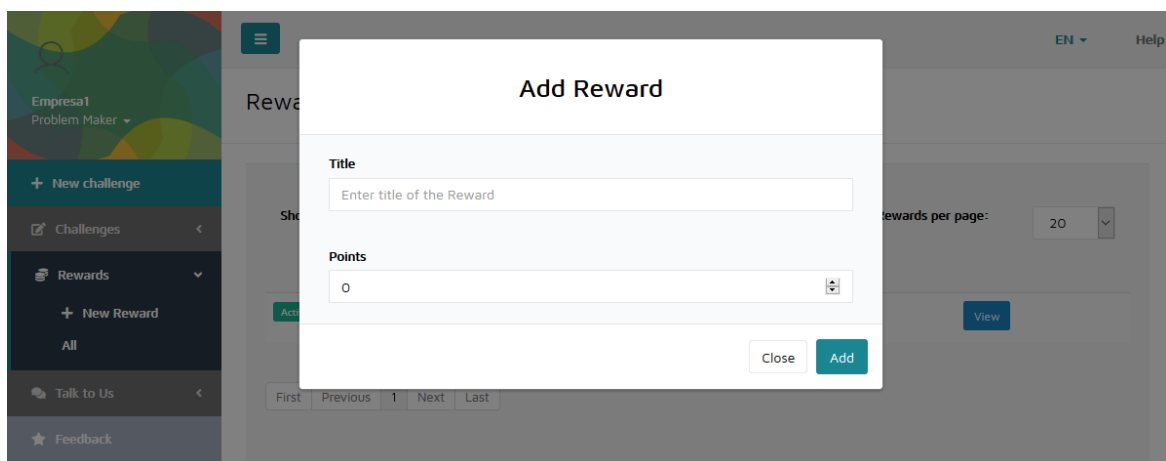


Fig. 6.4 Company/Organization profile - adding a reward

Figures 6.6, 6.7, 6.8, 6.9, and 6.10, show the flow of a company creating a challenge, the user seeing that challenge and volunteering to it. It also illustrates a company terminating the challenge by selecting the users that actually attended it. The last Figure shows that the number of points of the citizen was updated after completing the challenge.

We can see that this platform can be useful to promote WeDoCare. We can upload the WeDoCare project to the platform as research solution. A company/organization that registers on the platform and creates a challenge similar to what WeDoCare is trying to help with,

