FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Tiago Moisés Jacob Dinis

IFRIEND

IoT Platform for non-intrusive monitorization of elderly people

Thesis submitted to the University of Coimbra for the Master's degree in Electrical and Computer Engineering, Specialization in Telecommunications.

Supervisors:

Prof. Dr. Jorge Sá Silva (University of Coimbra)
Prof. Dr. André Rodrigues (Polytechnic of Coimbra)

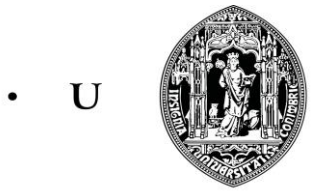
October 2022

This work was developed in collaboration with:

Universidad de Alcalá

UNIVERSIDAD DE EXTREMADURA

Department of Electrical EngineeringUniversity of Coimbra



UNIVERSIDADE DE COIMBRA


Hospital Universitario
Príncipe de Asturias
SaludMadrid


Fundación
General CSIC


Iceland
Liechtenstein
Norway
Active
citizens fund

Acknowledgments

This work was conducted under the guidance of Prof. Dr. Jorge Sá Silva and Prof. Dr. André Rodrigues, to whom I must express my gratitude for all the support and constant presence, throughout this project. I must also show my gratitude toward Marcelo Fernandes for all the companionship, support, and guidance in this project. I extend my gratitude to all my friends and colleagues, which toasted me over theyears with joy, happy moments, and friendship.

Resumo

Atualmente, as plataformas eHealth estão a ser cada vez mais utilizadas e desenvolvidas para melhorar a precisão no diagnóstico e tratamento, para uma observação mais personalizada do estado de saúde dos pacientes e para processar a informação em tempo-real usando tecnologias como WiFi, Artificial Inteligence, Chatbots e Internet Of Things. Atualmente existe rede WiFi na maior parte dos espaços públicos e privados. Por isso, aumenta a facilidade de conexão dos dispositivos e a integração de IoT na sociedade, principalmente na área da saúde. O objetivo deste projeto é desenvolver uma plataforma inovadora de monitorização sem fios através de WiFi, baseada numa app. Esta tecnologia baseia-se nas alterações que o corpo humano provoca nas ondas eletromagnéticas utilizadas em comunicações por WiFi. Essas alterações irão ser analisadas através do *Channel State Information* (CSI) entre dois dispositivos WiFi (Router e Access Point). O foco do nosso projeto é feito na deteção da frequência cardíaca, da frequência respiratória e na localização/atividade do paciente. O projeto iFriend vai permitir a monitorização inteligente do estado de saúde de idosos que sofrem de insuficiência renal, sendo primariamente testado em estudantes das Universidades de Coimbra, Alcalá, Extremadura e, como objetivo final, em ambiente real com pacientes do Hospital Príncipe das Astúrias. Esta monitorização irá ser feita através de duas aplicações, uma para smartphone e outra para smartwatch, ambas desenvolvidas em Xamarin (desta forma suportando dispositivos Android e iOS) para gerir e supervisionar as atividades do dia a dia dos idosos e para recolher alguns dos dados vitais destes. E, por fim, para complementar e agrupar os dados recolhidos foi desenvolvido um Dashboard onde os enfermeiros e responsáveis de saúde irão poder analisar, em tempo “quase-real” e em historial, a informação sobre os pacientes.

Abstract

Nowadays eHealth platforms are being increasingly used and studied to improve accuracy in diagnosis and treatment, for a more personalized view of a patient's health status and to process information in real-time using technologies such as WiFi, Artificial Intelligence, Chatbots, and Internet of Things. Currently, there are WiFi networks in most places, public and private. Therefore, it increases the ease of connecting devices and integrating devices IoT in society, especially in healthcare. The objective of this project is to develop an innovative wireless monitoring platform over WiFi with IoT validation, based on an app. This technology is based on the changes that the human body causes to electromagnetic waves of the WiFi network. These changes will be detected by analyzing the *Channel State Information* (CSI) between two WiFi devices (Router and Access Point) and the respiration rate validated with a respiration belt. The focus of our work is based on the heart rate, respiratory rate, and location/activity of the patient. This project will allow intelligent monitoring of the status of the health of elder people, especially those suffering from insufficient kidney function. Being initially tested on students from the Universities of Coimbra, Alcalá, and Extremadura, and as a final goal, in a real environment with patients at the Príncipe of Asturias's Hospital. In addition to this monitoring, there will be a smartphone and smartwatch app, both developed in Xamarin, which supports both Android and iOS devices to manage and supervise the day-to-day activities of the elderly. And finally, develop a dashboard where nurses and healthcare professionals of the institutions will access "real-time" data and see the historical information of the patient.

List of Acronyms

AP Access Point.
API Application Programming Interface.

CPS Cypher-Physical System.
DB Database.
DTO Data Transfer Object.

GE's Generic Enablers.
HITLCPS Human-in-the-Loop Cypher-Physical Systems.
HTTP Hypertext Transfer Protocol.

ID Identification.
IoT Internet of Things.
JSON JavaScript Object Notation.

LINQ Language Integrated Query.
MAC Media Access Control.
NDK Native Development Kit.
NLP Natural Language-Programming.
OS Operating System.

REST Representational State Transfer.
RF Radio Frequency.
UI User Interface.
URL Uniform Resource Locator.
VGS Version Control System.
VM Virtual Machine.

List of Figures

List of Tables

Contents

1 - Introduction

1.1 Context

The permanent monitorization of the health state has been an area of interest for the past few years in research, namely the study of efficient ways to monitor elderly people, provide them with the best healthcare, and facilitate the gathering of data. This monitoring is normally done by measuring the vital signs of the patient such as heart and respiration rates. One of the areas that have been thriving and taking advantage of this evolution of permanent monitoring is elderly healthcare. Specifically for this type of population, the use of the technology that currently exists isn't properly easy to use and implement. The way this technology is implemented can be usually very intrusive to this type of people for many reasons. As they aren't used to it or the tools that are being used interfere too much with their daily habits which becomes a challenge of another level such as comfortability and adaptability. One way to counteract this dynamic is to provide ways to support the maintenance of the quality of life of the elders. The longer people can remain mobile and care for themselves, the lower the cost of long-term care to families and society. Our project aims to provide a user-friendly way for elderly people to self-manage their general health and for their healthcare professionals and caregivers to monitor their health status with greater accompaniment and data precision.

1.2 Objectives

- One of our main objectives for this project is, precisely, to smoothly bring and adapt this type of technology, referred on the last section, to personally take care of elder people. For that, uninterrupted remote monitoring can be a solution of "passive wireless sensing", so that to avoid the necessary direct contact with the patient that is currently being done in an obstructive way. The iFriend project started last year whereas the mobile app is being developed by my colleague José Ramos and the part of monitoring the signal vitals through WiFi's Channel State Information is being done by my colleague Guilherme Lemos.

- With that in mind, my objective was to develop a smartwatch application to complement the data collected by the smartphone app and the WiFi monitoring system.

- Develop the Dashboard which is the main tool responsible for showing all the data collected by all the devices and that allows the health professional to check the health status of the patient. Whether viewing historical information or the "almost live" information it enables the doctor to have a more insightful, permanent knowledge and awareness of the medical record.

- Finally, my goal is to assure the continuation of the project so that can be grouped with the work of the other universities and make possible the implementation in a more real ambiance which is a hospital or nursing homes.

1.3 Thesis Structure

This master thesis is structured into six chapters: Introduction, State of the Art, System Overview, System Development, Tests/Results, and Conclusions/Future work:

- The first chapter serves as an introduction to the project, presenting the context in which it is inserted and the main objectives it had. This chapter is also presented a quick overview of the technologies used and the methodology.

- The second chapter, it is presented an overview of the state of the art.

- The third chapter is focused on the overall system development, that is the system architecture and its components, where it is given a general overview of all the system components, the smartwatch application iFriendWear, and the dashboard. And it is presented a general requirements overview for my part of the project.

- The fourth presents the development of the system and allits components as well as the several steps the project undertakes.

- The fifth focuses on testing and validation of the system, namely on the wear application.

- The sixth chapter is dedicated to the conclusions taken from this project as well as an overview of future work and open challenges.

1.4 Materials and Methodology

In this section, we state how the project was planned and developed throughout its duration and the materials we used to achieve its completion. The requirements and architecture of this project were previously designed so it could follow the evolution of the technologies used. Maintaining enough documentation to accompany the development of the project and the collaboration of our direct partners. Although, with the advances in the project some points were raised, and the plan had to be adopted. The iFriendWear app was made on JetBrains Rider [1] in Xamarin and tested on a smartwatch (Motorola Moto 360SP with a battery of 300 mAh capacity, Bluetooth 4.0, a Qualcomm Snapdragon 400 with 1.2 GHz quad-core processor, Adreno 305 with 450MHz graphics unit and an optical heart rate monitor amongst the other basic sensors of a smartwatch) provided by the University. The Dashboard was designed and developed on JetBrains IntelliJ PyCharm[2] in Python.

In this project, we used the Scrum [3] methodology. This methodology consists of doing several iterations in the project, to obtain a product that adapts itself to the evolutions of the market. That is, instead of focusing only on planning at the start and moving to the next phase of development only when the previous one is finished, we work with Sprints. Sprints are iterations of the project; they usually take from 2 to 3 weeks and follow the same phases as a normal software development plan, build, test and review. An example of a potential Sprint can be, for example, the redesign of the User Interface of the Dashboard. At the start of every Sprint, we plan the requirements to make the small implementations of this Sprint, in the build we develop those changes, then test them and at the end of the Sprint, we review those changes. At the end of every Sprint, we should have a potentially shippable product, that we can choose to deploy or not. At the end of one Sprint, we evaluate the project, and start a new iteration (Sprint), if one is needed. This methodology is very simple and makes the development of software easier since sometimes it is hard to have a full image of the final product right from the start. Also, as a form of communication, we utilized Slack [4], which allowed us to separate the different projects in different channels. This way we were able to send and share relevant information with the members of each project, we could also create topics for specific problems and find faster solutions this way. Skype [5] was used to do the weekly meetings to state the work done in the previous week and plan the work for the week ahead.

2 - State-Of-Art

In this section, we will introduce some of the concepts used during the development of this project, as well as some of the fundamental technologies applied.

2.1 Concepts

In this section, we address some of the concepts and paradigms used throughout the project. These concepts were fundamental for all phases of the development, and as such we think that a summary of them should be made.

2.1.1 Internet-of-Things

The Internet of Things (IoT) describes the network of physical objects – "things" – that are embedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the internet. These devices range from ordinary household objects to sophisticated industrial tools. With more than 10 million connected IoT devices today, experts are expecting this number to grow up to 16 million in 2025 and 25 million in 2030 [6].

Over the past few years, IoT has become one of the most important technologies of the 21st century. Now that we can connect everyday objects—kitchen appliances, cars, thermostats, baby monitors—to the Internet via embedded devices, seamless communication is possible between people, processes, and things.

Through low-cost computing, the cloud, big data, analytics, and mobile technologies, physical things can share and collect data with minimal human intervention. In this hyperconnected world, digital systems can record, monitor, and adjust each interaction between connected things. The physical world meets the digital world—and they cooperate.

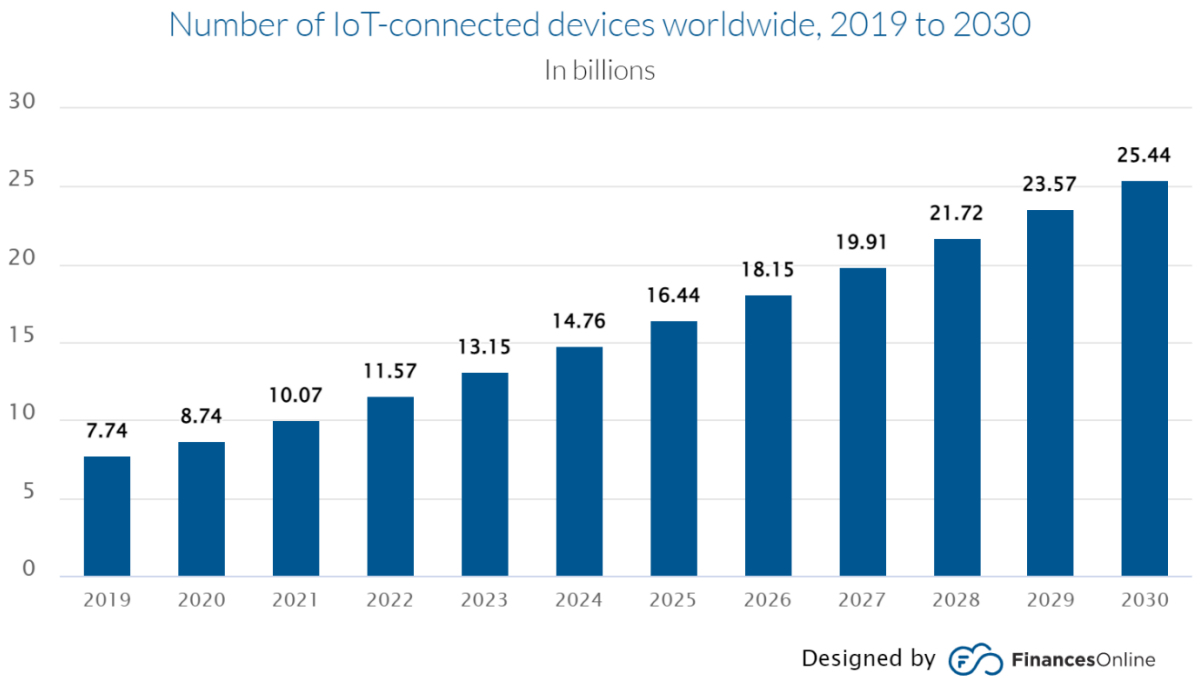


Figure 1: *Number of connected devices installed base worldwide from 2019 to2030 (in billions)* [6]

A collection of recent advances in several different technologies has made the idea of IoT more practical:

- **Access to low-cost, low-power sensor technology**. Affordable and reliable sensors are making IoT technology possible for more manufacturers.
- **Connectivity**. A host of network protocols for the Internet has made it easy to connect sensors to the cloud and other "things" for efficient data transfer.
- **Cloud computing platforms.** The increase in the availability of cloud platforms enables both businesses and consumers to access the infrastructure they need to scale up without having to manage it all.
- **Machine learning and analytics.** With advances in machine learning and analytics, along with access to varied and vast amounts of data stored in the cloud, businesses can gather insights faster and more easily. The emergence of these allied technologies continues to push the boundaries of IoT, and the data produced by IoT also feed these technologies.
- **Conversational artificial intelligence (AI).** Advances in neural networks have brought natural-language processing (NLP) to IoT devices (such as digital personal assistants Alexa, Cortana, and Siri) and made them appealing, affordable, and viable for home use.