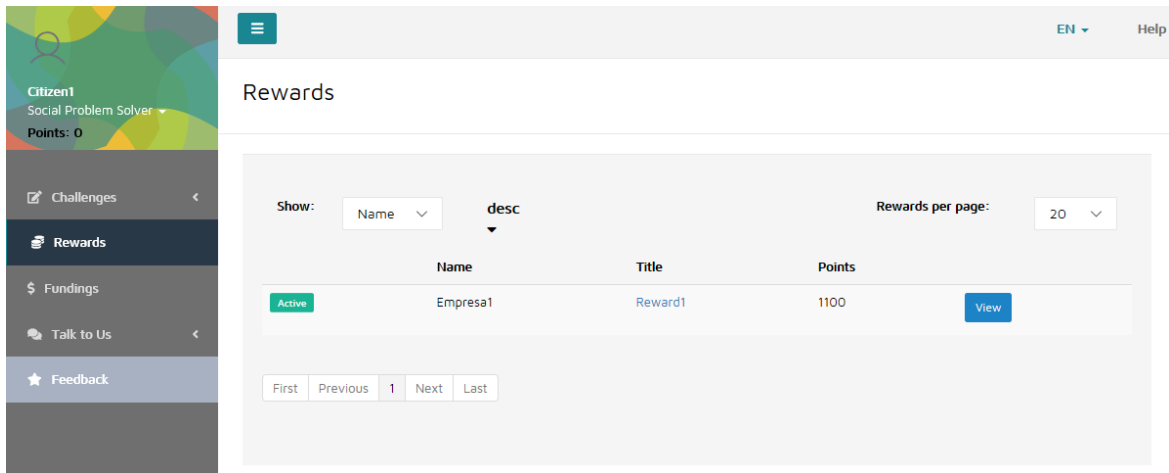


Fig. 6.5 Citizen profile - listing rewards

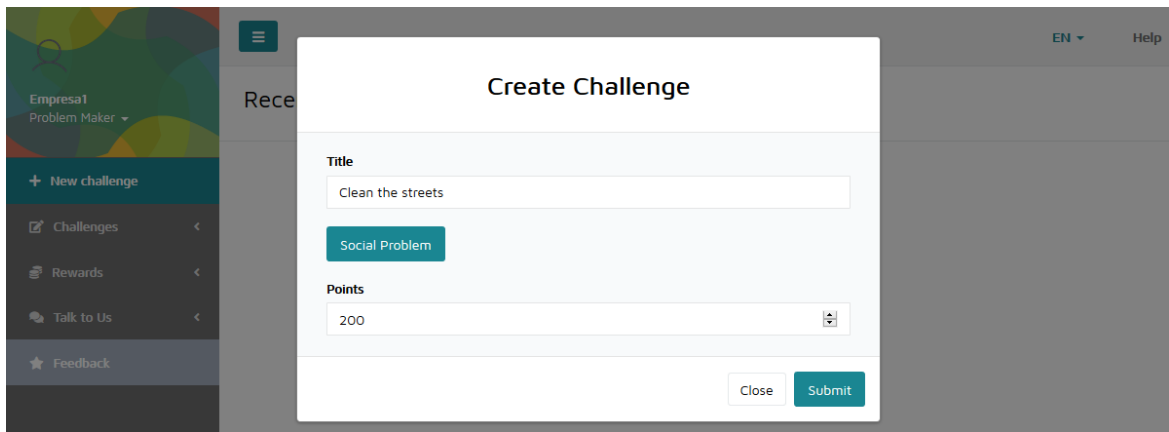


Fig. 6.6 Company/Organization profile - adding a challenge

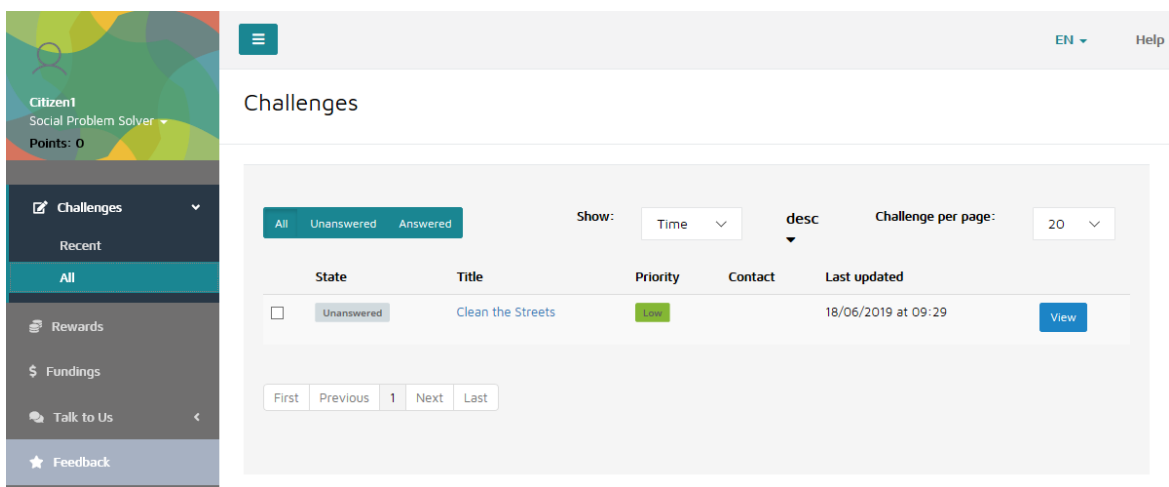


Fig. 6.7 Citizen profile - listing challenges

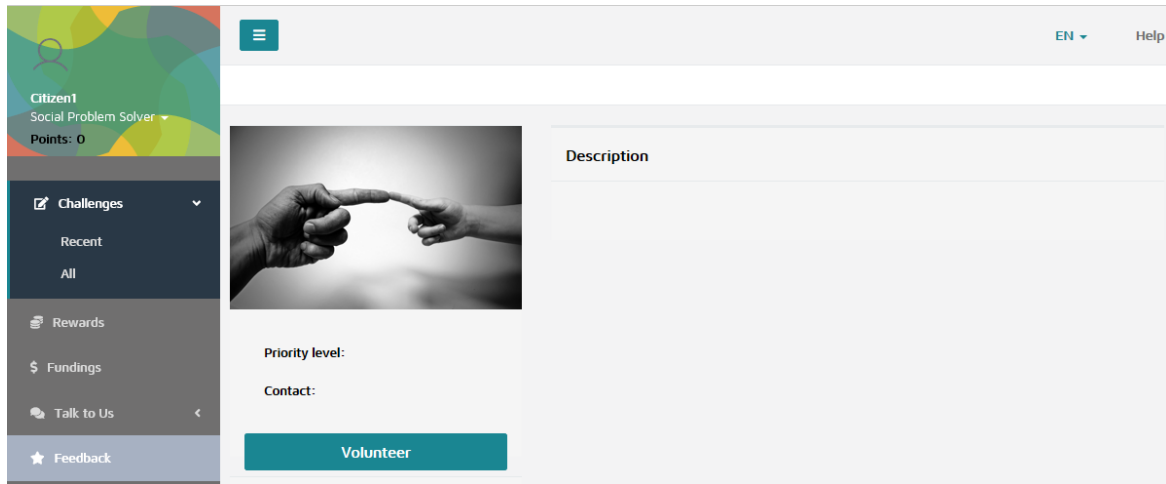


Fig. 6.8 Citizen profile - volunteering to a challenge

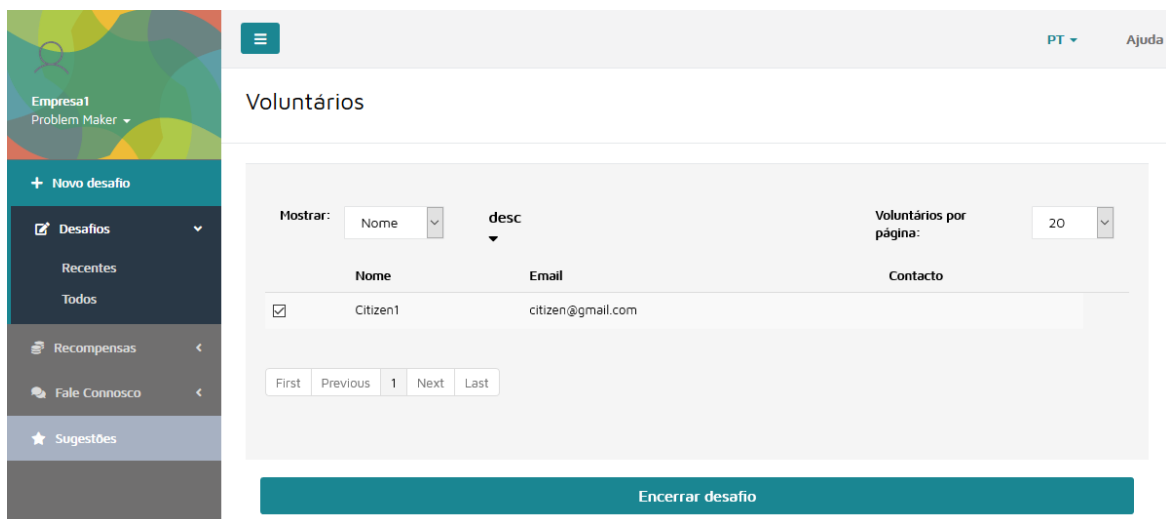


Fig. 6.9 Company/Organization profile - Terminating a challenge

The screenshot displays a user profile for 'Citizen1' with the role 'Social Problem Solver' and a score of 250 points. The main section is titled 'Desafios' (Challenges). It features a navigation menu on the left with options like 'Desafios', 'Recompensas', 'Financiamentos', 'Fale Connosco', and 'Sugestões'. The challenge list includes filters for 'Todas', 'Sem Resposta', and 'Respondidas', and sorting options for 'Tempo' and 'desc'. The table headers are 'Estado', 'Título', 'Prioridade', 'Contacto', and 'Última atualização'. The pagination shows 'First', 'Previous', '1', 'Next', and 'Last'.

Fig. 6.10 Citizen profile - Number of points is now updated

can just choose WeDoCare as a solution. It will help the company in tackling that problem, and it will promote us, researchers. Another possibility is the fact that we can form new partnerships and WeDoCare may have more functionalities in the future, to adhere to an even more expanded target audience.

Chapter 7

Conclusion

7.1 Final results and Future work

The smartphone is arguably the gadget that we most use nowadays. Furthermore, even the cheapest smartphones have the most basic sensors and technologies, like GPS, Bluetooth, accelerometer, amongst others. One of the goals of this internship was to improve the existing WeDoCare application, making it more reliable and adding new features, so it could be more helpful to our target audience. However, the main goal was to put WeDoCare 3.0 in a real life scenario. We were able to do that, as WeDoCare 3.0 is being used by the people from the Saúde em Português organization. WeDoCare 2.0 was not tested in a real life scenario, as it had some bugs and it was not stable. We took WeDoCare 2.0, corrected the bugs, improved it, added new features, and now it is being tested in a real life scenario. With that said, we were able to accomplish our main goal.

Throughout the year, we tried to form a new partnership, this time with the State Secretary for Citizenship and Equality. We wanted to expand our target audience to people in relationships for the reasons explained in chapter 1. However, there is a huge delay, which is out of our control. This is the reason why we could not test the application during Queima das Fitas, as initially planned. We also tried to finalize our partnership with the Police, but due to the reasons presented in section 4.1, our partnership is still not official. This was beyond our influences and made it impossible to test the application with Ergue-te members during this year, as we also planned. As a result, we do not have as much feedback as we were hoping to. Nonetheless, Saúde em Português accepted to test our application, even without our partnership with the Police. The two previously mentioned scenarios might not have gone as we were hopping for, through no fault of our own, but we do have WeDoCare being tested in a real life scenario, thanks to the people from Saúde em Português.

Conclusion

All of the specified requirements were implemented, or accomplished, in the case of the non-functional requirements. All of the functional requirements, from both the servers and the Android application, were successfully implemented. Regarding the non-functional requirements, the Privacy one was accomplished, as the credentials of each user are encrypted before being saved in the application. Furthermore, the application stores the location information in the phone only, so only the user is able to access it. In terms of battery usage, as we saw in section 5.3, the application never consumes more than 3% of the phone's battery, so even if WeDoCare runs in the background all day, the phone's battery will last at least one day, as specified by this requirement. The application is also reliable since it does not produce 'false positives', as the defects were corrected after those tests with Ergue-te, which are mentioned in section 5.5. The application also fulfills the usability requirement, since all users that performed the usability test considered that the application was easy to use. As for maintainability, the code was still written following the MVP pattern in Android, and the MVC pattern was used for the back-end server. As mentioned in chapter 4, these patterns make it easier to understand the code, to add new features, and to perform unit testing. For this reason, this requirement was also achieved. For the availability requirement, what we have to say is that the server has been up since the beginning of April, and it never crashed or stopped working. Finally, the performance requirement is also met, as the application is smooth in changing screens, and apart from the video streaming, the servers always reply to requests from the application very fast.

One of our key features is reliability. We developed an application that activates accurate emergency location alerts using keyword and gesture recognition, a panic button, and a IoT node - iTag. Since the beginning that the keyword recognition system was the least reliable feature, as the API used by Google uses a brazilian dictionary instead of a portuguese one. This was the main reason to integrate an IoT node with the application, so we would still have a reliable method to detect a dangerous situation, even if the user did not have the phone on his/her hands. It is safe to say that, now, with these 4 detection methods, the application is truly reliable. For future work, the speech recognition module should be improved, maybe through machine learning techniques. Also, when the partnerships with the Police and State Secretary for the Citizenship and Equality are official, we will have more feedback and the application will improve even further.

Although there were some unforeseen events, which kept us from doing what we initially planned, we consider that we build a reliable and promising application, that for sure will be

useful to help people feel more safe.

7.2 Personal consideration

I took this internship because of my passion for building android applications, and because it is good to work on a project that actually tries to help people. I did not want to do a project just for the sake of completing my master's degree. I wanted to do something I thoroughly enjoy, and if I can help people along the way, even better.

However, this journey was not easy. Trying to understand somebody else's code is not always trivial, hence why in the beginning, it was somewhat hard to fix the bugs. Also, it was not easy to implement the video streaming feature, as I never had done that before. The fact that some of our partnerships were suffering from huge delays to become official also did not help, since it is easier to work when you have certainties, which we did not have. Despite this, I continued to push hard to have a stable version of WeDoCare at the beginning of April, so it would be ready to be tested during Queima das Fitas. That scenario unfortunately did not come to fruition, and so, I worked on the T4SC project, since it would be useful to promote WeDoCare. This project was also not easy to work on, since I never really dealt with JavaScript and AngularJS. It took me a while to understand the front-end code, but once I understood it, it was easy to make the necessary changes. I learned a lot, in regards to mobile developing, to back-end developing and to unit testing. I also learned how to cope better with having a lot of tasks to do at the same time, as I also had a scholarship whilst doing this work for the thesis.

This journey had its ups and downs, but I finish this internship knowing I give all that I had. I worked extremely hard to develop an application that would correspond to what was expected for it to be. I always woke up at least at 7am and went to the Department to work. There were several days where I even woke up at 6am, specially during May and June. I wanted to have everything done ahead of time, to avoid dealing with unexpected situations in a short amount of time. I am happy to know that I accomplished that, and even happier to know that I was able to develop an application that will help people feel safer.

References

- [1] *A Índia é o país mais perigoso para se ser mulher*, <http://visao.sapo.pt/actualidade/mundo/2018-06-27-A-India-e-o-pais-mais-perigoso-para-se-ser-mulher> - last checked: 26/12/2018
- [2] *Claudia Garcia-Moreno, Henrica AFM Jansen, Mary Ellsberg, Lori Heise, Charlotte H Watt*, Prevalence of intimate partner violence: findings from the WHO multi-country study on women's health and domestic violence - last checked: 14/01/2019
- [3] *Dating Abuse Statistics*, <https://www.loveisrespect.org/resources/dating-violence-statistics/> - last checked: 26/12/2018
- [4] *Prepared for Our Watch*, The Line campaign Summary of research findings - last checked: 26/12/2018
- [5] *me too.*, <https://metoomvmt.org/> - last checked: 26/12/2018
- [6] *D. Woodlock*, "The Abuse of Technology in Domestic Violence and Stalking," Violence against women, vol. 1, 2016 - last checked: 26/12/2018
- [7] *J. P. Carles Gomez, Joaquim Oller*, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology," Sensors, August 2012 - last checked: 26/12/2018
- [8] *Ergue-te*, <http://erguete.pt> - last checked: 24/12/2018.
- [9] *Saúde em Português*, <https://www.saudeportugues.org> - last checked: 24/12/2018
- [10] *ISCAC - Coimbra Business School*, <http://www.iscac.pt> - last checked: 10/01/2019
- [11] *CLEI 2019 - Home*, <http://clei2019.utp.ac.pa/en/> - last checked: 10/01/2019
- [12] *T4SC - Tech 4 Social Change*, <http://t4sc.dei.uc.pt> - last checked: 03/06/2019
- [13] *MobiWise*, <http://mobiwise.av.it.pt> - last checked: 05/06/2019