2.1.2 Cyber-Physical-Systems

Cyber-physical systems (CPSs) are built from, and depend upon, the seamless integration of computational algorithms and physical components. These systems combine digital and analog devices, interfaces, sensors, networks, actuators, and computers with the natural environment and with human-made objects and structures. Just as the Internet has transformed the way people interact with information, cyber--physical systems are transforming the way people interact with the physical world. At the same time, the scale and inherent heterogeneity of these systems pose tremendous engineering challenges [7]. New technological approaches are needed to formalize their design, manage and control them in a scalable, efficient, and secure way, and ensure their usability. The components and domains that compose the CPS area are increasingly used to build user-friendly environments that can:

- Monitor the health of patients at home or a hospital, though wearable sensors or non-intrusive environmental monitors, to ensure that sub-optimal vital signs are recognized early, and emergencies are responded to immediately.

- Deliver integrated public transport and safe, efficient road traffic systems. Time spent traveling can be reduced if travelers have simple, cost-effective ways to switch travel modes with integrated tickets that are accepted across independent transport providers and up-to-date, accurate information on where there are currently jams or backlogs, and where in the network there is spare capacity not being used.

- Secure cost-effective, traceable food supplies. Distributed sensors, vehicles, and complex decision-making support software are needed in agriculture to allow farmers to achieve the best possible yield in return for their investments and to react to conditions on the ground, whilst CPSs could be deployed throughout the food chain to ensure that our food supplies are traceable.

- Provide secure and energy-optimized buildings. Going beyond simple temperature and humidity sensors, smart homes and offices of the future will use varied data inputs such as weather forecasts and knowledge about the time of day, season, and building usage to provide comfortable environments with minimum energy consumption.

2.1.3 Human in the Loop Cyber-Physical Systems

Human-in-the-loop cyber-physical systems (HiLCPSs) comprise a challenging and promising class of applications with immense potential for impacting the daily lives of many people. A typical HiLCPS consists of a loop involving a human, an embedded system (the cyber component), and the physical environment. The embedded system augments a human's interaction with the physical world. A HiLCPS infers the user's intent by measuring human cognitive activity through body and brain sensors. The embedded system in turn translates the intent into robot control signals to interact with the physical environment on the human's behalf via robotic actuators. Finally, the human closes the loop by observing the physical world interactions as input for making new decisions. Examples of HiLCPSs include brain-computer interfaces (BCIs), controlled assistive robots, and intelligent prostheses [8].

HiLCPS applications offer benefits in many realms— for example, the population of functionally locked-in individuals would benefit tremendously from such systems. Because these individuals cannot interact with the physical world through their movement and speech, they often must rely heavily on support from caregivers to perform fundamental everyday tasks, such as eating and communicating. A HiLCPS could aid in restoring some autonomy by offering alternative interfaces to the cyber-physical environment for interaction, communication, and control.

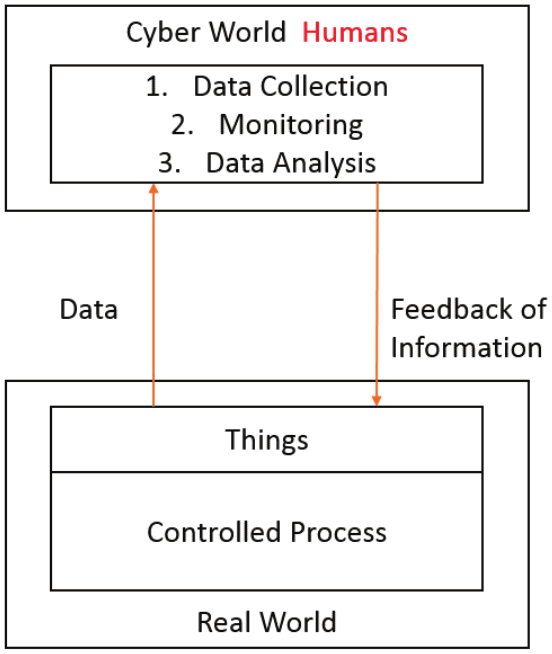


Figure 2: *Control loop for a HITLCPS*

2.1.4 Tracking vital signs using WiFi networks

Tracking human vital signs of breathing and heart rates during sleep is important as it can help to assess the general physical health of a person and provide useful clues for diagnosing possible diseases. Traditional approaches (e.g., Polysomnography (PSG)) are limited to clinic usage. Recent radio frequency (RF) based approaches require specialized devices or dedicated wireless sensors and are only able to track breathing rate. The traditional way to monitor vital signs during sleep requires a patient to perform hospital visits and wear dedicated sensors, which are intrusive and costly. The obtained results may be biased because of the unfamiliar sleeping environments in the hospital. Moreover, it is difficult, if not possible, to run long-term sleep monitoring in clinical settings. Thus, a solution that can provide non-invasive, low-cost, and long-term vital signs monitoring without requiring hospital visits is highly desirable. This solution aims to perform continuous long-term vital signs monitoring at low cost and without the requirement of wearing any sensor. It is possible to track breathing and heart rates during sleep by using off-the-shelf WiFi, exploiting fine-grained channel information, Channel State Information (CSI) [9]. This is the method that was used within our project for the part of the wireless system.

Using channel state information has significant implications on how fine-grained minute movements can be captured for vital signs monitoring. Compared to the traditional RSS, which only provides a single measurement of the power over the whole channel bandwidth, the fine-grained CSI provides both amplitude and phase information for multiple OFDM subcarriers. For instance, mainstream WiFi systems such as 802.11 a/g/n are based on OFDM where the relatively wideband 20MHz channel is partitioned into 52 subcarriers. Due to the frequency diversity of these narrowband subcarriers, the multipath effect and shadow fading at different subcarriers may result in a significant difference in the observed amplitudes. This means that a small movement in the physical environment may lead to a change of CSI at some subcarriers, whereas such change may be smoothed out if we examine the signal strength over the whole channel bandwidth.

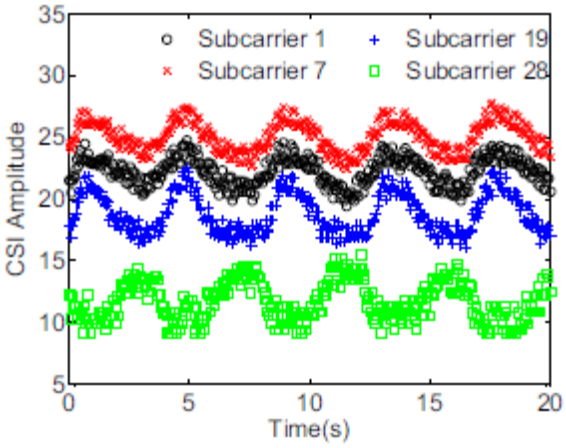


Figure 3: *CSI amplitude of four subcarriers over time when a person is asleep* [9]

Analyzing the CSI at each subcarrier thus provides a great opportunity to capture the minute movements from not only breathing but also heartbeats. Figure 3 shows the CSI amplitude of four subcarriers (i.e., subcarriers 1, 7, 19, and 28) extracted from a laptop in an 802.11n network over time when a person is asleep. His bed is in between an AP and the laptop 3 meters apart. The person does not carry any sensor in his body. We observe that the CSI amplitude of these four subcarriers exhibits an obvious periodic up-and-down trend. Such a pattern could be caused by the person's breathing during sleep. This observation strongly suggests that it is possible to achieve device-free fine-grained vital signs monitoring by leveraging the CSI from off-the-shelf WiFi devices.

2.2 Technologies used

Because the main components of my part of the project are the Xamarin application and the Dashboard web application, the most used technology was Android [10], FIWARE for the communication processes, Git as our version controller, and Plotly Dash.

2.2.1 Android

Android is an Operating System (OS) designed especially for mobile devices. It is based on the Linux Kernel [11] and is designed by Google. This OS is made to run on devices with a touchscreen, like smartphones, and the main interaction with it is through touch gestures it also has a virtual keyboard for text inputs. When the System started it was only designed for Smartphones and Tablets but over the years new versions of the OS appeared, such as the Android TV for Smart TVs, Android Auto for cars, and Android Wear [12] for smartwatches. The last one is used in this project and so it requires further explanation.

Android Wear as the name indicates was meant to be used in items that people wear, but until this day it is only used in Smartwatches. This OS is for small devices, with even smaller screens, and was designed to be used without hands, that is, almost all the features are available using voice command. We can also interact with it through gestures like swipes and clicks and the newer version ofthis OS, Android Wear 2.0, comes with a virtual keyboard for text input. Android Wear was made to work as a complement to the Smartphone; with it, we can see notifications with a glance at the wrist instead of having to draw our phone from our pocket. But these devices are also designed for fitness and sports purposes and are embedded with some sensors that are not available for Smartphones, like the heart rate sensor and the pedometer sensor. This OS is, as stated before, based on Linux Kernel but it also runs a *Java Machine* (JVM) developed especially for these devices. The main programming language used in the development of Android apps is Java, but it is also very common to use languages like C++ and C# with the *Native Development Kit* (NDK).

It is an open-source technology, and it is the most used mobile platform in the world. In addition to it, the Android platform has also a large community of developers and many third-party libraries with good documentation that facilitates the development of apps.

2.2.2 FIWARE

FIWARE is an open-source initiative that works toward building a set of standards to develop Smart applications for different domains such as Smart Cities, Smart Ports, Smart Logistics, Smart Factories, and others. Smart Applications require collecting data from different sources about what is going on that is relevant to the application at any moment, what we refer to as “context information”. Current and historic context information is then processed, visualized, and analyzed at a large scale, thus producing the expected intelligent behavior.

It promotes a standard that describes how to collect, manage and publish context information, and additionally adds certain elements that allow exploiting collected data. Such a standard doesn’t exist today, and it would be instrumental in building a Digital Single Market for Smart Applications where apps/solutions can be ported from one customer to another without major changes. It also solves multiprotocol communication in multisensory networks. It offers a solution to the diversity in IoT protocol and languages and translates the information gathered from the sensors into a common language [13].

FIWARE is based on a library of components called Generic Enablers (GEs) that are meant to implement Application Programming Interface (API)s. GEs offer reusable and commonly shared functions “as a Service". Through APIs, GEs allow developers to put into effect functionalities making programming much easier by combining them.

GEs are classified into seven technical chapters: Cloud Hosting, Data/Context Management, Architecture of Applications/Services Ecosystem and Delivery Framework, Interface to Networks and Devices, Security, Internet of Things Enablement, and Advanced Web-based User Interface.

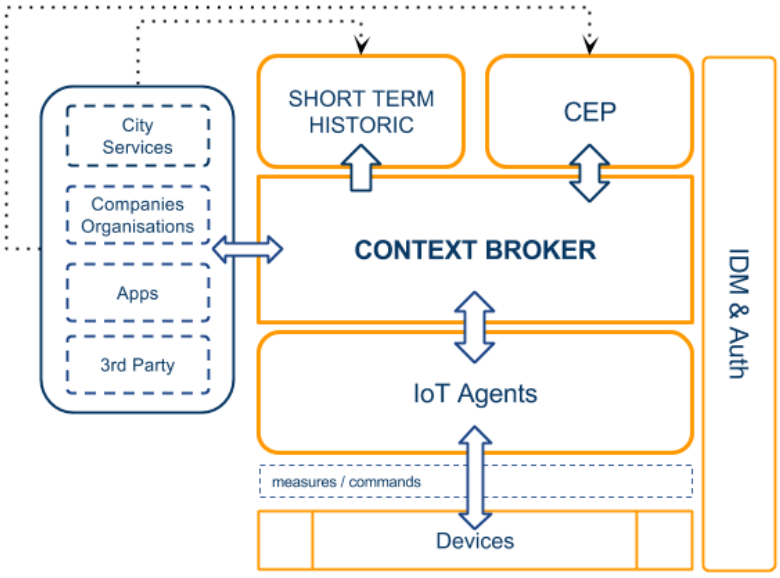


Figure 4: *FIWARE Architecture* [14]

2.2.3 Git - Version Control System

The use of Version control systems (VGS) is one of the best practices for software development. These systems allow us to store all the developed software remotely to ensure the project's continuation and prevent any drawbacks in case of computer malfunction. But this tool's capabilities go beyond the repository functionalities. They also allow us to keep track of all the changes on the project files, letting us know when the files were changed and by whom, and even what section of the file. This helps prevent several errors and save time as it is very easy to make a rollback in the project when something undesirable happens [15]. Other functionalities also include the merging and branching of projects allowing us to keep different versions of the same software, for instance, we can keep a stable version on a branch while working on new functionalities in a different branch of the project.

The version control system used during this project was a local instance of GitLab. Using a local instance, with authentication assured that all the applications and code were secured and private.

In our case, the version control system is also the team management tool. Allowing us to keep track of everyone's work, as well as coordinate the work of the different project members. Creating issues, be it for debugging or for new tasks, allow us to keep track of the work that had to be done.

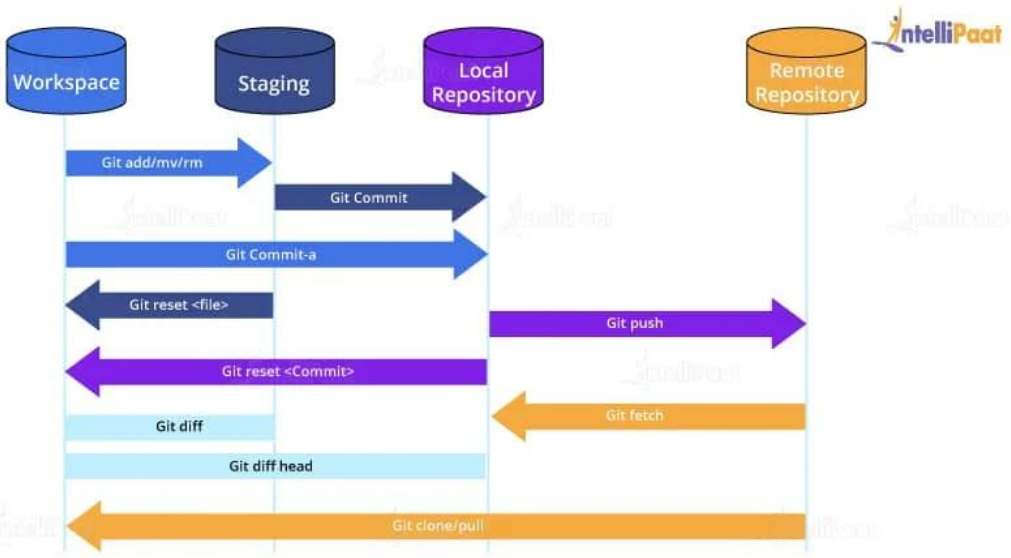


Figure 5: *Git Workflow*

2.2.4 Xamarin

So Xamarin is a developer's tool for cross-platform mobile application development acquired by Microsoft in 2016 and it's built for mobile applications in Android, iOS, and Windows. It has the bindings for all the platform SDKs for Android and iOS; these are easy to use and navigate and provide robust compile-time type checking allowing the development of more error-free and higher-quality applications. The platform provides facilities to apply high-level language libraries directly. Allowing to use of wide arrays of third-party codes, Xamarin has project binding capabilities that let us tie Java libraries or native Objective-C by using declarative syntax. All Xamarin applications are developed in the C#, as it is a modern language that features more dynamic functional constructs like parallel programming, lambdas, LINQ, and more [16].