References

- [14] *CISUC*, <https://www.cisuc.uc.pt/projects/show/230> - last checked: 05/06/2019
- [15] *The 21st Century Cell Phone*, <https://www.streetdirectory.com/travel-guide/132116/cell-phones/the-21st-century-cell-phone.html> - last checked: 26/12/2018
- [16] *Evolution of the Mobile Phone*, www.tigermobiles.com/evolution - last checked: 26/12/2018
- [17] *Living without a Smartphone in the 21st Century*, <https://www.vagabomb.com/Living-without-a-Smartphone-in-the-21st-Century-The-Story-of-Hunger-Pangs-and-Lost-Cars/> - last checked: 26/12/2018
- [18] *What is Internet of Things*, <https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT> - last checked: 26/12/2018
- [19] *That 'Internet of Things' Thing*, <https://www.rfidjournal.com/articles/view?4986> - last checked: 26/12/2018
- [20] *Inter net of Things forecast*, <https://www.ericsson.com/en/mobility-report/internet-of-things-forecast> - last checked: 26/12/2018
- [21] *Connecting the World: IoT Ideas for Business*, <https://dzone.com/articles/connecting-the-world-iot-ideas-for-business> - last checked: 26/12/2018
- [22] *Sensors Overview*, https://developer.android.com/guide/topics/sensors/sensors_overview - last checked: 26/12/2018
- [23] *Sensors*, <https://source.android.com/devices/sensors/> - last checked: 26/12/2018
- [24] *S. Munir, J. A. Stankovic, C. M. Liang, S. Lin*, "Cyber Physical System Challenges for Human-in-the-Loop Control", 8th Int. Workshop Feedback Computing, San Jose, CA, USA, 2013 - last checked: 16/12/2018
- [25] *Society-in-the-Loop*, <https://medium.com/mit-media-lab/society-in-the-loop-54ffd71cd802> - last checked: 24/12/2018
- [26] *Nordic Thingy:52*, <https://www.nordicsemi.com/Software-and-Tools/Development-Kits/Nordic-Thingy-52> - last checked: 03/06/2019
- [27] *nRF52 DK development kit*, <https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52-DK> - last checked: 03/06/2019

References

- [28] *Nordic Thingy - Apps on Google Play*, <https://play.google.com/store/apps/details?id=no.nordicsemi.and> - last checked: 03/06/2019
- [29] *Nordic Thingy:52*, <https://makezine.com/product-review/boards/nordic-thingy52/> - last checked: 04/06/2019
- [30] *Nordic App Source Code*, <https://github.com/NordicSemiconductor/Android-nRF-Toolbox> - last checked: 04/06/2019
- [31] *Nordic Thingy:52*, <https://www.nordicsemi.com/Software-and-Tools/Development-Kits/nRF52-DK/Download#infotabs> - last checked: 04/06/2019
- [32] *Embedded Studio | The Cross Platform IDE by SEGGER*, <https://www.segger.com/products/development-tools/embedded-studio/> - last checked: 04/06/2019
- [33] *Segger Embedded Studio*, <https://www.segger.com/products/development-tools/embedded-studio/> - last checked: 04/06/2019
- [34] *iTAG User Manual*, https://www.1dayfly.com/cms_img/track_en_trace_manual.pdf - last checked: 04/06/2019
- [35] *What Is the Android Operating System*, <https://www.lifewire.com/what-is-google-android-1616887> - last checked: 25/06/2019
- [36] *iPhone vs Android Market Share*, <https://www.macworld.co.uk/feature/iphone/iphone-vs-android-market-share-3691861/> - last checked: 25/06/2019
- [37] *Model View Presenter*, <https://medium.com/@cervonefrancesco/model-view-presenter-android-guidelines-94970b430ddf> - last checked: 24/12/2018
- [38] *Women's Safety XPRIZE*, <https://www.xprize.org/prizes/womens-safety> - last checked: 24/12/2018
- [39] *WearSafe*, <https://wearsafe.com> - last checked: 24/12/2018
- [40] *Nimb*, <https://nimb.com/app/> - last checked: 24/12/2018
- [41] *Guardian Circle*, <http://guardiancircle.com> - last checked: 24/12/2018
- [42] *Watch Over Me*, <http://watchovermeapp.com> - last checked: 24/12/2018
- [43] *bSafe*, <https://getbsafe.com/features/> - last checked: 24/12/2018

References

- [44] *Leaf*, <https://www.leafwearables.com> - last checked: 24/12/2018
- [45] *Slack*, <https://slack.com> - last checked: 05/01/2019
- [46] *GitLab*, <https://about.gitlab.com> - last checked: 05/01/2019
- [47] *Kanban vs Scrum vs Agile*, <https://www.softwaretestinghelp.com/kanban-vs-scrum-vs-agile/> - last checked: 05/01/2019
- [48] *Kanban - A brief Introduction*, <https://www.atlassian.com/agile/kanban> - last checked: 05/01/2019
- [49] *Trello*, <https://trello.com> - last checked: 05/01/2019
- [50] *Android Studio*, <https://developer.android.com/studio/> - last checked: 15/01/2019
- [51] *Java*, <https://www.java.com/en/> - last checked: 15/01/2019
- [52] *Kotlin Programming Language*, <https://kotlinlang.org> - last checked: 15/01/2019
- [53] *Kotlin Java Interoperability*, <https://www.baeldung.com/kotlin-java-interoperability> - last checked: 04/06/2019
- [54] *IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains*, <https://www.jetbrains.com/idea/> - last checked: 15/01/2019
- [55] *TeamGantt*, <https://prod.teamgantt.com> - last checked: 15/01/2019
- [56] *draw.io*, <https://www.draw.io> - last checked: 15/01/2019
- [57] *Online Diagram Software & Chart Solution*, <https://online.visual-paradigm.com> - last checked: 15/01/2019
- [58] *Overleaf, Online LaTeX Editor*, <https://www.overleaf.com> - last checked: 15/01/2019
- [59] *Introdução a Requisitos de Software*, <https://www.devmedia.com.br/introducao-a-requisitos-de-software/29580> - last checked: 06/01/2019
- [60] *MVC Architecture*, <https://www.tutorialsteacher.com/mvc/mvc-architecture> - last checked: 10/06/2019
- [61] *Spring Boot*, <https://spring.io/projects/spring-boot>: 10/06/2019
- [62] *Spring Data JPA*, <https://spring.io/projects/spring-data-jpa>: 10/06/2019

- [63] *MySQL :: Download Connector/J*, <https://dev.mysql.com/downloads/connector/j/> - last checked: 10/06/2019
- [64] *JUnit 5*, <https://junit.org/junit5/> - last checked: 10/06/2019
- [65] *AssertJ / Fluent Assertions for Java*, <https://joel-costigliola.github.io/assertj/> - last checked: 10/06/2019
- [66] *Mockito framework site*, <https://site.mockito.org> - last checked: 10/06/2019
- [67] *Which is better? Saving files in Database or File System*, <https://habiletechnologies.com/blog/better-saving-files-database-file-system/> - last checked: 11/06/2019
- [68] *TestNG*, <https://testng.org/doc/> - last checked: 10/06/2019
- [69] *greenDAO: Android ORM for your SQLite database*, <http://greenrobot.org/greendao/> - last checked: 11/06/2019
- [70] *Dagger*, <https://dagger.dev> - last checked: 11/06/2019
- [71] *Butter Knife*, <http://jakewharton.github.io/butterknife/> - last checked: 11/06/2019
- [72] *EventBus: Events for Android*, <http://greenrobot.org/eventbus/> - last checked: 11/06/2019
- [73] *Retrofit*, <https://square.github.io/retrofit/> - last checked: 11/06/2019
- [74] *LibVLC for Android, Android TV and Chrome OS*, <https://github.com/videolan/vlc-android> - last checked: 11/06/2019
- [75] *PermissionsDispatcher*, <https://github.com/permissions-dispatcher/PermissionsDispatcher> - last checked: 11/06/2019
- [76] *Overview of Google Play services*, <https://developers.google.com/android/guides/overview> - last checked: 11/06/2019
- [77] *MaterialDialog*, <https://github.com/afollestad/material-dialogs> - last checked: 11/06/2019
- [78] *Android Ripple Background*, <https://github.com/skyfishjy/android-ripple-background> - last checked: 11/06/2019
- [79] *Sensey*, <https://github.com/nisrulz/sensey> - last checked: 11/06/2019

References

- [80] *Bootstrap*, <https://getbootstrap.com> - last checked: 16/06/2019
- [81] *WOW.js*, <https://www.delac.io/wow/> - last checked: 16/06/2019
- [82] *Animate.css*, <https://daneden.github.io/animate.css/> - last checked: 16/06/2019
- [83] *Font Awesome*, <https://fontawesome.com> - last checked: 16/06/2019
- [84] *jQuery*, <https://jquery.com> - last checked: 16/06/2019
- [85] *WeDoCare*, <https://wedocare.dei.uc.pt> - last checked: 25/06/2019
- [86] *What is Software Testing? Introduction, Definition, Basics & Types*, <https://www.guru99.com/software-testing-introduction-importance.html> - last checked: 12/06/2019
- [87] *Defect Management Process in Software Testing*, <https://www.guru99.com/defect-management-process.html> - last checked: 12/06/2019
- [88] *Unit Testing*, <http://softwaretestingfundamentals.com/unit-testing/> - last checked: 12/06/2019
- [89] *Why Performance Testing is Necessary | Abstracta*, <https://abstracta.us/blog/performance-testing/why-performance-testing-is-necessary/> - last checked: 13/06/2019
- [90] *Performance Testing Tutorial, What is, Types, Metrics & Example*, <https://www.guru99.com/performance-testing.html> - last checked: 13/06/2019
- [91] *What is performance testing*, <https://searchsoftwarequality.techtarget.com/definition/performance-testing> - last checked: 13/06/2019
- [92] *Apache JMeter*, <https://jmeter.apache.org> - last checked: 13/06/2019
- [93] *Profile battery usage with BatteryStats and Battery Historian*, <https://developer.android.com/studio/profile/battery-historian> - last checked: 13/06/2019
- [94] *Usability Testing*, <https://www.usability.gov/how-to-and-tools/methods/usability-testing.html> - last checked: 12/06/2019
- [95] *BroadcastReceiver*, <https://developer.android.com/reference/android/content/BroadcastReceiver> - last checked: 13/06/2019

- [96] *App Manifest Overview*, <https://developer.android.com/guide/topics/manifest/manifest-intro> - last checked: 13/06/2019
- [97] *Google's new SMS and call permission policy is crippling apps used by millions*, <https://www.androidpolice.com/2019/01/05/googles-new-sms-and-call-permission-policy-is-crippling-apps-used-by-millions/> - last checked: 13/06/2019
- [98] *Use of SMS or Call Log permission groups*, <https://support.google.com/googleplay/android-developer/answer/9047303> - last checked: 13/06/2019
- [99] *Declare permissions for your app*, <https://support.google.com/googleplay/android-developer/answer/9214102> - last checked: 13/06/2019
- [100] *A. Reis, D. Nunes, H. Aguiar, H. Dias, R. Barbosa, A. Figueira, S. Sinche, D. Raposo, V. Pereira, J. Sá Silva, F. Boavida, A. Rodrigues, C. Herrera*, "Tech4SocialChange: crowd-sourcing to bring migrants experiences to the academics", June 2016 - last checked: 17/06/2019
- [101] *AngularJS*, <https://angularjs.org> - last checked: 18/06/2019

Appendix A

WeDoCare Paper

In this Appendix it is presented the paper about WeDoCare that was written during the second semester, and was submitted to the CLEI 2019 conference [11].

WeDoCare: an IoT system for vulnerable social groups

Abstract

One of the biggest problems in the current society is people's safety. Safety measures and mechanisms are especially important in the case of vulnerable social groups, such as migrants, homeless, and victims of domestic and/or sexual violence. In order to cope with this problem, we witness an increasing number of personal alarm systems in the market, most of them based on panic buttons. Nevertheless, none of them has got widespread acceptance mainly because of limited Human-Computer Interaction. In the context of this work, we developed a mobile application that recognizes an attack through speech and gesture recognition. This paper describes such a system and presents its features, some of them based on the emerging concept of Human-in-the-Loop Cyber-physical Systems and new concepts of Human-Computer Interaction.

Keywords: Android, IoT, HitL, Domestic Violence

1. Introduction

We often hear stories of violence against women on the news. In fact, a study from the World Health Organization (WHO) recognized domestic and sexual violence as a worldwide problem and that women are exposed to a greater risk of violence from their husbands/partners than from other people [2]. Even though there are worldwide initiatives that fight this type of violence and try to call the attention of the general public to this pressing problem, such as the "me too" movement started in 2006 to help victims of sexual assault [1], a large percentage of women who are victims of this type of violence are still not reporting it to the authorities. Similar problems can be pointed out in the case of other types of victims, such as migrants, homeless, or even prostitutes.

On the other hand, existing personal safety systems are failing to be successful, either because they require specific hardware, or because they rely on explicit action from the victim, or both. The reality is that technology has still to improve considerably in order to be of assistance to a large variety of potential or actual victims of violence, especially if the victims belong to vulnerable social groups.

In this paper, we propose a personal alarm system named WeDoCare, based on mobile phones, optionally complemented by an unobtrusive, auxiliary device to be attached to victim's clothes. This solution would help in the detection of attacks and in the triggering of help requests - the whole of which would be performed in a very short time (i.e. less than 10 seconds). This small device has an accelerometer, a barometer and a button, so that these elements can be used for detection. Its button, when pushed, triggers the help request. Moreover, the victim's violent movements while under attack will be detected by the accelerometer, triggering a help request. This is particularly important in the case of female victim, as sometimes they leave their mobile phones in their handbag. This device has another important advantage: it can be hidden from perpetrators, as it would be placed covertly, and can detect sudden and unusual

movements. Both the mobile phone application and this additional device can trigger alerts.

WeDoCare comprises a Human-in-the-Loop Android application that recognizes an attack through speech and gesture recognition, or lets the user trigger it manually through an alarm button. This is partly because we also considered the concept of Human-Computer Interaction (HCI). HCI is a multidisciplinary field of study that focuses on the design of all forms of information technology and, in particular, the interaction between humans (the users) and computers, phones, and many other devices [16]. With this in mind, WeDoCare was developed in such a way that allows an easy and efficient interaction between the user and the application, which is very important for the problem we are trying to solve. We can see that HCI is a rather important field when it comes to help resolve social problems like the one WeDoCare is trying to solve. The easier and more efficient the interaction between the user and the application, the better. As mentioned before, the user can trigger an alarm through 3 different ways. As a result of the alarm, a text message is sent to the police with the victim's location. The system is reliable in detection, alert, communication, dissemination, and actuation, and ensures the user privacy at all times, since it does not store any type of personal data in remote systems under any circumstance and it minimizes the locally stored information. We partnered with local associations that work with sexual violence victims to develop WeDoCare. Our key partners are "Ergue-te" (Rise) [13], "Saúde em Português" (Health in Portuguese) [14] and the PSP (Polícia de Segurança Pública, the Portuguese Police). "Ergue-te" is a social intervention association that aims to help prostitutes to quit and find a new project for their lives. These are women who suffer a high level of violence, not only from their clients but from their bosses and partners as well, which makes them a target audience for WeDoCare. "Saúde em Português" is a Non-Governmental Organization that promotes social and community integration and human rights. We specifically worked with one of its departments that provides assistance and protection to victims of trafficking. Since smartphones usually are carried by users everywhere they go, we can make the most of these devices since even the cheapest mobile phones have sensors. The Android platform supports three generic types of sensors that provide data: motion sensors, environmental sensors, and position sensors [3]. Not included in these groups are camera, fingerprint sensor, microphone and touch screen [4]. These sensors are already used in millions of Android applications which are, sometimes, connected to other pieces of hardware (a smartwatch or a wristband, for example). However, continuously using these sensors has costs in terms of battery life. To evaluate the phone's battery life when using WeDoCare, we conducted several tests with the aim of comparing the phone's battery life with and without WeDoCare. We also performed some tests to ensure the reliability of speech and gesture recognition. Although speech recognition results were below what we expected, gesture recognition and battery life test results were promising.

The remainder of this paper is organized as follows: Section 2 describes some applications similar to WeDoCare that make use of mobile phone sensors and compares them with WeDoCare. Section 3 explains how the proposed application works, describes its architecture and design as well as its

privacy features. Section 4 presents the conducted tests, along with a discussion of the results. Section 5 discusses a case study, conducted in a real environment. Section 6 presents the conclusions and guidelines for further work.

2. Similar applications

In this section we will describe applications similar to WeDoCare. Some projects below are the semifinalists of the Anu & Naveen Jain Women's Safety XPRIZE [5], a competition that challenges teams to develop a project to help women in India feel safer. In some cases, there is the possibility of integrating additional hardware. Even though this can improve the reliability of the solution, it implies extra costs for the user and, therefore, is not a solution that is within reach of several potential victims.

2.1. Wearsafe

Wearsafe [6] comes as a personal or enterprise solution. Its core is a free mobile application with a button that sends an emergency alert to a chat group previously defined by the user that will coordinate the emergency response. The alert can be activated through a button inside the application home screen or by a wearable device which can be both a smartwatch or a Wearsafe Tag, useful when the smartphone is not within the user's reach. The pairing of such devices with the mobile phone implies a subscription cost. The application features also include audio streaming, location updates and the possibility for friends and family to call emergency services right from the application.

2.2. SafeTrek

SafeTrek [7] is a paid mobile application that sends emergency alerts directly to the police. When the user feels threatened, he/she can press a button inside the application home screen. When safe, the user can simply release the button and enter a pin to confirm. If in danger, the user simply releases the button without entering the pin and his/her location will be shared with the police. The minimum pricing is \$29.99 per year.

2.3. Nimb

Nimb [8] is a personal safety app that optionally pairs with the Nimb Ring, which costs \$249 and is useful when the user cannot reach the phone. The system allows users to send alert to friends, family, nearby users or emergency responders. The option to ask emergency responders for help can be accessed only through a subscription plan that costs at least \$155.

2.4. Guardian Circle

The Guardian Circle [9] is a free mobile application that works almost as a social network for personal safety. Users can ask their guardians (that can be whoever they want, from family to neighbors) for help inside the application. Guardians can then communicate with the authorities and launch an emergency response. When facing an emergency, users have to first select the alert type (emergency, urgent, request or only a test) and can optionally write about what is going on, as there is not a quick 'help' button.

2.5. Watch over me

Watch Over Me [10] is a freemium application, meaning its basic features are free of charge, while more advanced features are paid for. The application sends emergency alerts to a safety network through push notifications for free. However, since this feature does not work in areas with low connectivity, for \$4.99 per year users can unlock the 'Total safety feature' that sends alerts via SMS. The company covers the cost of those text messages. When the user feels insecure, he/she can set a timer inside the application so that it remains alert for a specified amount of time and additionally share details. If the user does not tell the system that he/she is safe before the timeout, an alert message is sent to specified users.

2.6. bSafe

bSafe[11] is a mobile application that works without the need for additional hardware. Inside the application, the user can create a safety network of friends with whom to share his/her location and send emergency alerts by tapping an emergency button. Additional features include a Fake Call and a timer that works the same way as the timer in Watch Over Me described in the previous subsection.

2.7. Leaf

Leaf [12] is a wristband with incorporated GPS that costs \$37 and pairs with the smartphone. It lets the user share live location with guardians through a mobile application. The wearable has a panic button that lets the user send emergency alerts and share live location with guardians. It also helps users find safe places like police stations and hospitals. This solution won the Anu & Naveen Jain Women's Safety XPRIZE.