Like all cross-platform development tools, it eliminates the need to employ additional developers to create apps for other operating systems. In addition, it can reduce maintenance costs as a single team can do the troubleshooting after deployment. This mobile development tool is part of Microsoft's open-source .NET platform. This means that it is free and has strong community support.

2.2.5 Dash

Dash is the original low-code framework for rapidly building data apps in Python, R, Julia, and F# (experimental). Started as a public proof-of-concept on GitHub 2 years ago, written on top of Plotly.js and React.js, it is ideal for creating analytical web applications with emphasis on data analytics, data exploration, visualization, modeling, instrument control, and reporting. Through a couple of simple patterns, abstracts away all the technologies and protocols that are required to build a full-stack web app with interactive data visualization [17].

Dash apps are rendered in the web browser. They can be deployed to VMs or Kubernetes clusters and then share through URLs since they're viewed in the web browser. The web servers run on Flask and communicate via JSON packets over HTTP requests. Flask is widely adopted by the Python community and deployed in production environments everywhere. Dash components are Python classes that encode the properties and values of a specific React component and that serialize as JSON. This toolset uses dynamic programming to automatically generate standard Python classes from annotated React prototypes. The components are user-friendly as they come with automatic argument validation, docstrings, and more. Since the Dash application is stored in the frontend (web browser) it allows it to be used in a multitenant setting where multiple users can have independent sessions while interacting with the Dash app at the same time.

3 - System Overview

In this chapter, we present the system in an overall way. In the first section, we present the contextualization and description of the project, and in the second section, the architecture of the system and its components separately. In the last section, we present the requirements for this part of the project.

3.1 Contextualization

The iFriend has as its main objective to monitor, assist and improve the health and daily life of elder people. With this solution, it is possible to better understand the day-to-day of elderly people and try to prevent some bad habits that may lead to worsening their health condition of them or prevent some emergencies.

Currently, older people don't have regular and constant monitoring or check-up on their healthcare mostly because of misinformation, access difficulties, or lack of technology that can help track them. The increased risk of health diseases associated with natural aging, health behaviors, social isolation, or side effects from medications, are some factors that can contribute to the risk of cardiac arrest or respiratory problems in this type of population. By controlling these factors by monitoring them, it would be possible to achieve health improvement and a better understanding of them. In this application, a HITLCPS architecture is implemented, where seniors are included in it, making them the first actuators to change their behavior. It is also important that seniors are willing to participate in the process and improve with it because in many cases, the main problem is to change some of the behavioral habits that they have, that is not the most suitable for a healthy life.

3.2 Description of the project

iFriend is a HITLCPS system that uses 3 sets of devices, to monitor the vital signs (heart rate and breathing rate) and the daily life of elderly people.

This set of devices, described in detail in section 3.3, is composed of the monitorization through a WiFi network (*Channel State Information* using both time and frequency) between a router and an AP (which will be settled in each senior's division) and the breathing rate validated with a Go Direct respiration belt (with a response time of 50 ms, maximum chest circumference of 140 cm, with a wireless connection through Bluetooth or wired connection through USB cable) [18]. It is also composed of a mobile application to monitor the elder's daily activity and help them track their habits and a smartwatch application that will help to validate the heart rate measured by the WiFi monitoring system. The main goal is to improve the health of the elderly, facilitate the work for the health responsible and improve their access to better health services.

This system monitors the seniors continuously, 24 hours a day, to gather the best data and analyses, and deliver them to health responsible. The CSI's data, the smartphone, and the smartwatch sensor's data are retrieved and sent to the FIWARE in real-time or when a connection to the Internet is available. After the data is retrieved from the FIWARE with specific time intervals is processed and shown in the Dashboard to the doctor or health responsible. As said earlier this overall monitorization system will keep track of the vital signs but it will as well help the seniors improve their health habits through the mobile application that has an interactive display for them to interact and keep track of their habits and health routine. The simple smartwatch application and the Dashboard will be explained in more detail in Chapter 4., as well as the processing of the information, what data is being retrieved, and how we made data acquisition.

3.3 General Architecture

The general architecture of the overall system is shown in figure 5. It is characterized by all the mechanisms that allow the structure to integrate the system HITLCPS as well as the acquisition and processing of the data by the WiFi monitorization devices, respiration belt, smartphone and smartwatch, and finally, all the information collected is shown on the dashboard web application.

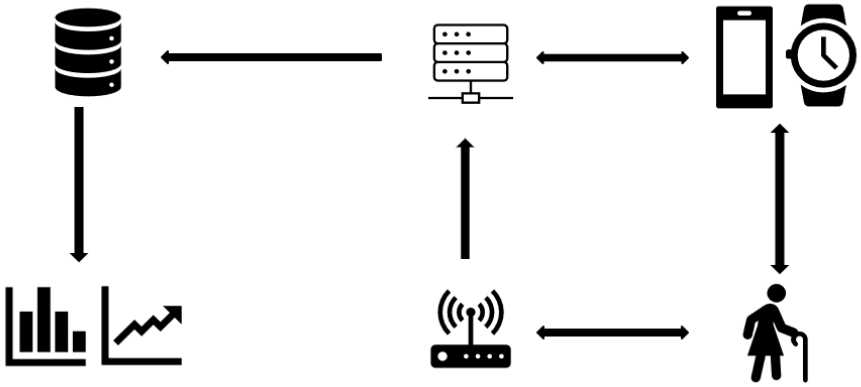


Figure 6: Overall architecture of the system

Firstly, this system has the vital sign WiFi monitorization through the Channel State Information (both time and frequency) collected by the connection between an Access Point and a Router using an RF link.

The basic idea of this system is to track vital signs during sleep by capturing the unique patterns embedded in WiFi signals. As illustrated in Figure 7, the system takes as input time-series CSI amplitude measurements, which can be collected at an off-the-shelf WiFi device by utilizing existing WiFi traffic or system-generated periodic traffic (if network traffic is insufficient) during people's sleep. The data is then processed to filter out the CSI measurements that contain sleep events (e.g., going to bed and turn over) or large environmental changes such as people walking by via *Coarse Sleep Event Detection and Filtering*. The measurements belonging to the regular sleep events can be further classified into detailed events such as going to bed, getting off bed, and turnovers. Moreover, our work is based on the fact that the breathing and heart rates of resting people have different frequency ranges (e.g., breathing rate ranges from 10 to 37 bpm, and heart rate ranges from 60 to 80 bpm). This useful information leads us to work on different frequency bands of the CSI measurements for accurate vital signs estimation.

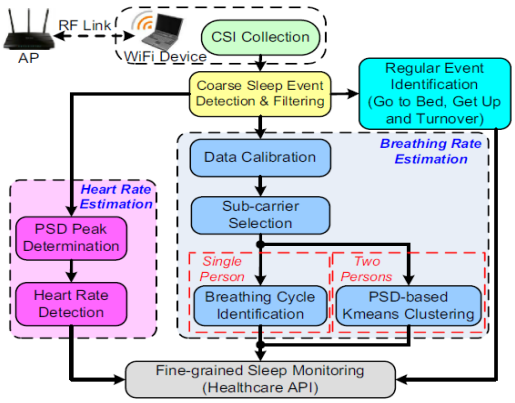


Figure 7: *Vital Sign WiFi monitorization* [9]

The core components of our system are *Breathing Rate Estimation* and *Heart Rate Estimation*. After coarse sleep event detection and data filtering, based on the different frequency information embedded inside the CSI measurements, the input is fed into *Breathing Rate Estimation* and *Heart Rate Estimation* respectively. In particular, the lower frequency information of the CSI measurements is processed by the *Breathing Rate Estimation* component. Our system first performs *Data Calibration* and *Subcarrier Selection* to preprocess the data and select only the subcarriers sensitive to minute human body movements (i.e., subcarriers with large variances). We then develop two methods, Breathing Cycle and PSD-based K-means Clustering, to estimate the breathing rate for single and two-person in-bed scenarios respectively. PSD denotes power spectral density. Following a similar principle, PSD-based K-means Clustering can be easily extended to handle the case of estimating breathing rates for multiple people simultaneously given the number of people under study is known. The higher-frequency information of the CSI measurements is fed into the Heart Rate Estimation component. The heart rate is then derived in the frequency domain by examining the peaks in power spectral density (PSD) of CSI measurements.

This is our most important part of monitorization because we are using innovative technology, as we can see from the explanation above. Which is being developed by my colleague Guilherme Lemos.

Secondly, we have a smartphone application that will allow the senior to follow his daily habits like medication, meals, or medical issues, and track his sleep routine or activities. This application is being developed by my colleague José Ramos and it will have a smartwatch as a companion application which is explained in the 3.3.2 section.

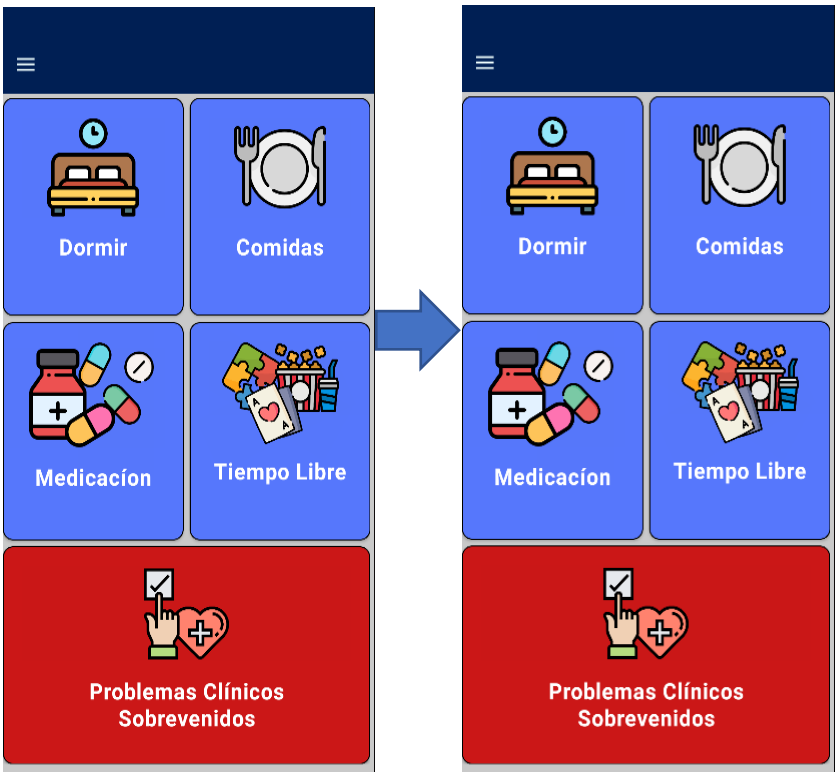


Figure 8: View of the smartphone application main page

This system uses FIWARE as a cloud-based module that implements the storage capabilities and allows communication between all the devices, the smartphone is also where all states are inferred and where the actuation is made through notifications and messages. The smartwatch is the only device that doesn't communicate directly with FIWARE, because of its architecture. This model is based on entities and attributes. Each entity has its type and it is represented by the attributes, using the JSON format.

3.3.1 FIWARE Architecture

The FIWARE is our back-end component that implements the storage and communications capabilities normally seen in mobile applications systems. In this implementation there are used 4 General Enablers: ORION, CYGNUS, KEYROCK, and COMET, with different functions [19]:

- **ORION** is a context broker that allows to create of context. More precisely it allows to create of virtual entities to represent objects in the real world or even people. These entities are like classes: they have a type, a specific *id,* and attributes. However, unlike classes these entities can have different attributes even if they are of the same type; for example, in our case study, some entities of the type WearData can have the attribute heart rate while others may not have it, depending on if the senior in the case has a smartwatch or not. This is of course a must-have capability in an IoT architecture where we want to create a connection between the sensors and the applications that consume the information. For this purpose, this module implements a Representational State Transfer (REST) API, which allows us to create, update and delete entities or attributes. Although this module allows us to create entities and save their attributes, it is only able to store the last instance of that data. That is if we update an attribute value, the module only retains the last value. To save data for historic context we need to implement other modules such as the CYGNUS and the COMET.

- **CYGNUS** module is responsible for coordinating the storage of the data. That is, in this module we can create subscriptions from the entities to a specific third-party storage system, such as MongoDB [20] or MySQL [21], creating this way a historic view of that data. These subscriptions are made by type, and after the subscription is made, for a specific type, whenever an entity of that type is created, updated, or altered the CYGNUS automatically saves those changes to the third-party database to which the subscription was made.

- **KEYROCK** is an authentication module that manages all the other module's accesses, which is by creating accounts we can restrict the access of certain information and functionalities to certain accounts. This allows us to solve several issues with users' access to networks, applications, and services, also this way we can secure the data and assure privacy. This module implements a single sign-on service, which is the user’s credentials (email and password), these are the same in all modules and are hosted inside of this module. When the user signs, he is given an Access Token that is then used in the requests he makes to

the remain modules. Those modules query then the KEYROCK, to validate the user request, before allowing the user to change or access any data.
This is a very important module since the privacy of the data is one of the main requirements of this project and with this module, we can prevent personal data from leaking to third parties, or even from being accessed by users who shouldn't have access to them.

- **COMET** module is a short-time history, in charge of managing the historical context information, storing and retrieving it. This module implements REST APIs to communicate with the ORION and with third parties. The API used to communicate with external applications was developed to facilitate the retrieving of data with time context, it allows us to aggregate the data by time or even query specific time intervals. The API also allows us to aggregate data with sums or by occurrence (to discrete values, like strings). This module is from where we get most of the data in the application, and as such that makes it a very important component to keep the platform working. This module also addresses the requirements for storing information in the project as well as the requirements to have information contextualized in time.