2.8. WeDoCare vs. Similar applications

As we can see from the applications above, location sharing is a common feature implemented among them and is also implemented in WeDoCare. Since WeDoCare is an anonymous system, the location of our users will never be shared with others with the exception of the police. This 'sharing location with the authorities' feature is only implemented in three of the described applications and is only available when users adhere to a subscription plan. WeDoCare is a totally free application, as users do not have to pay for extra functionalities and they do not need to buy additional hardware to pair with the application. A further strength of WeDoCare is the possibility of activating an alarm without any wearable device when it is not possible to reach the mobile phone. As will be seen in section 3, WeDoCare can detect an attack through a keyword recognition module. WeDoCare shares some features with other applications but it clearly stands out for the privacy it offers. Also, it is the only system capable of detecting violent attacks with a complete 'hands-free' way. Since it is free as well, WeDoCare is in advantage over other solutions when it comes to low-cost and reliable safety applications. A summary of the comparison of WeDoCare with the other mobile applications is presented in Table 1.

3. WeDoCare

3.1. Main features

WeDoCare is a native Android application whose main goal is to detect violent attacks in an automated way using speech recognition and gesture detection. During installation, WeDoCare collects the serial ID of the SIM card and asks the user to grant all necessary permissions. Then, once the application is running, it requests location updates every five minutes if the location access is turned on. The user can see if the location access is turned on/off in the main screen of the application.

WeDoCare's main screen contains an emergency button to manually trigger alerts, a button to activate speech recognition, and another that pops-up an address form, as shown in figure 1.

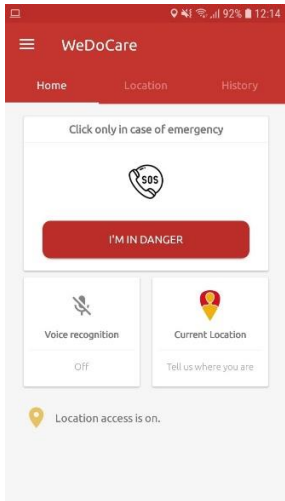


Figure 1: WeDoCare main screen

This address form contains fields the user should fill with information about his/her current location, because sometimes the coordinates WeDoCare gets might not be accurate enough. These fields are optional and can be modified anytime. The speech recognition button changes icon and color according to the speech recognition state, so that the user can know whether it is turned on or off just by looking at it. Figure 2 shows the address form.

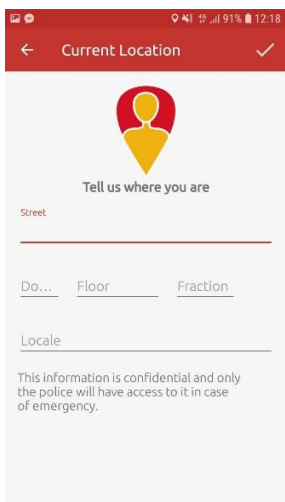


Figure 2: WeDoCare address form

The main screen is just one of the 3 tabs that the application holds. It also has a "Location" tab with an embedded map and a pin pointing to the user's current location, and a "History" tab that contains a record with the messages sent to the police.

On top of the screen there is an ActionBar that holds the current activity or application title, and a navigation button which opens and closes a NavigationDrawer. This navigation drawer lets the user access the application's privacy policy, frequently asked questions, and entities responsible for WeDoCare, as shown in figure 3, below.

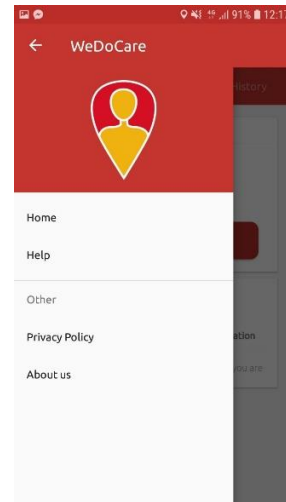


Figure 3: WeDoCare Navigation Drawer menu

The user can activate the alarm in three different ways: by saying the word "help" or "socorro" (the portuguese word for requesting assistance), by doing a chop gesture with the phone, or by clicking the panic button. In all cases, a notification appears with the text "Don't forget to buy bread" and a button. The content of the notification is, apparently, meaningless so that attackers are not able to understand what it truly means. When the notification appears, the user has then seventeen seconds to click on the notification button and cancel the alarm. If the timer finishes without the user clicking the button, an SMS is sent to the police with the location of the attack. Before sending it, however, WeDoCare checks if the SIM card ID is the same as the one collected during installation. If the values are different, it may be because the mobile phone has been stolen or sold. The notification can be seen in figure 4.

Even though good practices say that Android critical permissions must be asked right before the execution of critical actions, we opted to ignore this guideline because we consider response to threat situations to be so important that we do not want anything to get in the way of attack detections or the sending of the SMS. We will further explain this problem with a practical example. Let's consider that the user has the mobile phone locked, with the application running in the background, and activates an alarm through speech recognition without having yet guaranteed the permission to send SMS. When the timer finishes, instead of sending the SMS to the police, that application would show a dialog to the user asking them to grant the permission to send the SMS and block until the user grants it or denies it.

Every time the application is started, WeDoCare checks the language settings of the Android device and shows WeDoCare

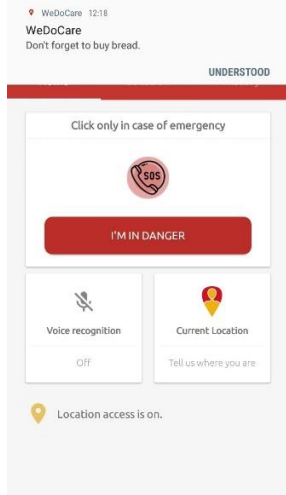


Figure 4: WeDoCare notification alert

in the same language as the mobile phone. The default language is English.

3.2. UI and UX

As mentioned in the first section, WeDoCare was developed with the concept of HCI in mind. WeDoCare's design was thought out very carefully, with the aim of giving the user a satisfactory experience with an easy to use application to improve the interaction between the user and the application.

WeDoCare does not have a lot of screens and the main features are comprised in the main screen. As it is shown in figure 1, in the main screen there is a button which the user can press when in danger. When the button is pressed there is an animation on the button as shown in figure 4. This is an interaction between the user and the application, and it shows to the user that the threat was recognized and the alarm was triggered. Another interaction is when the user wants to activate the voice recognition feature. As seen in the previous figures, the voice recognition button shows that the feature is deactivated. But when the button is pressed it changes color to show the user that it is activated.

When an alarm is triggered a notification appears, which the user can cancel very easily, just by pressing a button in the notification, as shown in figure 4. We aimed to make WeDoCare as simple and efficient as possible, providing a simple, yet elegant design and experience.

3.3. Privacy

The application is for personal use only, and since every necessary information is collected during runtime, processed and then forgotten, there is no need to create a profile. Thus, it is not necessary to implement authentication methods. The application will only be installed in devices that belong to people authorized by the associations to participate in the tests. For this reason, WeDoCare will not be available on the Google Play Store for everyone to download. It is important, however, that the application is not accessed by people outside the testing group, which could happen if the mobile phone is sold or stolen. The simplest way to make sure that alerts from unauthorized

people are not sent to the police is to verify that the SMS comes from the same SIM card ID as the one collected during the installation.

4. Tests and results

We ran several application tests. First, we distributed the application to our team members and our partner associations for two weeks, so that they could provide general evaluation of the application and report errors. Subsequently, reliability tests were performed. We defined testing protocols together with our team leaders that would let us conclude about the application performance.

4.1. Battery life

Long battery life is, perhaps, the most important aspect of user experience. No one wants to install an application that drains all the phone's battery because a dead mobile phone is, obviously, useless. If this were the case of WeDoCare, not only our users would likely uninstall it, but it could fail them in an emergency as well, leading them to believe that WeDoCare is not a viable solution.

Instead of developing the application for minimum battery consumption first, we chose to develop it for reliability first. Embedding location awareness in an application can be very battery-consuming and, in

general, there is a direct proportionality ratio between the precision of the location data and energy consumption. We use the Fused Location Provider API, a Google Play Services API, to make our application context aware. The `setAccuracy()` method accepts four different values as argument:

PRIORITY_HIGH_ACCURACY: For the most accurate location possible. Uses whatever resources available (GPS, Wi-Fi, cell towers) to compute location.

PRIORITY_BALANCED_POWER_ACCURACY: Avoids using GPS to compute location in order to save battery, preferring Wi-Fi and cell tower information.

PRIORITY_LOW_POWER: Provides minimum accuracy location data and minimum battery consumption. Uses cell towers to find the phone's location.

PRIORITY_NO_POWER: Receives location information from other location-aware applications.

WeDoCare first requests location updates every twenty seconds using the `PRIORITY_HIGH_ACCURACY` value.

Constantly trying to convert physical addresses into coordinates can be energy consuming. Instead, WeDoCare listens for changes in the connectivity state, meaning that it detects when the user turns on/off the Wi-Fi or mobile data. If the conversion has not already happened, it tries to compute the coordinates when an Internet connection is available. There are of course other components like the Speech Recognition service and the number of times the user opens the application which are also accountable for battery consumption. First, we wanted to have a general idea of the battery consumption. For this purpose, we used the following test protocol:

A. Creation of control values

1. Charge the phone completely. Register the time of full charge.
2. Register the battery level 12h later.
3. Repeat for 5 days.

B. Measure WeDoCare battery consumption

1. Charge the phone completely. Register the time of full charge.
2. Open WeDoCare.
3. Turn on location access and speech recognition.
4. Use the phone normally but with WeDoCare always running on background.
5. Register the battery level 12h later.
6. Repeat for 5 days.