3.3.2 Wear App Architecture

The architecture of the iFriendWear application, which I developed, is like a mobile app that has a thread to handle the graphic display and background services to handle the more complex tasks. From figure 8, we can see that the architecture is more minimal with just two background services. This architecture allows us to have an application where the display is not affected by the tasks that are running in the background. The Main service takes care of the sensor's and context's information acquisition, and the Wear Service handles the communication with the smartphone. It was also necessary to have a database on the smartwatch to ensure that no data is lost before being sent to the Smartphone. The background thread handles the more complicated tasks like business logic, HTTP communication, data storage, or even repetitive tasks like sensor data retrieval.

As the main goal of this app is to help with the monitorization of the vital signals and the activity of the seniors, the frontend of this app is very simple and just displays the last heart rate taken and the number of steps taken during that day. On the backend, it has a background service that is constantly running while the app is open that monitors the heart rate and the steps, and saves the data on the local database.

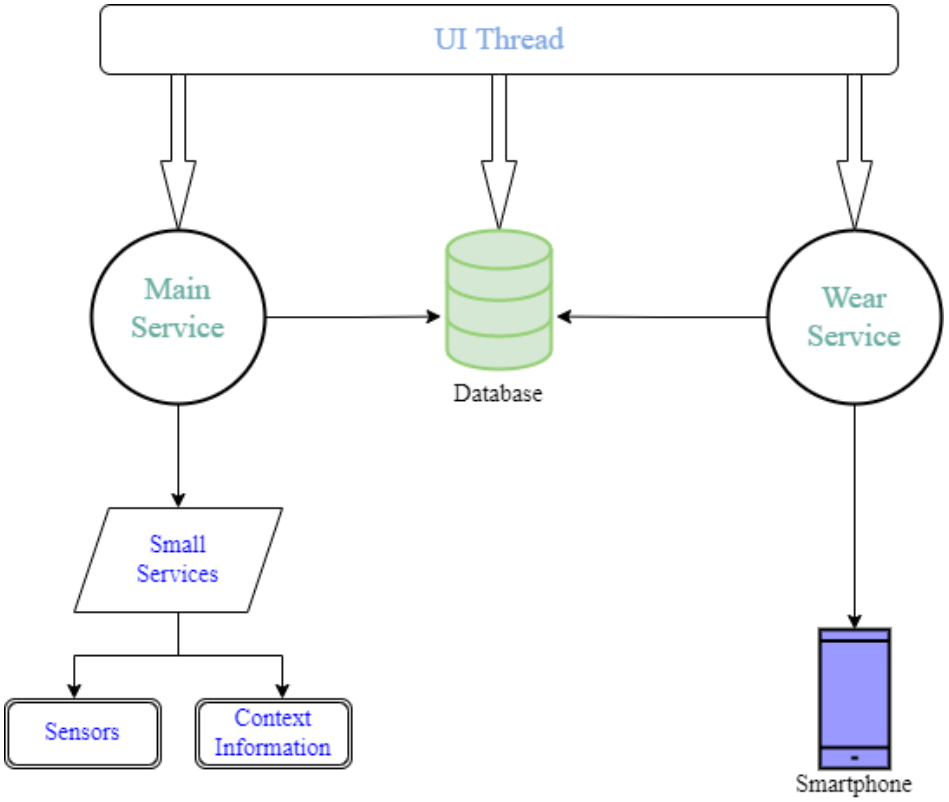


Figure 9: Wear Application Architecture

3.3.3 Dash Architecture

The architecture of the Dashboard is based on the usual Dash Plotly architecture that has minimal interaction between the modules and extensibility with each module, where each one has a specific functionality: web-based client; visualization management, and data analysis modules. This architecture permits the optimization of each component independently by separate groups with appropriate skill sets resulting in a flexible, extensible, and efficient dashboard. The need for a close synchronization between the client module and the back-end visualization management module. For this to work correctly it has a web framework than can support the modules, seamless communication, ease of use, and a strong open-source community for future development and extensions. The python-based web framework Flask is minimalistic, interactive, flexible, and extensible. Plus, visualization tools like Plotly are supported exhaustively by Flask. It gives maximum flexibility and control due to granular tuning for customization and makes no assumptions about how data is stored, thus becoming a viable choice for a wider spectrum of applications [22].

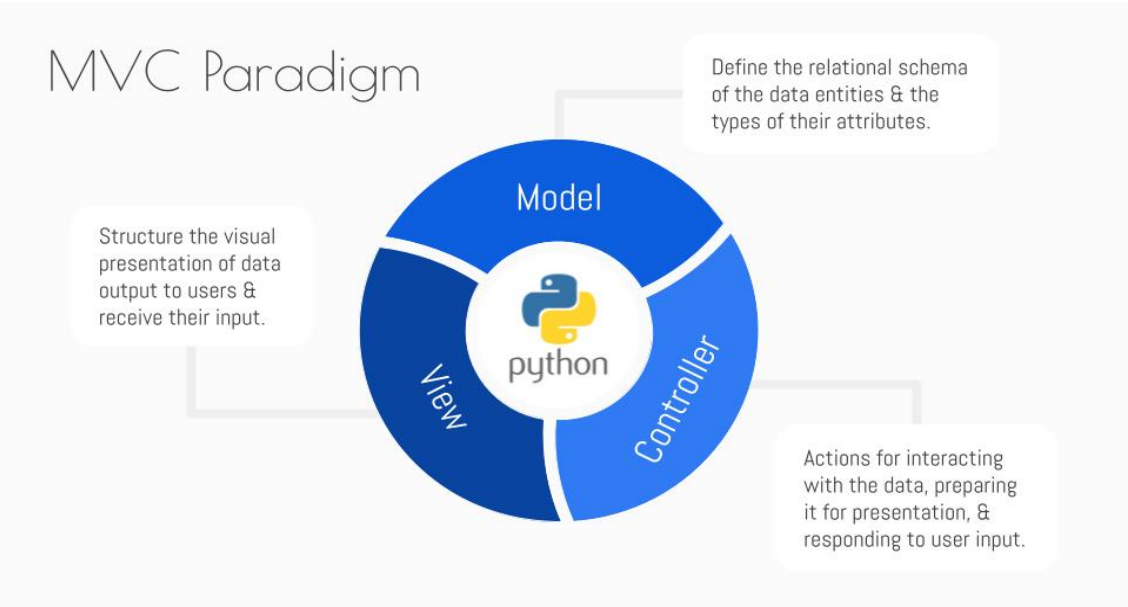


Figure 10: *Dash overall architecture* [22]

1. View: The main requisite of this dashboard is to present all the information retrieved about the patient to the health professional. This client module only has display components and barely any user interaction. It's a multi-page app in which all the components are reusable. It has a hidable sidebar that is independent of the main view that loads the pages that are mainly divided into 2 tabs: one for "Live Metrics" and the other for "Historical Record". These components are very important for a clear view of the patient's information, so it must be also easy to identify all the different aspects of the medical record. For this, there are presented some simple graphics for every variable that is being measured in the patient's daily life through the various devices used in this project. The various elements of this module are supported using HTML and CSS.

2. Model: Functionally, this is an important module, which handles several tasks: visualization generation – using either base data or computed results – from the analysis module or from directly the 2 servers that were designed to expose two APIs that return the data through the querying requests. These APIs establish a connection with two main databases (MongoDB and MySQL) that have all the data sent from all the devices stored.

3. Controller: This module is responsible for the analysis of some data that is being gathered by the WiFi monitorization through the *Channel State Information* like heart rate and breathing rate and needs to be filtered and smoothed to show this information in the most readable and clearable way to the doctor.

3.4 Requirements

In this section, we present wear and dashboard applications, with functional and non-functional requirements.

Parameters	Parameter's Description
Priority	**Must:** the requirement is essential to the project. **Should:** the requirement is important to the project, but the system should work without it. **Could:** not implement requirements that won't affect the implementation of the important requirements. **Will:** not implemented requirements for the project, considered as future work.
Description	Overall exposure of the requirement.
Actors	Systems that intervene with the application.
Pre-Conditions	Conditions necessary to the requirement functionality.
Events Flow	Description of the actions that an actor needs to fulfill to get the expected outcome.
Expected Outcome	The expected result from the action.

Table 1: *Requirement's parameters*

3.4.1 Functional Requirements

3.4.1.1 Wear Application

- **Intro**

- **Priority:** Should.
- **Description:** The application should present an intro with a reference to the University of Coimbra.
- **Actors:** The user.

- **Pre-Conditions:** None.
- **Event-flow:** The user clicks the application's icon from the smartwatch's menu, and the application starts and shows the Main screen.
- **Expected outcome:** The screen shows a new activity with the text "Made by the University of Coimbra". The screen then changes to the Main screen after 5 seconds.

➢ **Sensors**

- **Priority:** Should.
- **Description:** The application must collect all the information needed from the sensors and process it to present it to the user.
- **Actors:** The user.
- **Pre-Conditions:** The user must give all the requested permissions.
- **Event-flow:** The user either clicks accepting or not accepting options.
- **Expected outcome:** If the user accepts to give the permissions, then it's presented on the Main screen. If the user doesn't accept to give the permissions the application closes.

➢ **Store Information**

- **Priority:** Must.
- **Description:** The application must be able to store information until a connection with the respective smartphone is available.
- **Actors:** None.
- **Pre-Conditions:** None.
- **Event-flow:** When the information is gathered from the sensors it is stored on a local database on the smartwatch and remains there until it is sent to the Smartphone.
- **Expected outcome:** If there's no connection available, the application stores the sensor's information on a database in the smartwatch. When the information is sent to the smartphone whether, via Bluetooth or WiFi, the information that has been sent is deleted from the smartwatch's database.

➢ **Main Screen**

- **Priority:** Must.
- **Description:** The application must be able to show the information updated and with accuracy.
- **Actors:** None.

- **Pre-Conditions:** None.
- **Event-flow:** After the user gives the right permissions, the information is gathered from the sensors, and it is presented to the user's screen.
- **Expected outcome:** If there's information being collected, it's presented on the screen with the last heart rate measured and the number of steps taken that day. If there's no information being retrieved from the sensors it is presented the last heart rate and the number of steps is stored in the database.

3.4.1.2 Dashboard Application

- **Main Screen**

- **Priority:** Must.
- **Description:** The application must be able to present a responsive/hidable sidebar with multi-page options and the respective tab information.
- **Actors:** None.
- **Pre-Conditions:** Must have Internet access.
- **Event-flow:** After the user access the respective URL of the Medical Dashboard, it must present with the Main Screen.
- **Expected outcome:** The screen shows all the information depending on which sidebar option and tab are selected by the user. It must show all the information on the screen according to the language selected.

- **Sidebar Options**

- **Priority:** Must.
- **Description:** The application must be able to show the information depending on which option the user selected.
- **Actors:** User.
- **Pre-Conditions:** None.
- **Event-flow:** After the user selects the option on the sidebar, it's presented with a new page depending on the option that contains 2 tabs.
- **Expected outcome:** When the user chooses one option on the sidebar, it should refresh the screen except for the sidebar and present the tabs and information corresponding to the option selected.

- **Live Metrics Option**

- **Priority:** Must.
- **Description:** The application must be able to refresh the page and show the information regarding the patient.

- **Actors:** User.
- **Pre-Conditions:** Live option is selected, the patient is selected, the tab is selected, and the measure button is selected on the corresponding tab.
- **Event-flow:** After the user chooses the Live option on the sidebar, the patient, and the tab, should be presented with the live information regarding these options.
- **Expected outcome:** If the "Vital Sign Estimation" tab is selected and after the button is pressed, it should present live metrics of the heart rate and breathing rate of the patient, which are being measured by the WiFi Signal (CSI) and respiratory belt. If the "Daily Information" tab is selected and the "Measure" button is pressed, it should be presented the last Location, Activity, Heart Rate, Number of steps taken, and the clinical problems during the last day, that were measured by the smartphone and smartwatch applications.

➢ **Historical Record Option**

- **Priority:** Must.
- **Description:** The application must be able to refresh the page and show the information regarding the patient.
- **Actors:** User.
- **Pre-Conditions:** Historic option selected, the patient selected, tab selected, and date selected on the corresponding tab.
- **Event-flow:** After the user chooses the Historic option on the sidebar, the patient, the tab, and the date, should be presented with the historical record of the patient for that specific date.
- **Expected outcome:** If the "Vital Sign Estimation" tab is selected and after the interval date is chosen, it should present the historical graphics of the heart rate and breathing rate of the patient, those vital signs were measured by the WiFi Signal (CSI) and respiratory belt. If the "Daily Information" tab is selected and the date is selected, it should be presented the pie graphics for Activity, Location, and Number of Steps (smartphone) during the day selected, a line graphic with heart rate measured during that day, and the number of steps (smartwatch), a sleep graphic, information of Meals, Medication and how free time was spent, and, finally, the clinical problems that the patient had during that day.

3.4.2 Non-Functional Requirements

These requirements are common to most smartwatch applications and dashboard applications. Most of them can be solved by implementing good design patterns. The non-functional requirements are very important in the development of an

application because sometimes the fact of missing one of these requirements is enough for the user to delete or not use the application anymore.

- **Accessibility:** Determines if all users can utilize the application, this means that the application should not suffer from limitations to the number of connected users.
- **Performance:** Shouldn't affect the battery life too much, the memory or the internet connection of the smartwatch, or the browser used.
- **Privacy:** All data collected must remain private and secure.
- **Support:** Should be supported by most of the Android devices or main browsers currently on the market.
- **Usability:** The application must have a pleasing UI and features that are important to the users, and more importantly should be easy to use.
- **Functionality:** The most important function of any app is the user's ability to navigate it. Users understand how to use them intuitively, and most menu items are always with a tapping of the thumb or mouse.

Accessibility is a requirement of the system and not of the wear application. The FIWARE itself handles accessibility. The good design of the FIWARE ensures that the system is always accessible independently of the number of connect users. Plus, the Flask server allows the Dashboard to handle thousands of requests for dynamic content at once.

The **performance** is handled using background services to perform the more complex tasks as was explained in section 3.3. By implementing background services, we are freeing the UI thread and ensuring a more responsive application.

The **privacy** of the data collected from the application is ensured by the offline wear database. Furthermore, the data is only stored on the smartwatch for short periods. After is sent to the smartphone, the data is encrypted before being sent to MongoDB through the FIWARE system.

Our application is **supported** by all devices with an Android version greater than 4.0.3. To ensure this compatibility is necessary to use some older APIs and classes that Android has deprecated.

Usability of the application is also a big requirement nowadays because users are becoming more demanding about the aesthetics of the applications, and it needs to be easy to use and understand for the health responsible. We took that into account when we were developing the application and tried to implement an interesting and usable UI, especially on the dashboard application. Some examples of that are the intuitive graphics, the navigation sidebar and tabs, and the theme of the application.