A group of eight volunteers participated in these tests. Due to human error, most of the records do not correspond to a twelve-hour period. Therefore, the best way to draw conclusions based on the registered values is by calculating the weighted mean for each set of data. The results of this test can be seen in Table 2. Phase A indicates the weighted mean for normal use of the phone without WeDoCare and Phase B indicates the weighted mean for normal use of the phone with WeDoCare. Every person uses the phone differently and that influences more the battery life than its capacity. Since these values are part of an empirical analysis and are so independent from one another, it does not make sense to find a value that represents them (e.g., average values). The only conclusion that can be drawn is that WeDoCare allows, in most of the cases, for at least one day of use when its most battery-consuming functionality (i.e., location updates) is continuously active. Only two out of eight mobile phones ran completely out of battery when WeDoCare was running. Since the Samsung Note 3 Neo device shows a high battery consumption even without WeDoCare, it was expected that the application would drain its battery completely. We cannot explain what happened with the LG G3 phone, but this shows that sometimes the way WeDoCare requests location updates might be excessive for some users or devices. For this reason, we decided to space the location updates for every five minutes and kept the accuracy mode because a precise location is crucial in case of emergency.

After these adjustments we installed the application on a Motorola Moto G3 and used the application under the same conditions. Instead of measuring the battery levels for twelve hours, we considered the statistics shown on the phone settings. The battery of this mobile phone has a capacity of 2470 mAh which lasts for approximately 24h on regular use.

The battery consumption related to the speech recognition service is accounted in the Google battery statistics. During this time, we took care of not opening Google applications, but it is not possible to be sure that WeDoCare was the only contributor for the Computer Power Use. Nevertheless, adding both Computed Power Use values, we conclude that WeDoCare consumes more or less 5% of the total battery of the smartphone.

4.2. Keyword recognition

The speech recognition system must work 97% of the time, which is the initial reliability requirement value. WeDoCare uses the Android Speech Recognition API to recognize keywords. The disadvantage of it is that we do not have access to the code and it is not possible to improve the algorithm behind it. Nevertheless, we tested the performance of this recognition system in five different scenarios, all of them as close to reality as possible. Since some of our users are not portuguese speakers we studied the recognition for both 'Socorro' and 'Help' words. Twenty volunteers participated in these tests. For each case, the participants had to repeat the words 'Help' and 'Socorro' five times in a loud tone, almost screaming, with and without background noise. When the system detects the word, it launches a push notification. The scenarios were the following:

1. Phone near the person;
2. Person is 5 meters away from the phone;
3. Phone is inside the trousers' pocket;
4. Phone is inside the person's purse/backpack;
5. Reading the sentence "Enviei pedidos de socorro aos meus amigos"/"I asked my friends for help".

A score between 0 and 5, representing the number of times the system detected the keyword, was assigned to each case. We then computed the average for each of the scenarios. The results can be seen in Table 3.

The system never generated false positives. None of the participants were native English speakers, which often resulted in poor pronunciation of the word 'help', leading to worse results for the English scenarios in general. The performance of the speech recognition system did not decrease significantly with the presence of background noise. However, as already said, the system gets the results on the onResults() method, and since the service does not deliver results until it detects that audio is no longer being received, the results did not come immediately after the participant saying the word. For the same reason, this also happened when the participant said the word in the middle of the sentence. In "Enviei pedidos de socorro aos meus amigos", the result of the recognition only came after the participant saying "amigos" and not immediately after "socorro".

As a result, in order to remove this delay WeDoCare now receives the recognition results from the onPartialResults() method. If we constantly parse the partial results, the keyword will be found sooner. Receiving the results in this method allows for near real-time processing.

Another approach to analyze the data was to convert the results into binary labels. If the system detects the keyword at least one out of five tries, then the result is a true positive and receives the label '1'. Otherwise, it receives the label '0'. This approach is closer to a real situation, since the user is not likely to scream for help only once in that case. The results of this alternative approach can be seen in Table 4.

Even for this approach, performance of the speech recognition service is far from desired, which makes this feature is inefficient for remote automated attack detection. There must be, therefore, another functionality in the application to trigger

the alarm without, at least, the necessity of unlocking the phone. To compensate for this, we integrated gesture detection in WeDoCare, as explained in more detail in the next section.

4.3. Gesture Detection

An Android phone is provided with various types of sensors, including sensors to measure motion, such as accelerometers, gravity sensors, gyroscopes and rotational vector sensors. WeDoCare can use these sensors to look for a pattern in movement and trigger alarms. The Android Development Kit has classes to create gesture-based events, but it requires writing considerable amounts of boilerplate code. To avoid this, we found Sensey [15], an Android library for Gesture Detection. This library provides a set of methods to detect pre-defined gestures, which include Wrist Twist gesture, Shake gesture, Touch types and, among others, the one we chose for WeDoCare: the chop gesture.

Our first approach was to implement the shake gesture which resulted in a high number of false positives. The chop gesture is a more precise and vigorous gesture. Since it is not a common movement to do with the mobile phone, it is less likely to trigger false positives. The library reads data from the accelerometer. The time defined for the chop gesture was 100 milliseconds and the threshold for the accelerometer data was 10ms-2. These are the only parameters required by the library and were obtained following a trial-and-error approach. The gesture detection was tested in three different devices. The procedure was to simply perform the gesture with the mobile phone and checking if the push notification appears. The results are presented in Figure 5.

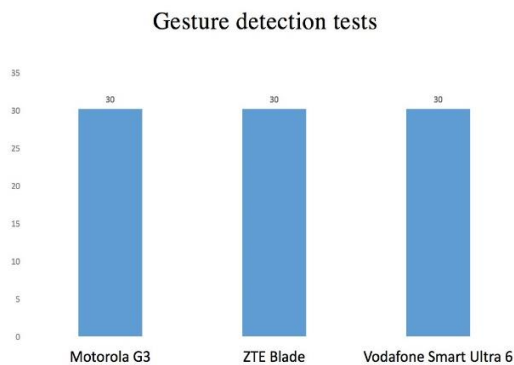


Figure 5: Gesture recognition results

For each mobile device we performed the gesture thirty times. The obtained results show that the system detects the gesture and triggers the alarm 100% of the times and is reliable. It is important to mention that when the application was tested in a real environment there were a few false positives, but since the notification appears, the phone vibrates and there's a 17 second window to cancel the help request, the users were able to abort it.

5. Case studies: Saúde em Português and Ergue-te

We tested the application in real environments. Saúde em Português and Ergue-te agreed to collaborate with us.

5.1. Concerns

Some concerns related to our beta testers and how they could jeopardize the testing of the system in a real scenario arose during the meetings with "Ergue-te" and "Saúde em Português". These concerns represent the risks of our project and are associated with situations that are beyond our control and are hard to prevent. These risks are:

- **Some members from the association might refuse to adhere to the test program**

Part of the people we want to include in this phase are female prostitutes. These women are very cautious with their privacy and with what they share with others and can be afraid of relying on technology because they know it implies sharing personal information at some point. Even if they are assured that their information is not stored or accessible to anyone, their lack of knowledge of digital tools prevents them from trusting the system. This is a limiting factor that may shorten our test population.

- **Even with the application installed, it is quite probable that users close it unintentionally at some point**

This is a very expectable situation. Even though our potential testing population has a smartphone, some of them do not take full advantage of their functionalities simply because they are not completely comfortable with the device. Closing the application will make all the system services to stop and, in case of emergency, the user will not be able to trigger an alarm automatically. This situation might discredit the application in the eyes of the victim who may stop using the system.

- **Low number of occurrences, which could lead to the entity responsible to answer help requests neglecting the system**

The system will be implemented and tested in a hostile environment. If during the test period a low number of attacks occurs, the entity responsible for answering help requests might forget about the phone that was given to them for emergency situations. If, by chance, the mobile phone runs out of battery power, the victim will not be assisted, and this situation might also discredit the application in the eyes of the victim. Besides, it is also difficult to analyze the applicability of the system if the number of occurrences is very low.

- **The user has no credit on the mobile phone**

The SMSs sent by the system are charged according to the user's operator. If the phone is without credit, it is impossible to inform the police that the victim needs assistance. In this case, the system works properly but it is not possible to meet its final purpose, which is to aid the victims. However, this can be avoided if a call is made to the emergency number.

- **The user does not activate access to location services and does not indicate the address of his/her current location**

In this scenario, it is not possible to send an SMS to the police, as there is no information about the location of the attack. Like in the scenario above, the system also loses its purpose in this situation. As already said, the testers will be assured that their location is not stored by the system and they can follow instructions on how to stop Google from keeping location

history in a user guide that was given to them. However, their lack of trust in technology might still lead them not to provide location information. It is hard to determine the exact likelihood of the above scenarios. Nevertheless, one can expect that the mentioned situations are likely to occur.

5.2. Success criteria

The success criteria are critical situations that need to happen in order to consider this case study a success. These are:

- **The panic button triggers the alert every time it is pressed**

The panic button is the most reliable way to trigger an emergency alert and get help from the police, but it is also the less convenient way to trigger an alarm because it requires the user to unlock the phone. However, if other alternatives fail, this button must work 100% of the times.

- **Gesture detection works every time the user does the Chop movement**

Gesture recognition is the most reliable way for the user to trigger an alert without unlocking the phone. In a panic situation, the user will probably perform the chop gesture more than once. It is very important that the recognition works almost 100% of the times.

- **The keyword recognition works, at least, 70% of the times**

Since it is not possible to improve the recognition system, the performance of the keyword recognition in this stage must at least equal the performance of the keyword recognition test described previously.

- **Low energy consumption**

WeDoCare must not drain the phone's battery completely so that it becomes unavailable in case of emergency.

5.3. Results

The application was distributed successfully to the testing group.

During a period of three weeks, no attacks occurred but the application is still on the field. However, "Saúde em Português" reported that its members feel safer already just by knowing that they can easily ask for help if necessary. We also got the information that the users feel comfortable using WeDoCare, proving that its UI/UX meets its purpose.