4 – System Development

In this chapter, we explain the development of my part in the system. We start by explaining the smartwatch development followed by the dashboard application development.

4.1 Smartwatch Application

The smartwatch application serves as a companion for the mobile application and as a way of validating the data the is being collected by the smartphone and the WiFi devices. In this section, we explain in detail the development of the wear application and we can divide the development of the application into 4 sections: the acquisition, the communication, the storage, and the display user interface of the application.

4.1.1 Data Acquisition

The smartwatch application only makes the acquisition of 2 values from the sensors:

- Heart Rate
- Step Count

To make this acquisition we created a service called *ValuesCollector* that is constantly running in the background after the user opens the application. This service implements the Handler class and the *ISensorEventListener* interface that allows us to know when a value of a sensor is changed, and the accuracy of the samples collected by it. By using the *SensorManager* class we can register this listener to the desired sensors; in our case the heart rate sensor and the step count sensor. We have a listener that is triggered every time a registered sensor or its accuracy is changed. This listener saves the sensor's updated value on a variable and the last value of those 20 seconds of collecting values is stored in a local smartwatch's database. This was made because the sensors take a certain amount of time to adjust and be at their highest accuracy. So, following this, we set the values collected to only accept the highest accuracy readings on a scale of 3 qualities(Low, Medium, and High)

2 timer functions allow us to only have the sensors registered for 20 seconds, which means that there's only going to be collected data from the sensors during those

20 seconds and after those, we unregister the sensors, and another timer to set the interval of 2 minutes that determines the time that the sensor will be turned on again. This was set after a sequence of battery tests that are going to be covered in chapter 5.

The physical sensors are sensors that are implemented in the smartwatch's hardware. To obtain the values of the smartwatch's sensor the Android OS provides an API. This API has classes dedicated to managing the sensor acquisition, as referred to before the *SensorManager* class. This class allows us to retrieve the values of all the sensors of the smartwatch. The acquisition follows the following steps:

1. Get the sensor manager instance from the application context, through the method *GetSystemService()*.

2. Get the required sensor from the sensor manager through the *GetDefaultSensor()* and refer to the parameter for the specific type of sensor.

3. Use the *SensorEvent* on *OnSensorChanged()* default method of the *Service Handler* where it is all the code that needs to run when that sensor value changes.

These steps need to be replicated in every sensor we want to use. As was stated before these *SensorEvents* are triggered by the change of a sensor value. This event returns an object from the *SensorEvent* class. It has four attributes that can be retrieved - which are: accuracy, sensor type, timestamp, and the value from the sensor. Using these four attributes we can gather a lot of information that allows us to build the most robust data possible.

4.1.2 Communication

The intended architecture for the Android wear does not allow us to connect the smartwatch directly to the Internet. This happens because the device manages the WiFi and Bluetooth connections automatically, to improve battery life. Once the device is connected to the smartphone by Bluetooth, it disconnects the WiFi to save the battery, unless it is charging. This of course would not allow us to send data when the smartwatch and the smartphone are close to each other (most of the time).

For this reason, we must use *Android Wear Message* API. To implement it on the smartwatch we use a timer inside the class *ValuesCollector* that reads, every 5 minutes, the database and tries to send the data to an available device. To send the data, we have a class called *MessageSender* in which we start a new service, which extends the *IntentService* class. This service uses the *Capabilities* API to make a scan for nodes (devices connected to the smartwatch, normally with Bluetooth) that have the desired capability. Once the API finds one, the service starts sending the saved instances of data

to the smartphone. When an instance of the data is sent to the smartphone it is also deleted from the smartwatch's DB. Once the smartphone receives the data it is stored on the smartphone's wear database and is ready to be sent to the MongoDB database.

4.1.3 Storage

We have two databases on the smartWatch with two models related to the data that is being collected: the *WearData* model and *AccuracyData*. Each database has its correspondent model:

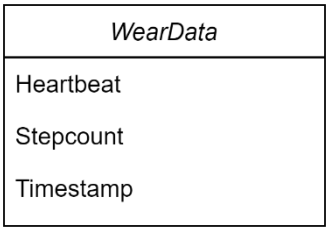


Figure 11: *WearData model for the Wear database*

As can be seen from figure 10 we save the heart rate and the step count on the *WearData* model. We also save the timestamp, because we need to record when these values were collected to help with the view of the graphics on the dashboard application and to ensure that we can relate the wear data with smartphone data.

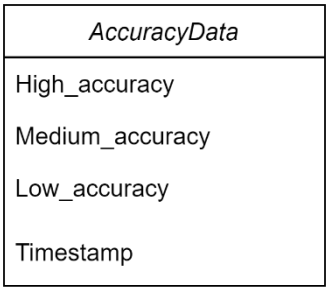


Figure 12: *AccuracyData model for the Accuracy database*

The *AccuracyData* model was designed especially for the wrist detector and accuracy tests, which are directly connected to the battery tests. For this, we save the accuracy of each heart rate value which is only stored if the accuracy of the sample collected is in the *Medium* or *High* category, and the time at each value was collected.

4.1.3 Display

The display of the smartwatch application is very simple for two main reasons. The first one is because this is going to be worn by people that don't use or have knowledge of technologies; and the second one is because this application was developed and designed for testing reasons, as stated before to validate the data collected by the other devices used in this project. The display on the watch is small and for that, we can't display a lot of information. So, for that, we opted for a simple layout where we can see from figure 12, the name of the application and the 2 sensors in real-time, with a symbol representing each of the two values and the actual value above it.

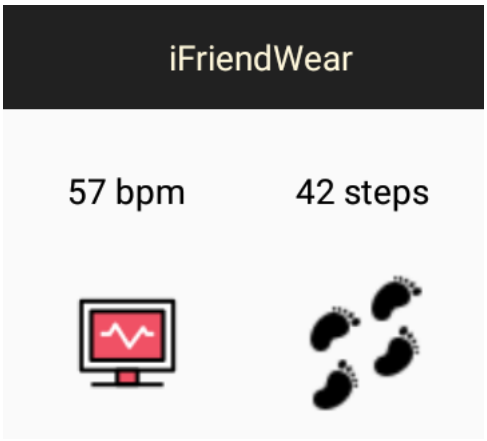


Figure 13: *Wear app screen*

4.2 Dash Application

In this section, the dashboard's development details are explained, including 3 sections: Callbacks and Data Processing, Communication, and Layout/Dashboard.

4.2.1 Callbacks and Data Processing

Callbacks are functions that are automatically called by dash whenever an input component's property changes, to update some property in another component, the output. These callbacks were divided into two groups:

- HrBrCallbacks are specifically designed to set the callbacks for the graphics that contain the information (heart rate and breath rate) from the respiration belt and the WiFi devices. In this section, Callbacks were developed first with the processing of the data that is being collected by

the devices referred to before and then to construct the layout for the graphics of the two variables stated before.

- AppDataCallbacks were developed to set the callbacks for all the other graphics and information on the dashboard that doesn´t require any data processing, like the heart rate graphic of the wear application or the activity recorded by the smartphone.

The data processing was made on the *UtilsHr* where the vital signs estimation is done, only for the information collected by the WiFi devices:

1. Firstly, the CSI information from the WiFi devices is retrieved from the server “vital_wifi_server” through a GET request;
2. The data retrieved will get through a set of data processing procedures like smooth, identify the sub-carrier of the CSI wave, remove the fake peaks and then filter, to be presented with clearness on the graphics, where both heart rate and breath rate are presented.

As for the respiration belt, the only variable that is being measured live (breath rate) is directly sent and fetched from the same server thought a different GET request, and no data processing is needed because we already have the values and related times stored there.

4.2.2 Communication

Regarding communication, it is made by queries that go inside HTTP requests, which relate to two microservices (servers) with just the function of exposing 2 APIs that establish the connection with the two main databases (MongoDB and MySQL). So, the connection isn’t directly made with the databases, which allows the APIs to facilitate the retrieving of data with time context, which permits us to aggregate the data by time or even query specific intervals. It also addresses the requirements for storing information in the project as well as the requirements to have information contextualized in time, which is crucial for the monitoring of senior people.

These two servers are divided into one that has the function to get the information from the MySQL database with the CSI data and respiration rate which is being collected by the WiFi devices and the respiration belt, respectively, and the other server is responsible for getting all the other data which includes the smartphone and smartwatch’s data.

The second server is part of the COMET module which implements a REST API to communicate with the ORION and with third parties (like the iFriend application), composing a FIWARE Ecosystem with several microservices.

These requests are made on the requestUtils class and most of them are GET requests with the user ID as the main and common parameter. For the retrieval of the data, we organize it by the time that was acquired to make a sequential graphic which is called inside the respective *Callback* and presented on the app layout.

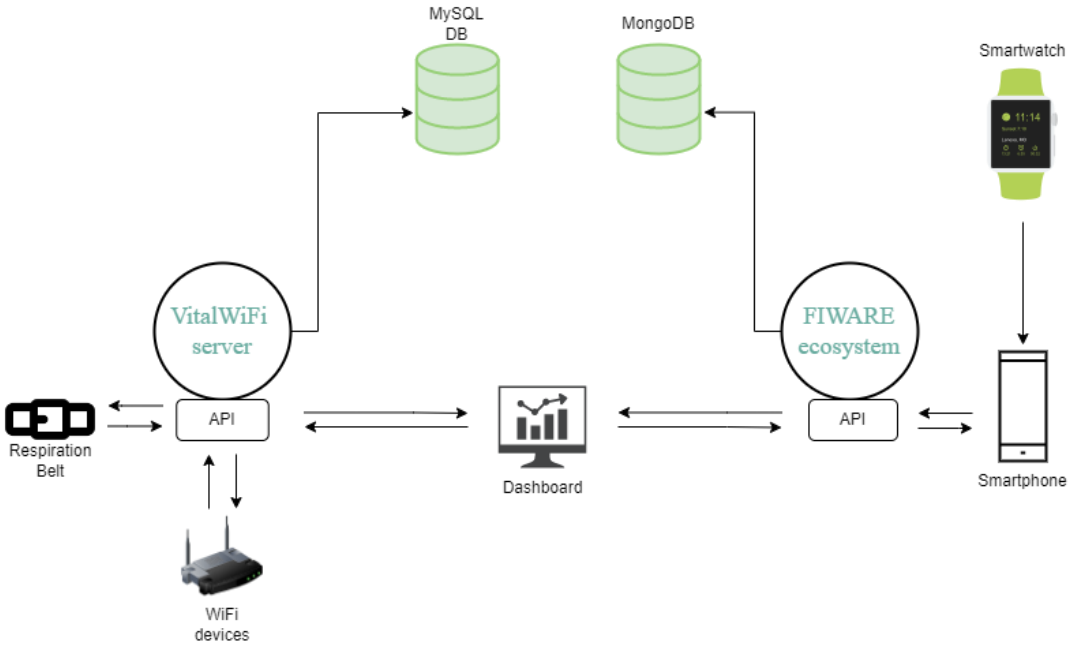


Figure 14: *Dash Communication Architecture*

It is important to mention that all the data that is being measured is sent to the two main databases and then retrieved through the APIs. The live metrics are not exactly real-time due to the dependency of having a WiFi connection and the delay of the data transfer.

4.2.3 Layout

The dashboard layout was designed through the conditions of satisfaction that Madrid's hospital provided us and to have a simple and clear visual for the health responsible to consult plus with all the data of the patient right in front of him/her. The dashboard is composed of the principal elements which are the sidebar and the tabs:

- Sidebars give users quality, valuable information that might not otherwise fit on the page or with the content. It adds a call-to-action button above the fold and leads to relevant pages easily. The sidebar can be presented or collapsed with a button to increase the view of the page. It has the selection of the patient, the three options of navigation links plus the list to choose the language of the page:

- The select dropdown allows the user to choose between a list of all patients that use the project devices;
- The three navigation options are: “Live”, “History”, and “About” (which contains information and a description about the project and its sponsors), each of these navigation links is a newly loaded page with the respective content;
- Finally, the select dropdown with the list of languages that the website supports which are English and Spanish.

- Tabs are on the loaded pages of the options “Live” and “History” and are divided into “Vital Sign Estimation” and “Daily Information”. These layout elements are the most common in dashboards and allow the user to travel with ease between diverse types of information which turns the experience of navigating and consulting the dashboard much better and smoother. It acts as the highest-order sections or categories, and they are powerfully broad so the user can gradually channel into the user interface to access more specific content without having to scan all the available contents that the page has to offer at once.

4.2.2.1 Live

As mentioned before, the “Live Page” is divided into two tabs, “Vital Sign Estimation” and “Daily Information”:

- The “Vital Sign Estimation” is dedicated to the almost “real-time” collected data from the WiFi devices and the respiration belt. So, in practice, the patient is wearing the respiration belt and the WiFi devices are installed in the evaluation room, so the data can be gathered and automatically be seen by the doctor on his screen:

Figure 15: *View of the live graphic of the heart rate measured by the WiFi devices*

As we can see, we have each point of the heart rate collected on the CSI of the WiFi devices and the time that it was measured. Because this is a live graphic, the user can start and stop the measure whenever he/she desires, and the graphic will increase/construct itself if the measure is being done. At the same time, the average heart rate is presented below the graphic.

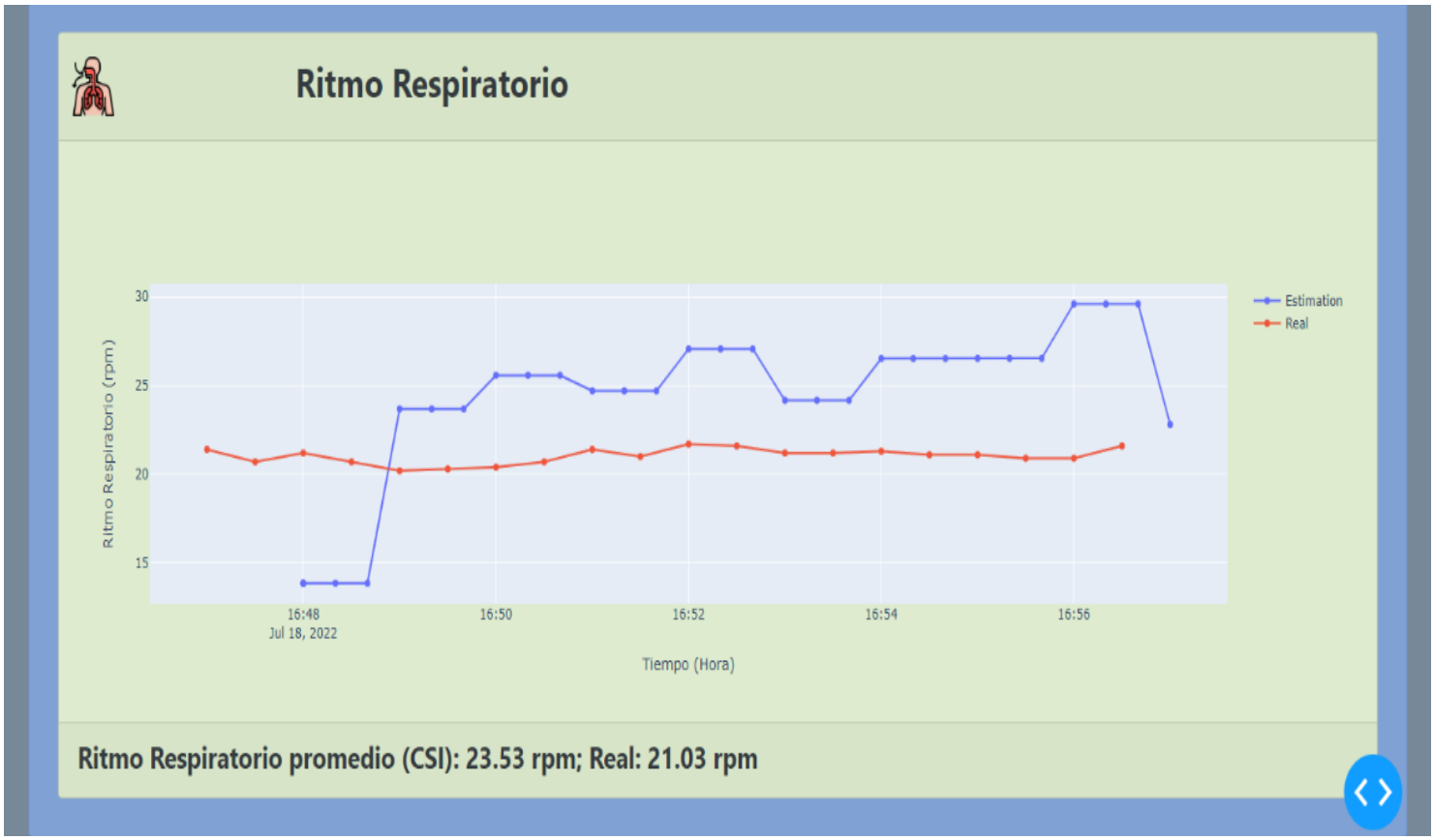


Figure 16: *Live graphic of the breathing rate measured by the WiFi devices and the respiration belt*

This graphic shows the breath rate collected by the information on the CSI of the WiFi devices and the respiration belt. As we can see on the right side of the graphic it is the identification of the two types of measuring: “estimation” (blue line) made by the WiFi monitoring system and the “real” (red line) value validated by the respiration belt. Below the graphic is the average breath rate measured during the time interval for each type of device.

- The Daily Information tab is dedicated to the “almost live” information about the smart device's data, where we gather the most vital information (activity, location, heart rate, number of steps taken so far) and present the last value taken of each one those elements:

Figure 17: *Live Estimation Tab with smart devices crucial information*

4.2.2.2 History

For this page option, there is the same choice of tabs because the options pages are related to each other in terms of the type of data they treat and present. Just to differentiate from the Live page, we decided to design a different background color for each page to distinguish better which page the user is navigating.

This page was designed to present to the user the global information of the patient, with that he can evaluate the patient’s health and get an overview of his medical record. This allows the doctor to have full information about the patient even before he goes to the medical appointment:

- The “Vital Sign Estimation” on the “History” option is the tab where we can see the data related to the gathering of data that the WiFi devices and the respiration belt measured during a specific time. This time can be chosen with a date and an interval of time. After that, there are presented two sets of graphics, one for the heart rate and the other for the breath rate:

Figure 18: *Heart Rate graphics of the two types of devices*

In the figure above, we can see the Interval of time chosen by the user and the graphic of the heart rate measured by the WiFi devices (blue line) and the smartwatch (gray line), with the respective averages below. In theory, the two graphics should overlap each other because the values measured should be the same at each point. But as we can see they are not overlapped, as they differ by about 11 bpm at each point. This happens because there is a difference in the accuracy of the sensor of the smartwatch and the information retrieved on the CSI of the WiFi devices, as the information that goes on the CSI is an estimation of the real value. In future work, the

accuracy of the heart rate measured in the CSI will be improved and, hopefully, this will make the two graphics overlap each other.

As the continuation of the tab on the page, we have the graphics of the breath rate, with respiration belt (blue line) and WiFi devices (gray line) data measured during the time interval selected, and below the respective averages. Identical to the heart rate graphics, the CSI information is an estimation of the real values which origins a difference of about 6 rpm between each point of the two graphics, being the respiration belt graphic the truthful data, and our guide. As has been said before, these CSI measures still need improvements to reach the desired line, which will overlap the respiration belt one.

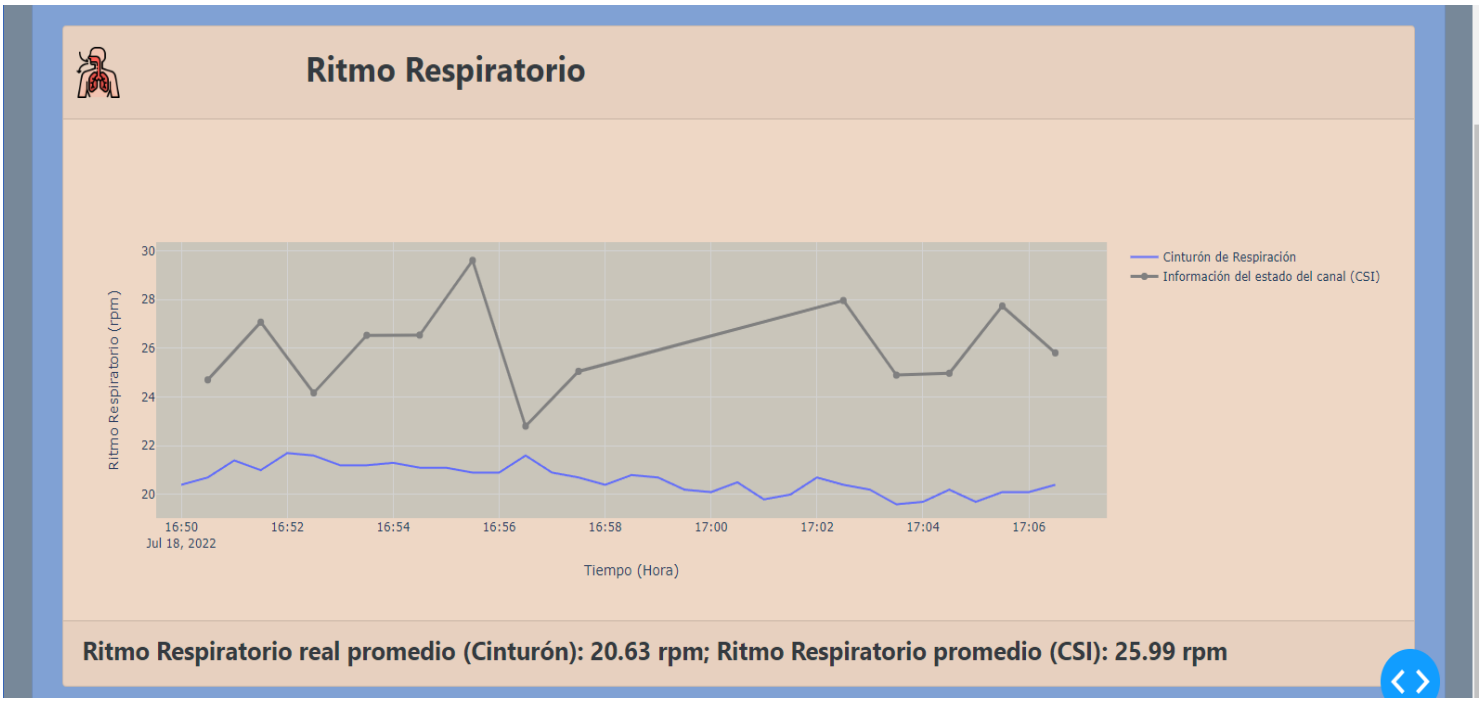


Figure 19: *Breathing Rate graphics of the two types of devices*

- Regarding the "Daily Information", this tab is the most complete one in terms of the overall information and daily record. In this tab, the user selects the day, and then it is presented with all the information that is being collected from the smartphone application and the smartwatch application. All this information will help the health responsible to detect any problem regarding the symptoms that the patient has been feeling, track the senior's activity throughout the day, or view his/her sleep routine:

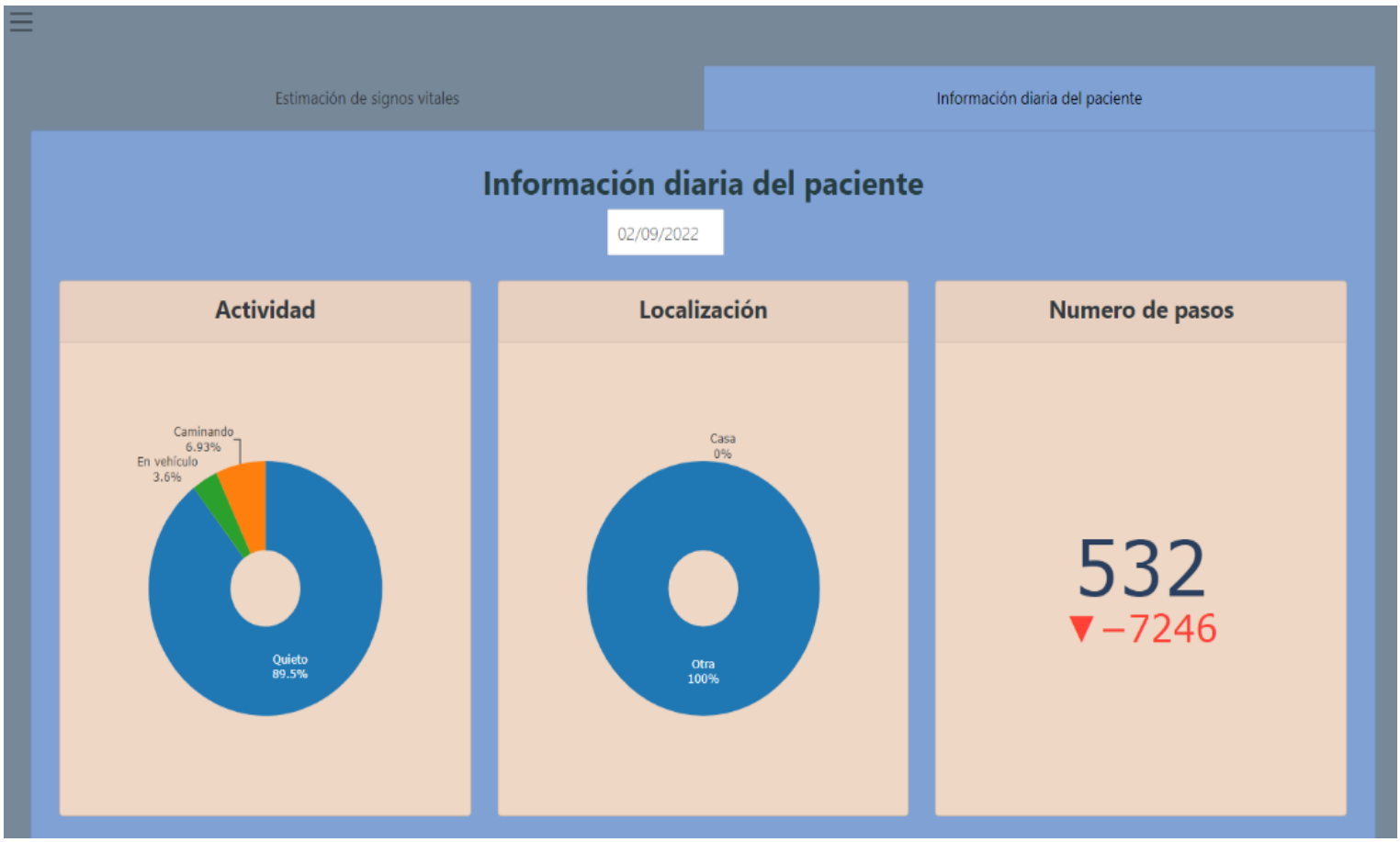


Figure 20: *Smartphone daily activity*

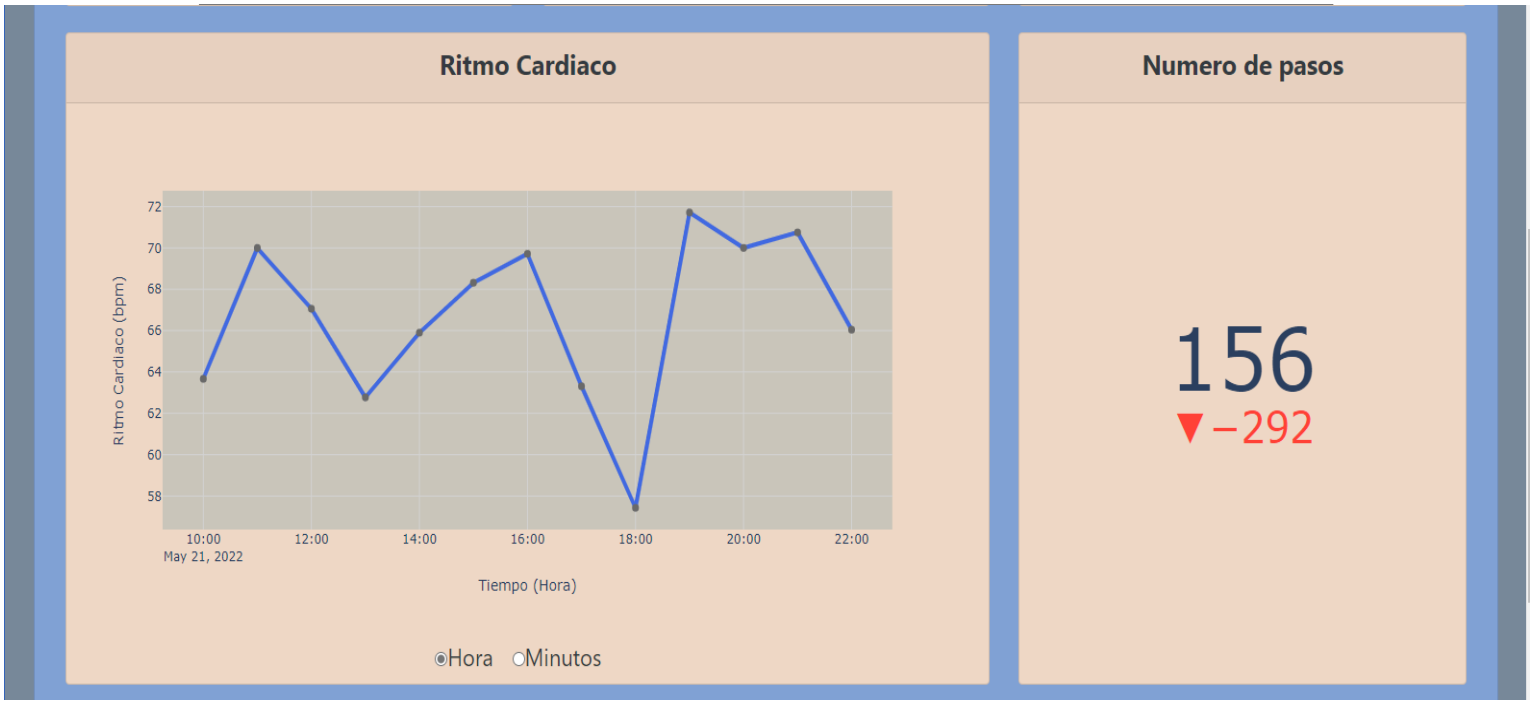


Figure 21: *Smartwatch daily metrics*

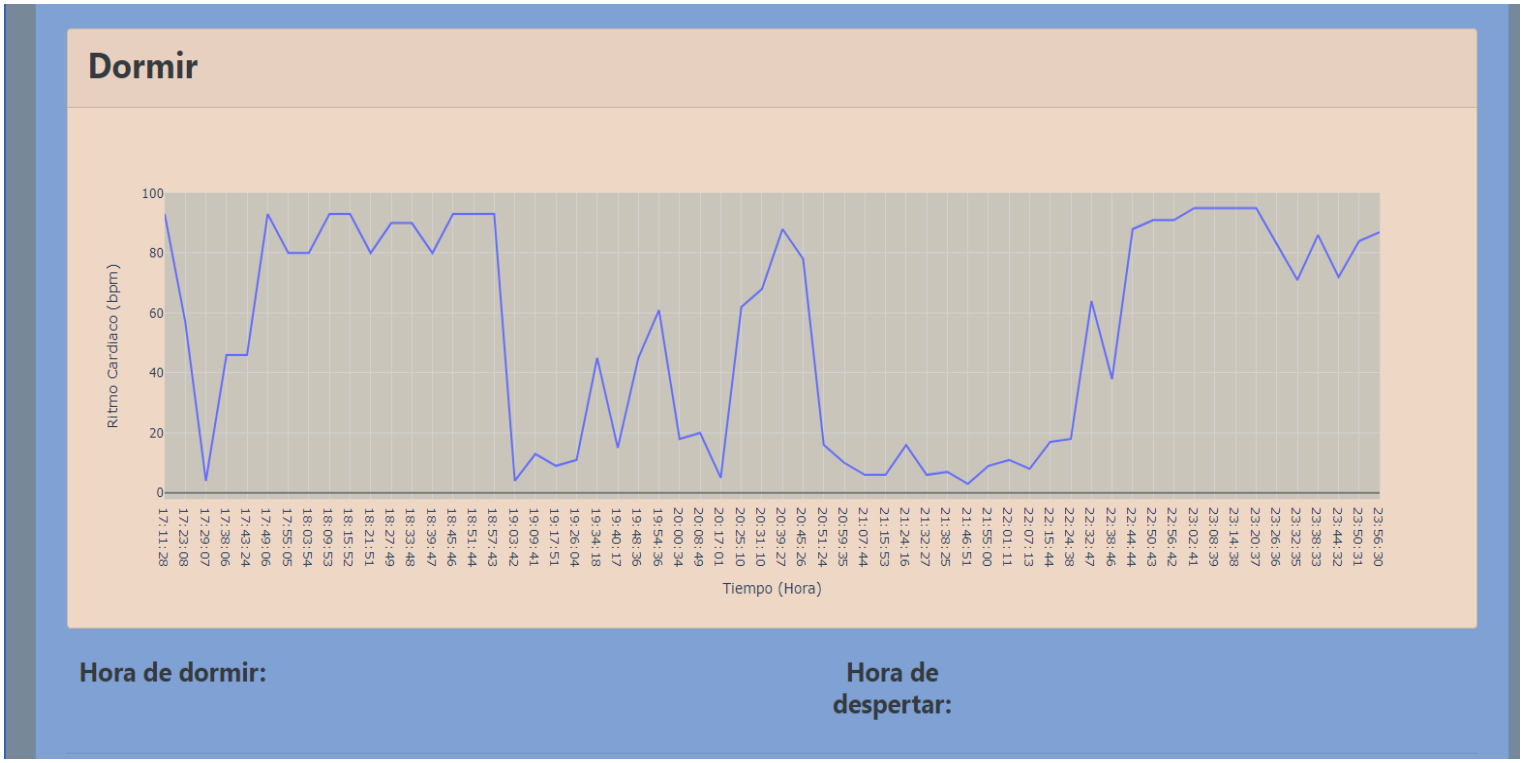


Figure 22: *Sleep Activity*

Figure 23: *Daily habits*

For the activity, localization, and free time layouts, we opted for circle graphics which can show all types of information with percentages, making it easier to analyze. Then, in figure 20, we can see a graphic of the heart rate of the selected day in which the user can see the graphic by the hour or by minute, to have an overall or detailed view of the data measured, and the number of steps taken during that day and the difference (in red) to the last 7 days average of steps per day taken with the smartwatch.

Figure 21 presents in a graphic the heart rate during sleep of the patient. This allows for to detection of spikes or exceedingly small intervals in which the heart can stop or skip a beat; this is quite common in elder people with heart problems. After in figure 22, there are the food and medication habits are tracked by the patient on the smartphone application.

5 - Tests and Results

As I had the work of developing the smartwatch application, one of the objectives was to improve and make sure that the application's battery life was extended, and that the accuracy of the data was robust.

For this, I made some tests on the battery life and sensors of the smartwatch, being these tests made at home, with myself wearing the smartwatch and keeping track of these two major variables in the development of the iFriend Wear application.

5.1 Battery Life

One of the major requirements in the wear applications is low battery consumption, especially in the area of health monitoring. As the sensors need to be available most of the time, the battery will drain a bit faster than usual.

So, since the beginning of the project, we took that into account and tried to develop the application in a way that did not affect the normal use of the watch.

As mentioned in the 1.4 section, the smartwatch used for these tests was the Motorola Moto 360SP with a battery of 300 mAh capacity, which gives up to more than a full day of battery life based on an average user profile that includes both usage and standby. The battery performance depends on network configuration, signal strength, operating temperature, features selected, voice, data, and other application usage patterns.

For these tests, we tried to approximate, as much as we could, the real ambient that is going to be implemented which is almost "no usage", since it is a monitoring application, at the ambient temperature and with only the Bluetooth turned on because this is the form of connection used to send the smartwatch's data to the Smartphone companion connected. These tests were made with the application running to see how much of the battery life would be affected by the background services and the time of sensors being turned on.

As was mentioned in the 4.1 section, the way of testing the battery life was to change the time that the sensors were turned on (Ton) and the time interval between measures (sensors turned off (Toff)), so the tests were divided into 3 scenarios of time intervals:

- Ton: 10 seconds
- Toff: 30 seconds

Date	Total test duration time	% of battery used
18/06/2022	8 hours	33
19/06/2022	10 hours	52
20/06/2022	10 hours	49
21/06/2022	14 hours	65
22/06/2022	16 hours	79
23/06/2022	16 hours	74

Table 2: *Battery life tests for 10 seconds (on) and 30 seconds (off)*

- Ton: 20 seconds
- Toff: 60 seconds

Date	Total test duration time	% of battery used
25/06/2022	8 hours	31
26/06/2022	8 hours	29
27/06/2022	16 hours	59
28/06/2022	22 hours	79
29/06/2022	24 hours	91
30/06/2022	24 hours	94

Table 3: *Battery life tests for 20 seconds (on) and 60 seconds (off)*

- Ton: 20 seconds
- Toff: 120 seconds

Date	Total test duration time	% of total battery used
10/07/2022	8 hours	25
11/07/2022	8 hours	28
12/07/2022	16 hours	55
13/07/2022	16 hours	52
14/07/2022	24 hours	81
15/07/2022	24 hours	85

Table 4: *Battery life tests for 20 seconds (on) and 120 seconds (off)*

As we can see, the differences between each table reflect the improvement of the battery life of the device just by increasing the time that the sensors were turned off. There's an average decrease, for a test duration of 8 hours, of the percentage of battery drained of about 9% from just the increase of the disconnecting sensors time interval from 30 to 60 seconds and 11% from 60 to 120 seconds. And for the test duration of 24 hours, there was a decrease of 6.5%, in the percentage of battery drained, from increasing the time interval that the sensors were disconnected from 30 to 60 seconds and 10.3% from 60 to 120 seconds.

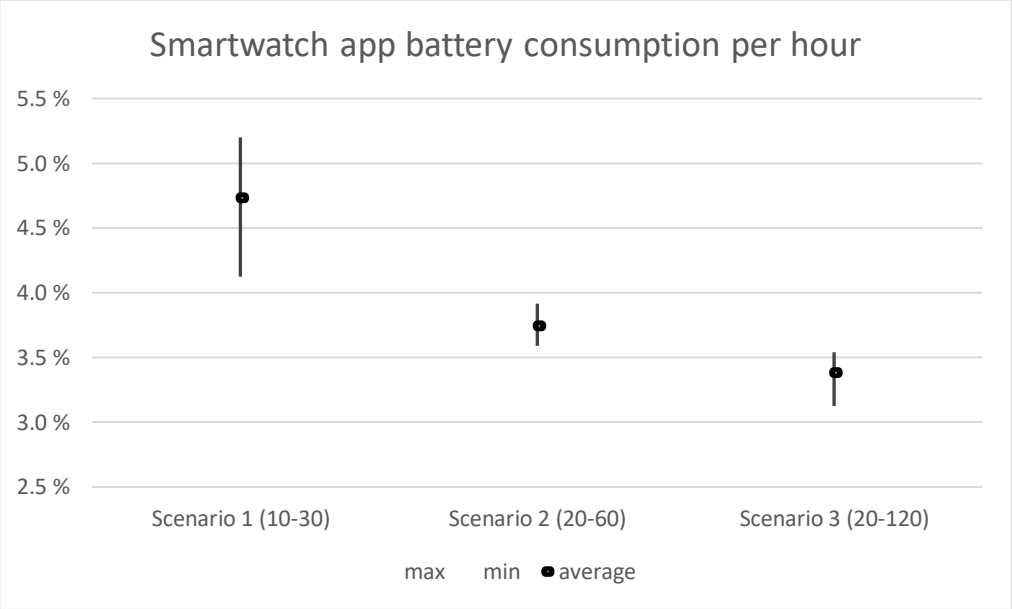


Figure 24: *Graphic with the 3 scenarios tested for the battery*

There was also made a graphic for the battery consumption per hour, in which we can see that there was a decrease in the life battery draining of about 1% by the hour from scenario 1 to scenario 2 and a decrease of about 0.4% by the hour from scenario 2 to scenario 3, as we can see on figure 22.

These “manual” tests were made to see, as well, what would be the ideal time, during the day, for the sensors to be turned on, which reflects on how many samples of heart rate were taken, to be able to present a consistent and solid graphic to the health professional. This is fairly connected with the accuracy sample tests that will be addressed in the next section.

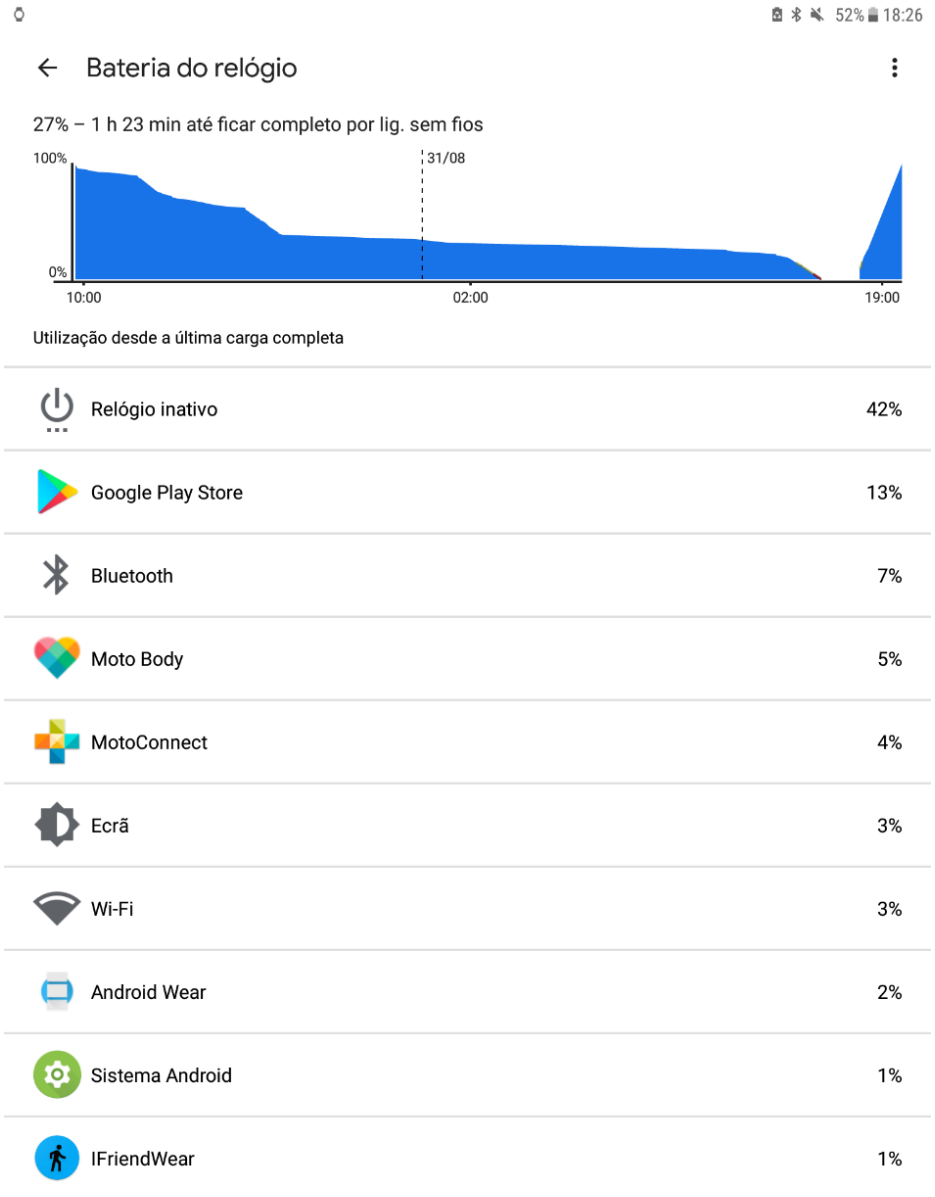


Figure 25: *Application battery usage*

5.2 Sensor's accuracy

The accuracy tests were developed and made while the battery life tests were made. These tests had the purpose of having the most accurate possible measurement of the heart rate and steps, by working with the property *SensorStatus* which gives the state of the sample's accuracy measured by the sensor of the smartwatch. The *SensorStatus* has the following constants: *AccuracyLow*, *AccuracyMedium*, *AccuracyHigh*, *NoContact*, and *Unreliable*. The first three constants are represented on a scale of 3 the level of accuracy of the sample being the lowest accuracy where calibration with the environment is needed, *AccuracyMedium* means the sensor is reporting data with an average level of accuracy, and calibration with the environment may improve the readings, and, finally, the highest accuracy means that the sensor is reporting with maximum accuracy. *NoContact* means that the values returned by the sensor cannot be trusted because the sensor had no contact with what is measuring (meaning that the smartwatch is not on the user's wrist). The *Unreliable* constant means that the values returned by the sensor cannot be trusted and calibration is needed, or the environment doesn't allow readings.