No errors or malfunctions were reported during this time.

6. Conclusions

The purpose of this work was to develop a reliable mobile application with social purpose, while exploring HitL and IoT concepts, and evaluate its utility in real context. Throughout the year, we partnered with local associations and the local police to decide how WeDoCare could be of use to social minorities. Our key feature was reliability. We were able to develop an application that activates accurate emergency location alerts using keyword and gesture recognition and a button. Two out of these three methods work 100 % but the keyword recognition method proved to be insufficient for attack detection. The additional hardware helps increasing the reliability of WeDoCare whenever the user cannot reach the mobile phone. As future work, aspects like developing for low battery consumption, threads and processes, reactive programming and unit testing must be prioritized as they are important for the application performance.

References

- [1] me too., <https://metoomvmt.org/>
- [2] Claudia Garcia-Moreno, Henrica AFM Jansen, Mary Ellsberg, Lori Heise, Charlotte H Watts. Prevalence of intimate partner violence: findings from the WHO multi-country study on women's health and domestic violence, Lancet
- [3] Sensors Overview, https://developer.android.com/guide/topics/sensors/sensors_overview.html
- [4] Sensors, <https://source.android.com/devices/sensors/>
- [5] Anu and Naveen Jain Women's Safety XPRIZE, <https://safety.xprize.org/teams>
- [6] wearsafe labs, <https://www.wearsafe.com/>
- [7] SafeTrek, <https://www.safetrekapp.com/>
- [8] NIMB, <https://nimb.com/app/>
- [9] GuardianCircle, <http://guardiancircle.com/>
- [10] Watch Over Me, <http://watchovermeapp.com>
- [11] bSafe, <https://getbsafe.com/features/>
- [12] LEAF, <https://www.leafwearables.com/>
- [13] ergue-te, <http://erguete.pt>
- [14] Saúde em Português, <http://www.saudeemporugues.com>
- [15] Sensey, <https://github.com/nisrulz/sensey>
- [16] Human Computer Interaction (HCI), <https://www.interaction-design.org/literature/topics/human-computer-interaction>

Annex

Application	Anonymity	Location Sharing	Cost	Additional Hardware	Intelligent Attack Detection
WeDoCare	Yes	Yes	Free	No	Yes
WearSafe	No	No	Freemium	Yes	No
SafeTrek	No	Yes	Freemium	No	No
Nimb	No	Yes	Freemium	No	No
Watch Over Me	No	Yes	Freemium	No	No
bSafe	No	Yes	Free	Yes	No
Guarian Circle	No	Yes	Free	No	No
Leaf	No	Yes	Paid	Yes	No

Table 1: WeDoCare vs Similar applications

Mobile Phone	Phase A	Phase B
Vodafone Smart Ultra 6	68%	29%
Samsung Note 3 Neo	10%	0%
Vodafone Smart Ultra 6	59%	37%
ZTE Blade	75%	51%
Xiaomi Remi Note 3	62%	40%
Samsung J5000 FN	63%	41%
Samsung J3 2017	17%	11%
LG G3	57%	0%

Table 2: Results of the first battery test

	Portuguese		English	
	Without background noise	With background noise	Without background noise	With background noise
Near	3.8	3.5	4.4	3.4
5 meters distance	1.7	1.1	0.1	0.3
Pocket	2.5	2.2	0.6	0.7
Purse	1.8	2.0	1.0	0.8
Sentence	3.4	2.9	3.4	3.1

Table 3: Results of the keyword recognition system

	Portuguese		English	
	Without background noise (%)	With background noise (%)	Without background noise (%)	With background noise (%)
Near	89.5	89.5	94.7	94.7
5 meters distance	73.7	63.2	5.3	21.1
Pocket	73.7	68.4	23.6	42.1
Purse	68.4	63.2	42.1	26.3
Sentence	100.0	84.2	89.5	94.7

Table 4: Results of the keyword recognition system: another approach

Appendix B

WeDoCare installation guide

Instalação Inicial

Sempre que instalamos uma aplicação da Google Play Store é executado o download de um ficheiro com extensão .apk, como por exemplo, aplicação.apk. É esse ficheiro que permitirá instalar a aplicação no telemóvel. Assim, irá ser fornecido o ficheiro WeDoCare.apk que permitirá instalar a aplicação WeDoCare. De seguida apresentam-se os passos necessários para a instalação:

1. No seu computador navegue até à pasta que contém o ficheiro WeDoCare.apk;
2. Ligue o telemóvel ao computador através de um cabo USB;
3. Verifique que no telemóvel está escolhida a opção para “Transferência de ficheiros”, como se pode ver na figura B.1;

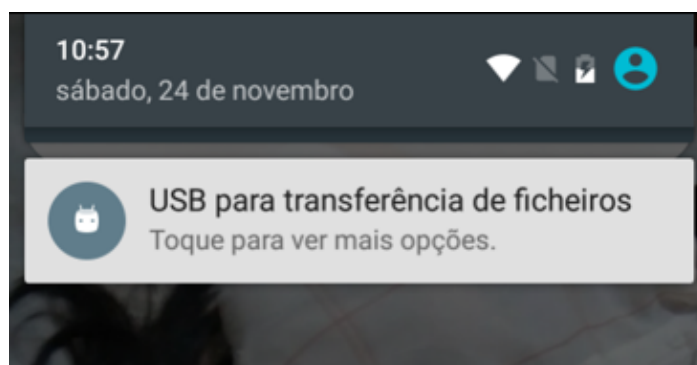


Fig. B.1 USB para tranferência de ficheiros

WeDoCare installation guide

4. No seu computador navegue até à pasta “Este PC” ou “Este computador” e aí deverá aparecer a pasta do telemóvel (figura B.2);

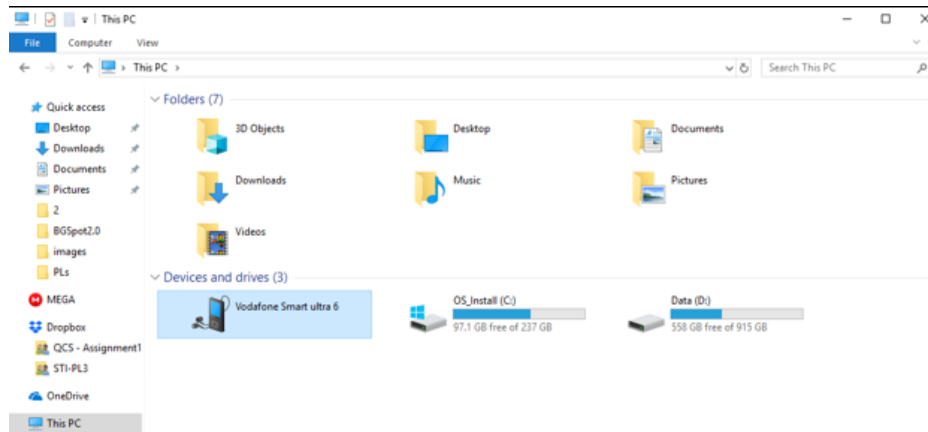


Fig. B.2 USB para transferência de ficheiros

5. Abra essa pasta e navegue até à pasta que diz “Armazenamento Interno”;
6. Agora mova o ficheiro WeDoCare.apk da sua pasta no seu computador para essa pasta do telemóvel que diz “Armazenamento interno” (figura B.3);

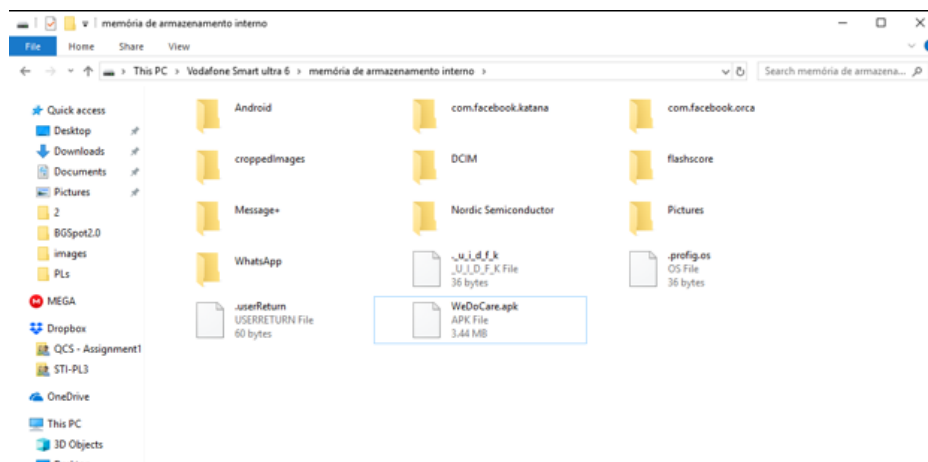


Fig. B.3 USB para transferência de ficheiros

7. Quando a transferência do ficheiro acabar, pode desconectar o telemóvel do computador;

-
8. No seu telemóvel deverá procurar uma pasta com o nome “Gestor de Ficheiros”. O nome pode variar. Também pode ter o nome “Explorador”. Basicamente é a pasta onde podemos procurar todos os ficheiros do telemóvel;
9. **NOTA: Se encontrou a pasta referida no ponto anterior, avance para o ponto 11.** Em certas situações, os telemóveis podem não ter instalada a referida pasta no ponto anterior. Nesse caso, é sugerido que vá à loja Google Play e faça o download da aplicação “Astro”. Esta aplicação permitirá que consiga procurar por todos os ficheiros do telemóvel. A figura B.4 mostra a aplicação;



Fig. B.4 Download do gestor de ficheiros Astro

WeDoCare installation guide

10. Depois de instalada a aplicação “Astro”, clique nela para a abrir e poder procurar ficheiros;
11. Quando abrir o “Gestor de Ficheiros” ou o “Astro”, terá de seleccionar a pasta de armazenamento interno do telemóvel, porque foi onde colocámos o ficheiro WeDoCare. No caso da imagem esquerda da figura B.5, essa pasta tem o nome “memória de armazenamento interno”. No caso da imagem do lado direito apenas tem o nome “Telemóvel”;