With these constants, we can set the sensor to only accept and save the readings that we want. So, for these tests, the sensor will only allow readings with the highest accuracy. In theory, we could also accept *Medium* accuracy readings but after testing these types of readings, we concluded that it was taking false readings, especially when the watch was on the wrist. So, by excluding all the non-high readings of the sensor, we made sure that the data that was being measured was solid and accurate, and that the watch was on the user's wrist. I would like to note that the accuracy of the step counter sensor was the highest constantly, so this specific sensor wasn't tested.

Regarding the heartbeat sensor, as was mentioned before, the accuracy readings were made at the same time as the battery life tests were made, so the duration time of each test was the same and for the same time intervals of section 5.1.

Our objective was to find how many samples with High accuracy were obtained during the duration of each test and how many samples were taken, on average, during each time the sensors were online. The time interval variables were the time that the sensors were turned on (Ton) and the time interval between measures (sensors turned off (Toff)), so the tests were divided into 3 groups of time intervals:

- Ton: 10 seconds
- Toff: 30 seconds

Date	Total test duration time	Accuracy High
18/06/2022	8 hours	198
19/06/2022	10 hours	316
20/06/2022	10 hours	310
21/06/2022	14 hours	460
22/06/2022	16 hours	510
23/06/2022	16 hours	567

Table 5: *Accuracy tests for 10 seconds (on) and 30 seconds (off)*

- Ton: 20 seconds
- Toff: 60 seconds

Date	Total test duration time	Accuracy High
25/06/2022	8 hours	238
26/06/2022	8 hours	254
27/06/2022	16 hours	859
28/06/2022	22 hours	1985
29/06/2022	24 hours	2095
30/06/2022	24 hours	2130

Table 6: *Accuracy tests for 20 seconds (on) and 60 seconds (off)*

- Ton: 20 seconds
- Toff: 120 seconds

Date	Total test duration time	Accuracy High
10/07/2022	8 hours	301
11/07/2022	8 hours	231
12/07/2022	16 hours	620
13/07/2022	16 hours	698
14/07/2022	24 hours	1330
15/07/2022	24 hours	1612

Table 7: *Accuracy tests for 20 seconds (on) and 120 seconds (off)*

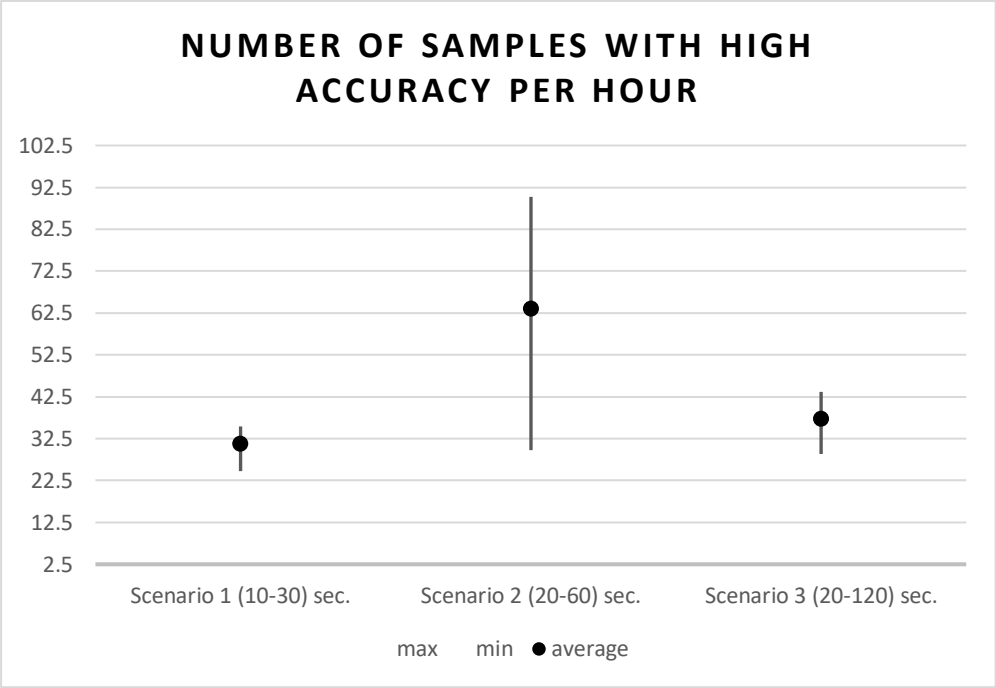


Figure 26: *Graphic with the 3 scenarios tested for the accuracy of the heart rate sensor*

After the analysis of the graphic in figure 26, we can conclude that by increasing the Ton interval from 10 to 20 seconds, we increased the number of samples taken during the day, and by increasing the Toff from 60 to 120 seconds we decrease the number of samples taken during the day which makes sense because the sensor will be turned on fewer times for the same time interval.

The sensor works in a way that when the surface and environment are constant during a big period, the sensor readings will stabilize, and on that time interval it's when it reads most samples with the highest accuracy. So, when the sensor stabilizes, for a Ton of 10 seconds, it reads on average 2 samples per time interval. And, for a Ton of 20 seconds, it reads on average 3.5 samples per time interval. This led us to the conclusion that the ideal Ton would be 20 seconds.

6 - Conclusion and Future work

6.1 Conclusion

With this project, we aimed to create a system that could monitor the senior's daily health habits, improve their lives and help the health responsible to better analyze the health situation of each patient. For that, we used the concepts of HITLCPS and WiFi monitorization and explored the concepts of IoT, CPS, and mobile/wear sensing. With those concepts, we evaluated how we could use them to create such a system, which we tried to make most completely.

With all these concepts we were able to develop a complex system to deliver the best information and tools for senior monitoring, which is very helpful in many fields, especially healthy ones. The main and innovative tool is WiFi monitorization which will provide a non-intrusive way of measuring the vital signs with the help of the smartwatch application and activity of each person with a simple system that still needs a lot of improvements, but it will have great success in the future. With the mobile application, the user will be able to track their habits which will be easily related to the rest of the information that is being collected.

We can conclude that in the end, the main objectives of this project were all accomplished and, we believe, with the advance and improvements that will be made this system can be easily implemented in real cases, like the one we have planned.

Also, I can conclude with certainty that this project had also a great improvement in my personal development, whether with all the concepts that I've learned but also gave me a lot of work ethic, which I believe will help me in the future.

6.2 Future work

As it was mentioned before, this project still needs some improvements to be implemented in a real situation. With the work of other Universities, hopefully, we can reach the main goal which is testing this system with real patients in a hospital.

Regarding the WiFi monitorization, as it was shown in the 4.2.3 section, the accuracy of the data that is being measured is not where we wanted it to be, but by using the respiration belt and the smartwatch as our guides, we can reach solid values that will reflect the real ones. This will allow the system to be much more accurate and ready to be implemented because it's all about the correct monitoring of the patient. And of course, these results will impulse the vital sign WiFi monitorization to other implementations like multi-room presence detection.

As for the smartwatch application that was developed, some things still need some work and improvement:

- Adding the activity recognition through the sensors of the device will allow us to better tell what type of activity the person is doing - this will help as well to corroborate the data that is being measured whether by the smartphone as well as the WiFi devices.
- Another thing that can be added to the equation is the blood oxygen levels, also known as SpO_2, which will be measured through the smartwatch sensors, which will be added as one of the major variables being collected for the vital signs, especially by helping with the readings of the breathing rate of the patient.
- Since this app is developed in a cross-platform architecture (Xamarin), the iOS application still needs to be developed and improved in the future.

Our dashboard application is nearly complete but still needs some things that can be improved and tested before being deployed, like the connections with the servers that can be made through some performance or health check testing. Also, we need to improve our security, by retrieving the data through DTO, encrypting all the data, and applying authentication/access control. Another thing that needs to be improved, which is one of the most necessary ones, is the live tab of daily information since our data is not received and shown in "real-time" because there's a layer of delays regarding the communication, especially because it is required Internet connection to send the data to the main database.

In conclusion, the improvements above will require some extra work and research but once implemented will allow this project to be implemented on real patients.

Bibliography

[1] "Rider: The Cross-Platform .NET IDE from JetBrains." https://www.jetbrains.com/rider/ (accessed Sep. 07, 2022).

[2] "PyCharm: the Python IDE for Professional Developers by JetBrains." https://www.jetbrains.com/pycharm/ (accessed Sep. 07, 2022).

[3] "What Is Agile Scrum Methodology? - businessnewsdaily.com." https://www.businessnewsdaily.com/4987-what-is-agile-scrum-methodology.html (accessed Sep. 07, 2022).

[4] "Where work happens | Slack." https://slack.com/ (accessed Sep. 07, 2022).

[5] "Skype | Stay connected with free video calls worldwide." https://www.skype.com/en (accessed Sep. 07, 2022).

[6] "Number of Internet of Things (IoT) Connected Devices Worldwide 2022/2023: Breakdowns, Growth & Predictions - Financesonline.com." https://financesonline.com/number-of-internet-of-things-connected-devices/ (accessed Sep. 07, 2022).

[7] "What are Cyber-Physical Systems? | Cyber-Physical Systems Research Center." https://cps.soe.ucsc.edu/ (accessed Sep. 07, 2022).

[8] G. Schirner, D. Erdogmus, K. Chowdhury, and T. Padir, "The future of human-in-the-loop cyber-physical systems," *Computer (Long Beach Calif)*, vol. 46, no. 1, pp. 36–45, 2013, doi: 10.1109/MC.2013.31.

[9] J. Liu, Y. Wang, Y. Chen, J. Yang, X. Chen, and J. Cheng, "Tracking vital signs during sleep leveraging off-the-shelf WiFi," in *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Jun. 2015, vol. 2015-June, pp. 267–276. doi: 10.1145/2746285.2746303.

[10] "Android | A plataforma que redefine tudo o que é possível." https://www.android.com/intl/pt_pt/ (accessed Sep. 07, 2022).

[11] "Linux Foundation - Decentralized innovation, built with trust." https://www.linuxfoundation.org/ (accessed Sep. 07, 2022).

[12] "Wear OS Smartwatches." https://wearos.google.com/#uniquely-you (accessed Sep. 07, 2022).

[13] "FIWARE, the standard that the IoT needs | TM Forum." https://www.tmforum.org/press-and-news/fiware-standard-iot-needs/ (accessed Sep. 07, 2022).

[14] "Build your own IoT platform with FIWARE enablers | FIWARE." https://www.fiware.org/2015/03/27/build-your-own-iot-platform-with-fiware-enablers/ (accessed Sep. 08, 2022).

[15] K. Hinsen, K. Läufer, and G. K. Thiruvathukal, "Essential tools: Version control systems," *Comput Sci Eng*, vol. 11, no. 6, pp. 84–91, Nov. 2009, doi: 10.1109/MCSE.2009.194.

[16] "What is Xamarin? - Xamarin | Microsoft Docs." https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin (accessed Sep. 07, 2022).

[17] "✳ Introducing Dash ✳. Create Reactive Web Apps in pure Python | by plotly | Plotly | Medium." https://medium.com/plotly/introducing-dash-5ecf7191b503 (accessed Sep. 07, 2022).

[18] "Go Direct® Respiration Belt - Vernier." https://www.vernier.com/product/go-direct-respiration-belt/ (accessed Oct. 10, 2022).

[19] "Developers Catalogue | FIWARE." https://www.fiware.org/catalogue/ (accessed Sep. 08, 2022).

[20] "MongoDB: The Developer Data Platform | MongoDB | MongoDB." https://www.mongodb.com/ (accessed Sep. 07, 2022).

[21] "MySQL." https://www.mysql.com/ (accessed Sep. 07, 2022).

[22] "Dash is Deeper than Dashboards. The missing link that makes Python a… | by Layne Sadler | Better Programming." https://betterprogramming.pub/dash-is-deeper-than-dashboards-5ab7414f121e (accessed Sep. 07, 2022).

[23] "Manifest.permission | Android Developers." https://developer.android.com/reference/android/Manifest.permission (accessed Sep. 08, 2022).

[24] "Request app permissions | Android Developers." https://developer.android.com/training/permissions/requesting (accessed Sep. 09, 2022).

Appendices

A

List of Permissions

In this appendix, we present a list of all the permissions used by the wear application [23] as well as an explanation as to why we use each one of them [24]. We are aware that some of these permissions raise privacy issues and as such, we believe that the transparency of why and how we used them is very important.

- **INTERNET:** This permission is necessary to access the Internet from the application. We may need this permission to communicate with the smartphone to send the wear data if requested.

- **ACCESS WIFI STATE:** To get the list of the configured networks, the WiFi needs to be turned on. This makes it easier to find a connection if one is needed.

- **ACCESS_FINE_LOCATION:** It allows to access the precise location of the user. This permission isn't being currently used, but we pretend to add location as part of the monitorization in future work.

- **ACTIVITY_RECOGNITION:** This permission is necessary to use the Google Activity Recognition API.

- **FOREGROUND_SERVICE:** It shows a status bar notification so that users are actively aware that the app is performing a task in the foreground and is consuming system resources.

- **BATTERY_STATS:** It allows to verify the state of the battery. This permission was specifically used for the battery tests.

- **BODY_SENSORS:** Allows the application to access data from sensors that in this case, we use to measure the heart rate.