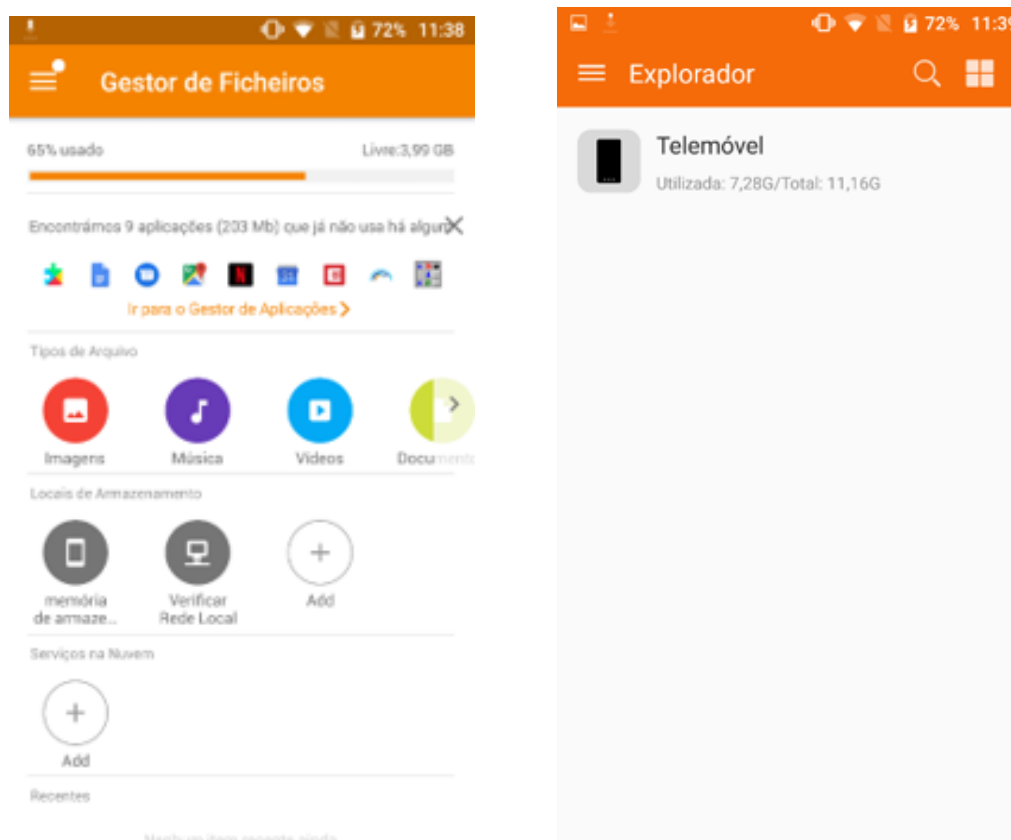


Fig. B.5 Armazenamento interno do telemóvel

12. De seguida é necessário procurar o ficheiro WeDoCare.apk, que deverá estar quase no fundo da lista uma vez que a lista é ordenada alfabeticamente;
13. Clique no ficheiro e a instalação será efetuada;
14. Em alguns casos, poderá ser necessário dar permissão para instalar aplicações de fontes desconhecidas. Se quando clicou no ficheiro do WeDoCare e a aplicação não foi instalada devido a este problema, faça o seguinte:

-
- Vá a Definições;
 - Clique em Seguranças;
 - Navegue até encontrar a opção Fontes desconhecidas;
 - Clique para ativar e confirme;
 - Agora já poderá instalar o WeDoCare.

Reconhecimento de voz em modo offline

Depois de instalado o WeDoCare, muitas vezes é necessário ativar o reconhecimento de voz offline para o WeDoCare poder captar melhor a palavra “socorro”. Para ativar o reconhecimento de voz em modo offline, siga os seguintes passos:

1. Vá às **definições** do telemóvel;
2. Clique em **Idioma e Entrada**;
3. Clique em **Voz**, ou **Escrita por Voz Google**;
4. Em **Idiomas** procure **Português (Brasil)**, e preme sem soltar para seleccionar como idioma principal. Depois clique em **Guardar**;
5. Em **Reconhecimento de voz offline** verifique se o idioma **Português (Brasil)** se encontra instalado. **Caso não se encontre instalado**, no separador **Tudo** procure **Português (Brasil)** e clique no idioma para instalar.

Appendix C

WeDoCare document sent to the Police

WeDoCare

Contexto

O presente relatório tem como objetivo principal apresentar e explicar o funcionamento da aplicação móvel WeDoCare.

Infelizmente casos de violência contra as mulheres são algo que ocorre com frequência. Esta aplicação surgiu como uma possível solução para tentar ajudar a combater este problema e trazer assim uma maior segurança para as mulheres. Um dos nossos parceiros mais próximos tem sido a equipa de intervenção social Ergue-te que tenta ajudar mulheres, na sua maioria prostitutas.

Funcionamento básico

O objetivo principal do WeDoCare é detetar uma situação de perigo e, quando tal acontece, despoletar um alarme. Se o alarme não for cancelado, então uma mensagem com a localização atual da vítima e com o tipo de alarme detetado será enviado para um telemóvel que estará em posse das autoridades policiais, para que estas consigam atuar.

A aplicação faz uso do sistema de GPS implementado no telemóvel para obter a localização da vítima, sendo esta localização atualizada a cada 5 minutos. Todavia, é necessário o utilizador ter o GPS ativado no seu telemóvel, algo que é recomendado pela aplicação. Se o utilizador preferir não ativar o GPS, a aplicação tem como alternativa a disponibilização de um pequeno formulário onde o utilizador pode colocar uma morada onde se encontre frequentemente. Esta informação também é guardada no telemóvel num formato não visível ao utilizador comum. No caso da deteção de uma situação de violência, se o GPS não estiver ativo, então é a morada que o utilizador colocou no formulário que é enviada na mensagem para o telemóvel em posse das autoridades policiais.

A aplicação apresenta 3 formas diferentes de detetar uma situação perigosa: i) através de um botão de alerta no ecrã principal, ii) através do grito da palavra “socorro”, iii) através de gestos bruscos pré-determinados.

Quando a vítima sentir que está numa situação de perigo iminente, pode premir o botão de perigo no ecrã principal da aplicação. Se o botão for premido, então uma notificação

aparece no telemóvel. Esta notificação contém um texto aparentemente casual que inclui a frase “Não se esqueça de comprar pão!”. Este conteúdo casual é útil porque se o agressor tiver acesso ao telemóvel, muito provavelmente não irá detetar nada fora do normal. A notificação fica presente durante 17 segundos. Esta janela de 17 segundos tem a finalidade de evitar falsos alarmes. Durante esse tempo, o utilizador pode premir um botão na notificação que diz “Compreendi”, e assim a notificação e o alarme serão cancelados. Se isto se verificar, nenhuma mensagem é enviada para o telemóvel em posse da polícia.

Porém o telemóvel pode não estar ao alcance da vítima quando de um possível ataque por parte de um agressor. Para tentar combater essa possibilidade, o WeDoCare possui a funcionalidade de detetar a palavra “socorro” como alternativa para sinalizar uma situação perigosa. Para tal é necessário ativar o reconhecimento de voz na aplicação. Se o telemóvel se encontrar fora do alcance da vítima, esta pode gritar a palavra “socorro” que a aplicação reconhecerá uma situação de perigo. O restante processo é igual à situação descrita previamente: a notificação fica presente durante 17 segundos, e se não for cancelada a mensagem, é enviado alerta para a polícia com a localização atual da vítima e o tipo de alarme.

Por fim, a aplicação também é capaz de detetar uma situação perigosa através de um gesto brusco, denominado em inglês de “*chop*”. Quando um agressor ataca uma vítima, a ocorrência de gestos bruscos é muito provável. O WeDoCare consegue detetar esses gestos bruscos e gerar a mesma notificação de perigo já referida. O restante processo é o mesmo.

Como já foi referido, a informação do cartão SIM e da localização da vítima são guardados no telemóvel, de um modo em que não é visível ao utilizador comum. Também é guardada informação relativa às mensagens que são enviadas para o telemóvel em posse das autoridades policiais. Contudo o que é guardado é apenas se a mensagem foi enviada ou não.

Arquitetura e Privacidade

A arquitetura da aplicação é simples, pois apenas recorre ao telemóvel da vítima e ao telemóvel que receberá a mensagem de perigo. Quando a aplicação é iniciada pela primeira vez, é recolhida informação sobre o cartão SIM do telemóvel, que será posteriormente guardada no mesmo, num formato não visível ao utilizador comum. Assim, sempre que o

utilizador voltar a iniciar a aplicação, a informação do cartão SIM é comparada com a informação que foi recolhida durante a primeira iniciação da aplicação. Esta verificação é útil se, porventura, um telemóvel com esta aplicação for roubado. Se o autor do roubo colocar o seu cartão SIM no telemóvel, não irá conseguir usar a aplicação, visto que esta ao ser iniciada irá comparar a informação do novo cartão com a informação guardada previamente. Como a informação do cartão SIM não é igual, a aplicação fecha automaticamente, evitando assim um uso indevido da mesma, e claro, evitando desta forma falsos alarmes.

Relativamente à privacidade das vítimas, esta é intocável, visto que nenhuma informação pessoal sobre as vítimas é guardada na aplicação, e os dados que são guardados, estão presentes no telemóvel mas inacessíveis a um potencial agressor. Além do mais, a instalação da aplicação é feita pelos elementos responsáveis da Ergue-te, pelo que a identidade de cada uma das vítimas não será revelada. Com isto conseguimos garantir que nenhuma das vítimas é capaz de ser identificada.